# On Non-Computable Functions

**By T. RADO**

*(Manuscript received November 12, 1961)*

**Abstract**

*The construction of non-computable functions used in this paper is based on the principle that a finite, non-empty set of non-negative integers has a largest element. Also, this principle is used only for sets which are exceptionally well-defined by current standards. No enumeration of computable functions is used, and in this sense the diagonal process is not employed. Thus, it appears that an apparently self-evident principle, of constant use in every area of mathematics, yields non-constructive entities.*

## I Introduction

The purpose of this note[1] is to present some very simple instances of non-computable functions. Beyond their simplicity, these examples throw light upon the following basic point. If a function $f(x)$ is to serve as an example of a non-computable function, then $f(x)$ must be *well-defined* in some generally accepted sense; hence the efforts to construct examples of non-computable functions reveal the general conviction that over and beyond the class of computable (general recursive) functions there is a much wider class, the class of *well-defined functions*. The scope of this latter class is vague; in some quarters, there exists a belief that this class will be defined some day in precise terms acceptable to all. The examples of non-computable functions to be discussed below will be well defined in an extremely primitive sense; we shall use only the principle that a non-empty finite set of non-negative integers has a largest element. Furthermore, we shall use this principle only for exceptionally well-defined sets; and thus our construction will rest upon considerations which occur constantly in every area of mathematics. It may be of interest to note that we shall not use an enumeration of computable functions to show that our examples are non-computable functions. Thus, in this sense, we do not use the diagonal process.

## II Terminology

We shall use binary Turing machines (that is, Turing machines with the binary alphabet 0, 1), in the sense of the excellent presentation of Kleene's *Metamathematics* (see Ref.), with the following exceptions. First, we do not permit a center shift; thus the machine must shift after the execution of an "overprint" instruction (the purpose is to simplify the following presentation). Second, we shall use the term "card" instead of "state". The reason is that the examples below were obtained as by-products of a logical game (the Busy Beaver game described below) which the writer made up to familiarize beginners with the idea of a Turing machine; and it appeared that terms such as state, internal configuration, and the like had a mysterious connotation for beginners. To illustrate some notational conventions to be used, let us consider the following example of a binary, 3-card Turing machine.

[1] Transcribed in March 2019 from the scanned original found at `https://ia801900.us.archive.org/0/items/bstj41-3-877/bstj41-3-877.pdf` by Sidney Cadot (`mailto:sidney@jigsaw.nl`). All footnotes and appendices were added during transcription; they are not part of the original manuscript.

| $C_1$ | |
|---|---|
| 0 | 102 |
| 1 | 113 |

| $C_2$ | |
|---|---|
| 0 | 111 |
| 1 | 102 |

| $C_3$ | |
|---|---|
| 0 | 112 |
| 1 | 100 |

Here $C_1$, $C_2$, $C_3$ stand for Card 1, Card 2, and Card 3. On each card, the left-most column contains the alphabet $0, 1$. The next column is the "overprint by" column; the next one is the "shift" column (where 0 is the code for a left shift and 1 is the code for a right shift). The last column is the "call card" column; it contains the index of the next card to be used, or 0 (zero), where 0 is the code for "Stop." This notation was found very convenient in situations where one wanted to enumerate (serialize) Turing machines with a given number of cards.

The reader is assumed to be familiar[2] with the meaning (in the sense of Kleene; see Ref.) of the statement that a binary Turing machine "computes" a function $f(x)$. It is understood that we consider only functions of non-negative integers with values which are again non-negative integers.

## III The Busy Beaver game

Consider a potentially both-ways infinite tape (see Ref.), where each square contains a 0 (all-zero tape). Start the 3-card machine described in Section II (with its Card 1) under any square. The reader will find that the machine stops after a few shifts[3], and when it stops, there are six ones on the tape. Actually, this particular machine is one of the four highest scorers (as of today) in the international BB-3 game (the 3-card deck classification of the Busy Beaver game). The rules in this game are as follows.

*i.* The contestant selects a positive integer $n$; and then makes up his own $n$-card, binary, Turing machine (using the notational conventions explained in Section II).

*ii.* He starts his machine (with its Card 1) on an all-zero tape, and satisfies himself that his machine stops after a certain number $s$ of shifts.

*iii.* He then submits his entry, as well as the shift-number $s$, to any member (in good standing) of the International Busy Beaver Club.

*iv.* The umpire first verifies that the entry actually stops exactly after $s$ shifts. If the entry fails to stop after $s$ shifts, it is rejected; if it stops after fewer than $s$ shifts, it is returned to the contestant for correction. After the entry has been verified, its score is the number of ones on the tape when it stops.

Naturally, the BB-$n$ champion is the contestant who achieved the highest score (so far) in the BB-$n$ classification. For example, in the BB-3 classification, the score of 6 was first achieved by R. Hegelman (U.S. Naval Weapons Laboratory, Dahlgren, Virginia). This score has been reached since by several others; but nobody knows as yet whether 6 is the highest possible score in the BB-3 classification[4]. The reader who tries to settle this question will soon realize the difficulties involved in this sort of problem. Beyond the enormous number of cases to survey, he will find that it is very hard to see whether certain entries do stop at all. This is the reason for the requirement that each contestant must submit the shift number $s$ with his entry.

## IV Highest score

There arises now the problem of determining the highest possible score in the BB-$n$ classification. In line with the point of view explained in the introduction, we formulate this problem with due care and caution.

[2]The necessary ideas are summarized in Appendix A.II.

[3]The behavior of this example Turing machine is shown in Appendix A.III.

[4]The score 6 for the BB-3 classification was proven optimal by Radó's student Lin in his doctoral thesis (1963).

Returning to rule *iv.* of the game, we see that a valid entry in the BB-$n$ classification is a pair $(M, s)$, such that the following holds.

(a) $M$ is an $n$-card binary Turing machine.

(b) $s$ is a positive integer.

(c) $M$ stops after exactly $s$ shifts if started (with its Card $C_1$) on an all-zero tape.

In discussing rule *iv.* above, we noted that we can actually decide whether or not an entry $(M, s)$ is valid. Also, if $(M_1, s_1)$, $(M_2, s_2)$ are valid entries such that $M_1 = M_2$, then evidently $s_1 = s_2$; hence the number of valid BB-$n$ entries cannot exceed the number $N(n)$ of all possible $n$-card, binary Turing machines. It is easy to see that

$$N(n) = [4(n+1)]^{2n} \tag{1}$$

Also, there exist valid BB-$n$ entries; for example, on choosing the 0-line of Card 1 as 110, one obtains an entry which stops after one shift.

Accordingly, if we denote by $E_n$ the set of all valid BB-$n$ entries $(M, s)$, we obtain a non-empty, finite set $E_n$ which has the following features.

(a) We actually exhibit elements of $E_n$; so $E_n$ is non-empty as a matter of concrete observation.

(b) We not only know that $E_n$ is finite, but for the number $N_e(n)$ of elements of this set of valid entries we have [see (1)] the inequalities.

$$1 < N_e(n) < N(n) = [4(n+1)]^{2n} \tag{2}$$

(c) For every pair $(M, s)$ we can actually decide whether $(M, s) \in E_n$.

Evidently, $E_n$ is (by current standards) an *exceptionally well-defined* non-empty, finite set. Yet, we shall show below that $N_e(n)$, the number of elements of $E_n$, is not a computable function of $n$. Next, each valid entry $(M, s) \in E_n$ has a definite score $\sigma(M, s)$ assigned to it (see Section III)[5]. Thus, for the same reasons, the set of these scores is an exceptionally well-defined non-empty finite set of non-negative integers. We denote by $\Sigma(n)$ the largest element of this set.
Thus

$$\Sigma(n) = \max\,[\sigma(M, s)] \text{ for } (M, s) \in E_n. \tag{3}$$

We shall see presently that $\Sigma(n)$ is not a computable function of $n$. Let us note, however, that it is entirely possible that $\Sigma(n)$ can be effectively determined for particular values of $n$. For example, evidently $\Sigma(1) = 1$. Also, it has been proved that $\Sigma(2) = 4$. We noted above that we know several BB-3 entries with a score of 6; hence $\Sigma(3) \geqq 6$, and it seems plausible[6] that $\Sigma(3) = 6$. Now while for low values of $n$ it is quite hard to achieve a respectable score, Dr. C. Y. Lee observed (in a letter to the writer) that for higher values of $n$ one can achieve very large scores. The following proof for the non-computability of $\Sigma(n)$ was obtained by developing this comment of Dr. Lee.

# V The growth of $\Sigma(n)$

Let $f(x), g(x)$ be two functions (as specified in Section II). We shall write

$$f(x) \rhd g(x)$$

to state that $f(x) > g(x)$ for $x$ greater than a certain $x_0$. Using this notation[7], we shall now prove the following theorem.

[5] Recall that rule *iv.* of the Busy Beaver game defines the score as *the number of ones on the tape when it stops*.

[6] As of March 2019, it is known that $\Sigma(3) = 6$, $\Sigma(4) = 13$, and $\Sigma(5) \geqq 4098$.

[7] Radó writes $f(x) >\!-\, g(x)$, see Appendix A.I. We recommend to pronounce either as '$f(x)$ dominates $g(x)$'.

*Theorem*. $\Sigma(n) \rhd f(n)$ for every computable (that is, general recursive) function $f(n)$. Hence $\Sigma(n)$ is not computable.

*Proof*. Assign a computable function $f(x)$. Introduce the auxiliary function

$$F(x) = \sum_{i=0}^{x} [f(i) + i^2]. \tag{4}$$

Then (see Ref.) $F(x)$ is also computable. Evidently[8]

$$F(x) \geqq f(x). \tag{5}$$

$$F(x) \geqq x^2. \tag{6}$$

$$F(x+1) > F(x). \tag{7}$$

Now since $F(x)$ is computable, we have a binary Turing machine $M_F$, with a certain number $C$ of cards (states) which computes $F(x)$ (in the sense described in Kleene; see Ref.). Now assign any integer $x \geqq 0$. We have then a binary Turing machine $M^{(x)}$, with $x+1$ cards (states) which prints on an all-zero tape $x+1$ consecutive ones and stops under the right-most one of these. For $x = 2$, for example, $M^{(2)}$ has the three cards:

| $C_1$ | |
|---|---|
| 0 | 112 |
| 1 | 110 |

| $C_2$ | |
|---|---|
| 0 | 113 |
| 1 | --- |

| $C_3$ | |
|---|---|
| 0 | 101 |
| 1 | --- |

Now consider the binary Turing machine $M_F^{(x)}$ given by the symbolic diagram:

$$M_F^{(x)} : M^{(x)} \to M_F \to M_F.$$

If the cards of $M_F^{(x)}$ are written out with consecutive indices, then it is seen to have $1 + x + 2C$ cards. If started on an all-zero tape, $M_F^{(x)}$ will first print (going to the right) a string of $x+1$ consecutive ones; then, beyond a 0 to the right, it will print a string of $F(x)+1$ consecutive ones; finally, beyond a 0 to the right, it will print a string of $F[F(x)]+1$ consecutive ones, and then will stop (under the right-most 1 it printed). Thus evidently, $M_F^{(x)}$ is a valid entry[9] in the BB-$(1+x+2C)$ classification with a score equal to

$$3 + x + F(x) + F[F(x)].$$

Hence, the maximum score $\Sigma(1+x+2C)$ in this classification satisfies the inequality

$$\Sigma(1+x+2C) \geqq 3 + x + F(x) + F[F(x)]. \tag{8}$$

Now since evidently $x^2 \rhd (1+x+2C)$ and $F(x) \geqq x^2$ [see (6)], it follows that

$$F(x) \rhd (1+x+2C). \tag{9}$$

Also, $F(x)$ is monotonically increasing by (7); hence (9) yields

$$F[F(x)] \rhd F(1+x+2C). \tag{10}$$

From (8) and (10) we see that

$$\Sigma(1+x+2C) \rhd F(1+x+2C);$$

hence (since $F(x) \geqq f(x)$)

$$\Sigma(1+x+2C) \rhd f(1+x+2C).$$

[8]Recall that Section II states that *we consider only functions of non-negative integers with values which are again non-negative integers.* With that in mind, inequalities (5)—(7) are indeed evident from definition (4).

[9]Radó's original manuscript has $N_F^{(x)} s$ here instead of $M_F^{(x)}$, which is a typo.

On setting $n = 1 + x + 2C$, we obtain finally

$$\Sigma(n) \rhd f(n).$$

and the theorem is proved.

The rate at which $\Sigma(x)$ grows is illustrated by the following intuitive observation. A Turing machine $M_H$ for computing $H(x) = x!$ can be constructed with no more than 26 states. Let us consider the chain of Turing machines:

$$M^{(x)} \to M_H \to M_H \to M_H \to M_H.$$

It follows from (8) that the number of ones which is produced by this chain is more than $(((x!)!)!)!$. Using the construction of the machine $M_H$ mentioned above, we may show that by combining these machines properly, the number of states required for this chain of machines for $x = 7$, for instance, is no more than 100. Therefore, $\Sigma(100)$ is at least $(((7!)!)!)!$. Since $\Sigma(100)$ is probably far bigger than this lower bound, it would be interesting to know how large a lower bound one can get for $\Sigma(100)$.

# VI The function $S(n)$

It is evident from our definitions that the set $E_n$ of valid BB-$n$ entries coincides with the set of the $n$-card stoppers, where by a stopper we mean a (binary) Turing machine which, if started on an all-zero tape with its card $C_1$, will stop after a while. Now the second coordinates $s$ of the valid BB-$n$ entries $(M, s)$ constitute a finite, non-empty set of positive integers; we denote by $S(n)$ the largest element of this set. Thus $S(n)$ is the maximum of the shift numbers of the $n$-card stoppers. Clearly

$$S(n) \geqq \Sigma(n). \tag{11}$$

Indeed, since we do not permit center-shifts, a BB-$n$ entry must shift after it prints a 1; thus (11) is obvious. From the theorem in Section V and from (11) we see that

$$S(n) \rhd f(n) \tag{12}$$

for every computable function $f(n)$. Thus $S(n)$ is non-computable (the reader will readily see that this result is equivalent to the undecidability of the so-called halting problem[10]).

# VII The function $N_e(n)$

This function, defined above as the number of elements of the set $E_n$ (that is, the number of $n$-card stoppers) does not grow unreasonably fast [see (2)]. However, we can discuss it as follows. Let us denote by $N(s, n)$ the number of those BB-$n$ entries which stop after exactly $s$ shifts. Evidently, the computation of $N(s, n)$ can be readily programmed; informally, one finds the value of $N(s, n)$ by running each one of the $n$-card binary Turing machines [whose number is given by (1)], persisting through no more than the given number $s$ of shifts, and noting the number of those that stop after exactly $s$ shifts. Let us put

$$G(s, n) = \sum_{i=1}^{s} N(i, n), \tag{13}$$

$$\Phi(s, n) = N_e(n) - G(s, n). \tag{14}$$

[10] To be clear: it is stated here that the non-computability of $S(n)$ is equivalent to the undecidability of the halting problem, which can indeed be shown rather easily, as we do in Appendix A.VI. It is not claimed that it is easy to see that the fact that $S(n) \rhd f(n)$ is equivalent to the undecidability of the halting problem. That statement is rather less obvious, as it is not immediately clear how the undecidability of the halting problem implies directly that $S(n)$ needs to dominate any computable function.

Clearly, $G(s,n)$ is the number of those BB-$n$ entries that stop after not more than $s$ shifts; thus $G(s,n) \leqq N_e(n)$, and hence $\Phi(s,n) \geqq 0$. Since evidently $G(s,n) = N_e(n)$ for $s = S(n)$, we see that $S(n)$ is the smallest value of $s$ for which $\Phi(s,n) = 0$; in symbols:

$$S(n) = (\mu s)[\Phi(s,n) = 0], \tag{15}$$

where $(\mu s)$ means "the smallest $s$ such that." From (13)–(15) it follows (see Ref.) that if $N_e(n)$ were computable then $S(n)$ would be computable too; since we know that $S(n)$ is not computable, it follows that $N_e(n)$ is non-computable.

### VII.I Remark

Suppose that, for a certain integer $n_0$, we somehow succeeded in determining the exact value of $N_e(n_0)$. From (13)–(15) it follows that we can then determine $S(n_0)$ also, and hence finally $\Sigma(n_0)$. Various other comments will readily occur to the reader. For example, the easily proved inequality

$$S(n) \leqq (n+1)\Sigma(5n)2^{\Sigma(5n)}$$

gives rise to some curious observations.

## VIII Summary

Inspection of the preceding presentation shows that we used in our constructions only the following "principle of the largest element": If $E$ is a non-empty, finite set of non-negative integers, then $E$ has a largest element. This principle is used constantly, as a matter of course, in every field of mathematics. Our examples above show that this principle, even if applied only to *exceptionally well-defined sets* $E$, may take us beyond the realm of constructive mathematics. Of course, common everyday experiences may be used to illustrate this sort of phenomenon. For example, when the writer wanted to find a certain highway on an automobile trip, he received the following directions from the foreman of a construction crew: "Drive straight ahead on this road; you will cross some steel bridges; and after you cross the last steel bridge, make a left turn at the next intersection." Luckily, the unsolvable problem implied by this advice was resolved by a member of the construction crew who volunteered the information that "after you cross the last steel bridge, there isn't another steel bridge until you reach Richmond, 130 miles away." The reader may find it amusing to verify, by detailed study of the excellent book of Kleene (Ref.), that this little story illustrates, in a concrete manner, some truly basic points in the theory of computable functions.

## IX Acknowledgement

The writer takes pleasure in thanking Dr. C. Y. Lee (of Bell Telephone Laboratories) for a number of stimulating comments.

## Reference

1. Kleene, S. C., *Introduction to Metamathematics*, D. Van Nostrand Co., Princeton, N. J., 1952.

# A Appendix: Transcription notes

The appendices that follow were added while transcribing the original manuscript.

## A.I List of changes to the manuscript

The following changes were made to the manuscript in the process of transcription:

- In Section V, Radó's original manuscript uses '$f(x) >\!- \; g(x)$' rather than '$f(x) \rhd g(x)$'. The change was made to improve readability.
- In Section V, Radó accidentally writes $N_F^{(x)}$ where $M_F^{(x)}$ is clearly intended. This was corrected.
- Footnotes and appendices were added to the manuscript to give updated information, small hints, and some missing easy proofs that, we hope, help to understand the manuscript.

## A.II Computation on a binary Turing machine

In this manuscript, in particular in Section V, Radó follows Kleene's convention on how to represent sequences of nonnegative integers on a binary tape, and how computation is carried out. Here, we summarize these conventions to the extent needed to understand the manuscript.

An integer $x$ is represented by $x + 1$ consecutive '1' symbols; this allows the number 0 to be properly represented, as a single '1' symbol. Multiple integers are separated by a single '0' symbol on the tape.

At the start of a computation, a Turing machine points to a certain integer by positioning the tape on the right-most '1' symbol of that number. Applying a function $f$ to $x$, the last (rightmost) number in the sequence of numbers present on the tape, starts with the tape positioned on the rightmost '1' symbol of the tape (this pointing to $x$), and then running a Turing machine $M_f$ until it halts.

After completing its computation, the Turing machine must leave the tape to the left of its initial starting point unaffected, i.e., $x$ and the numbers preceding it will remain on the tape. To the right of its initial starting point, it must have added the representation of $f(x)$, separated from the representation of $x$ by a single '0' symbol, and it should have repositioned the tape to point at the rightmost '1' symbol of the number $f(x)$ just calculated. The tape to the right of its end position should be all-zero.

## A.III Structure and behavior of the example Turing machine

Figure 1 shows the state transition diagram for the example Turing machine presented in Section II. The machine starts with card $C_1$ on an all-blank tape. Each of the non-halting states has two outgoing edges. These are marked 'symbol→overwrite;direction', where *symbol* is the symbol read from the current position of the tape (0 or 1), *overwrite* is the value to be written on the tape at the current position before moving (0 or 1), and *direction* gives the direction the tape head should move to after the overwrite step (L for left, or R for right). Note that this particular machine specifies an overwrite of a '1' symbol on each transition.

Figure 2 shows the behavior of the example Turing machine when started from an all-blank tape.

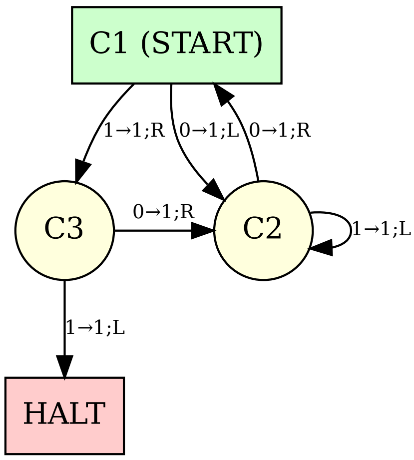


Figure 1: State transition graph for the example Turing machine.

```
 0:  0  0  0 <0> 0  0  0  0    C1
 1:  0  0 <0> 1  0  0  0  0    C2
 2:  0  0  1 <1> 0  0  0  0    C1
 3:  0  0  1  1 <0> 0  0  0    C3
 4:  0  0  1  1  1 <0> 0  0    C2
 5:  0  0  1  1  1  1 <0> 0    C1
 6:  0  0  1  1  1 <1> 1  0    C2
 7:  0  0  1  1 <1> 1  1  0    C2
 8:  0  0  1 <1> 1  1  1  0    C2
 9:  0  0 <1> 1  1  1  1  0    C2
10:  0 <0> 1  1  1  1  1  0    C2
11:  0  1 <1> 1  1  1  1  0    C1
12:  0  1  1 <1> 1  1  1  0    C3
13:  0  1 <1> 1  1  1  1  0    HALT
```

Figure 2: The thirteen steps it takes for the example Turing machine to halt.

## A.IV Four basic lemmas about $f(x) \rhd g(x)$

In Section V, there are a number of steps in the proof that Radó takes as evident, which may however not be so evident to the reader. We state and prove them below, and refer to them as Lemmas A—D in Appendix A.V.

*Lemma A*: If $f(x) \rhd g(x)$, and $h(x) \geqq f(x)$, then $h(x) \rhd g(x)$.

*Proof of Lemma A:*

Since $f(x) \rhd g(x)$, there exists a value $x_0$ such that, for all $x > x_0$, $f(x) > g(x)$. Since $h(x) \geqq f(x)$, it also follows that, for all $x > x_0$, $h(x) > g(x)$. This means that $h(x) \rhd g(x)$, which concludes the proof.

*Lemma B:* If $f(x) \rhd g(x)$, and $g(x) \geqq h(x)$, then $f(x) \rhd h(x)$.

*Proof of Lemma B:*

Since $f(x) \rhd g(x)$, there exists a value $x_0$ such that, for all $x > x_0$, $f(x) > g(x)$. Since $g(x) \geqq h(x)$, it also follows that, for all $x > x_0$, $f(x) > h(x)$. This means that $f(x) \rhd h(x)$, which concludes the proof.

*Lemma C:* If $f(x) \rhd g(x)$, and $h(x)$ is an increasing function, i.e., $x_2 > x_1 \iff h(x_2) > h(x_1)$, then $h[f(x)] \rhd h[g(x)]$.

*Proof of Lemma C:*

Since $f(x) \rhd g(x)$, there exists a value $x_0$ such that, for all $x > x_0$, $f(x) > g(x)$. By virtue of the fact that the function $h(x)$ is increasing, we then also know that, for all $x > x_0$, $h[f(x)] > h[g(x)]$. This means that $h[f(x)] \rhd h[g(x)]$, which concludes the proof.

*Lemma D:* If $f(x + \alpha) \rhd g(x + \alpha)$ for some value $\alpha \geqq 0$, then $f(x) \rhd g(x)$.

*Proof of Lemma D:*

Since $f(x+\alpha) \rhd g(x+\alpha)$, there is some value $x_0$ for which, for all $x > x_0$ , $f(x+\alpha) > g(x+\alpha)$. This is precisely the same as saying that $f(x) > g(x)$ for any $x > (\alpha + x_0)$. This means that $f(x) \rhd g(x)$, which concludes the proof.

## A.V Structure of the proof in Section V

Figure 3 on the next page gives a graphical representation of the central argument of the manuscript, as presented in Section V, where it is proven that $\Sigma(n)$ is not a computable function. In this figure we have tried to make explicit all steps made in the argument, including steps that the original manuscript doesn't explicitly mention because they are more or less evident. In the figure's node texts, we write 'BB$\Sigma$' instead of just $\Sigma$ when referring to the $\Sigma$ function, to avoid confusion with the summation operator. The nodes are color coded as follows:

**the light green nodes** are statements that appear in the manuscript with an equation number.

**the light blue nodes** are statements that appear in the manuscript without an equation number.

**the light yellow nodes** are applications of the lemmas we stated and proved in Appendix A.IV. Radó doesn't make these explicit.

**the light red node** gives a statement that is important for the flow of the argument, but is not explicitly stated in the manuscript.

**the white node** gives the explicit conclusion of the proof. Radó considers the proof done with the proof that $\Sigma(n) \rhd f(n)$.

## A.VI The equivalence of the non-computability of $S(n)$ and the halting problem

In Section VI, it is remarked without proof that $S(n)$ being non-computable is equivalent to the undecidability of the halting problem. We prove that result here for completeness.

*To be proven: the non-computability of the function $S(n)$ is equivalent to the non-decidability of halting problem.*

We prove this equivalence by separately proving the two implication directions (left-to-right, and right-to-left) as subproofs (a) and (b) below.

*Subproof (a): the non-computability of the function $S(n)$ implies the non-decidability of halting problem.*

First, suppose the halting problem is non-decidable; then the implication becomes trivially true. So we only need to consider the case where the halting problem is decidable. To compute $S(n)$, we can then enumerate all possible $n$-state Turing machines, and reject those that do not halt, using the algorithm that solves the halting problem. The Turing machines that remain are all known to halt. Simulate them all, counting for each of the number of steps it takes until they halt. The maximum of these step counts is, by definition, equal to $S(n)$.

*Subproof (b): the non-decidability of the halting problem implies the non-computability of the function $S(n)$.*

First, suppose the non-computability of the function $S(n)$ to be true; then the implication becomes trivially true. So we only need to consider the case where the non-computability of $S(n)$ is false, i.e., the case where $S(n)$ is computable. This allows us the decide the halting behavior (yes or no) of any $n$-state Turing machine by simulating it up to $S(n)$ steps, which we can know as a concrete value because $S(n)$ is computable. If the machine halts, it is now known to halt (by concrete demonstration). If it didn't halt, we known that it will *never* halt, since $S(n)$ is the maximum number of steps that an eventually-halting $n$-card Turing machine can run before it halts.

*Subproofs (a) and (b) together prove the equivalence we set out to prove.*

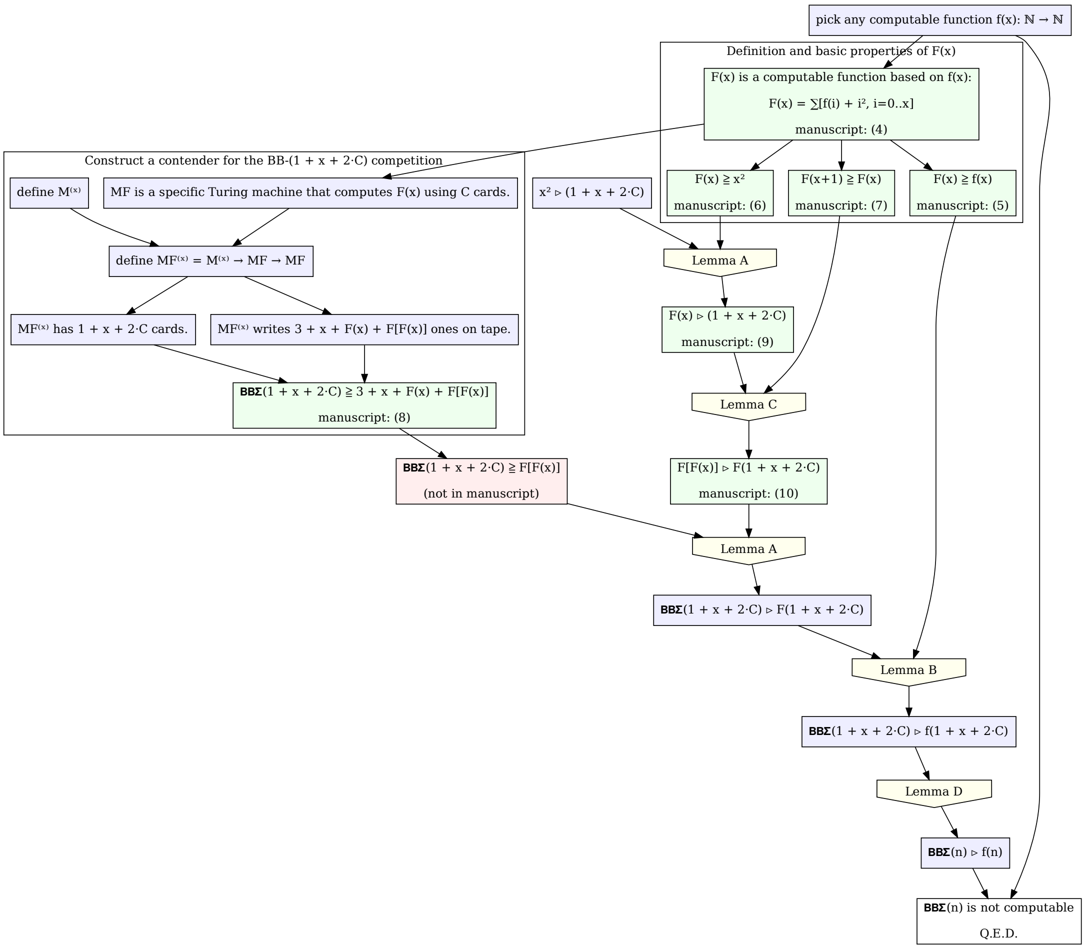


Figure 3: Structure of the proof in Section V.