1 Introduction

Providing a comprehensive introduction to issues in physics and computation within 30,000 words is quite a challenge. Without further ado, here is what to expect. In Section 2, I introduce the reader to Turing's work on the concept of computation, focusing on his 1936 paper. In Section 3, I present the Church–Turing thesis and its ramifications for physical limits on computational possibilities. In Section 4, I present the problem of how to characterize computation in physical systems and some of the accounts that attempt to address it. In Section 5, I introduce the reader to three different types of quantum computers. Finally in Section 6, I explore proposed explanations of quantum speedup.[1] My hope is that by the end of the Element the reader will be equipped with the resources to further pursue investigations into the relation between physics and computation in any of the three different areas articulated above: the limits of computational possibility, accounts of computation in physical systems, or investigations into quantum speedup. That said, this Element isn't merely introductory. New arguments and novel criticisms of different views are distributed throughout.

2 Turing's 1936 Paper

In this section I will briefly present some of the ideas and results from Turing's landmark (1936) paper "On computable numbers, with an application to the Entscheidungsproblem." This is without a doubt one of the great papers of the twentieth century, widely cited and widely misunderstood. I want to inoculate the reader from at least a few misunderstandings by discussing some key points. Most importantly, I will emphasize that, when Turing uses the term "computer," he means "a human performing computations," which is very different from what we now mean by "computer." This fundamental fact is often overlooked and is a key source of confusion that infects discussions of computation. Setting the historical stage properly and explaining Turing (1936) in detail are not among the goals here. For that, I would refer the reader to Copeland (2004). Instead, I want to give the reader the bare-bones understanding of Turing (1936) to serve as a springboard for discussing physics and computation quite generally.

[1] In an Element like this, one might expect to find a discussion on Landauer's principle, which states that erasure of n bits of information always incurs a cost of $k \ln n$ in thermodynamic entropy. Unfortunately, there is no space for a discussion of the principle in this Element. See O. Maroney (2009) for an introduction to the issues associated with Landauer's principle.

2.1 Formal Representation of Human Computers

Turing, on the first page of his paper (1936), writes

> We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. […] For the present I shall only say that the justification lies in the fact that the *human* [emphasis added] memory is necessarily limited.

Front and center, it is obvious that in an attempt to characterize the computable numbers, Turing's intent is to characterize the class of numbers whose decimal expansion could be provided by a human. To do so, Turing provides a mathematical representation of what he views as the essential features of humans that enable and limit human computational abilities. Turing then uses this mathematical representation of humans to indicate what the class of computable numbers is.

Turing's mathematical representation of what is essential to human computing defines what a Turing machine is. A Turing machine is not an actual machine, but a mathematical construct like circles or triangles. A Turing machine consists, in part, of an endless tape partitioned into various cells that are capable of bearing symbols. A Turing machine is equipped with a read/write head that is capable of reading symbols from a finite set on the cell in the tape that it is currently positioned over, writing or erasing symbols, and shifting left or right one cell. A Turing machine also has a finite set of internal states. Particular Turing machines are characterized by their machine table, which corresponds to what we would now call a program. The machine table determines the behavior of the machine. Given an internal state and scanned symbol, which we refer to jointly as the *configuration* of the machine, the program indicates whether the read/write tape head should erase or write a symbol, whether the head should move or not, and finally, what internal state to update to. Machine tables can be represented as sets of instructions of the form $\langle q_i, s_i, q_j, s_j, d\rangle$, where $q_{i/j}$ are the current/subsequent internal states, $s_{i/j}$ are the current/subsequent symbols on the tape, which include the blank symbol, and $d \in \{-1, 0, 1\}$ indicates how the read/write tape head should move.[2] See Figure 1.

Turing devotes Section 9 of his paper to arguing that Turing machines capture the essential properties of humans performing computations. He draws our attention to several important aspects of human computations. First, when humans perform computations, they typically utilize a spatial medium with distinguishable locations (e.g. a piece of paper with a grid on it). Turing views

[2] This notation is not Turing's, but it has the advantage of being more transparent than Turing's.

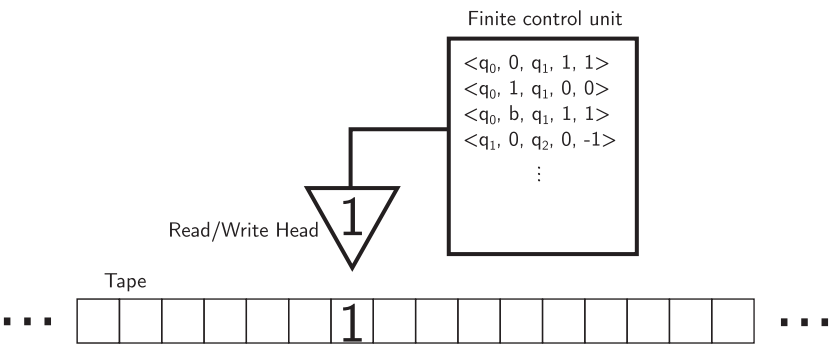

Figure 1 A Turing machine with an unbounded tape divided into cells, a read/write tape head that can move one left or right one cell or stay stationary, and a finite control unit.

the number of dimensions of the medium as being unimportant; hence he represents this aspect of computation with a one-dimensional tape with distinguishable cells. Humans distinguish only a finite number of symbols, hence the limitation of Turing machines to a finite set of symbols. Humans can distinguish only small sequences of symbols at a glance. Turing uses the convincing example of the two (different) numbers 9999999999999999 and 999999999999999 to illustrate the point. Distinguishing these two numbers would involve some sort of procedure. Turing thinks that it is no computational limitation to distinguish only one symbol at a time and use memory to recover the whole number. Hence, the read/write head on a Turing machine reads a single symbol at a time. Humans can change only a single symbol at a time, and so too can the read/write head. Turing portrays humans as having only a finite number of states of mind, hence a finite number of internal states for Turing machines. The state of mind of a human computer and the symbol or symbols in view determine the next state of mind and action on the page in terms of where attention is directed to and what is done to symbols on the page. Similarly, the internal state and scanned symbol determine what a Turing machine does next to update its configuration. Humans utilize a finite set of primitive operations when performing a computation, and computers do too. Those primitive operations involve a change of symbol on the page with a possible change of mind, or a possible change of observed squares, together with a change of mind. Primitive operations of a Turing machine are similar. They involve a change of symbol on the tape along with a possible change of internal state, or a change of scanned tape cell together with a change in internal state. Clearly Turing aimed to represent the essential features of human computers mathematically with his machines.[3]

[3] Turing's portrayal of human computational abilities involves no elements of chance, insight, intuition, or ingenuity, and his results should be understood accordingly.

2.2 The Results

What are the computable numbers? Computable numbers are any infinite decimal that can be generated by a Turing machine and may or may not be prefaced by an integer.[4] So, Turing machines that halt (i.e. stop after some finite number of basic operations) do not correspond to computable numbers, as defined by Turing. Copeland (2004, 33) notes that this use of "computable number" might seem too restrictive. Modern writers consider finite sequences generated by Turing machines as corresponding to computable numbers as well.

In addition to developing a representation of human computers and providing a definition of computable numbers, Turing developed the notion of a *universal* Turing machine. A universal Turing machine was a machine that could take as an input on a tape a description of a particular Turing machine and then emulate its output behavior. As remarked above, Turing machines can be described by their machine tables. These tables are just lists of instructions. Turing came up with an encoding scheme so that each distinct list of instructions was associated with a distinct number, called the *description number of the machine*. From that number one can produce the the *standard description* of a Turing machine and use it as input to a universal machine, and it will produce the output of the machine associated with the standard description. By showing how to assign description numbers to Turing machines, Turing demonstrated that the class of computable numbers was enumerable.

Turing also argued that the Entscheidungproblem is not solvable. The Entscheidungsproblem was articulated in Hilbert and Ackermann's (1928) *Grundzüge der theoretischen Logik*. The problem, as described therein, is to determine whether or not a given formula of the predicate calculus is universally valid, or whether those formulas are deducible from the axiom system of the predicate calculus, or whether a sentence of the predicate calculus is satisfiable. These are three equivalent ways of stating the problem (Davies, 2013). The problem was understood as inquiring into *our* limitations. Turing's proof of the undecidability of the Entscheidungsproblem proceeded by showing the following (Del Mol, 2019, Sec 2.4.2):

1. how, for each Turing machine T, it is possible to construct a corresponding formula **T** in first-order logic and
2. if there is a general method for determining whether **T** is provable, then there is a general method for proving that T will ever print 0.

[4] Note that the concept of a computable number is still highly idealized. Computable numbers cannot actually be computed!

Turing proved that there was no general method for proving that T will ever print 0, hence the Enscheidungsproblem is undecidable.[5]

In an appendix to his paper, Turing showed that the computable numbers coincided with the class of numbers identified by Church (1936) as the "effectively calculable" numbers.

It is worth emphasizing that our concept of computation arose from discussions in the foundations of mathematics between Church, Kleene, Gödel, and Post, among others. That said, for the purposes of this Element, what concerns us are the possible implications of this work in foundations of mathematics for what is computable, what a computer is, and how to identify what number, function, or other thing a computer is computing.[6] We are interested in computation in physical systems. It is to these topics that we now turn.

3 The Church–Turing Thesis and the Physical Church–Turing Thesis

In this section we explore possible limitations to our computational capacities. Both the Church–Turing Thesis and the physical Church–Turing thesis will be discussed. The Church–Turing thesis is widely misunderstood as making claims about what is computationally possible, but, as will be argued below, it is almost definitional in character. The physical Church–Turing thesis, something completely different, can be very coarsely construed as the claim that the computational power of Turing machines cannot be exceeded.[7] Given what we have learned in Section 2, this would be a remarkable fact, as Turing machines are just formal representations of human computational abilities. Can no physical system do better? Let us investigate.[8]

3.1 The Church–Turing Thesis

As we have seen, Turing set out to describe the class of numbers that are computable by humans. Even so, it is not obvious that the class of numbers that he identified is the right class. The kind of processes that Turing machines represent are those that leave no room for insight, intuition, or ingenuity. Given that humans do have insight, intuition, or ingenuity, it would seem that there

[5] For more details about how Turing carried out the proof of the undecidability of the Entscheidungsproblem, see Del Mol (2019).
[6] I am grateful to an anonymous referee for suggesting that this point be emphasized.
[7] See Pitowsky (2007) for a description of how theses about computation change over time.
[8] For a more detailed discussion of these matters, see Copeland (2019) and Piccinini (2015), chapters 15 and 16. Copeland, Shagrir, and Sprevak (2018) also discuss these matters and related ones.

are functions that might be computable by humans that have not been captured by Turing. It is perfectly reasonable if the reader is puzzled.

Puzzlement can be dispelled, in part, by considering a problem that had become pressing in the foundations of mathematics in the 1920s and 1930s: that of defining what an effectively calculable function was, where *effectively calculable* was understood in an informal way. Copeland (2019) provides a useful definition:

> A method, or procedure, *M*, for achieving some desired result is called "effective" (or "systematic" or "mechanical") just in case:
> *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);
> *M* will, if carried out without error, produce the desired result in a finite number of steps;
> *M* can (in practice or in principle) be carried out by a human being unaided by any machinery except paper and pencil;
> *M* demands no insight, intuition, or ingenuity, on the part of the human being carrying out the method.[9]

An effectively calculable function (or number, or whatever) is one that is calculable by an effective method.[10]

The precision of Copeland's characterization of effective method and corresponding characterization of effectively calculable function should not lead one to believe that the concept was just as clear in the minds of mathematicians and logicians in the 1920s and 1930s, though no doubt something of the sort was present.

Several candidate precisifications for the intuitive concept of effectively calculable function were given in the 1930s. Church (1937) suggested that the class of λ-definable functions was the set of effectively calculable functions. In footnote 3 of that paper, Church notes that work by Kleene, Church, and Rosser shows that the λ-definable functions are also the recursive functions, characterized in Gödel's 1934 lectures at Princeton. It is worth noting that Gödel was at the Institute for Advanced Studies, Church was a professor of mathematics at Princeton, and Kleene and Rosser were his graduate students. The claim that the effectively calculable functions were the λ-definable functions or the recursive functions came to be known as *Church's thesis*.[11]

[9] See Piccinini (2015, 247) for an alternative characterization for an effective process.

[10] Copeland (2019) provides a useful example of the connection between computable number and computable function: "For example, the computable number .14159... (formed of the digits following the decimal point in π, 3.1419...) corresponds to the computable function: f(1) = 1, f(2) = 4, f(3) = 1, f(4) = 5, f(6) = 9,...".

[11] Copeland (2019) calls Church's thesis the claim that a function of positive integers is effectively calculable *only if* λ-definable (or, equivalently, recursive), and distinguishes it from the

Turing can be seen as providing a precisification of the concept of effectively calculable function.

It is reasonable to assume that Turing had something like the concept of an effective method in mind when he wrote his 1936 paper. The first line of the paper reads, "The 'computable' numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means." On that page, he writes that the justification for this definition of computable numbers "lies in the fact that human memory is limited." The reference to finite means and human memory on that first page suggests that Turing did have in mind something like an effectively calculable number (or function) as defined above, though he used the phrase "computable function" instead.

Now, back to our puzzle: Why think that Turing had succeeded in identifying the *humanly* computable numbers, functions, and so on when human abilities like insight, intuition, or ingenuity are not taken into account? It was because Turing had in mind the concept of effective method when he considered human computational abilities, and that excluded insight, intuition, or ingenuity. Such a concept had been of increasing importance in the 1920s and early 1930s, in part because of Hilbert's program to put mathematics on solid epistemological footing, and also due to Gödel's incompleteness results.[12]

The reader has likely anticipated what Turing's thesis is, but it does deserve to be explicitly stated. *Turing's thesis* is that a number, function, or anything else is effectively calculable only if computable by a Turing machine. Turing (1937) demonstrated that the class of λ-definable functions was equivalent to the class of functions computable by a Turing machine and that that class of functions was equivalent to the class of recursive functions. So, Church's thesis and Turing's thesis pick out the same class of functions. One rendering of the Church–Turing thesis is that a function is effectively calculable only if Turing computable or λ-definable or recursive. This captures Kleene's intent when he first used the phrase "Church–Turing Thesis" in his 1967 book.

The question of whether the Church–Turing thesis is true naturally arises. The concept of an effectively computable function was imprecise, but shared among those working in foundations of mathematics and logic in the early twentieth century. Whether the thesis is "true" depends on whether Church or Turing provided the appropriate precisification of the concept of an effectively

converse of the thesis, that a function of the positive integers is effectively calculable *if* it is λ-definable. He does note that Church himself didn't distinguish between the two.

[12] Sieg (1994) is an excellent resource for understanding why people interested in the foundations of logic and mathematics were concerned about clarifying the concept of an effectively calculable function or the related concept of effective method.

calculable function as it was understood by those employing the concept. On that matter, it seems like the thesis was accepted by those concerned. Church (1937, 43) writes, "It is thus immediately clear that computability, so defined, can be identified with (especially is no less general than) the notion of effectiveness as it appears in certain mathematical problems (various forms of the Entscheidungsproblem, various problems to find complete sets of invariants in topology, group theory, etc., and in general any problem which concerns the discovery of an algorithm)." It is of note, and well known, that Gödel did not accept Church's thesis, but did accept Turing's thesis. What is of significant difference between Church's presentation of his thesis (1936) and Turing's presentation (1936) was that Turing made it transparent how his machines could represent humans using an effective method for calculation of numbers (and functions too). Insofar as Turing machines adequately represented effective methods for the calculation of numbers, one should be confident that the limitations of the machines would be limitations of effective calculability.[13] So, the Church–Turing thesis does seem to be true, but it is almost trivially so, given that it has an almost definitional status. Further evidence that the thesis is true comes from the fact that alternative representations of effective methods arrive at the same class of functions that Turing and Church arrived at as detailed in Kleene (1953).[14]

Given the above discussion, it should be reasonably clear that the Church–Turing thesis, even if true, is rather limited. It does not imply anything about what computers can do, when "computers" is understood as it is today as "a system that we use to perform computational tasks" rather than how Turing understood the term. Using Turing's version of the thesis, recall that it states that *if* a function is effectively calculable, *then* it is calculable by a Turing machine. In the contrapositive, if a Turing machine cannot compute a function, then it is not effectively calculable. "Effectively calculable" has the essential connection to a human process lacking insight, intuition, or ingenuity. The thesis, if true, circumscribes what is possible *given certain limited means*. There seems to be no prima facie reason for thinking that the Church–Turing thesis, if true, circumscribes what is computationally possible in the contemporary sense. One can reasonably expect that human capabilities extend beyond what Turing machines are capable of because we do have insight, intuition, or ingenuity. Moreover, we could expect that systems characterized by physical processes very different from those a Turing machine has available to it would have

[13] Turing also offered another argument in Section 9 of his paper (1936). See Copeland (2019) for a discussion.

[14] Kripke (2013) argues that the Church–Turing thesis does admit proof, and he attempts to provide one. See Copeland (2019) for an overview of Kripke's proof.

different computational abilities. It is also of note that the Church–Turing thesis, even if true, does not provide any indication about what computations can be done efficiently. Recall that computable numbers are computed by Turing machines that never halt! If one wants to be in the business of making claims about what is computationally possible or what is efficiently computable, then one has to go beyond the Church–Turing thesis.[15]

3.2 Beyond the Church–Turing Thesis

Given what the Church–Turing thesis actually claims, it may seem rather underwhelming compared to the grandiose claims that have been made regarding it. That is because many of the claims attributed to the Church–Turing thesis ought to be attributed to a completely different set of theses: the *simulation thesis* and the *maximality thesis*.[16]

Simulation thesis: Turing machines can simulate any physical system to any degree of approximation.[17]

Maximality thesis: All functions that can be generated by machines (working in accordance with a finite program of instructions) are computable by effective methods (Copeland, 2019).

These distinct theses are often combined into what is referred to as the *physical Church–Turing thesis*, which has many nonequivalent formulations. For conceptual clarity, it is useful to distinguish the simulation and maximality theses.

It has been emphasized in Copeland (2002) and Piccinini (2015, Ch. 15) that, when investigating issues relating to the physical Church–Turing thesis, one should distinguish between theses as they apply to physical possibilities broadly or narrowly construed. Construed narrowly, one can consider the truth of the above theses with respect to our best physical theories and conditions present in the actual world. Construed broadly, one can consider the truth of the above theses with respect to our best theories, ignoring actual conditions, or even more broadly constrained only by logical possibility. Let us examine the above theses with such distinctions in mind.[18]

[15] See Copeland (2019, sec. 2) for a discussion of the many ways in which the Church–Turing thesis has been misunderstood.

[16] With respect to confusions regarding the Church–Turing thesis, one might also add to the list a claim to the effect that the efficiency of simulations on Turing machines is correlated to the efficiency of other physical processes. Quantum speedup, as discussed in section 6, seems to refute this.

[17] See Copeland (2019) and Pitowsky (2002) for alternative formulations of the simulation thesis.

[18] For an alternative conceptualization of the various issues involved in the physical Church–Turing thesis, see Piccinini (2015, Ch. 15).

3.3 The Simulation Thesis

The simulation thesis claims that Turing machines can simulate any physical system to any degree of approximation. The simulation thesis appears to be straightforwardly false when properly understood. Piccinini (2015) points out that one ought to disallow Turing machines that instantiate input output tables that are programmed directly into a Turing machine from counting as simulating physical systems. This stands to reason, if we want to capture the concept of simulation as it is used in science. Simulations are used in practice to *predict* how systems of interest will behave. If one programs a Turing machine using a lookup table, that means that one needs to have the lookup table in hand prior to programming the Turing machine. To do so, one must already know how the system of interest behaves in every possible case. So, rather than predicting how a system of interest might behave in a particular situation, Turing machines that are programmed with a lookup table are just describing how the system will behave. The simulation thesis, if it is to have any significance, ought to make claims about simulation in practice, else it suffers from triviality. Let us turn to the evidence against the thesis, understood in its nontrivial sense.

The existence of some chaotic systems is devastating to the simulation thesis. Some chaotic systems are such that no matter how precisely we specify initial conditions, there will be a time at which we fail to predict the behavior of the system. Evidence of the existence of such systems does not come only from highly idealized mathematical models. We need only turn our attention to the double pendulum. Shinbrot et al. (1992) take pains to create two identical pendulums that are released in as identical as possible initial conditions, yet the two systems diverge in behavior very quickly. Our theoretical models of such systems predict this behavior as well. If there exist systems that diverge so quickly, the simulation thesis doesn't even get a foot in the door. Even if we knew exactly what the laws of the world are, we don't have the right control to have systems evolve identically. So, even if the laws of physics were computable, we cannot create, much less specify, initial conditions that would allow one to simulate the behavior of some physical systems for any extended length of time.

Advocates of the simulation thesis might retort that this violates the spirit of the simulation thesis. In such cases, it is no fault of Turing machines that they fail to simulate a system. They might think that there is no reason to believe that, *if* we could precisely specify the initial conditions, Turing machines would fail to be able to predict behavior. This would be a significant weakening of the