Philosophies **2015**, *1*, 15-27; doi:10.3390/philosophies1010015

OPEN ACCESS

philosophies

ISSN 2409-9287

http://www.mdpi.com/journal/philosophies

Article

Should Computability Be Epistemic? a Logical and Physical Point of View

Florent Franchette

INSA Lyon, Humanities Department, 1 rue des Humanités, 69621, Lyon, IHPST (Paris 1, ENS, CNRS), 13 rue du four, 75006 Paris, France; E-Mail: florent.franchette@gmail.com

Academic Editor: Vincent C. Müller

Received: 20 July 2015 / Accepted: 23 October 2015 / Published: 29 October 2015

Abstract: Although the formalizations of computability provided in the 1930s have proven to be equivalent, two different accounts of computability may be distinguished regarding computability as an epistemic concept. While computability, according to the epistemic account, should be based on epistemic constraints related to the capacities of human computers, the non-epistemic account considers computability as based on manipulations of symbols that require no human capacities other than the capacity of manipulating symbols according to a set of rules. In this paper, I shall evaluate, both from a logical and physical point of view, whether computability should be regarded as an epistemic concept, *i.e.*, whether epistemic constraints should be added on (physical) computability for considering functions as (physically) computable. Specifically, I shall argue that the introduction of epistemic constraints have deep implications for the set of computable functions, for the logical and physical Church-Turing thesis—cornerstones of logical and physical computability respectively—might turn out to be false according to which epistemic constraints are accepted.

Keywords: computability; physical computability; Church-Turing thesis; physical Church-Turing thesis; physical process; usability

1. Introduction

Computability historically deals with functions of positive integral arguments with values 0 or 1 that are effectively computable—even though whether an arbitrary function is computable, e.g., on arbitrary strings of symbols, can also be investigated. Several formal definitions of computability which are thought to correspond satisfactorily to the intuitive notion of effective computation were proposed in the 1930s—see, e.g., [1–4]. Although such definitions have proven to be equivalent, two different accounts of computability may be distinguished regarding computability as an epistemic concept.

The epistemic account of computability regards computability as an epistemic concept, which should be based on epistemic constraints related to the capacities of human computers. For instance, Turing computability [4] refers to the computation of functions or numbers that can be computed with paper and pencil in a finite number of steps by a human clerk working effectively—though such a definition of Turing computability may be controversial [5]. Computability can thus be considered as an epistemic concept, in the sense that the process of computing functions is closely related to the capacities of human computers. The non-epistemic account of computability, however, does not take into account human capacities in the process of computing functions: computability is only based on manipulations of symbols, which require no human capacities other than the capacity of manipulating symbols according to a set of fixed rules. According to Church computability, for instance, a function f is effectively computable if f is λ-definable, *i.e.*, if there is a λ-term H such that $\forall n \in \mathbb{N}$, $H(CN(n)) = CN(f(n))$, where $CN(n)$ is the n-th Church numeral [1].

In this paper, I shall answer the following question: does considering computability as an epistemic concept has consequences for the concept of a computable function? Answering this question means evaluating, both from a logical and physical point of view, whether computability should be regarded as an epistemic concept, *i.e.*, whether epistemic constraints should be added on (physical) computability for considering functions as (physically) computable. Specifically, I shall argue that the introduction of epistemic constraints have deep implications for the set of computable functions, for the logical and physical Church-Turing thesis—cornerstones of logical and physical computability respectively—might turn out to be false according to which epistemic constraints are accepted.

From a logical point a view, the logical Church-Turing thesis (CTT)—according to which effectively computable functions are computable by Turing machines—seems to be preserved whatever the account that is considered. Although different, Turing and Church computability are equivalent in terms of computable functions: Turing-computable functions correspond to λ-definable functions. Nevertheless, arguments against the CTT can be developed if the relationship between human computers and computability is strengthened, *i.e.*, if specific epistemic constraints—such as the one proposed by [6]—are added to computability. It will be explained, in Section 2, why the non-epistemic account of computability is required for preserving the CTT.

From a physical point of view, the situation is somewhat reversed though. Although the physical extension of the CTT is usually called the physical Church-Turing thesis (PCTT), no universally accepted definition can be found in the literature. For example, the PCTT may be understood as a thesis claiming that any physical process can be simulated by some Turing machine [7], or as suggesting that the laws of nature can be expressed, not only in the language of mathematics, but also in an algorithmic language [8]. The formulation of the PCTT that is considered in this paper may be set out as follows: functions that are computable by physical processes are computable by Turing machines. According to such a formulation, I shall argue that adopting the non-epistemic account leads to trivialize physical computability by falsifying the PCTT. In Section 3, it will be explained how the epistemic constraints proposed by [9] could preserve the PCTT against trivial counterexamples—even though all Piccinini's constraints should not be regarded as constraints on physical computability, in my opinion.

2. Should Logical Computability be Epistemic?

In what follows, I begin by presenting the epistemic and non-epistemic account of computability. Then I explain why the non-epistemic account is required for preserving the CTT.

2.1. Epistemic and Non-Epistemic Account of Computability

The aim of the logicians of 1930s such as Turing, Church and Kleene was to identify the functions on natural numbers that can be regarded as effectively computable. Although all characterizations of effectively computable functions have proven to be equivalent, two different accounts of computability may be distinguished regarding computability as an epistemic concept.

According to the epistemic account, computability should be based on epistemic constraints related to the capacities of human computers. Turing computability, for instance, is based on two main epistemic constraints specifying that computations (1) can—in principle—be carried out by a human being unaided by any machinery save paper and pencil; (2) demands no insight or ingenuity on the part of the human being carrying them out [10].

The real question underlying the epistemic account is therefore "what are the possible processes which can be carried out in computing a number?" In particular, [4] identified the concept of 'effectively computable functions' with that of "functions computable with paper and pencil in a finite number of steps by an idealized mathematician". Turing explained that an effective computable function is one whose values can be computed with paper and pencil in a finite number of steps by a human computer following fixed rules, and without understanding. A human computer is an idealized human who computes values of a function by following a finite set of instructions—each instruction expressed by means of a finite number of symbols, and that may lead to any of the function's defined values in a finite number of steps. The human computer is idealized in that he is not in principle limited to physical resources such as time and memory space: The human computer has as much resources as required for arriving at a value of the function [11].

According to the non-epistemic account, however, computability should not take into account human capacities in the process of computing functions: computability should only be based on manipulations of symbols, which require no human capacities other than the capacity of manipulating symbols according to a set of rules. In the words of Rogers,

> An algorithm is a clerical (*i.e.*, deterministic, book-keeping) procedure which can be applied to any of a certain class of symbolic inputs and which eventually yield, for each such input, a corresponding symbolic output [12] (p. 1).

More formally, a function is computable if and only if there is an algorithm A such as that, given a $Gn\ulcorner n\urcorner$, A applied to $\ulcorner n\urcorner$ yields to $Gn\ulcorner m\urcorner$ if and only if $f(n) = m$, where $Gn\ulcorner n\urcorner$ denotes the *Gödel number* of n, which is the code assigned to n via the method used by [13].

From a logical point a view, the CTT seems to be preserved whatever the account that is considered, although arguments against the CTT can be developed if the epistemic relationship between human computers and computability is strengthened, *i.e.*, if specific epistemic constraints are added to computability.

2.2. Bowie's Counterexample

The non-epistemic account of computability was supported by [14] in order to preserve the CTT against a counterexample put forward by [6]. Although the CTT is regarded as a true thesis by the scientific community—see, e.g., [15–17], Bowie claimed it was possible to exhibit a function which was not intuitively computable, yet recursive. For preserving the CTT, Ross then argued that computability should be regarded as a non-epistemic concept.

According to Bowie, claiming that a unary function f is computable means claiming there is an algorithm A able to compute $f(x)$ for every x within the domain of f. If $\Diamond$ denotes 'it is in principle possible that', then a function f is computable if and only if **(I)** $\Diamond\exists A\forall x$ (A would compute $f(x)$ with input x); and **(II)** $\exists y$ (A would establish that $f(x) = y$ where y is given in some appropriate notation).

It is worth noting that Bowie's definition of computability is an unorthodox one, for an additional epistemic constraint has been added, which represents the fact that human computers—who are carrying out A—are to establish that $f(x) = y$.

From his definiton, Bowie exhibited a function which was supposed to be recursive and non-computable. To this end, he began by defining a statement r as an undecidable statement, *i.e.*, as one which it is impossible to determine whether it is true or false. He then described a function f over $\mathbb{N}$ such as

$$f(n) = \begin{cases} 1 \text{ if } r \text{ is true} \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

For Bowie, f is a genuine recursive function which is not computable. Firstly, if f was computable then there would be—from **(I)** and **(II)**—an algorithm from which a human computer could establish that $f(n) = y$ for a specific y. Since r is either true or false, establishing that $f(n) = y$ for a specific y means establishing that $f(n) = y$ for every n and therefore determining the value of r. This conclusion, however, stands in contradiction with the hypothesis according to which r is an undecidable statement.

Secondly, let C_1 and C_0 be two constant functions defined as follows: $\forall n \in \mathbb{N}$, $C_1(n) = 1$ and $C_0(n) = 0$. Since r is either true or false, $\forall n \in \mathbb{N}$, $f(n) = 0$ or $f(n) = 1$. Thus, $f = C_1$ or $f = C_0$ depending on the truth value of r. And since C_1 and C_0 are both recursive functions, one can establish, by using the theorem (2), that f is a recursive function as well – with $h = C_1$ or $h = C_0$, and $g = f$.

$$\forall h \forall g [(h = g \text{ and } h \text{ is recursive}) \rightarrow g \text{ is recursive}] \tag{2}$$

Even though Bowie claimed that f is a genuine counterexample to the CTT, it is possible to show that his argument is fallacious. In his article, [14] criticized Bowie's argument by explaining that computability, for considering a given function f as computable, does not need to refer to an observer who must establish that $f(n) = m$. For Ross, computability should be understood according to the non-epistemic account, *i.e.*, only as a manipulation of symbols that requires no human capacities other than the capacity of manipulating symbols according to a set of rules.

As a result, the function f described by Bowie turns out to be computable since one of the two functions, C_1 or C_0, yields to the values of f. If two logicians were computing C_1 and C_0 in parallel, then the function f would be necessarily computed—although neither logician would be able to establish who is actually computing f. In other words, f would be computable despite the impossibility of exhibiting which algorithm would be computing f; rather, f would be computable because there would be a recursive function—C_1 or C_0 —having an input-output behavior identical to f.

According to the analysis of Bowie's counterexample, it would seem that logical computability should be regarded as a non-epistemic concept. Nevertheless, is that still the case when computability is extended to physics? The following section is intended to show that, in the case of physical computability, the situation is somewhat reversed though.

3. Should Physical Computability be Epistemic?

In this section, I begin by explaining why adopting the non-epistemic account results in falsifying the PCTT from specific physical processes such as random processes. The epistemic constraints proposed by [9], from which it might be possible to preserve the PCTT against trivial counterexamples, are then presented. Finally, although Piccinini's epistemic constraints might be able to preserve the PCTT, I argue that some of his constraints should not be regarded as genuine constraints on physical computability.

3.1. Going beyond Physical Computability with Randomness

From a physical point of view, adopting the non-epistemic account of computability would result in trivializing the PCTT: functions that are computable by physical processes are computable by Turing machines. If the non-epistemic account is preferred to its epistemic analogue, then the PCTT would become falsifiable, which disagrees with current researches showing that the question remains—see, e.g., [18–20].

Specifically, the PCTT would be falsified by a physical system based on processes that generate random strings of digits. Two types of random processes may be distinguished though: Pseudo random processes and genuine random processes. Pseudo random processes generate strings of digits from pseudo random methods, whose numbers appear to be random but are actually provided by algorithms. It is hardly surprising that pseudo random processes lead to effective computations: in theory, pseudo random strings come from algorithms and so can be provided, according to the CTT, by effective procedures; in practice, pseudo random strings can be provided by computers, which are able to compute exactly the same functions as Turing machines.

By contrast, genuine random processes generate genuine random strings of digits—or random strings if there is no ambiguity. Although the concept of a random string is technically complex—see [21–23], it can be shown that, if a string of digits $r_1, r_2, \ldots, r_n, \ldots$ is genuinely random, then there is no function $f(n) = r_n$ that is Turing-computable [24]. Basically, why genuine random strings would lead to strings of digits which are not Turing-computable may be explained as follows:

> There are uncountably many infinite strings of digits. (Even more strongly, there are uncountably many infinite strings of digits with any given limiting frequency of '0's and '1's.) But there are only countably many Turing-computable infinite strings. Therefore, assuming that each infinite string (or each infinite string with a certain limiting frequency) has the same probability of occurring as a result of a random process, the probability that a random process would generate a Turing-computable string of digits is zero, whereas the probability that the string of digits is not Turing computable is one [9] (p. 750).

In fact, the very idea of using randomness to go beyond Turing machines is not a new one: already Turing pointed out that a machine equipped with a random element could do more than Turing machines [25]. In particular, [26,27] proposed to use as oracles specific physical processes having random behaviors, in order to go beyond Turing machines.

The idea of computing with the aid of an oracle comes from [28]. Turing is indeed behind a kind of machine called oracle Turing machine or *o*-machine, which is a Turing machine equipped with an oracle, namely a black box whose behavior is not specified. The particularity of the oracle lies in its capacity to provide non-computable functions values:

> Let suppose that we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle as it were. We shall not go any further into the nature of this oracle apart from saying that it cannot be a machine [28] (p. 167).

According to its architecture and computational power, an *o*-machine is not a formalization of computability but rather a hypermachine, namely a machine capable of computing non-Turing-computable functions [29].

Since computing with the aid of oracles allows in theory to go beyond Turing machines, some researchers proposed to use random processes as oracles. There are two main proposals moving in this direction, which are based on random processes coming from radioactive decay [27] and quantum randomness [26], respectively.

Calude's proposition, for instance, consists of fixing on a computer a device able to generate a random string from a quantum process. As an example, the device might operate like *Quantis*, a generator created by the ID quantique company (http://www.idquantique.com/.) which is supposed to be able to provide a random string from an elementary quantum optics process. Photons are sent one by one onto a semi-transparent mirror and detected. The exclusive events—reflection and transmission—are associated to 0, 1 bit values, and each of them is supposed to have a probability at 50% to occur. The operation of Quantis is continuously monitored to ensure immediate detection of a failure and disabling of the random bit stream. By assuming that strings provided by Quantis are genuinely random, it would be possible to provide non-Turing-computable functions values, and therefore to claim that a task that lies beyond the reach of Turing machines has been carried out. Quantis would be seen as an oracle able to provide non-computable information from nature.

As a result, Calude's proposal might be able to falsify the PCTT under the condition that the epistemic account was adopted. Since the strings of digits that would be generated by Quantis would be able to represent values of non-computable functions, Quantis would compute non-computable functions according to the epistemic account, for an input-output behavior could not be reproduced by Turing machines. A possible way of preserving the PCTT from random processes might be to consider an epistemic account of computability, though.

3.2. Piccinini's Usability Constraints

According to [9], a physical process is a genuine computation only if it is usable in a certain way, *i.e.*, only if the following epistemic constraint – called the usability constraint – is satisfied: if a physical process is a computation, it can be used by a finite observer to obtain the desired values of a function.

The usability constraint is an epistemic one primarily because it is defined from the concept of a finite observer understood in a broad sense as including both human and other intelligent beings having finite capacities. To be more specific, a physical process is usable if and only if it is executable, automatic, uniform and reliable—for details, see [9] (p. 741). Actually, only executability is needed for the sake of argument. In the words of Piccinini,

> An executable physical process is one that a finite observer can set in motion to generate the values of a desired function until it generates a readable result. This requires that the observer can discover which function is being computed, that the process's inputs and outputs be readable by the observer, and that the finite observer be able to construct the system that exhibits the process. An executable physical process is also a process that, like an effective procedure, in principle can be repeated if a finite observer wishes to run it again. Each of these aspects of executability will be spelled out as a usability sub-constraint [9] (p. 741).

Executability may be specified from the following constraints:

Readability: Inputs and outputs have to be readable by an observer, *i.e.*, inputs and outputs can be prepared and measured to the desired degree of approximation. For instance, inputs and outputs of current computers are readable because they are strings of symbols coming from a finite alphabet.

Definability: An observer can define the function being computed independently of the process of computing it. That means an observer must be able to identify what function is computed before running the process.

Repeatability: The process has to be in principle repeatable by every observer who wants to obtain the result provided by the process. Basically, that means a given result will occur whenever the process is prepared to its initial state and launched.

Settability: An observer can set the system by providing it for inputs different arguments of the function which is computed by the system. When the observer wants to restart a computation, the system has to be resettable to its initial state and a new argument—which may be different from the previous one—can be provided.

Reliability: To be useful, a computing system must operate correctly long enough to yield (correct) results at least some of the time. For this to happen, the system's components must not break too often. In addition, the system's design must be such that noise and other external disturbances are generally insufficient to interfere with the results [9] (pp. 745–746).

By accepting the usability constraint, it can be decided whether a given physical process is a genuine computation. In particular, Piccinini claimed that random processes do not satisfy definability, settability and repeatability, and therefore should not count as genuine computations. As a result, devices providing random strings cannot be seen as counterexamples of the PCTT.

However, all Piccinini's constraints should not be regarded as constraints on physical computability, in my opinion. In what follows, I explain why definability, settability and repeatability should not count as constraints on physical computation.

3.3. Arguments against Usability Constraints

According to Piccinini, random processes should not count as computations because they are not definable—computed functions cannot be identified, settable—an observer cannot set the system by providing it for inputs different arguments, and repeatable—the process cannot be repeated. Random processes should not be ruled out as genuine computations by such constraints though. Reasons are (1) that definability is actually satisfied by random processes; (2) that some processes usually regarded as genuine computations may not be able to satisfy settability; and (3) that repeatability is not relevant for deciding whether physical processes are computations. Let us begin with definability.

3.3.1. Definability

According to Piccinini, a random process P does not satisfy definability because

> there is no way to define the function $f : \mathbf{N} \rightarrow \{0, 1\}$ whose values are being generated by P without reference to P itself [...] if P is genuinely random, there is no way to specify f without generating the values of f by running P [9] (pp. 750–751).

However, a mathematical function may be defined in two different ways: *in comprehensio* or *in extensio*. Basically, a function f with n arguments can be defined *in extensio* if it can be defined from the set of tuples $(x_1, \ldots, x_n, m)$ that characterize it. A function f can be defined *in comprehensio* if it can be defined without the set of tuples $(x_1, \ldots, x_n, m)$ that characterize it.

If defining the function f that is generated by a random process means defining it *in extensio*, then no random process can satisfy definability. It is indeed impossible to enumerate the digits representing the values of f, that belong to a particular segment of the random string prior to running the random process. Because every segment that would be enumerated would take the form of finite strings of digits—contrary to the random string itself, which is an infinite one, there would always be a Turing-computable function having values corresponding to the enumerated segment.

If defining f means defining it *in comprehensio*, though, random processes actually satisfy definability. Definitions *in comprehensio* differ from definitions *in extensio* in that they may be regarded as operations providing a procedure—which is not necessarily an effective one—for generating the values of defined functions. For instance, some non-Turing-computable functions can be defined *in comprehensio*, as evidenced by the following definition of the halting function H for Turing machines. Let $M_1, \ldots, M_n, \ldots$ be an enumeration of Turing machines:

$$H(x,y) = \begin{cases} 1 \text{ if } M_x \text{ halts on input } y \\ 0 \text{ if } M_x \text{ never halts on input } y \end{cases} \tag{3}$$

Similarly, the function R whose values are generated by a genuine random process can be defined *in comprehensio* as follows:

$$R(x) = \begin{cases} 1 \text{ with a probability of } 50\% \\ 0 \text{ with a probability of } 50\% \end{cases} \tag{4}$$

Even though the probability distribution of a genuine random process cannot be set by programming the device that will achieve the process, it can be specified. Since each digit generated by a genuine random process must be identically distributed—so the resulting distribution can remain uniform, if the random string consists of two digits then each digit will have a probability at 50% to occur. As a result, it is possible, in a certain way, to say that functions whose values are generated by random processes are definable prior to running them. Hence, random processes satisfy definability.

3.3.2. Settability

Random processes are not settable because

> [...] a 'user' of a random process P cannot select the value of f she wishes P to generate. If she wishes to obtain the n-th value of f, all she can do is let P take its course and wait until the n-th value is generated. If this won't happen until a million years from now, that's too bad [9] (p. 751).

The argument that is used for arguing that random processes do not satisfy settability is unconvincing: the fact that a user must wait the generation of $n - 1$ values before to obtain the nth value does not involve unsettability.

Computer scientists usually use recursive algorithms, which require the computation of the first $n-1$ values to reach the nth value. Let us assume that a recursive algorithm is implemented for computing the Fibonacci series $\{F_n\}_{n\geq 1}$ such as $F_1 = F_2 = 1$ and $F_{n+1} = F_n + F_{n-1}$. In addition, let us assume that a user wants to compute $F(100)$. By definition, the user must wait until the computer has computed all the values that are required for computing F_{100}. Yet, is the waiting time an obstacle to the usability of the computer in which F is implemented? No, for the Fibonacci series has several practical applications. For instance, financial technical analysis uses a tool called Fibonacci retracement allowing to predict financial markets according to specific thresholds corresponding to the Fibonacci series.

Someone might object that a recursive function – taken in the sense of recursive algorithm—does not always require the computation of previous values. For example, the computation of the nth term of the function F': $F'_{n+1} = F'_n + r$ with $n, r \in \mathbb{N}$ can be directly achieved from the formula $F'_n = F'_0 + (n-1) \times r$ without computing all the previous values. If there were such a 'shortcut' for the Fibonacci series—and more generally for every recursive function, computing all the previous values to reach outputs would only be required in the case of random processes—and Piccinini would be right.

However, whether there are such shortcuts for every recursive function has not been decided yet. Researchers working on this problem are trying to formally define the concept of a shortcut.

In particular, [30] have proposed a formal definition of the computational irreducibility, namely the fact to be able to compute $f(n)$ without having to compute $f(i)$ for every $i < n$. Although no function has been proved to be computationally irreducible, Zwirn and Delahaye have suggested some possible candidates such as the function f defined by: $f(n) =$ the first bit of the sum of the kth bit of $3k$ for all $k \geq n$. The existence of irreducible functions would show (1) that computing all the previous outputs values is an intrinsic property to computation; and (2) that random processes cannot be considered as spurious computations only on the basis they require to wait until the nth value is generated. Since no function has proven to be irreducible, however, there is no reason to exclude random processes because they are unsettable.

3.3.3. Repeatability

> [...] for a physical process to count as a genuine computation, it must be in principle repeatable by any competent observer who wishes to obtain its results [9] (pp. 742–743).

Although I am about to argue that repeatability should not be regarded as a constraint on physical computation, it is worth noting that Piccinini is right by claiming that genuine random processes are not repeatable: it is in effect impossible to generate a string which would be identical to the previous one, even though the random process can be reiterated. Because the process is genuinely random, we have no way of generating a particular random string. By definition, the random string is to obey the law of large numbers: there must be the same amount of 0 and 1 within the random string, and each number must occur with probability 1/2. The law of large numbers allows, in particular, to compute the probability of obtaining a second string that would be identical to the previous one. Let S be the first random string that is generated by the random process:

$$S = (x_0, x_1, x_2, x_3, \ldots, x_n, \ldots) \tag{5}$$

The probability that the first n digits of the second string S' are identical to the first n digits of S is

$$\frac{1}{2^n} \tag{6}$$

And since the string S consists of an infinite number of digits, the probability that $S = S'$ is

$$\lim_{n\to\infty} \frac{1}{2^n} = 0 \tag{7}$$

Hence, a particular random string cannot be generated twice.

Even though random processes are not repeatable, repeatability should not count as a constraint on physical computation. Piccinini justifies repeatability by emphasizing that computation matters to us and to the logicians who created computability theory—because we can learn from it, and that no one can learn from an unrepeatable process. Specifically,

> Unrepeatable processes can only be observed by the lucky few who happen to be in their presence. They cannot be used by others who might be interested in learning from them. If no one is present, no one can learn from an unrepeatable process. If someone is present but makes a mistake in recording a result, that mistake can never be corrected [9] (p. 743).

This passage is not entirely correct though. While it is true that unrepeatable processes can only be observed by the few who happen to be in their presence, it is false that if no one is present then no one can learn from an unrepeatable process.

Random processes may actually be sufficiently appropriate to learn from them. For that purpose, only devices capable of recording values provided by the random generator are needed, so people who would not be observing the process could learn from them in deferred time. In addition, if the number of recording devices is important enough, then mistakes that might occur in the recording process would be very low. If a mistake would nevertheless occur, the fact it could never be corrected would not be specific to random processes. For example, although a device could only achieved a single computation—a computation might require a quantity of resources that could never be collected again, results should be recorded as well, and therefore mistakes that could never be corrected might still occur.

Even from a computability viewpoint, it is possible to learn from random processes. Computability theory, which is based on the formalization of an effective procedure, was developed with the aim of identifying functions that cannot be effectively computed. Random processes, in particular, can be precisely used for exhibiting non-Turing-computable functions, whose values are represented by the digits that are randomly generated. As a result, a specific feature of functions—namely whether they are Turing-computable or not—can be identified, making it possible to learn from random processes.

To summarize, random processes should not count as spurious computations because they are not usable in the sense of Piccinini. Definability, settability and repeatability should not be regarded as criteria for deciding whether a physical process is a genuine computation.

4. Conclusions

In this paper, two accounts of computability have been presented, namely the epistemic and the non-epistemic account. It has been argued that each of these accounts may have deep conceptual implications for the set of computable functions depending whether logical or physical computability was considered. From a logical viewpoint, the non-epistemic account should be required for protecting the CTT from Bowie's epistemic argument, which is based on the description of a recursive function which is not computable. From a physical viewpoint, however, the epistemic account as described by Piccinini might allow to preserve the PCTT against trivial counterexamples such as random processes. As a result, the set of computable functions may be modified depending on the account—epistemic or non-epistemic—and the domain—logical or physical—which are considered, the issue being whether the two accounts are consistent with our intuitive idea of a computable function in the relevant domain. On this point, Piccinini's usability constraints—although being able to preserve the PCTT, should not be regarded as genuine constraints on physical computability.

Acknowledgments

I would like to thank the editors and referees for their very helpful comments during the preparation of this paper. I would personally like to thank Gualtiero Piccinini for many of our discussions and exchanges.

Conflicts of Interest

The author declares no conflict of interest.

References

1. Church, A. An UnsolvableProblem of Elementary Number Theory. In *The Undecidable (1965)*; Davis, M., Ed.; Dover: Mineola, NY, USA, 1936; pp. 88–107.
2. Kleene, S. General Recursive Functions of Natural Numbers. In *The Undecidable (1965)*; Davis, M., Ed.; Dover: Mineola, NY, USA, 1936; pp. 236–254.
3. Post, E. Finite Combinatory Processes. Formulation I. In *The Undecidable (1965)*; Davis, M., Ed.; Dover: Mineola, NY, USA, 1936; pp. 289–303.
4. Turing, A. On Computable Numbers, with an Application to the Entscheidungsproblem. In *The Undecidable (1965)*; Davis, M., Ed.; Dover: Mineola, NY, USA, 1936; pp. 116–151.
5. Hodges, A. Did Turing and Church Have a Thesis about Machines. In *Church's Thesis After 70 Years*; Olszewski, J.W.A., Janusz, R., Eds.; Ontos: Frankfurt, Germany, 2006; pp. 242–252.
6. Bowie, G. An Argument Against Church's Thesis. *J. Philos.* **1973**, *70*, 66–76.
7. Pitowsky, I. Quantum Speed-Up of Computations. *Philos. Sci.* **2002**, *69*, 168–177.
8. Dowek, G. The Physical Church-Turing Thesis and non-Deterministic Computation over the Real Numbers. *R. Soc. Lond. Philos. Trans. Ser. A* **2012**, *370*, 3349–3358.
9. Piccinini, G. The Physical Church-Turing Thesis: Modest or Bold? *Br. J. Philos. Sci.* **2011**, *62*, 733–769.
10. Copeland, B. The Church-Turing Thesis. Stanford Encyclopedia of Philosophy. Available online: http://plato.stanford.edu (accessed on 19 August 2002).
11. Shagrir, O. Effective Computation by Humans and Machines. *Minds Mach.* **2002**, *12*, 221–240.
12. Rogers, H. *Theory of Recursive Functions and Effective Computability*; The MIT Press: Cambridge, MA, USA, 1987.
13. Gödel, K. On Formally Undecidable Propositions of the Principia Mathematica and Related Systems, I. In *The Undecidable (1965)*; Davis, M., Ed.; Dover: Mineola, NY, USA, 1931.
14. Ross, D. Church's Thesis: What its Difficulties Are and Are not. *J. Philos.* **1974**, *71*, 515–525.
15. Folina, J. Church's Thesis: Prelude to a Proof . *Philos. Math.* **1998**, *6*, 302–323.
16. Black, R. Proving Church's Thesis. *Philos. Math.* **2000**, *8*, 244–258.
17. Dershowitz, N.; Gurevich, Y. A Natural Axiomatisation of Computability and Proof of Church's Thesis. *Bull. Symb. Logic* **2008**, *14*, 299–350.
18. Pour-El, M. Abstract Computability and its Relation to the General Purpose Analog Computer (Some Connections between Logic, Differential Equations and Analog Computers). *Trans. Am. Math. Soc.* **1974**, *199*, 1–28.

19. Earman, J. *A Primer on Determinism*; Reidel: Dordrecht, The Netherlands, 1986.
20. Wolfram, S. Undecidability and Intractability in Theoretical Physics. *Phys. Rev. Lett.* **1985**, *54*, 735–738.
21. Martin-Löf, P. The Definition of a Random Sequence. *Inf. Control* **1966**, *9*, 602–619.
22. Chaitin, G. On the Length of Programs for Computing Finite Binary Sequences. *J. Assoc. Comput. Machinery* **1966**, *13*, 547–569.
23. Schnorr, C. A Unified Approach to the Definition of a Random Sequence. *Math. Syst. Theory* **1971**, *5*, 246–258.
24. Church, A. On the Concept of a Random Sequence. *Bull. Am. Math. Soc.* **1940**, *46*, 130–135.
25. Copeland, B. Narrow Versus Wide Mechanism : Including a Re-Examination of Turing's Views of the Mind-Machine Issue. *J. Philos.* **2000**, *1*, 5–32.
26. Calude, C. Algorithmic randomness, quantum physics, and incompleteness. In *Machines, Computations, and Universality*; Margenstern, M., Ed.; Springer: Berlin, Germany, 2005; Volume 3354, pp. 1–17.
27. Stannett, M. Computation and Hypercomputation. *Minds Mach.* **2003**, *13*, 115–153.
28. Turing, A. Systems of Logic Based on the Ordinals. In *The Undecidable (1965)*; Davis, M., Ed.; Dover: Mineola, NY, USA, 1939; pp. 154–222.
29. Copeland, B. Hypercomputation. *Minds Mach.* 2002, *12*, 461–502.
30. Zwirn, H.; Delahaye, J. Unpredictability and computational irreducibility. In *Irreducibility and Computational Equivalence;* Zenil, H., Ed.; Springer: Berlin, Germany, 2013; Volume 2, pp. 273–295.