Article

Not All Computational Methods Are Effective Methods

Mark Sprevak

School of Philosophy, Psychology and Language Sciences, University of Edinburgh, Edinburgh EH8 9JU, UK; mark.sprevak@ed.ac.uk

Abstract: An effective method is a computational method that might, in principle, be executed by a human. In this paper, I argue that there are methods for computing that are not effective methods. The examples I consider are taken primarily from quantum computing, but these are only meant to be illustrative of a much wider class. Quantum inference and quantum parallelism involve steps that might be implemented in multiple physical systems, but cannot be implemented, or at least not at will, by an idealised human. Recognising that not all computational methods are effective methods is important for at least two reasons. First, it is needed to correctly state the results of Turing and other founders of computation theory. Turing is sometimes said to have offered a replacement for the informal notion of an effective method with the formal notion of a Turing machine. I argue that such a view only holds under limited circumstances. Second, not distinguishing between computational methods and effective methods can lead to mistakes when quantifying over the class of all possible computational methods. Such quantification is common in philosophy of mind in the context of thought experiments that explore the limits of computational functionalism. I argue that these 'homuncular' thought experiments should not be treated as valid.

Keywords: computation; effective method; algorithm; human computer; Turing; hypercomputation; quantum computation; homuncular functionalism; computational functionalism; unconventional computation

check for updates

Citation: Sprevak, M. Not All Computational Methods Are Effective Methods. *Philosophies* **2022**, 7, 113. https://doi.org/10.3390/philosophies7050113

Academic Editors: Diane Proudfoot and Zhao Fan

Received: 9 May 2022
Accepted: 30 September 2022
Published: 10 October 2022

1. Introduction

What is the relationship between the notion of a computational method and that of an effective method? A number of authors assume that the two notions are coextensive. Indeed, some treat the terms "effective method" and "computational method" not just as extensional equivalents but also as synonyms. The claim made by this paper is that any such equation is false: not all computational methods are effective methods.

Distinguishing effective methods from computational methods is important for a number of reasons. First, it is needed to accurately represent the historical motivations of the founders of computation theory, such as Turing, and to correctly state their results in a modern context. Second, not distinguishing between the two has the potential to adversely affect our reasoning when we quantify over the class of all possible computational methods. For example, if one thinks that some mental processes are computational processes, one might be led to an incorrect view about the possible nature of those processes—that they are the kinds of things that "little men" might perform. While this kind of homuncular thinking might serve as a rough heuristic or explanatory device when first introducing computational ideas about the mind, it is simply not the right way to understand computational processes. Not all computational methods are human executable, even in principle.

Turing famously developed a formal predicate that aimed to make the informal idea of an effective method more precise. This formalisation, the Turing machine, is sometimes described as offering a "definition", an "analysis", or a "replacement" for the informal notion of an effective method. I argue that care should be taken in interpreting these claims. Turing's formalisation may serve as an adequate replacement for the informal notion in certain contexts, but not in all. In particular, if one chooses to individuate computational methods so as to preserve fine-grained differences between methods that compute the same function—for example, if one is a functionalist about the mind or one cares about the complexity profiles of different methods for computing the same function—then the methods available to a system that uses an effective method cannot be identified with those available to a system that can use any computational method.

The argument of this paper runs as follows. In Section 2, I distinguish my argument from two superficially similar arguments in the literature: the first is that hypercomputers provide examples of non-effective computational methods; the second is that all computational methods should be individuated extensionally (by their overall input–output profile). In Section 3, I analyse the notion of an effective method; I argue that an essential requirement is that an effective method should, in principle, be human executable. In Section 4, I consider the objection that Turing offered a precisification of the notion of an effective method that would allow us to dispense with the informal notion. In Section 5, I examine instances of the claim that all computational methods are effective methods and explore some of their damaging consequences. In Section 6, I describe two examples, taken from quantum computing, of computational methods that are not effective methods. In Section 7, I consider the objection that quantum computing methods may still be executed by hand if a human were to simulate, step-by-step, the evolution of the underlying quantum wave function.

2. Distinguishing Features of This Argument

The argument in this paper should be distinguished from similar arguments in the literature that (i) depend on hypercomputation; or (ii) concern differences in functions rather than in methods.

2.1. No Dependence on Hypercomputation

Hypercomputers are hypothetical (real or notional) systems that compute functions that cannot be computed by any effective method.[1] These machines generally deploy some deliberately "non-effective" element as part of their design—some special extra resource that is not available to a human being working by themself. The exact nature of this special resource may vary between different machines. It might, for example, take the form of being able to complete an infinite number of steps in finite time, of being able to store arbitrary real numbers with infinite precision, or of having an "oracle" that provides the machine with the answer to uncomputable problems via some non-effective means.[2]

It might seem natural to appeal to hypercomputers to justify the claim that not all computational methods are effective methods. Shagrir and Pitowsky develop an argument along exactly these lines. After introducing various hypercomputer designs, they write:

> "... 'effective computation' (i.e., calculation by means of effective procedures) encompasses a wide, and an important, class of computations, but not necessarily all computations ... none of the hyper-machines described in the literature computes by means of effective procedures." Shagrir and Pitowsky [6] (p. 94)

If one accepts that hypercomputers are computers in the full and ordinary sense of the word, then it appears that no more needs to be said. Not all computational methods are effective methods because the computational methods used by hypercomputers are (by design) not effective methods.

Complicating this conclusion, however, are two issues.

First, the hypercomputers that have been proposed to date are only notional constructs. It is unclear whether they correspond to possibilities that are in any reasonable sense physically or practically available to us. It is unknown whether the kinds of non-effective resources required by hypercomputers could be physically implemented in our universe, and even if they could, whether they could be exploited by us in a practical way.[3] This may prompt one to wonder whether we should treat hypercomputers as being exactly on a par with more ordinary types of computer. Notwithstanding the properties of notional hypercomputers, perhaps all computational methods that *can be physically implemented*, or *implemented in some practicable way*, are effective methods. If one's primary interest is in methods that can be, or actually are, physically implemented—for example, the computational methods that are implemented in the brain—then perhaps one can ignore or bracket off considerations about non-effective methods based solely on hypercomputation.

Second, even if one ignores issues about the physical implementation of hypercomputers, it is common for both advocates and critics of hypercomputation to characterise hypercomputers as not computing in the full or ordinary sense of the term. Németi and Dávid [10] talk of their machines using computational methods in a "broad" or "extended" sense. Copeland [11] describes them as satisfying a "nonclassical" conception of computation. Turing [12] refers to oracle machines as instances of "relativised" computation: computation relative to the assumption that some problem uncomputable in the ordinary sense has been solved. These qualifications seem to suggest that a distinction should be drawn between an ordinary conception of computation and an extended or relativised notion. As with the previous point, this threatens to deaden the force of the claimed result. Not all hypercomputational methods are effective methods, but perhaps all *ordinary* computational methods are.

This paper deliberately avoids appeal to hypercomputation to justify the claim that not all computational methods are effective methods. This is not to endorse either of the two concerns above, but only to show that one does not need to rely on hypercomputational methods in order to establish the relevant claim.

The examples I use to justify the claim are taken from quantum computing. These have been chosen because (i) they are known to be physically implementable (and are already physically implemented and practically used); and (ii) they are commonly regarded as computing in the ordinary (non-hyper, non-extended) sense.[4]

In their seminal paper on hypercomputation, Copeland and Sylvan wrote:

> "It is perhaps surprising that not all classical algorithms are manual methods. That this is in fact the case has emerged from recent work on quantum computation . . . Algorithms for quantum Turing machines are not in general manual methods, since not all the primitive operations made available by the quantum hardware can be performed by a person unaided by machinery." Copeland and Sylvan [2] (p. 55)

After making this observation, they immediately turn to consider hypercomputation ("non-classical" algorithms). They do not return to, or explore further, non-hypercomputational methods ("classical" algorithms) that are not effective ("manual") methods. This paper could be understood as an attempt to expand on and defend Copeland and Sylvan's original observation.

2.2. *Computations Should Be Individuated by Their Internal Workings*

It is common for textbook discussions of effective methods to focus on questions of computability—questions about which functions can be computed.[5] In that context, computational methods are normally individuated *extensionally*: by their overall input–output behaviour.[6] My focus in this paper is not on the question of which functions are computable, but on which *method* is used for computing a given function. None of the examples I consider involve computation of a function that cannot also be computed by some effective method. The question considered here is whether the deployment of a *computational method* always entails the deployment of an *effective method*. This should be interpreted as not a question about computability, but as a question about the conditions under which effective methods are and are not instantiated in a given computational system. In order to be able to state this question correctly, and to prevent it collapsing into the question about computability, it is important that we do not individuate computational methods in a purely extensional fashion. To this end, in the context of this paper, I will assume that computational methods should be individuated, at least in part, by their internal workings.[7]

It should be stressed that this assumption is not ad hoc or unmotivated. Questions of computability and relations of extensional equivalence are important, but more fine-grained differences between computational methods matter too. Such differences are relevant to proposals about computational functionalism regarding the mind. According to these views, what is required for having a mental life is not only having the right behavioural responses—computing the right input–output function—but also the computational method by which that behaviour is generated—how the system calculates its function. If one wishes to reproduce or model cognition in an artificial system, then reproducing that method—not merely its overall input–output behaviour—is essential.[8]

Fine-grained differences between computational methods also matter to computer science. Different methods for computing the same function sometimes impose significantly different demands on resource usage, rendering some computational methods more or less feasible to implement. Measures of that resource usage—often summarised by a function that bounds how much time or space a method uses in the worst case—are of considerable theoretical and practical interest in computer science.[9]

BUBBLESORT and MERGESORT are widely regarded as distinct computational methods even though they compute the same function. Both methods take an unordered list of elements as input and yield a sorted list of the same elements as output. BUBBLESORT works by swapping pairs of adjacent elements in place until the entire list is sorted. MERGESORT works by splitting a list to create sublists which it then recursively merges to produce a final sorted version. BUBBLESORT can be shown to have a worst-case run-time complexity of $O(n^2)$ and space complexity of $O(1)$, whereas MERGESORT has a worst-case run-time complexity of $O(n \log n)$ and space complexity of $O(n)$.[10] A powerful motivation for distinguishing between these computational methods—for treating them as two distinct computational methods rather than as one—is that they have different worst-case complexity profiles. Their different complexity profiles are strong indicators that they place significantly different demands on the resources of any system that implements them. The implied general principle—that different worst-case complexity profiles indicate different computational methods—will be important later in this paper.

Worst-case complexity profiles are not the only considerations of relevance when individuating computational methods. Variants of either BUBBLESORT or MERGESORT might share the same worst-case complexity profile but still count as different methods. Indeed, for any computational method one might imagine introducing a range of variations from minor (e.g., extra debugging checks) to major changes (e.g., new data structures) into the sequence of its operations without changing its worst-case complexity profile. At which point does a variation in a method's internal workings produce a new computational method? Which factors—above and beyond differences in worst-case complexity profile—matter when individuating computational methods?

This question is a hard one to answer. Currently, there is no agreed answer, or at least none that takes the form of an exhaustive set of necessary and sufficient conditions.[11] It is difficult to give a fully general theory for the individuation of computational methods. There are multiple reasons for this. One is that there are "borderline" cases where no one seems to be certain whether a theory should say that two computational methods are the same or not. Another is that the standards regarding what we treat as the "same" computational method sometimes appear to vary depending on context and what features are currently of most interest to the interlocutors.[12] Notwithstanding these challenges, however, and the presence of "hard" cases for a general theory to handle, there are also plenty of clear-cut cases where we *can* say that computational methods are the same or different.

BUBBLESORT and MERGESORT are examples of such cases. They are paradigmatic examples of different computational methods, and classified as such both clearly and relative to any interests. As remarked above, a powerful consideration in their specific case—

one that makes the relevant identity judgement relatively clear-cut—is their demonstrable difference in worst-case complexity measure. In computer science, it is unheard of for two computational methods with different worst-case complexity profiles to be classified as the same for any purpose other than extensional equivalence.[13] My claim is that, relative to this widely accepted, clear, and robust standard for individuating computational methods, there are computational methods that are not effective methods.

3. What Is an Effective Method?

Copeland provides a clear characterisation of an effective method:

"A method, or procedure, *M*, for achieving some desired result is called 'effective' (or 'systematic' or 'mechanical') just in case:

1. *M* is set out in terms of a finite number of exact instructions (each instruction being expressed by means of a finite number of symbols);
2. *M* will, if carried out without error, produce the desired result in a finite number of steps;
3. *M* can (in practice or in principle) be carried out by a human being unaided by any machinery except paper and pencil;
4. *M* demands no insight, intuition, or ingenuity, on the part of the human being carrying out the method." Copeland [24]

Or more briefly:

"A mathematical method is termed 'effective' or 'mechanical' if and only if it can be set out in the form of a list of instructions able to be followed by an obedient human clerk ... who works with paper and pencil, reliably but without insight or ingenuity, for as long as is necessary." Copeland [25] (p. 12)

What Copeland says is consistent with a wide range of historical and contemporary sources:

"Turing examined ... *human* mechanical computability and exploited, in sharp contrast to Post, limitations of the human computing agent to motivate restrictive conditions ... Turing asked in the historical context in which he found himself *the* pertinent question, namely, what are the possible processes a human being can carry out (when computing a number or, equivalently, determining algorithmically the value of a number theoretic function)?" Sieg [26] (p. 395)

"[Computable problems are those] which can be solved by human clerical labour, working to fixed rule, and without understanding" Turing [27] (pp. 38–39)

"[With regard to what is effectively calculable] Both Church and Turing had in mind calculation by an abstract human being using some mechanical aids (such as paper and pencil)." Gandy [28] (p. 123)

"*Turing's analysis makes no reference whatsoever to calculating machines*. Turing machines appear as a result, as a codification, of his analysis of calculation by humans [previously defined as 'effective calculability']." Gandy [29] (p. 77)

"Roughly speaking, an algorithm [previously defined as an 'effective procedure'] is a clerical (i.e., deterministic, book-keeping) procedure which can be applied to any of a certain class of symbolic *inputs* and which will eventually yield, for each such input a corresponding symbolic *output*." Rogers [30] (p. 1)

"*Effectiveness*. An algorithm is also generally expected to be *effective*, in the sense that its operations must all be sufficiently basic that they can in principle be done exactly and in a finite length of time by someone using pencil and paper." Knuth [31] (p. 6)

"[an effective procedure is] a list of instructions ... that in principle make it possible to determine the value $f(n)$ for any argument n ... The instructions

> must be completely definite and explicit. They should tell you at each step what to do, not tell you to go ask someone else what to do, or to figure out for yourself what to do: the instructions should require no external sources of information, and should require no ingenuity to execute . . . ” Boolos et al. [32] (p. 23)

Common to all these suggestions is the idea that an effective method should be capable of being executed *by a lone human being unaided by any resources except paper and pencil*. The human is allowed an unlimited but finite amount of time, they are assumed not to make errors or get bored, and they have an unlimited but finite supply of paper and pencils. An effective method is a method that can be implemented by such an idealised human worker. Correspondingly, the kinds of operations that can be executed by this idealised human set limits on the class of effective methods.

Some authors have argued for revisionist accounts of “effective method”. Cleland [33,34] proposes that an effective method is a “quotidian” procedure that has essentially physical, causal consequences, such as baking a cake or assembling a child’s bicycle. Although a human might follow an effective method, human executability is not a necessary condition on such a method—a non-living particle travelling through a vacuum might follow an effective method that no human could replicate. Etesi and Németi [35] suggest that “effective method” should refer to any method that can be realised in any physical system, whether that system is an idealised human being or not. Shagrir [36] argues that the term “effective method” has undergone a meaning shift: in 1936, it meant a method that was in principle human executable, but today it means any symbolic operation that makes use of a finite procedure, and so it may refer to methods executable by humans, physical systems, or abstract automata.[14]

What our words mean is ultimately up to us and, in principle, there is nothing to stop a sufficiently determined revisionist from electing to define or redefine “effective method” so that it includes non-human-executable methods. However, there are good reasons for not choosing to define “effective method” in this way. Or rather, there are good reasons for maintaining a term in our vocabulary that refers specifically to only human-executable computational methods, and this role is normally occupied in mathematics and computer science by the term “effective method”.

Shapiro [38] provides a helpful discussion that places these attempts at revision in context. He describes how our various different ideas about effective computation *might* have been sharpened in many competing ways. He argues that the notion of effectiveness exhibited “open texture”, meaning that the full range of possible cases to which it correctly applied was not entirely pinned down by our pre-theoretic intuitions. Shapiro’s point about flexibility, however, pertains primarily to the historical development of the concept: our early, relatively inchoate ideas about what was or was not an effective method *could* have been sharpened in different ways. He does not suggest that today we are free to adopt different conceptions (as suggested by the revisionist proposals above), or that adopting an alternative, e.g., non-human, conception of what an effective method is would be equally good for the purposes of doing computer science or mathematical logic. Indeed, he argues that this is not the case. In line with others working on the foundations of computing, he suggests that the idealisation described above—a human working with unbounded time and computing space—is a way of signalling that one is talking about a particular subtype of procedure, one that has important pre-existing connections to ideas about mathematical provability, decidability, and surveyability. Irrespective of the rise of non-human machines or shifting interests within computer science, there is a persistent need to refer to this subset of methods. The term “effective method” is standardly the one used to fulfil this function.[15] A revisionist might insist that a different term should play this role—not “effective method” but something else. In that case, the argument of this paper may be rephrased to employ that alternative term.

4. Didn't Turing Define "Effective Method"?

In textbooks on mathematical computation theory, the term "effective method" often disappears once an appropriate formal predicate has been introduced. It is rare to see it persist after the first few introductory pages. Its disappearance is often explained by saying that Alan Turing provided a formal *definition*, *analysis*, or *replacement* for the informal notion. Thanks to Turing, we can replace "effective method" with a more formal, mathematically precise term, "Turing machine". A small wrinkle in the story is that there is more than one definition, analysis, or replacement for "effective method" available—Church introduced one with the λ-calculus, Gödel introduced another with general recursive functions, and there are many others. However, all these formal terms can be shown to be extensionally equivalent, so the choice between them may be glossed as largely a matter of convention. In light of this, the informal, human-centric notion of an "effective method" can be systematically replaced with the formal, precise notion of a "Turing machine" (or an extensionally equivalent term):

> "Turing's work is a paradigm of philosophical analysis: it shows that what appears to be a vague intuitive notion has in fact a unique meaning which can be stated with complete precision." Gandy [29] (p. 79)

> "Church's thesis is the proposal to identify an intuitive notion with a precise, formal, definition." Folina [42] (p. 311)

> "In 1928, the notion of an algorithm [effective method] was pretty vague. Up to that point, algorithms were often carried out by human beings using paper and pencil ... Attacking Hilbert's problem forced Turing to make precise exactly what was meant by an algorithm. To do this, Turing described what we now call a *Turing machine*." Matuschak and Nielsen [43]

> "If Turing's thesis is correct, then talk about the existence and non-existence of effective methods can be replaced throughout mathematics, logic and computer science by talk about the existence or non-existence of Turing machine programs." Copeland [24]

Turing himself, perhaps in a relatively unguarded moment, appears to endorse this too:

> "... one can reduce it [the definition of a solvable puzzle] to the definition of 'computable function' or 'systematic [effective] procedure'. A definition of any one of these would define all the rest. Since 1935 a number of definitions have been given [Turing machines, the λ-calculus, the μ-recursive functions, etc.], explaining in detail the meaning of one or other of these terms, and these have all been proved equivalent to one another ..." Turing [44] (p. 589)

I call this the "replacement theory" of effective methods. If the replacement theory is correct, then the notion of an effective method can be exchanged for that of a Turing machine (or an appropriate equivalent) without loss or distortion. The question this paper asks could then be rephrased as a question about which computational systems can and cannot instantiate Turing machines (or an appropriate equivalent).

It is important to appreciate that this is not the case. The replacement theory only holds—and it was only justified by Turing—under certain limited circumstances. To see this, it should be clear that an entirely unrestricted version of the replacement theory would not be plausible. "Effective method" and "Turing machine" do not have the same meaning—they do not have identical semantic content. If they did, it would have taken little or no insight on Turing's part to establish a relationship between them. In order to make sense of Turing's work, and the breakthrough that it represents, one needs to set aside the idea that "effective method" is a mere synonym for "Turing machine". The key question is when it is, and is not, legitimate to replace one notion with the other.

In Section 9 of his 1936 paper [45] ("The extent of the computable numbers"), Turing says that his goal is to show that both a human working by hand and a machine (later known as a Turing machine) can compute the same numbers. If this relationship between the two were to hold, then a certain kind of intersubstitutability between the corresponding terms would be warranted. Provided one's concern is only to identify *which numbers are computable*, then talk of effective methods could be safely replaced with that of Turing machines (or an extensionally equivalent formalism). For replacing one term with the other would have no effect on the validity of one's reasoning about the extent of the computable numbers.

One of the key arguments that Turing gives to justify this claim is to say that his machine and a human clerk go through a similar process when they compute a number. He does not say, however, that they go through an *identical* process, or that the operations that a Turing machine may take include all and only those that an idealised human worker may take. Turing instead suggests that the *results* the human worker can obtain without insight or ingenuity must meet a series of constraints, and that in light of these constraints, they are also reproducible by an appropriate series of steps of a Turing machine. He does not say that effective methods *are* Turing machines, but that the numbers that can be computed by any effective method turn out to be the same as the numbers that can be computed by a Turing machine.[16] If one's primary concern is to identify those numbers (i.e., to determine which numbers are computable), then talk of effective methods can be replaced with that of Turing machines (or another extensionally equivalent formalism).

It is worth noting that Turing did not argue—he did not need to argue—that the computational methods available to a Turing machine are identical to the methods available to a human clerk. Indeed, such a claim would be almost certainly false, and for reasons independent of the main argument of this paper. The steps and operations of a Turing machine—the basic operations that change the state of the head and that make marks on the tape—are not the only ways for a human or any other system to effectively calculate a number. The alternative models of Church, Gödel, and others show that there are many other ways to effectively calculate that do not involve exactly those basic operations. A sequence of basic operations might, for example, involve reduction operations in the λ-calculus, or minimisation and recursion operations over the μ-recursive functions. Different computational formalisms support different types of computational method, and porting methods between different computational formalisms is often non-trivial. One cannot always take a computational method that runs on a Turing machine and run *exactly the same method*, without changes, on a system that operates according to the rules of the λ-calculus. One might attempt to create a similar process—one with different internal characteristics expressed in terms of the basic operations and idioms of the λ-calculus—that computes the same function. The computational methods available to a user of the λ-calculus are not identical to those available to Turing machines. Given that a human clerk might follow any one of these various effective methods when computing a number, Turing machines cannot be identified with effective methods.

5. All Computational Methods Are Effective Methods

Here are some examples of the claim that all computational methods are effective methods:

> "An algorithm or effective method … is a procedure for correctly calculating the values of a function or solving a class of problems that can be executed in a finite time and mechanically—that is, without the exercise of intelligence or ingenuity or creativity … A computation is anything that … calculates the values of a function or solves a problem by following an algorithm or effective method." Burkholder [46] (p. 47)

> "The logician Turing proposed (and solved) the problem of giving a characterization of *computing machines* in the widest sense—mechanisms for solving problems by effective series of logical operations." Oppenheim and Putnam [47] (p. 19)

> "We have assumed the reader's understanding of the general notion of effectiveness, and indeed it must be considered as an informally familiar mathematical notion, since it is involved in mathematical problems of a frequently occurring kind, namely, problems to find a method of computation, i.e., a method by which to determine a number, or other thing, effectively. We shall not try to give here a rigorous definition of effectiveness, the informal notion being sufficient to enable us, in the cases we shall meet, to distinguish methods as effective or non-effective ... The notion of effectiveness may also be described by saying that an effective method of computation, or algorithm, is one for which it would be possible to build a computing machine." Church [48] (p. 52)

> "Sometimes computers are called information processors ... *How* they process or manipulate is by carrying out effective procedures ... Computation [means] the use of an algorithm ... also called an 'effective method' or a 'mechanical procedure' ... to calculate the value of a function." Crane [49] (pp. 102, 233)

> "The functional organisation of mental processes can be characterized in terms of effective procedures, since the mind's ability to construct working models is a computational process." Johnson-Laird [50] (pp. 9–10)

> "... [a] procedure admissible as an 'ultimate' procedure in a psychological theory [will fall] well within the intuitive boundaries of the 'computable' or 'effective' as these terms are presumed to be used in Church's Thesis." Dennett [51] (p. 83)

The quotations above illustrate that the claim has been made in a variety of contexts. The final three quotations provide examples of how it can constrain thinking about the mind.[17]

Searle's Chinese room argument provides a particularly clear example of the latter phenomenon [53]. Searle's argument may be challenged on many points, but among them is his assumption that any computational method can be executed by the human being inside the room who generates Chinese responses. Searle needs a claim like this in order to connect the specifics of his thought experiment (a lone human working without insight or ingenuity inside a room) to the general conclusion that *no possible* computational method can suffice for understanding. He needs some way to make the inferential leap from the person inside the room not understanding Chinese regardless of which method *they follow* to the conclusion that *no possible computational method* could be sufficient for Chinese understanding. Searle cites Turing's analysis of computation to justify this key step.[18] However, as we have seen in Section 4, the required claim is not attributable to Turing, and as we will see in the next section, it is false.[19]

However, the identification of computational methods with effective methods is more deep-seated in the philosophical literature than just Searle's argument. It is employed not only by the critics of computational accounts of cognition, but also by their advocates. A common philosophical move when reasoning about a computational model of cognition is to assume that one may freely replace *any* computational system's internal workings with a human working by rote without changing the computational method. Computational methods may always be swapped out during the course of reasoning, without loss or distortion, for effective methods. This has given rise to a widespread and mistaken form of what I call "homuncular thinking" about computational models of cognition. Here are some examples.

Fodor [57] describes how an account of knowledge-how, e.g., knowledge of how to tie one's shoes, could be given in computational terms. In the course of his analysis, he moves between a formulation of that knowledge-how in terms of a computation performed by the brain and a formulation of it in terms of elementary steps taken by an imaginary "little man" who reads basic instructions and follows them. The unstated assumption is that whatever computational method actually underlies knowledge-how (and that is implemented in the brain), one can be sure that it can be described, without loss or distortion, in terms of a series of steps taken by a little man who reads instructions and follows them without insight or ingenuity. Fodor does not, of course, suggest that a little man actually lives inside the head. However, he does think that talk of "the little man stands as a representative *pro tem* for psychological faculties which mediate the integration of shoe-tying behavior by applying information about how shoes are tied." (ibid., p. 629). He does not consider the possibility that "little man" talk might provide a blinkered, distorted, or misleading picture of a computational method. He simply assumes that it can always stand in for a computation method without colouring assumptions about the kind of computational method that is being considered. In other words, he assumes that all computational methods are human executable.

Dennett [51] famously developed a highly influential view known as homuncular functionalism.[20] In the course of defending the view, he moves between two different formulations of it that, like Fodor, he treats as freely interchangeable. According to one formulation, Dennett characterises homuncular functionalism as the view that a cognitive capacity can be explained by breaking it down in terms of operations of simpler computational subsystems, which are each explained in terms of the operations of simpler subsubsystems, and so on, until one reaches systems whose operations are so basic that they do not require further explanation of this kind. This model of explanation is treated as equivalent to the idea that one should explain the cognitive capacity by breaking it down into the capacities of a series of notional "little men" inside the head who each work without insight or ingenuity. The unstated assumption is, again, that whatever computational processes actually underlie cognition, they may *always* be described—without any loss or distortion—as a series of operations capable of being executed by little men each working to an effective method.

Block [59] provides a range of arguments against computational theories of consciousness based on intuitions about what a collection of little men can and cannot do. In his "homunculi-headed" thought experiment, the computation that normally takes place inside a human brain via neuronal activity is imagined to be reproduced by a sequence of steps taken by a group of little men each working according to an effective method. Block argues that is implausible that this group of little men would instantiate a new qualitative conscious experience, and hence that any purely computational account of conscious experience is unlikely to be true. A crucial premise in Block's argument is, again, that such a collection of little men could reproduce, without loss or distortion, any computational method, and in particular any computational method allegedly characteristic of conscious experience. Like Fodor and Dennett, Block does not justify this assumption. He simply takes for granted that all computational methods are human executable.

Why do these authors make this assumption?

One possible explanation might come from misguided intuitions about multiple realisability. Computational methods are multiply realisable: they can be implemented in more than one physical system. They are multiply realisable because the kind of description that is needed to characterise a computational method does not tie it to being implementable in just one physical medium. When the steps of a Turing machine are described, there is no requirement intrinsic to that description that a system which implements those steps must be made out of, for example, lead or wood or steel. In other words, computational methods are not tied by virtue of their specification to being implemented in one type of physical medium. However, there is a different and much stronger claim about multiple realisability that is regularly associated with computation and which is much less plausible. This second claim is that any computational method can be realised, in principle, in *any* physical medium. As Putnam put it: "We could be made of Swiss cheese and it wouldn't matter" [60] (p. 291). In the specific case of the human clerk, this second claim would suggest that any computational method could be implemented in the specific physical system of the clerk (provided they were to take an appropriate sequence of steps).

However, unlike the first claim, there is no reason to think that this second claim is true. It does not follow from computational methods being multiply realisable: just because it is possible for a computational method to be implemented in *more than one* physical medium that does not entail that it could be implemented in *any* medium (or in a human clerk). Different physical media have different causal powers. Those causal powers afford them the ability to implement some computational operations, but not others. There is no reason to think that an idealised human clerk has the causal powers to implement any possible computational method.[21]

Before closing this section, it is worth saying a few words about the definition of the term "algorithm". A number of the authors cited above suggest that "algorithm" should be regarded as a synonym for *both* "effective method" *and* "computational method". My claim is that "effective method" and "computational method" have different meanings and different extensions. If we are to distinguish these two terms, how then should we understand "algorithm"? Should it be treated as a synonym for "effective method", as suggested by Button [7], Smith [41], and Cutland [62]; or should it be treated as a synonym for "computational method", as suggested by Copeland [11], Copeland and Sylvan [2], Soare [63], and Gurevich [64]?[22] In this paper, I will follow the latter convention and treat "algorithm" as a synonym for the broader term, "computational method". This will allow us to say that there are quantum computing algorithms, even if there are no quantum computing effective methods. This convention has the virtue that it preserves how computer scientists already talk about quantum computing methods. Nothing important turns on this decision, however, and the argument of this paper may be rephrased if one prefers to define the term "algorithm" differently.[23]

6. Quantum Computations That Are Not Effective Methods

Quantum computers are able to move from input to output using computational methods that are not open to any idealised human clerk. A human working by hand may be able to compute the same functions as a quantum computer—they may be able to simulate a quantum computer's input–output behaviour—but they are not able to use the same computational method to do so.

Deutsch et al. [68] describe a simple quantum computer that uses a non-effective method. The computer uses *quantum interference* to compute the NOT function. The NOT function maps an input of 0 to an output of 1 and an input of 1 to an output of 0. Clearly, there is no question here of computing a function that cannot also be computed by hand. The question is whether the computational method that the quantum computer uses to calculate NOT could also be used by an idealised human clerk.

Deutsch's proposed quantum computer is composed of two half-silvered mirrors (mirrors that reflect a photon with 50% probability and allow a photon to pass through with 50% probability). The presence of a photon along one path to a half-silvered mirror denotes an input of 1, the presence of a photon along the other path denotes an input of 0; the presence of a photon along one exit path denotes an output of 1, the presence of a photon along the other exit path denotes an output of 0.

A single half-silvered mirror implements a quantum computational gate that Deutsch calls $\sqrt{\text{NOT}}$. If the input to the gate is 0, then the output is measured as either 0 or 1 with 50% probability; similarly, if the input is 1, the output is measured as either 0 or 1 with 50% probability. Formally, if the input is prepared in quantum state $|0\rangle$ (i.e., 0), then the output occurs in superposition state $(|0\rangle - i\,|1\rangle)/\sqrt{2}$ (which, on measurement, results in a 0 or 1 with 50% probability). If the input is prepared in quantum state $|1\rangle$ (i.e., 1), then the output occurs in superposition state $(-i\,|0\rangle + |1\rangle)/\sqrt{2}$ (which also, on measurement, results in a 0 or 1 with 50% probability).[24]

If two half-silvered mirrors are connected together in series, as shown in Figure 1, then the overall system computes NOT ($0 \rightarrow 1$, $1 \rightarrow 0$). If one did not know better, one might have guessed that this arrangement would produce a random result, perhaps with the output evenly distributed over 0s and 1s. Individual half-silvered mirrors appear to be randomisers, so one might guess that chaining two mirrors together would produce equally random results. However, due to the rules by which superposition states evolve in quantum mechanics, the relevant states interfere with each other, so that an input of 0 to the first mirror always yields an output of 1, and an input of 1 to the first mirror always yields an output of 0. This occurs even if a single photon is sent into the system, a phenomenon known as single-particle interference.

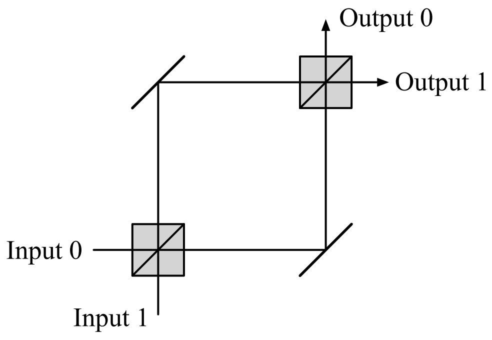

Figure 1. An example of a quantum NOT gate.

Formally, this can be seen as follows. The first half-silvered steps maps $|0\rangle \rightarrow (|0\rangle - i\,|1\rangle)/\sqrt{2}$. The second half-silvered mirror applies the same quantum operator to that superposition state, mapping $(|0\rangle - i\,|1\rangle)/\sqrt{2} \rightarrow -i\,|1\rangle$, which, on measurement, results in an output of 1 with 100% probability ($|-i|^2 = 1$). Combining the two operations, if the input is 0, then the output is 1. Similarly, the first half-silvered mirror maps $|1\rangle \rightarrow (-i\,|0\rangle + |1\rangle)/\sqrt{2}$. The second half-silvered mirror applies the same operator, mapping $(-i\,|0\rangle + |1\rangle)/\sqrt{2} \rightarrow -i\,|0\rangle$, which, on measurement, results in an output of 0 with 100% probability.

The same function, NOT, can of course be calculated by a human, but not using the same method.

It is important to stress that the claim here is not about the physical implementation of the quantum computation. The claim is not that the same *photon-and-mirrors* process cannot be reproduced by an unaided human. That is obviously true. The claim is that the same abstract *computational method* cannot be used. There is no suitably equivalent physical process that a human worker can engage in, even if they are idealised in the manner suggested, that calculates input–output in the same way. The computational method used by the quantum NOT computer is multiply realisable: it might be implemented with photons, electrons, fields, or atomic nuclei. All of these physical resources exhibit interference patterns that might be exploited to compute NOT using this computational method. But that method cannot be implemented in an unaided human working by hand—or at least, not in a controllable way. The computational method for calculating NOT is multiply realisable, but it cannot be realised at will in an unaided human.[25]

Interference is one non-effective computing method. Another example is *quantum parallelism*. Quantum parallelism underlies the claimed speedup of some quantum computers over more conventional computers.

Quantum parallelism allows a quantum computing system to calculate multiple values of a function $f(x)$ in a single step. In the simplest case, if an arbitrary 1-bit function $f(x)$ is applied to an input superposition state $(|0\rangle + |1\rangle)/\sqrt{2}$, then the output state would include $(|0, f(0)\rangle + |1, f(1)\rangle)/\sqrt{2}$. This state contains information about *both* $f(0)$ and $f(1)$, but it was obtained using only a single application of $f(x)$.[26] In a more complex case, every value of an arbitrary n-bit $f(x)$ could be calculated using a single application of $f(x)$. If $n+1$ bits are prepared in a superposition state, then one application of $f(x)$ would result in the superposition state $(2^{-n/2}) \sum_x |x, f(x)\rangle$, a state that encodes all values of $f(x)$.[27] Quantum parallelism is a non-effective method that allows a quantum computing system to calculate all values of an arbitrary function in one step. It is not a computing method freely available to a human working by hand.

A well-known limitation on methods that employ this kind of quantum parallelism is that it is only possible to recover a single value of $f(x)$ from the superposition state

$(2^{-n/2}) \sum_x |x, f(x)\rangle$ by measurement.[28] This limitation, however, is far from fatal to the use of quantum parallelism in computation. That is because before measurement all manner of computational operations might be performed on the quantum state that encodes all values of $f(x)$. These operations might affect different components of the superposition state in different ways. For example, certain components of the superposition state might be arranged to interfere with one another. These interference relations might be constructive or destructive, amplifying the probability of an outcome, or suppressing it. If correctly arranged, such interference relations might combine to allow a quantum computer to calculate some "global" property of $f(x)$: a mathematical property that depends on multiple values of $f(x)$—one that would require a conventional computer to explicitly evaluate $f(x)$ for several values of x individually. Consequently, even though only a single value of $f(x)$ can be recovered directly via measurement, all values of $f(x)$ are in principle available to compute global properties of $f(x)$, rendering this a potentially useful form of parallel computation.

The Deutsch–Jozsa algorithm provides an example of how this might work.[29] Suppose that Alice picks a function $f(x) : \{1, \ldots, n\} \to \{0, 1\}$ that is either balanced or constant and keeps it secret. A *constant* function yields the same value for all x; a *balanced* function yields 1 for half of x, and 0 for the other half. Bob can send Alice a number and ask her for the value $f(x)$. Bob's task is to determine, with as few queries as possible, whether Alice's function is constant or balanced. Using quantum parallelism, he can solve the problem using *just one* evaluation of $f(x)$. In the classical case, he requires at least $2^n/2 + 1$ operations in order to solve the same problem. Using quantum parallelism, Bob can apply Alice's $f(x)$ once to a superposition state which is then passed through a series of Hadamard gates.[30] If Alice's function is balanced, the various components of the superposition state $\sum_x |x, f(x)\rangle$ interfere with each other to yield the answer 0. If her function is constant, the components of the superposition state that encode all the values of $f(x)$ interfere to yield the answer 1.[31] The full details of the Deutsch–Jozsa algorithm are complex, but the key point is that the way in which Bob solves the problem requires only a single application of $f(x)$, which is not available to a human working by hand.

The problem that the Deutsch–Jozsa algorithm solves is of little practical interest, but similar techniques that employ quantum parallelism can be used to compute other, more useful properties. Shor's algorithm, for example, uses quantum parallelism to find the prime factors of integers [74]. Shor's algorithm factorises integers in polynomial time, making it almost exponentially faster than the most efficient known non-quantum factorisation method (the general number field sieve).[32] Shor's algorithm correspondingly has a different worst-case complexity profile to any known effective method for factoring numbers. Applying the principle described in Section 2.2—that different worst-case complexity profiles indicate different computational methods—it seems reasonable to conclude that Shor's algorithm is *different* from any known effective method. It is an example of a computational method that, as far as we know, cannot be executed by a suitably idealised human clerk.

Just as with interference, what is at issue here is not whether an unaided human could reproduce the same physical processes that take place inside a specific quantum computer. An unaided human is clearly not the same as an arrangement of half-silvered mirrors. The question at issue is whether that same *computing method* could be instantiated by a human clerk. Could a human following an effective method instantiate a computing method like Shor's algorithm? Quantum parallelism is a computational method that is multiply realisable: it might be implemented with photons, electrons, or atomic nuclei. Can it also be implemented by an unaided human working to an effective method? The answer appears to be no. The fact that such a human cannot, to our knowledge, engage in any process that would factorise numbers with the same worst-case complexity profile is strong evidence that they *cannot* instantiate the same computational method.

Quantum parallelism should not be conflated with other forms of parallelism found in modern electronic computers. In a modern electronic computer, multiple computational units are often executed simultaneously to compute more than one value of $f(x)$ within a single time step. In contrast, in a case of quantum parallelism, a single computational unit is executed *once* to evaluate all values of $f(x)$. Quantum parallelism should also not be conflated with a non-deterministic method of computation. A quantum computer that uses the superposition state $(|0, f(0)\rangle + |1, f(1)\rangle)/\sqrt{2}$ is not the same as a non-deterministic computer that yields $f(0)$ with 50% probability and $f(1)$ with 50% probability. In the case of a non-deterministic computer, the two alternatives $f(0)$ and $f(1)$ necessarily exclude each other: the machine computes either $f(0)$ or $f(1)$ on any given run. In a device that uses quantum parallelism, the two alternatives might interfere with each other to create an output that reflects a global property of $f(x)$—an output that would require a machine to know both $f(0)$ and $f(1)$.[33]

The two examples described in this section—quantum interference and quantum parallelism—are not meant to be an exhaustive list of all non-effective features of quantum computation. Other potential features might include quantum entanglement, quantum teleportation, or counterfactual computation.[34] Just like interference and quantum parallelism, these features are multiply realisable—they are not specific to any particular hardware implementation. Just like interference and quantum parallelism, it is hard to see how they might be implemented at will in an unaided human.

Quantum computers are unlikely to be the only systems that use non-effective computational methods. Other possible examples might include DNA computers [79,80], enzyme-based computers [81], slime moulds [82], fungi [83], reservoir computers [84], and optical computers [85]. An anonymous referee suggested that analog computers provide good examples of systems that use non-effective computing methods [86]. Shagrir [40] claims that the Game of Life is another case. According to Shagrir, an unbounded number of cells inside the Game of Life need to be updated simultaneously at each time step. He argues that this requires an unbounded number of parallel operations, which, he notes, building on work by Gandy [28], cannot be executed by a human clerk.[35] Gurevich suggests that although the idea of a computational method ("algorithm") originated with human-executable methods, it has since been generalised to other methods, and it continues to expand in ways that are hard to delimit:

> "In addition to classical sequential algorithms, in use from antiquity, we have now parallel, interactive, distributed, real-time, analog, hybrid, quantum, etc. algorithms. New kinds of numbers and algorithms may be introduced. The notions of numbers and algorithms have not crystallized (and maybe never will) to support rigorous definitions." Gurevich [64] (p. 32)

The argument of this paper is not intended to suggest that quantum computing methods are the only, or the most central, examples of non-effective computational methods. However, the case of quantum computing is a particularly helpful one with which to make the case that not all computational methods are effective methods. This is because it allows us to apply a relatively clear-cut, quantitative, and widely accepted principle—that different worst-case complexity profiles indicate different computational methods—to settle hard questions about how to individuate computational methods.

Any claim that a computational method is not an effective method is liable to face scrutiny or some degree of scepticism regarding its specific understanding of how to individuate computational methods. As noted in Section 2.2, this involves defence over a complex and unsettled territory. It is often hard to say definitely whether two computational methods are the same or not. From one perspective—at a particular level of abstraction, or with a focus on the similarity of certain features rather than others—two computational methods might appear to be the same. However, if looked at in a different way—at a different level of abstraction, or with an emphasis on different features—those same methods might be classified as different.[36] In general, it is not obvious what counts as a "superficial" versus a "genuine" difference between the internal workings of computing methods (e.g., in Shagrir's case it is not obvious that the Game of Life really does require all cells to be updated simultaneously). The question therefore always potentially arises about whether a suggested computational method is genuinely different from an effective method—whether it is a legitimate case of a non-effective computational method—or only one that differs in some superficial respects.

The principle described in Section 2.2 provides a way of cutting through this uncertainty. Different complexity profiles provide a sufficient reason for distinguishing between computational methods that compute the same function. This principle only really gets its teeth, however, in the quantum computing case. Generally speaking, it is common inside complexity theory to assume that any two "reasonable" models of computation are able to simulate each other with at most a polynomial penalty in time or space. This is sometimes known as the Cobham–Edmonds thesis or the extended Church–Turing thesis [87,88]. It is sometimes glossed as showing that the abstract computational method "does not matter" to the worst-case complexity profile associated with a task—worst-case complexity profiles are robust under changes in computing method or computing paradigm [87] (pp. 33–34). However, quantum computers appear to offer an exception to this [88–91]. Some quantum computational methods provide near-exponential speedup for some problems (e.g., factorisation). They appear to be examples of cases where the computational method *does* matter to the worst-case complexity profile associated with solving a task. My claim is that because of these differences in complexity profile we have good reason to think that we have genuinely different computing methods on our hands. We can apply the principle from Section 2.2 to show that a quantum computational method is *different* from any effective method. This kind of result is not known for other types of non-human computing method. As such, quantum computing methods stand out as particularly clear-cut, egregious cases of computational methods that are not effective methods.

7. Simulating the Quantum System by Hand

Someone might object to the treatment of the computational methods described in the previous section by replying that the relevant quantum computational methods *are* executable by a human. All a human would need to do is calculate the evolution of the relevant quantum wave function. This is exactly what we appeared to do in the case of Deutsch's NOT quantum computer: we applied the relevant quantum operator step-by-step (by performing matrix multiplication) to calculate the output state from an input state. In principle, a similar kind of procedure could be performed for the Deutsch–Jozsa algorithm or for Shor's algorithm. Calculating the evolution of a quantum wave function by hand can be extremely laborious, but there is no reason to think it cannot be done, in principle, with an effective method. There is no overtly "non-effective" mathematical step within any of the formal theory of quantum mechanics. Hence, the objection goes, there is no reason to think that a suitably idealised human clerk cannot reproduce the computational methods of any quantum computer.

It is important to note that although it might be possible to calculate the evolution of a quantum computer's wave function by hand, doing so is *not* the same computational method as letting a quantum computer evolve by itself. There is a difference between applying the relevant quantum operators by hand (e.g., by doing a sequence of matrix multiplications) and letting the target physical system run to produce its output. That there is a difference can be justified by appealing, again, to the principle, described in Section 2.2.

Feynman [92] famously showed that simulating the evolution of a quantum system by hand is a computationally intractable problem. This means that a quantum computer undergoing natural evolution of its wave function, and a human simulating it by an effective method, e.g., by repeatedly performing matrix multiplications, have qualitatively different worst-case complexity profiles. The human working by hand will use exponentially more space (or time) than the quantum computer to produce the same overall output. Calculating the evolution of an n-bit quantum system by hand would require (at least) 2^n classical bits.[37] For a quantum computer with 400 quantum bits (say, consisting of 400 atomic nuclei), an effective method that calculates the wave function by hand would require more bits for storage than there are estimated particles in the universe. The relevant issue here is not, however, a practical limitation on storage—after all, the imagined clerk is allowed unlimited space and time. The problem is about how the clerk's resource use *grows* with the size of the problem. This growth is what the worst-case complexity profiles measure and it is what signals that these are different methods for tackling the same problem. Any effective method that an idealised human might adopt for stepping through the evolution of the quantum system by hand will be exponentially less efficient than running the quantum computer itself. Therefore, applying the principle from Section 2.2, running the quantum computation is not the same—in terms of which computational method is instantiated—as having a human work through the evolution of its wave function by hand.

8. Conclusions

In summary, the argument of this paper is as follows. It relies on two premises:

1. If two putative computational methods have different worst-case complexity profile, then they are genuinely different computational methods.
2. There are abstract quantum computational methods that have different worst-case complexity profiles to that of any known effective method.

The conclusion follows that:

3. There are computational methods that are not effective methods.

Premise 1 was introduced and defended in Section 2.2. I argued that it is embedded as a principle into the practice of both theoretical and engineering computer science. Knuth [22] suggests that it may be a defining feature of thinking like a computer scientist rather than thinking like a mathematician. Premise 2 is not proven, but widely believed to be true within the quantum computing community. It is widely thought that certain quantum methods (e.g., Shor's algorithm) provide a true "quantum advantage" in terms of worst-case space or time usage. I discuss examples of such methods, and the unusual basic steps that they employ, in Sections 6 and 7.

Someone might take issue with either premise 1 or 2. I have supplied here some reasons why someone might accept them, but I offer nothing original. The primary argument of this paper is that *if* they are true, then conclusion 3 follows.

It is worth stressing that neither premise 1 nor premise 2 reference the specifics of the physical implementing hardware. Quantum computing methods are commonly implemented in non-human physical systems (e.g., with mirrors and photons). However, the argument of this paper is not that quantum computing methods are not effective because they are implemented in some non-human physical system. It is not merely the non-human character of their typical implementation that means that quantum computing methods are not effective methods. A non-human physical system (e.g., an electronic PC) might implement an effective computational method and the relevant quantum methods might be implemented in many—an unlimited number of—different kinds of physical system. The argument of this paper is rather that quantum computing methods cannot be implemented in a suitably idealised human clerk because, at least to the best of our knowledge, the required clerk cannot implement any computational method that displays the same worst-case complexity profile.

In Section 9 of his 1936 paper, Turing wrote:

"The real question at issue is 'What are the possible processes which can be carried out in computing a number?'" Turing [45] (p. 249)

Turing had in mind a human computer (a "computor", to use Gandy's term [29]), and he went on to answer this question by describing the operations and methods of what has come to be known as a Turing machine. This appears to suggest that the *possible processes which can be carried out in effectively computing a number* should be identified with the *methods that can be executed by a Turing machine*. As Wittgenstein said, "Turing's … machines are *humans* who calculate." [93] (§1096).

We have seen that care should be taken in how this claim is interpreted. The possible processes that might be carried out in computing a number outrun both (i) those that might be carried out by a Turing machine and (ii) those that might be carried out by an idealised human following an effective method. There are processes for computing that are human executable but not Turing-machine executable (e.g., that involve sequences of operations in the λ-calculus, or over the μ-recursive functions) and there are processes for computing that are not executable by a human but which are executable by certain other systems (e.g., quantum computers).

Turing ignored these issues in his 1936 paper because his focus was on relationships between computing processes of extensional equivalence. If one's primary focus is on questions of computability, then these fine-grained differences between computational methods—differences that do not affect which numbers are computable—can be ignored. However, if one is interested in differences in internal workings between computing methods—as is commonly the case in philosophy of mind and in many areas of computer science—then an identification between computing methods and effective methods cannot be made.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: I would like to thank three anonymous referees for their helpful comments on this paper. An early version of the paper greatly benefited from feedback at the Philosophy Workshop at the Department of History and Philosophy of Science, University of Cambridge.

Conflicts of Interest: The author declares no conflict of interest.

Notes

1 See Copeland and Proudfoot [1].

2 For examples of proposed hypercomputers designs, see Copeland and Sylvan [2]; Copeland [3]; Copeland [4]; Syropoulos [5].

3 For a range of objections to hypercomputation along these lines, see Button [7]; Davis [8]; Piccinini [9].

4 I do not claim that quantum cases are the only examples of non-effective computational methods; see the end of Section 6 for discussion of other possible examples.

5 Or which numbers can be computed (see Section 4), or which puzzles can be solved by computational means (see Section 5).

6 In other words, by the function that they compute, where "function" is understood in a purely extensional way, i.e., as a set or ordered pairs corresponding to the overall input and output.

7 In the terms of Church [13], we consider differences in the function-in-intension rather than the function-in-extension. In the terms of Marr [14], we individuate computations as they are at the algorithmic level rather than at the computational level (pp. 22–24).

8 See Block [15] for a classic discussion of this.

9 Worst-case measures of space or time complexity are not the only ones used to describe this resource usage, but they are the most commonly employed. Thanks to an anonymous reviewer for this point.

10 In this notation, n is the size of the list and $O(g(n))$ provides an asymptotic upper bound on the resource consumption: for large enough n, resource consumption is always less than or equal to some constant times the $g(n)$ function named inside the $O(\cdot)$. For more on complexity theory and use of big-O notation to measure resource usage, see Papadimitriou [16].

11 See Dean [17] for a review of contemporary analytic approaches to this problem, including those of Gurevich [18,19] and Moschovakis [20].

12 For a helpful analysis of these two problems in relation to creating a general theory, see Blass et al. [21].

13 See Knuth [22], (p. 97), who suggests that a distinguishing feature of computer science is that algorithms should be individuated by their complexity class. He argues that this "algorithmic" mode of thinking separates the thought processes of earlier mathematicians from those of later computer scientists (pp. 96–98). See Dean [17], (pp. 20–29); Shagrir [23] for further discussion of how and why complexity profiles matter to the individuation of computational methods.

14 Some critics of Turing argued that his human-centric characterisation of an effective method was not too narrow (as the authors above suggest), but too broad: the definition should be narrowed by adding a requirement that the number of steps taken by the human clerk should be somehow bounded or determinable in advance. For criticism of such proposals, see Gandy [29] (pp. 59–60); Mendelson [37] (p. 202); Rogers [30] (p. 5).

15 For further defence of the human-centric condition regarding "effective method", see Black [39]; Button [7]; Copeland [24]; Gandy [29]; Shagrir [40] (p. 40); Smith [41].

16 See Shagrir [36] (pp. 225–226); Shagrir [40] (pp. 36–39).

17 Copeland [25,52] criticises a number of the same authors for committing what he calls the "Church–Turing fallacy". The fallacy is to assume that any possible physical mechanism could be simulated by some Turing machine. My claim is that the authors make a second mistake in that they assume that any possible computational method is also an effective method. Copeland argues that although "effective" and "mechanical" sometimes appear to be synonyms in mathematical logic, the relationship between them should be handled with caution. "Mechanical" should be understood as a term of art and defined in the way described in Section 3. It does not correspond in any straightforward way to the concept of a physical mechanism.

18 See Searle [54] (p. 202) and Searle (personal correspondence).

19 See Copeland [55]; Sprevak [56] for a detailed analysis of the role of this assumption in the Chinese room argument.

20 See Lycan [58] for the name "homuncular functionalism" and a clear reconstruction of the view.

21 See Shagrir [61] for a helpful analysis and criticism of this second claim about the multiple realisability of computation.

22 See also Blass and Gurevich [65]:

"In fact the notion of algorithm is richer these days than it was in Turing's days. And there are algorithms . . . not covered directly by Turing's analysis, for example, algorithms that interact with their environments, algorithms whose inputs are abstract structures, and geometric or, more generally, non-discrete algorithms." (p. 31)

23 It is worth noting that the term "algorithm" has a long history of its own and semantic associations that predate its current connections with either "computational method" or "effective method". See Chabert et al. [66]; Knuth [67].

24 If a superposition state $\alpha\,|0\rangle + \beta\,|1\rangle$ is measured, then the result is 0 with probability $|\alpha|^2$, and 1 with probability $|\beta|^2$, with $|\alpha|^2 + |\beta|^2 = 1$. A $\sqrt{\text{NOT}}$ gate performs the operation on the quantum state vector $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ described by the complex-valued matrix $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$.

25 Cf. Nielsen and Chuang [69] (p. 50): ". . . it is tempting to dismiss quantum computation as yet another technological fad . . . This is a mistake, since quantum computation is an *abstract paradigm* for information processing that may have many *different* implementations in technology."

26 More accurately, a unitary (reversible) operator U_f is applied to the input, U_f: $|x, y\rangle \to |x, y \oplus f(x)\rangle$, where $\oplus$ indicates addition modulo 2. U_f is used because there is no guarantee that an arbitrary f itself is unitary, and the evolution of a quantum mechanical system must be governed by unitary operators. This modification does not affect the point above.

27 See Nielsen and Chuang [69] (pp. 30–32).

28 Strictly, a pair of values can be recovered, $x, f(x)$. The output is a pair because the evolution of the quantum state is governed by unitary operators (quantum computations must be reversible).

29 See Cleve et al. [70]; Deutsch and Jozsa [71]. A simplified version of the algorithm was first proposed by Deutsch [72].

30 A Hadamard gate is a quantum operator that works in a similar way to Deutsch's $\sqrt{\text{NOT}}$ operator, but defined over the real numbers. The transformation provided by a Hadamard gate is given by the real-valued matrix $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Like Deutsch's $\sqrt{\text{NOT}}$, a Hadamard gate may be physically implemented with half-silvered mirrors; see Barz [73].

31 See Nielsen and Chuang [69] (pp. 32–36) for the details of the algorithm.

32 Ibid.

33 For further discussion of this point, see Nielsen and Chuang [69], pp. 30–34.

34 See Ekert and Jozsa [75] for algorithms that use quantum entanglement, and Bennett et al. [76], Gottesman and Chuang [77] for algorithms that use teleportation. Counterfactual computation is a counterintuitive method in which the intermediate steps of the computations do not take place in the actual world (according to measurement), yet the desired output is still produced; for a proposed application, see Hosten et al. [78].

35 Shagrir [40], pp. 46–47.

36 I am assuming the methods in question have the same overall input–output profile and that one is trying to individuate them based on their internal workings. As discussed in Section 2.1, I am setting aside the use of hypercomputers for establishing the claim that not all computational methods are effective methods.

37 Nielsen and Chuang [69] (pp. 48, 204–206).

References

1. Copeland, B.J.; Proudfoot, D. Alan Turing's forgotten ideas in computer science. *Sci. Am.* **1999**, *280*, 77–81. [CrossRef]
2. Copeland, B.J.; Sylvan, R. Beyond the universal Turing machine. *Australas. J. Philos.* **1999**, *77*, 46–66. [CrossRef]
3. Copeland, B.J. Hypercomputation. *Minds Mach.* **2002**, *12*, 461–502. [CrossRef]
4. Copeland, B.J. Hypercomputation: Philosophical issues. *Theor. Comput. Sci.* **2004**, *317*, 251–267. [CrossRef]
5. Syropoulos, A. *Hypercomputation: Computing Beyond the Church–Turing Barrier*; Springer: New York, NY, USA, 2008.
6. Shagrir, O.; Pitowsky, I. Physical hypercomputation and the Church-Turing thesis. *Minds Mach.* **2003**, *13*, 87–101. [CrossRef]
7. Button, T. SAD Computers and two versions of the Church-Turing Thesis. *Br. J. Philos. Sci.* **2009**, *60*, 765–792. [CrossRef]
8. Davis, M. The myth of hypercomputation. In *Alan Turing: Life and Legacy of a Great Thinker*; Teuscher, C., Ed.; Springer: Berlin, Germany, 2004; pp. 195–211.
9. Piccinini, G. The physical Church–Turing Thesis: Modest or bold? *Br. J. Philos. Sci.* **2011**, *62*, 733–769. [CrossRef]
10. Németi, I.; Dávid, G. Relativistic computers and the Turing barrier. *Appl. Math. Comput.* **2006**, *178*, 118–142. [CrossRef]
11. Copeland, B.J. The broad conception of computation. *Am. Behav. Sci.* **1997**, *40*, 690–716. [CrossRef]
12. Turing, A.M. Systems of logic based on ordinals. *Proc. Lond. Math. Soc. Ser. 2* **1939**, *45*, 161–228. [CrossRef]
13. Church, A. *The Calculi of Lambda-Conversion*; Princeton University Press: Princeton, NJ, USA, 1941.
14. Marr, D. *Vision*; W. H. Freeman: San Francisco, CA, USA, 1982.
15. Block, N. Psychologism and behaviorism. *Philos. Rev.* **1981**, *90*, 5–43. [CrossRef]
16. Papadimitriou, C.H. *Computational Complexity*; Addison-Wesley: Reading, MA, USA, 1994.
17. Dean, W. Algorithms and the mathematical foundations of computer science. In *Gödel's Disjunction: The Scope and Limits of Mathematical Knowledge*; Horsten, L., Welch, P., Eds.; Oxford University Press: Oxford, UK, 2016; pp. 19–66.
18. Gurevich, Y. The sequential ASM thesis. *Bull. Eur. Assoc. Theor. Comput. Sci.* **1999**, *67*, 93–124.
19. Gurevich, Y. Sequential Abstract State Machines capture sequential algorithms. *Acm Trans. Comput. Log.* **2000**, *1*, 77–111. [CrossRef]
20. Moschovakis, Y.N. What is an algorithm? In *Mathematics Unlimited—2001 and Beyond*; Engquist, B., Schmid, W., Eds.; Springer: Berlin, Germany, 2001.
21. Blass, A.; Dershowitz, N.; Gurevich, Y. When are two algorithms the same? *Bull. Symb. Log.* **2009**, *15*, 145–168. [CrossRef]
22. Knuth, D.E. Algorithms in modern mathematics and computer science. In *Algorithms in Modern Mathematics and Computer Science*; Ershov, A.P., Knuth, D.E., Eds.; Springer: Berlin/Heidelberg, Germany, 1981; pp. 82–99.
23. Shagrir, O. Advertisement for the philosophy of the computational sciences. In *The Oxford Handbook of Philosophy of Science*; Humphreys, P., Ed.; Oxford University Press: Oxford, UK, 2016; pp. 15–42.
24. Copeland, B.J. The Church-Turing thesis. In *The Stanford Encyclopedia of Philosophy*, Summer 2020 ed.; Zalta, E.N., Ed.; 2020. Available online: https://plato.stanford.edu/archives/sum2020/entries/church-turing/ (accessed on 23 September 2022).
25. Copeland, B.J. Narrow versus wide mechanism. *J. Philos.* **2000**, *97*, 5–32. [CrossRef]
26. Sieg, W. Calculations by man and machine: Conceptual analysis. In *Proceedings of the Reflections on the Foundations of Mathematics (Essays in Honor of Solomon Feferman)*; Volume 15 of Lectures Notes in Logic, Association of Symbolic Logic; Sieg, W., Sommer, R., Talcott, C., Eds.; Cambridge University Press: Cambridge, UK, 2002; pp. 390–409.
27. Turing, A.M. Proposals for development in the Mathematics Division of an Automatic Computing Engine (ACE). Report to the Executive Committee of the National Physics Laboratory. In *Collected Works of A. M. Turing: Mechanical Intelligence*; Ince, D.C., Ed.; Elsevier: Amsterdam, The Netherlands, 1992; pp. 1–86.
28. Gandy, R.O. Church's thesis and principles of mechanisms. In *The Kleene Symposium*; Barwise, J., Keisler, H.J., Kunen, K., Eds.; North Holland: Amsterdam, The Netherlands, 1980; pp. 123–145.
29. Gandy, R.O. The confluence of ideas in 1936. In *The Universal Turing Machine: A Half-Century Survey*; Herken, R., Ed.; Oxford University Press: Oxford, UK, 1988; pp. 55–111.
30. Rogers, H. *Theory of Recursive Functions and Effective Computability*; McGraw-Hill: New York, NY, USA, 1967.
31. Knuth, D.E. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 3rd ed.; Addison-Wesley: Upper Saddle River, NJ, USA, 1997.
32. Boolos, G.; Burgess, J.P.; Jeffrey, R.C. *Computability and Logic*, 5th ed.; Cambridge University Press: Cambridge, UK, 2007.
33. Cleland, C.E. The concept of computability. *Theor. Comput. Sci.* **2004**, *317*, 209–225. [CrossRef]
34. Cleland, C.E. On effective procedures. *Minds Mach.* **2002**, *12*, 159–179. [CrossRef]
35. Etesi, G.; Németi, I. Non-Turing computations via Malament–Hogarth space-times. *Int. J. Theor. Phys.* **2002**, *41*, 341–370. [CrossRef]
36. Shagrir, O. Effective computation by humans and machines. *Minds Mach.* **2002**, *12*, 221–240. [CrossRef]
37. Mendelson, E. On some recent criticism of Church's Thesis. *Notre Dame J. Form. Log.* **1963**, *4*, 201–205. [CrossRef]
38. Shapiro, S. Computability, proof, and open-texture. In *Church's Thesis After 70 Years*; Olszewski, A., Woleński, J., Janusz, R., Eds.; Ontos Verlag: Heusenstamm, Germany, 2006; pp. 420–455.
39. Black, R. Proving Church's Thesis. *Philos. Math.* **2000**, *8*, 244–258. [CrossRef]
40. Shagrir, O. *The Nature of Physical Computation*; Oxford University Press: Oxford, UK, 2022.
41. Smith, P. *An Introduction to Gödel's Theorems*, 2nd ed.; Cambridge University Press: Cambridge, UK, 2013.
42. Folina, J. Church's Thesis: Prelude to a proof. *Philos. Math.* **1998**, *6*, 302–323. [CrossRef]

43. Matuschak, A.; Nielsen, M.A. *Quantum Computing for the Very Curious*; San Francisco, 2019. Available online: https://quantum.country/qcvc (accessed on 23 September 2022).
44. Turing, A.M. Solvable and unsolvable problems. In *The Essential Turing*; Copeland, B.J., Ed.; Oxford University Press: Oxford, UK, 2004; pp. 582–595. Originally published in *Science News*, **1954**, *31*, 7–23.
45. Turing, A.M. On computable numbers, with an application to the *Entscheidungsproblem*. *Proc. Lond. Math. Soc. Ser. 2* **1936**, *42*, 230–265.
46. Burkholder, L. Computing. In *A Companion to the Philosophy of Science*; Newton–Smith, W.H., Ed.; Blackwell: Oxford, UK, 2000; pp. 44–55.
47. Oppenheim, P.; Putnam, H. Unity of science as a working hypothesis. In *Concepts, Theories, and the Mind–Body Problem*; Feigl, H., Scriven, M., Maxwell, G., Eds.; Minnesota Studies in the Philosophy of Science, University of Minnesota Press: Minneapolis, MN, USA, 1958; Volume 2, pp. 3–36.
48. Church, A. *Introduction to Mathematical Logic*; Princeton University Press: Princeton, NJ, USA, 1956.
49. Crane, T. *The Mechanical Mind*, 2nd ed.; Routledge: London, UK, 2003.
50. Johnson-Laird, P.N. *Mental Models*; Cambridge University Press: Cambridge, UK, 1983.
51. Dennett, D.C. *Brainstorms*; MIT Press: Cambridge, MA, USA, 1978.
52. Copeland, B.J. Turing's O-machines, Searle, Penrose and the brain. *Analysis* **1998**, *58*, 128–138. [CrossRef]
53. Searle, J.R. Minds, brains, and programs. *Behav. Brain Sci.* **1980**, *3*, 417–424. [CrossRef]
54. Searle, J.R. *The Rediscovery of the Mind*; MIT Press: Cambridge, MA, USA, 1992.
55. Copeland, B.J. The curious case of the Chinese gym. *Synthese* **1993**, *95*, 173–186. [CrossRef]
56. Sprevak, M. Chinese rooms and program portability. *Br. J. Philos. Sci.* **2007**, *58*, 755–776. [CrossRef]
57. Fodor, J.A. The appeal to tacit knowledge in psychological explanation. *J. Philos.* **1968**, *65*, 627–640. [CrossRef]
58. Lycan, W.G. Form, function, and feel. *J. Philos.* **1981**, *78*, 24–50. [CrossRef]
59. Block, N. Troubles with functionalism. In *Perception and Cognition: Issues in the Foundations of Psychology, Minnesota Studies in the Philosophy of Science*; Savage, C.W., Ed.; University of Minnesota Press: Minneapolis, MN, USA, 1978; Volume 9, pp. 261–325.
60. Putnam, H. Philosophy and our mental life. In *Mind, Language and Reality, Philosophical Papers, Volume 2*; Cambridge University Press: Cambridge, UK, 1975; pp. 291–303.
61. Shagrir, O. Multiple realization, computation and the taxonomy of psychological states. *Synthese* **1998**, *114*, 445–461. [CrossRef]
62. Cutland, N. *An Introduction to Recursive Function Theory*; Cambridge University Press: Cambridge, UK, 1980.
63. Soare, R. The history and concept of computability. In *Handbook of Computability Theory*; Griffor, E.R., Ed.; Elsevier: New York, NY, USA, 1999; pp. 3–36.
64. Gurevich, Y. What is an algorithm? In *SOFSEM 2012: Theory and Practice of Computer Science. Lecture Notes in Computer Science, Volume 7147*; Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G., Eds.; Springer: Berlin, Germany, 2011; pp. 31–42.
65. Blass, A.; Gurevich, Y. Algorithms: A quest for absolute definitions. In *Church's Thesis after 70 Years*; Olszewski, A., Woleński, J., Janusz, R., Eds.; Ontos Verlag: Heusenstamm, Germany, 2006; pp. 24–57.
66. Chabert, J.L.; Weeks, C.; Barbin, E.; Borowczyk, J.; Guillemot, M.; Michel-Pajus, A.; Djebbar, A.; Martzloff, J.C. *A History of Algorithms: From the Pebble to the Microchip*; Springer: Berlin, Germany, 1999.
67. Knuth, D.E. Ancient Babylonian algorithms. *Commun. ACM* **1972**, *15*, 671–677. [CrossRef]
68. Deutsch, D.; Ekert, A.; Lupacchini, R. Machines, logic and quantum physics. *Bull. Symb. Log.* **2000**, *3*, 265–283. [CrossRef]
69. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*, 10th anniversary ed.; Cambridge University Press: Cambridge, UK, 2010.
70. Cleve, R.; Ekert, A.; Macchiavello, C.; Mosca, M. Quantum algorithms revisited. *Proc. R. Soc. Ser. A* **1998**, *454*, 339–354. [CrossRef]
71. Deutsch, D.; Jozsa, R. Rapid solution of problems by quantum computation. *Proc. R. Soc. Ser. A* **1992**, *439*, 553–558.
72. Deutsch, D. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. Ser. A* **1985**, *400*, 97–117.
73. Barz, S. Quantum computing with photons: Introduction to the circuit model, the one-way quantum computer, and the fundamental principles of photonic experiments. *J. Phys. B At. Mol. Opt. Phys.* **2015**, *48*, 083001. [CrossRef]
74. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
75. Ekert, A.; Jozsa, R. Quantum algorithms: Entanglement-enhanced information processing. *Philos. Trans. R. Soc. Lond. Ser. A* **1998**, *356*, 1769–1782. [CrossRef]
76. Bennett, C.H.; Brassard, G.; Crépeau, C.; Jozsa, R.; Peres, A.; Wootters, W.K. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.* **1993**, *70*, 1895–1899. [CrossRef]
77. Gottesman, D.; Chuang, I.L. Quantum teleportation is a universal computational primitive. *Nature* **1999**, *402*, 390–393. [CrossRef]
78. Hosten, O.; Rakher, M.T.; Barreiro, J.T.; Peters, N.A.; Kwiat, P.G. Counterfactual quantum computation through quantum interrogation. *Nature* **2006**, *949–952*. [CrossRef] [PubMed]
79. Adleman, L.M. Molecular computation of solutions to combinatorial problems. *Science* **1994**, *266*, 1021–1024. [CrossRef] [PubMed]
80. Lipton, R.J. DNA solution of hard computational problems. *Science* **1995**, *268*, 542–545. [CrossRef] [PubMed]
81. Barrett, H.C. Enzymatic computation and cognitive modularity. *Mind Lang.* **2005**, *20*, 259–287. [CrossRef]
82. Adamatzky, A. (Ed.) *Advances in Physarum Machines*; Springer: Berlin, Germany, 2016.

83. Adamatzky, A. Towards fungal computer. *Interface Focus* **2018**, *8*, 20180029. [CrossRef] [PubMed]
84. Tanaka, G.; Yamane, T.; Héroux, J.B.; Nakane, R.; Kanazawa, N.; Takeda, S.; Numata, H.; Nakano, D.; Hirose, A. Recent advances in physical reservoir computing: A review. *Neural Netw.* **2019**, *115*, 100–123. [CrossRef] [PubMed]
85. Wu, K.; García de Abajo, J.; Soci, C.; Ping Shum, P.; Zheludev, N.I. An optical fiber network oracle for NP-complete problems. *Light. Sci. Appl.* **2014**, *3*, e147. [CrossRef]
86. Ulmann, B. *Analog Computing*; Oldenbourg Wissenschaftsverlag: Munich, Germany, 2013.
87. Goldreich, O. *Computational Complexity: A Conceptual Perspective*; Cambridge University Press: Cambridge, UK, 2008.
88. Yao, A.C.C. Classical physics and the Church–Turing Thesis. *J. ACM* **2003**, *50*, 100–105. [CrossRef]
89. Aharonov, D.; Vazirani, U. Is quantum mechanics falsifiable? A computational perspective on the foundations of quantum mechanics. In *Computability: Turing, Gödel, Church, and Beyond*; Copeland, B.J., Posy, C.J., Shagrir, O., Eds.; MIT Press: Cambridge, MA, USA, 2013; pp. 329–349.
90. Bernstein, E.; Vazirani, U. Quantum complexity theory. *SIAM J. Comput.* **1997**, *26*, 1411–1473. [CrossRef]
91. Yamakawa, T.; Zhandry, M. Verifiable quantum advantage without structure. *arXiv* **2022**, arXiv:2204.02063v2. [CrossRef]
92. Feynman, R.P. Simulating physics with computers. *Int. J. Theor. Phys.* **1982**, *21*, 467–488. [CrossRef]
93. Wittgenstein, L. *Remarks on the Philosophy of Psychology*; Blackwell: Oxford, UK, 1980; Volume 1.