On The Busy-Beaver Problem

by Ulisses Ferreira

Abstract *- The present paper refutes some theoretical claims. One of them is that there exists some uncomputable function in the theory of computability. The present paper refutes the known proof of this and suggests just the opposite claim. The other refuted claim is the undecidability of the busy beaver problem. This paper refutes the known proofs of this. Both results together mean that the referring to proofs no longer support the hypothesis of the undecidability of the halting problem.*

1 Uncomputability

Uncomputable functions seem to be strange elements in computer science, unrelated to practice. Even Haskell programmers do not talk about such functions, at least when they are programming. In this section, it is shown that this concept is probably redundant.

1.1 Does Uncomputable Function Exist?

In this section, it is shown that if the halting problem is unsolvable, there is no uncomputable function.

As it has been known[1, 4, 5], unary computable functions can form a denumerable set according to some enumeration of Turing machines. Therefore, let such an enumeration of all of the computable functions be with the following monadic symbols f_i for $i \in \mathbb{N}$: $\{f_1(x), f_2(x), f_3(x), ...\}$, from natural numbers to natural numbers. Let $g : \mathbb{N} \longrightarrow \mathbb{N}$ be defined as follows:

$$g(x) = \begin{cases} 0 & \text{if } f_x(x) \text{ is undefined;} \\ f_x(x) + 1 & \text{if } f_x(x) \text{ is defined.} \end{cases}$$

Clearly, $g(x)$ above has been regarded as total. Following that proof, there is the assumption that all total unary functions are computable, and then a contradiction is obtained: For every $n \in \mathbb{N}$, $g(n)$ seems to be different from $f_n(n)$ when $f_n(n)$ is undefined, because g is defined, above. On the other hand, $g(n)$ differs from $f_n(n)$ when $f_n(n)$ is defined because $g(n) = f_n(n) + 1$ and the results are also different for all n. And if g differs from f in all cases, g does not belong to the enumeration. However, as it was assumed that all total unary functions were computable, it was also assumed that g would be in the enumeration, and a contradiction is, in this way, obtained. The traditional conclusion: there exists some total unary function that is not computable, and that the conclusion is correct.

1.1.1 However, Another View

It seems that the above function has side conditions which are not part of the computation, like misterious ghosts, which are at a higher level than the function level. The picture can be seen in a different way. If the halting problem is unsolvable as it has been thought to be since Turing, despite the apparency, the above $g(x)$ function is *not* a total function but instead a *partially computable function* since its value depends on the value of $f_x(x)$ which in turn has to be calculated beforehand, and can be undefined. In this case with $x = n$, the partially computable $g(x)$ corresponds to a Turing machine which loops in the

case of the particular value n. On the other hand, if $f_x(x)$ is defined, the partial $g(x)$ is also defined, and $g(x) = f_x(x) + 1$, like in the traditional view, and there is no contradiction while the function g, as partial, is not in the referred to list.

1.1.2 However, If the Halting Problem is Decidable?

Let us check how the above computation of $g(x)$ may be thought to be performed. Initially, the code of g is *outside* the tape (this Turing machine will be called G) while the tape contains only a particular value of x in some representation. Let X be this representation. As soon as the machine G starts running, it transforms the particular value of x into some $F_x(X)$ (some representation of $f_x(x)$), obtaining the forms of representation for x and for $f_x(x)$, both encoded on the tape. Then, G starts interpreting $F_x(X)$ until its result is obtained if any, with the value of the function application $f_x(x)$. Finally, if the computation of $F_x(X)$ halts, the result is added to 1 as the result of $G(X)$. Otherwise, the result of $G(X)$ is 0. Here, the absence of any unexpected effect during that interpretation is assumed.

A second scheme is the following: Initially, the Universal Turing Machine U is outside the tape, while both X and $G(X)$ are encoded on the tape. For a Universal Turing machine that always guarantees the absence of unexpected effects, the result is always exactly the same as in the above paragraph. There is still something naïve here. If one checked this explanation in a deeper way, one would observe that, in the case that $g(x) = f_x(x)$ assuming a total function, the computation with all its details would run forever during the decoding process, if there was not care. In other words, without care, after $F_x(X)$ is interpreted, a new copy $F_x(X)$ appears; then this new copy is started being interpreted; another copy $F_x(X)$ appears and so on. Therefore, in the particular case of $g(x) = f_x(x)$, no matter other parts of the definition of g, the Universal Turing machine computation should compare and know that two copies are exactly the same and, hence, it unconditionally decides that the partial function g does not halt, which is a contradiction when comparing to the assumption that g was total. Therefore, g is partial.

An observation is that no Turing machine can decode itself and then start interpreting itself to obtain any result, as there will be no result at all. Thus, the Turing machine may compare two instances of its code and decide.

If the halting problem is published as solvable as the present work suggests, for all x, $g(x)$ is defined where $f_x(x)$ is undefined and, in this case, results in 0. Moreover, if $f_x(x)$ is defined, the computation of the g function becomes equivalent to $g(x) = f_x(x) + 1$.

Note that, in the first case of the $g(x)$ function (see in the previous definition, where $f_x(x)$ is undefined) $g(x)$ is defined if and only if one algorithm that solves the halting problem is properly involved in the computation of $g(x)$. More generally, in the same view, any function $g(x)$ is regarded as total if and only if some algorithm deciding the halting problem is in the computation of $g(x)$ wherever another function is undefined.

Here, regardless of whether the halting problem is decidable, the function $g(x)$ is in the above list of computable functions.

Theorem 1 *No function (over denumerable sets) is uncomputable.*

[Proof] Every recursive function is a function. Let $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ be any function for any $k \in \mathbb{N}$ such that $k \geq 1$. Therefore, by definition, f has one domain $Dom(f)$, one codomain $Cod(f)$ and one image $Im(f)$, where $Im(f) \subseteq Cod(f)$. No matter how f is *represented* and whether f *is defined* as recursive or not, f is a function and can be described as a set of tuples where each tuple has $k + 1$ elements in $\mathbb{N}$ where the first k elements of every tuple (in some previously set order) denote one element of $Dom(f)$ while the $(k + 1)$th element is the result from f and denotes one element of $Im(f)$. In this way, the referred to sequence shows only those elements where f is defined. Where f is undefined, the tuple is not in the sequence and vice versa. $\mathbb{N}$ is simply a sample denumerable set. Therefore, any set of tuples such that each tuple has $k + 1$ natural elements is a denumerable set. To build the enumeration, every of

these numbers are represented in such a way that there is a sequence of tuples, where each tuple has $k+1$ elements, and where the last element is a result.

Given this ordered representation and clearly, some computation of f given one element in $Dom(f)$ is trivial searching the sequence in this established order. □

2 Possible Decidability of The Busy Beaver Problem

Briefly, the busy beaver problem[6] consists in the following: given a certain number of states, one is asked for programming a Turing machine (Tm), with the same number of states, such that the Tm writes the longest sequence of ones (each is denoted here as I) on the tape, in comparison to the results from other Turing machines (Tms) with the same alphabet and states - the Tms may draw while comparing such scores. The sequence of ones has to be unbroken. And for simplification the alphabet is binary (one symbol plus the blank symbol).

As equivalent problems, it has been known that if the halting problem is undecidable, the busy beaver problem is also undecidable and vice-versa. A premise which is alternative to Turing's proof of the undecidability of the halting problem is the well known undecidability of the busy beaver problem. From the best of the present author's knowledge, there are two proofs of the undecidability of the busy beaver problem. One of them is in the original paper[6] and the other one is in [2]. There might be variations of these proofs, but the essence is the same.

2.1 Analyzing a Known Proof

This section initially recasts [3], and then shows what is mistaken in the corresponding proof. The reader should check the reference for it is a very good book.

That chapter on uncomputability via the busy beaver problem starts defining the notion of productivity of a Turing machine as follows: if the machine M computation halts, its productivity is the number of the I symbol written on the tape. The tape contains occurrences of I or of the blank symbol only. Furthermore, the productivity is 0 if the computation is undefined. Then, it is set that $p(n)$ = the productivity of the most productive n-state machines. The following propositions are correctly proved: $p(1) = 1$, also $p(47) \geq 100$, also $p(n+1) > p(n)$, also $p(n+11) \geq 2n$.

Then the proof of the undecidability of the busy beaver problem starts by using these propositions above.

However, in [3], the proposition

$$p(n+2k) \geq p(p(n)) \qquad \text{if BB exists} \quad (1)$$

there states a little more than the truth[†]. In particular, it does not hold for $n+2k < p(n)$. That sentence could be restated with correction as the following one:

$$\begin{array}{ll} p(n+2k) < p(p(n)) & \text{if } n+2k < p(n) \text{ and} \\ p(n+2k) = p(p(n)) & \text{if } n+2k = p(n) \text{ and} \\ p(n+2k) > p(p(n)) & \text{if } n+2k > p(n) \text{ if BB exists} \quad (1) \end{array}$$

and the rest of the proof in [2] ought to be in accordance with the present observation. As part of the present piece of work, in the second line, in the above formula, the comparison can be in the same machine or in the same set of states, as there are, there, the same number of states. In this way, one cannot find any contradictory statement, at least which can negate the assumption "BB exists" that was initially set. Notice that, in [2] and for the busy beaver problem, it is inconsistent to make n be universally quantified

[†] BB is the k-state Turing machine that computes the function p.

along with k as existentially quantified. As a concluding remark on the proof, a Turing machine BB with k states can exist in such a way that the condition $p(n) > n + 2k$ is possible for BB. This is a novelty. Thus, still on the proof as presented in [2], the correct would be to state

$$p(n + 2k) \geq p(p(n)) \quad \text{if } BB \text{ exists } \textit{and } n + 2k \geq p(n).$$

Following the same proof, given the novel and updated assumption, the conclusion should be that BB does not exist *or* $n + 2k < p(n)$, which does not suffice for concluding that BB does not exist. The rest of the proof is now briefly and correctly shown:

$$p(i) > p(j) \text{ if } i > j \quad (2); \qquad j \geq i \text{ if } p(j) \geq p(i) \quad (3);$$

$$n + 2k \geq p(n) \quad \text{if BB exists and } n + 2k \geq p(n) \quad (4);$$

$$n + 11 + 2k \geq p(n + 11) \text{ if BB exists and } n + 2k \geq p(n) \quad (5);$$

$$p(n + 11) \geq 2n \quad (6); \qquad n + 11 + 2k \geq 2n \quad \text{if BB exists and } n + 2k \geq p(n) \quad (7);$$

$$11 + 2k \geq n \text{ if BB exists and } n + 2k \geq p(n) \quad (8);$$

for each positive integer n. In particular, with $n = 12 + 2k$, then $11 + 2k \geq 12 + 2k$ is obtained or, subtracting $11 + 2k$ from both sides

$$0 \geq 1 \text{ if BB exists and } n + 2k \geq p(n) \quad (9)$$

2.2 Analyzing the Proof of 1962

Briefly, in [6], the $\Sigma : \mathbb{N} \longrightarrow \mathbb{N}$ function was defined as follows:

$$\Sigma(n) = \max[\sigma(M, s)]$$

where $\sigma(M, s)$ denotes the number of ones on the tape after s steps of computation of the Turing machine M over a blank tape. $\Sigma(n)$ is initially assumed to give the solution of the busy beaver problem. The author proves the unsolvability of the busy beaver problem writing $f(x) >- g(x)$ to state that $f(x) > g(x)$ for all x greater than x_0 for some x_0. There, the functions f and g have signature $\mathbb{N} \longrightarrow \mathbb{N}$, and $x, x_0 \in \mathbb{N}$. It is thus proved that $\Sigma(n) >- f(n)$ for any computable function $f(n)$, and, in this way, $\Sigma(n)$ is not computable. Nevertheless, that proof of $\Sigma(n) >- f(n)$ for all computable functions f is actually mistaken. Continuing to follow that proof, the auxiliary function

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

is defined in the referred to article as also computable. That article, therefore, correctly infers that $F(x) \geq f(x)$, also $F(x) \geq x^2$ and also $F(x + 1) > F(x)$, which are later used in the same proof.

Following [7], since $F(x)$ is computable, there is some Turing machine M_F with C states that computes $F(x)$. Further, for any $x \geq 0$, the reader has a Turing machine $M^{(x)}$ with $x + 1$ states that writes $x + 1$ consecutive ones on the tape and halts. Therefore, there is a Turing machine $M_F^{(x)}$ with $1 + x + 2C$ states that initially writes $x + 1$ consecutive ones, then calculates $F(x) + 1$, then writes $F[F(x)] + 1$ consecutive ones, and eventually halts. For the problem, $M_F^{(x)}$ is also a valid machine with score $3 + x + F(x) + F[F(x)]$, while score means the number of ones left on the tape after computation. Until now, the deduction seems to be correct. That article deduces that the maximum score $\Sigma(1+x+2C)$ satisfies $\Sigma(1 + x + 2C) \geq 3 + x + F(x) + F[F(x)]$.

However, the present author discovered that the above sentence does not necessarily hold, in particular for the number of states equal to $1 + x + 2S$ with S small enough for not being able to implement $F(x)$, as $f(x)$ is any arbitrary function in $\mathbb{N} \longrightarrow \mathbb{N}$. That is, the above expression misses its condition. As an alternative, the above expression can be rewritten as $\forall x, S : \mathbb{N},\ x \geq 0 \wedge S \geq C \Rightarrow \Sigma(1 + x + 2S) \geq 3 + x + F(x) + F[F(x)]$.

The original statement is certainly incorrect with respect to the meaning of $S < C$. From this, S is an arbitrary number such that $S \geq C$. Thus, as we are talking about $\Sigma(1 + x + 2S)$, we are also stating that $1 + x + 2S$ denotes *the number of states* in a Tm that compute on an empty tape. Thus, both x and S are variables at the same level, that is, S is not more a constant with respect to x; both equally grow; and both are bound to a previously set number of states. After including the premise $S \geq C$, the proof can be correctly stated in the following way:

$$(x^2 \succ\!- (1 + x + 2S)) \wedge F(x) \geq x^2 \text{ if } S \geq C \; and \; \Sigma \text{ computable}$$

$$F(x) \succ\!- (1 + x + 2S) \text{ if } S \geq C \; and \; \Sigma \text{ computable} \quad (9)$$

$$F[F(x)] \succ\!- F(1 + x + 2S) \text{ if } S \geq C \; and \; \Sigma \text{ computable} \quad (10)$$

$$\Sigma(1 + x + 2S) \succ\!- F(1 + x + 2S) \text{ if } S \geq C \; and \; \Sigma \text{ computable}$$

$$\Sigma(1 + x + 2S) \succ\!- f(1 + x + 2S) \text{ if } S \geq C \; and \; \Sigma \text{ computable}$$

$$\Sigma(n) \succ\!- f(n) \text{ if } S \geq C \; and \; \Sigma \text{ computable}$$

As a less important comment, perhaps $\succ\!-$ defined of a unique variable is not very suitable for the problem. That is, the same definition could be used in contexts without S, or two variables could be used instead. However, note that the function $\succ\!-$ plays an essential role in that proof. In this way, adding the discovered premise, there is no contradiction, and that proof of the undecidability has also been refuted.

References

[1] George S. Boolos and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, third edition, 1989.

[2] George S. Boolos and Richard C. Jeffrey. *Computability and Logic*, chapter 4 Uncomputability via Busy Beaver Problem, pages 34–42. Cambridge University Press, third edition, 1989.

[3] George S. Boolos and Richard C. Jeffrey. *Computability and Logic*, chapter 4 Uncomputability via Busy Beaver Problem, page 38. Cambridge University Press, third edition, 1989.

[4] Nigel Cutland. *Computability: an introduction to recursive function theory*. Cambridge University Press, 1980. This book was reprinted.

[5] I. C. C. Phillips. *Handbook of Logic in Computer Science*, volume 1: Mathematical Structures, chapter Recursion Theory, pages 79–187. Oxford University Press, 1992.

[6] Tibor Rado. On non-computable functions. *The Bell System Technical Journal*, pages 877–884, May 1962.

[7] Tibor Rado. *On Non-Computable Functions*, page 881. The Bell System Technical Journal, May 1962.