Effective Computation by Humans and Machines

Oron Shagrir
Department of Philosophy, The Hebrew University of Jerusalem, Jerusalem 91905, Israel.
E-mail: shagrir@cc.huji.ac.il

There is an intensive discussion nowadays about the meaning of effective computability, with implications to the status and provability of the Church-Turing Thesis (CTT). I begin by reviewing what has become the dominant account of the way Turing and Church viewed, in 1936, effective computability. According to this account, to which I refer as the Gandy-Sieg account, Turing and Church aimed to characterize the functions that can be computed by a human computer. In addition, Turing provided a highly convincing argument for CTT by analyzing the processes carried out by a human computer. I then contend that if the Gandy-Sieg account is correct, then the notion of effective computability has changed after 1936. Today computer scientists view effective computability in terms of finite machine computation. My contention is supported by the current formulations of CTT, which always refer to machine computation, and by the current argumentation for CTT, which is different from the main arguments advanced by Turing and Church. I finally turn to discuss Robin Gandy's characterization of machine computation. I suggest that there is an ambiguity regarding the types of machines Gandy was postulating. I offer three interpretations, which differ in their scope and limitations, and conclude that none provides the basis for claiming that Gandy characterized finite machine computation.
Key words: effective computability, Gandy machines, human computation, machine computation, physical computation, The Church-Turing Thesis.

The Church-Turing thesis identifies the effectively computable functions with those functions that can be computed by a universal Turing machine (the Turing computable functions). But what exactly is an effectively computable function? What is the meaning of 'effective computation'? In the first section of this paper, I review what has become the dominant view of effective computability. According to this view, Church (1936a) and Turing (1936) considered the functions that can be computed by a human computer, namely, by a human who computes by means of an effective procedure. In addition, Turing (1936) provided a highly convincing argument for the Church-Turing thesis by analyzing the processes carried out by a human computer. An analysis of machine computation, something very different, was suggested much later, notably by Robin Gandy, in his 1980 paper "Church's Thesis and Principles for Mechanisms". This view – advanced, for example, in Copeland (1996), Shagrir (1997), and Shagrir and Pitowsky (forthcoming) – has its roots in Gandy's seminal papers (1980, 1988) and those of Wilfried Sieg (1994, 1997, 2001, forthcoming), where it is explicated, honed, and extended. It will be apt, therefore, to refer to it as the Gandy-Sieg account of effective computation.

In the second section, I argue that if the Gandy-Sieg account is correct, then since 1936 the way in which effective computation is understood has changed. In current computer science, effective computation is no longer identified with human computation, but, rather, it is identified with finite machine computation. The third section is devoted to Gandy's 1980 paper. After examining Gandy's analysis of machine computation, I attempt to better understand what kind of machines Gandy is really talking about. I argue that Gandy's analysis does not apply to all instances of finite machine computation.

1. Effective computation by humans

In 1936, Church, Kleene, Post, and Turing published pioneering papers on computability. Each paper provided a precise mathematical characterization of the class of effectively computable functions, namely the functions whose values can be computed by means of an effective procedure (algorithm).[1] Though the characterizations are quite different – the λ-calculus (Church), general recursive functions (Kleene), finite combinatory processes (Post), and Turing machines (Turing) – they are all extensionally equivalent: it was immediately proved that they identify precisely the same class of functions.[2] Church and Turing also put forward versions of the now renowned Church-Turing thesis:

> **The Church-Turing Thesis (CTT)**: Any effectively computable function (of positive integers) is Turing computable.[3]

What is an effectively computable function (intensionally speaking)? On the Gandy-Sieg account, an effectively computable function is one whose values can be computed by a human computer: "Both Church and Turing had in mind calculation by an abstract human being using some mechanical aids (such as paper and pencil). The word 'abstract' indicates that the argument makes no appeal to the existence of practical limits on time and space." (Gandy 1980: 123-124).[4] A human computer is, intuitively, an ideal human who computes values of a function, e.g., addition, by following an effective procedure. An effective procedure, in turn, can be understood as a finite set of instructions (each instruction expressed by means of a finite number of symbols), that, if followed correctly, can lead to any of a function's defined values in a finite number of steps. The execution of the instructions demands no special intellectual effort. The process, as Turing put it, is purely mechanical. The human computer is idealized in two ways. First, the human computer is not bounded by time and memory space. The computing agent can take as much finite time and as much finite memory space as he or she needs to arrive at a value of the function. Second, the computing agent follows the instructions correctly, making no errors. Following the procedure must lead to the correct values of the function. Given this understanding of effective computability, we can articulate this human version of CTT as following:

> **The Human version of the Church-Turing Thesis (HCTT)**: Any function (of positive integers) that can be computed by a human computer is Turing computable.

This does not mean that only human computers can effectively compute the function's values, but only that a human computer *can* compute these values.

This human version of effective computability is supported by two kinds of evidence. One pertains to the historical background to the 1936 developments. The wider context has to do with the central epistemic role of effective computation in logic and mathematics in the decades preceded the establishment of CTT. Starting with Frege (and perhaps with Leibniz), and culminating in Hilbert's program, mathematicians sought to justify mathematical theorems within the framework of formal systems, those systems whose constitutive condition is that the provability relation is effectively computable.[5] The more immediate context is Gödel's incompleteness theorems (Gödel, 1931), and the Entscheidungsproblem. Gödel's incompleteness theorems rest on the representation lemma, which asserts that any (primitive) recursive relation is representable in any formal system L that includes a minimal set of axioms for arithmetic.[6] It follows that if the metatheoretic relation "X is a proof of y" is recursive (Turing computable), it is representable (where X is a finite sequence of Gödel numbers of propositions and y is a Gödel number of a proposition in L). But since a condition on such a formal system is that the relation "X is a proof of y" must be effectively computable, it was possible, prior to 1936, to think that there might be complete formal systems for arithmetic in which the relation "X is a proof of y" is effectively computable but not recursive.[7] This possibility was eliminated only in 1936, when CTT was established: CTT ensures that all provability relations are representable.

The Entscheidungsproblem, formulated by Hilbert and Ackermann (1928: 72-73), concerns the decidability of first-order logic, namely, the question of whether there is an effective procedure for computing the validity (or provability) of any given first-order proposition.[8] Gödel's incompleteness theorems make it unlikely that there is such a decision procedure.[9] But again, all decidable relations must be examined to establish that none of them can determine whether or not a proposition is valid/provable. In order to establish the negative answer to the Entscheidungsproblem, one is forced to provide an exhaustive account of the effectively computable relations. When such an account became available in 1936, and CTT was established, Church and Turing could prove that first-order logic is undecidable (Church 1936a, 1936b; Turing 1936).

Given the mathematical problems that occupied the logicians of the time, it would not be implausible to say that they thought of effectively computability in terms of human computation. In the wider context of Frege and Hilbert, it is hard to make sense of a formal system unless we understand 'effectively computable' in terms of human computation. It would be of no relevance if the truth of a mathematical conjecture could be proven within a formal system with constructive rules governed by computational procedures that humans could not, even in principle, follow. In the more immediate context, the Entscheidungsproblem, like other questions of decidability, is understood as asking if a human can compute whether or not any first-order proposition is valid. It would be completely beside the point if no human, but some supermachine, could compute the validity of any first-order proposition. This might be thought to show that some machines can "solve" the Entscheidungsproblem, but inasmuch as the "solution" cannot be arrived at by human computation, it would not be considered a solution in the context of logic and mathematics.[10]

The second source of evidence for the claim that Turing and Church thought about effective computation in terms of human computation is the way they refer to computation in their writings. Turing (1936) often referred to computation in terms of human computation,[11] in addition to analyzing human computability (see below). Church was less explicit about this connection in 1936. But, in his review of Turing's 1936 paper, Church made clear that the decision problems are defined in terms of human computation: "In particular, a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a kind of a Turing machine. 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…)" (1937a: 42-43).

Let us now turn to examine the central arguments for CTT put forward by Church and Turing. Church's step-by-step argument for CTT appears in section 7 (pp. 100-102) of his 1936a. There are two versions of the argument. Following Gandy and Sieg, here we shall consider the second only. The argument can be reconstructed as follows:

Church's step-by-step argument:

1. A function *f* is effectively computable iff it is calculable in logic, namely, iff there is a logic L such that *f* is representable in L.
2. **Church's Central Thesis**: The steps of the computation governing the proofs in L are recursive.[12]
3. **Conclusion (CTT)**: Any effectively computable function is recursive.

In the first premise of the argument, Church identifies an effectively computable function as one that can be represented in some formal system L. This means that effective computability is equated with the provability of the formula representing *f* in L. This identification rests on results by Gödel (1931) and Kleene (1936) to the effect that any general recursive function can be represented in the system of *Principia Mathematica*.[13] The first premise thus seems to indicate that the logicians thought about the effective computable functions in the context of their role in mathematical systems such as formal systems or algebraic equations. We find this identification in Gödel (1946: 84, note *); we also see it in Turing (1936: 138-139), and in Hilbert and Bernays (1939). Given the epistemic nature of formal systems, the first premise of Church's argument attests, in my opinion, to the fact that the real goal of Church's argument is to secure HCTT, that is, to identify the functions computable by human computers.[14]

CTT, the argument's conclusion, clearly follows from the premises. The weakness of Church's argument lies at the second premise – Church's Central Thesis – according to which the relation "X is a proof of y" in L must be recursive. Church brings no argument for this premise. This is quite surprising given that he wants to rule out the possibility of a formal system in which this relation is effectively computable but not recursive. It thus seems viciously circular to assume the premise without any argument. But perhaps the verdict is too harsh. Church views the argument not as a conclusive proof, but as a "positive justification for the selection of a formal definition to correspond to an intuitive notion" (1936a: 100). Furthermore, in view of Kleene's analysis of general recursive functions and their representability, and in view of the "constructive" nature of formal systems, it is indeed hard to conceive a formal system in which the steps of a proof are not recursive. Still, this defense is not entirely convincing. The move from "it is hard to conceive such a logic" to "there is no such logic" calls for a justification. The stumbling block– as Sieg refers to it – impeding Church's argument is removed only if we can justify the implicit claim that the rules governing the system must be recursive. As we are about to see, Turing removes the impediment.

In sections 9 and 10 of his classic 1936 paper, Turing presents three arguments for CTT. Here, I consider only the first of these, thought by many to be the most convincing argument for HCTT. Gandy even considers this argument to constitute a proof of HCTT (1988: 76-78). In this argument, Turing removes the stumbling block besetting Church's argument, by analyzing human effective computation directly. The problem with Church's argument is that it does not explain why the steps of a computational process must be recursive. Turing analyzes these steps, demonstrating that they must be recursive.

He begins by recognizing that "the real question at issue is 'What are the possible processes which can be carried out in computing a number?'" (1936: 135). He then proceeds to formulate constraints on all such possible computational processes, constraints that are motivated by the limitation of human's sensory apparatus and memory. The constraints of determinacy, boundedness, and locality (in short, D-B-L) are reformulated in Sieg (1997) as follows:

(D) The immediately recognizable (sub-)symbolic configuration determines uniquely the next computation step.
(B.1) There is a fixed bound on the number of symbolic configurations that can be immediately recognized.
(B.2) There is a fixed bound on the number of states that need to be taken into account.[15]
(L.1) Only immediately recognizable symbolic configurations are changed.

(L.2) New observed configurations are within a bounded distance of an immediately previously observed configuration.[16]

The determinacy condition asserts that the computational process is deterministic. Each operation is uniquely determined by the current symbolic configuration (which includes the instruction and the input).[17] The boundedness conditions ensure that there is a bound on the symbolic configuration that can affect a (local) change, and the locality conditions guarantee that there is a bound on the number of local changes (it is in fact just one). The conditions themselves are justified, quite convincingly, in terms of the limits on human sensory and cognitive capacities.

The last step of the argument sketches a proof for the claim that every function whose values can be arrived at by a process constrained by D-B-L is Turing computable. The proof is straightforward. Conditions D-B-L ensure that each computation step involves a change in one bounded part of the relevant symbolic configuration, so that the number of types of elementary steps is bounded and simple. The proof is completed after demonstrating that each such step can be mimicked, perhaps by a series of steps, by a Turing machine.[18] We can summarize Turing's argument as follows:

<u>Turing's argument</u>:

1. **Turing's Thesis**: A human computer must satisfy the restrictive conditions D-B-L.
2. **Turing's Theorem**: Any function that can be computed by a computer that satisfies conditions D-B-L is Turing computable.
3. **Conclusion (HCTT)**: Any function that can be computed by a human computer is Turing computable.

Unlike Church, then, Turing does not assume that each step is recursive. Rather, Turing demonstrates that each step must be recursive. He analyzes the constraints imposed on each step of the computation, and then proves that each step can be carried out by a universal Turing machine.

Let me recapitulate: CTT should be understood against its historical background. In this context, effectively computable functions are those functions that can be calculated by a human computer, that is, by effective human computation, and, hence, CTT should be understood in terms of HCTT. Turing provided a highly convincing argument (Sieg), perhaps even a proof (Gandy), for HCTT. In this argument, Turing analyzes the processes of human computation, demonstrating that these processes satisfy certain restrictive constraints, and that functions computed via processes satisfying these constraints must be Turing computable. It is, therefore, highly likely that HCTT is true.

2. Effective computation by machines

According to the Gandy-Sieg account, effective computation had been understood and analyzed in 1936 in terms of human computation, and Turing argued for a human version of CTT by analyzing human computers. The Gandy-Sieg account is, I think, correct. As I am about to argue, however, after 1936 the prevailing view of effective computation began to change. In current computer science, effective computation in general and CTT in particular are understood in terms of *finite machine computation*, where 'machine' is taken in its broadest sense, that is, referring to humans, physical systems, or abstract automata.

When we look at the more recent textbooks in the theory of computation, automata, and formal languages, we observe that the notion of effective computation is applied to machines, and CTT is understood as a thesis about computers in general. Here are just a few examples: "Today the Turing machine has become the accepted formalization of an effective procedure. Clearly one cannot prove that the Turing machine model is equivalent to our intuitive notion of a computer, but there are compelling arguments for the equivalence, which has become known as Church's hypothesis." (Hopcroft and Ullman 1979: 147); "Because the Turing machines can carry out any computation that can be carried out by any similar type of automata, and because these automata seem to capture the essential features of real computing machines, we take the Turing machine to be a precise formal equivalent of the intuitive notion of 'algorithm'." (Lewis and Papadimitriou 1981: 223); "The claim, called *Church's thesis* or the *Church-Turing thesis*, is a basis for the equivalence of algorithmic procedures and computing machines." (Nagin and Impagliuzzo 1995: 611); "The **Church-Turing thesis** says that, from a theoretical standpoint, all computers have the same power. This is commonly accepted; the most powerful computers in the world compute the same things that Turing's abstract machine could compute." (Astrachan 2000: 397).

Though the texts are much less explicit about the question of when does a machine carry out an algorithm, or when does it compute effectively, they all implicitly assume that a machine computes effectively just in case its processes are symbolic discrete operations that make use of "finite means" (see the citations below). We can thus express the machine version of the Church-Turing thesis as follows:

> **The Machine version of the Church-Turing Thesis (MCTT)**: Any function (of positive integers) that can be computed by a finite machine is Turing computable.

As just stated, there have been, perhaps surprisingly, almost no attempts to analyze the restrictions on the processes of finite machine computation. But there are, I think, certain ingredients of finite machine computation that can be characterized. Following Gandy (1980), we can think of a finite machine as a pair <S,F>. S is a set of, potentially, infinitely many states, and F is a state transition operation. We will say that <S,F> computes by finite means if and only if it fulfills the following conditions:

(1) Each computational process can be described as a state transition process S_0, $F(S_0)$, $F(F(S_0))$, … . The process, if terminating, consists of finitely many transitions.
(2) Each state is assembled from finitely many tokens of atomic parts (or, alternatively, is a finite symbolic configuration). However, the number of atoms (symbols) from which a state is assembled may be unbounded.
(3) There are finitely many types of atomic parts (a finite alphabet) from which the states of S are assembled.
(4) Each type of state transition can be reassembled from a finite number of primitive operations. In addition, there is a bound on the number of types of primitive operations from which a state transition can be assembled.

I would like to emphasize that nothing in my argument in this section depends on the correctness of this definition. The definition will be used only in the argumentation in the next section. Moreover, the definition is not meant to be a precise characterization of a finite machine. The definition can be made precise only if we can characterize, more rigorously, the terms 'assembled' and 'primitive operation', but this is not an easy task. In fact, the widespread assumption nowadays is that a precise characterization of a finite machine cannot be given. We can never rule out the possibility that there is a finite machine that we never thought before, but that does not satisfy the restrictive conditions we formulate today:

> But since there is no end to the possible variations in detailed characterizations of the notions of computability and effectiveness, one must finally accept or reject the *thesis* (which does not admit of mathematical proof) that the set of functions computable in our sense is identical with the set of functions that men or machines would be able to compute by whatever effective method, if limitations on time, speed, and material were overcome. (Boolos and Jeffrey 1989: 20).

One example of a finite machine is Turing's human computer, where each state can be seen as a finite string of symbols, and each state transition is restricted by Turing's conditions D-B-L. A Turing machine, where each state transition can be described by a finite sequence of quadruples, is yet another example. One more example of a finite machine is a discrete neural net, whose size changes with the size of the problem. Take for example a net that solves the n-queens problem (Shagrir 1992). An initial state of the machine consists of n^2 nodes, each representing the presence/absence of a queen on a cell of the board, and a matrix of n^4 weights between pairs of nodes. The machine arrives at a solution through a finite number of state transitions. The transitions do not satisfy Turing's locality and boundedness conditions. Locality is violated because, at each state transition, changes can be made in parallel in arbitrarily many nodes. Boundedness is violated because each local change (in a node) results from the values of all the other nodes. Thus for every bound k there is a state that consists of n^2 nodes (n>k), in which a change in a node is affected by nodes that are not within the bound. Yet the machine is considered to be a finite machine. Each state transition is assembled (roughly by summations) from primitive operations of the type $a_i \cdot w_{ij}$, where a_i is the value (0 or 1) of $node_i$ at time t-1, and w_{ij} is the fixed value of the weight from $node_j$ to $node_i$.

This transition in the perspective from human computation to machine computation is not very surprising. Turing and Church conceived of effective computability in terms of human computation because the problems they aimed to solve were defined in these terms. In current computer science, however, the main focus is on problems that can or cannot be solved by machines. Yet, I do not mean to suggest that, prior to 1936, people thought about effective computation solely in terms of human computation. Hilbert's student Heinrich Behmann wrote in 1921 that "*mechanical calculations* according to given instructions" could be thought of as "*mechanical* or *machinelike* thought (Perhaps one could later let the procedure be carried out by machine)".[19] Also the identification of effective computation with finite computation is already found in the opening sentence of Turing's 1936 paper: "The 'computable' numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means" (p. 116). Church even reads Turing as providing the basis for the redefining of effective computation by finite machine computation. In his 1937 review of Turing (1936), Church writes: "The author proposes a criterion that an infinite sequence of digits 0 and 1 be 'computable' that it shall be possible to devise a computing machine, occupying a finite space with working parts of finite size" (1937a: 42). Church proceeds to claim, in his 1937 review of Post (1936), that this definition is appropriate: "To define effectiveness as computability by an arbitrary machine, subject to the restriction of finiteness, would seem to be an adequate representation of the ordinary notion" (1937b: 43).[20]

In fact, I do not even suggest that the extension of 'effective computability' has changed. Given that any Turing computable function can be computed by a human computer, and granted that MCTT is true, the extensions of 'effective machine computability' and 'human computability' is the same. My suggestion, rather, is that the intension of 'effective computability' has changed. In 1936, and prior to that date, logicians thought about effective computation in terms of human computation in the sense that effective computation is constrained by what human computers can and cannot do. Machines that satisfy these constraints compute effectively, and machines that do not satisfy these constraints do not compute effectively. In current computer science, however, effective computation ceases to be constrained by what human computers can and cannot do. Effective computation is constrained, rather, by what an arbitrary finite computer can do.

Other sources of evidence for the claim that there was a shift, after 1936, from human computation to machine computation, are the arguments for CTT. While it is widely recognized that Turing's is the best argument for CTT, when effective computation is understood in terms of human computation, this recognition has no echo in the textbooks in computer science. The two arguments for CTT that appear in most textbooks are the argument from confluence and the argument from non-refutation. The argument from non-refutation states that CTT has not been refuted, even though it is refutable: "Church's Thesis could be overthrown at some future date, if someone were to propose an alternative model of computation that was publicly acceptable as fulfilling the requirement of 'finite labor at each step' and yet was provably capable of carrying out computations that cannot be carried out by any Turing machine. No one considers this likely" (Lewis and Papadimitriou 1981: 223). "Although this thesis ('Church's thesis') is unprovable, it *is* refutable, *if false*… it is refutable by finding a counterexample; and the more experience of computation we have without finding a counterexample, the better confirmed the thesis becomes" (Boolos and Jeffrey 1989: 20). The argument from confluence states that many characterizations of computation, which differ in their goals, approaches, and details, encompass the same class of computable functions: "Turing machines can be imitated by grammars, which can be imitated by μ-functions, which can be imitated by Turing machines. The only possible conclusion is that all these approaches to the idea of computation are equivalent" (Lewis and Papadimitriou 1981: 224). "Indeed, given any other plausible, precise characterization of computability that has been offered, one can prove by careful, laborious reasoning that our notion is equivalent to it in the sense that any function which is computable according to one characterization is also computable according to the other" (Boolos and Jeffrey 1989: 20). Even Church's step-by-step argument is nowadays considered to be a more general argument for CTT than Turing's (Shepherdson 1988: 538). Thus, on the one hand, it appears that the dominant arguments for CTT nowadays are the arguments from confluence and non-refutation. While on the other hand, even though the arguments from confluence and non-refutation were well known in 1936, they were not at the front of Turing's and Church's argumentation.[21]

The transition in the centrality of arguments for CTT can be explained by the transition from human to machine computation. The arguments from confluence and non-refutation are very different from Turing's. They are not based on an analysis of the properties underlying effective computation, but rest on a query of the computational powers of given different mathematical systems and machines. As such, they are inherently inconclusive. It is always possible that all the precise characterizations examined so far address the same proper subclass of the effectively computable functions. In particular, all these characterizations rest on the very intuitive, but unanalyzed assumption – Sieg's stumbling block – that the most elementary steps of the computation are Turing computable. Given that Turing's analysis removes this obstacle, by analyzing the conditions underlying human computation, and proving that any computation restricted by these conditions falls under the limits of Turing computability, it is not surprising that Turing's argument is conceived as a far more convincing argument for HCTT.[22]

The situation is quite different when effective computation is understood in terms of machine computation. Turing's argument for MCTT is unsatisfactory, since it encompasses only a proper subclass of finite computers. From the perspective of machine computation, that is, Turing analyzed only one species of finite computers, i.e., human computers. This does not mean that Turing's analysis is irrelevant to other sorts of finite computation. Indeed, the analysis applies to any finite machine the computational steps of which involve a local change in one bounded part. In particular, Turing's theorem – the second premise in his argument – applies to any finite machine that satisfies conditions D-B-L. Nevertheless, Turing's analysis does not encompass all finite computers. Parallel distributed machines, like the neural net described above, are finite machine that do not satisfy conditions D-B-L, and thus do not fall within the scope of Turing's analysis. These machines might compute functions that are not Turing computable. To show that such machines do, nonetheless, fall within the scope of Turing computability, another argument has to be provided.

The current dominance of the arguments from confluence and non-refutation is explained by the fact that many think that the notion of a finite machine computation has not been characterized, and perhaps is even open-ended. As long as we do not have, and perhaps cannot have, a precise analysis of a finite machine, the best we can do is demonstrate that no function computed by any known machine refutes MCTT, and that all of the functions computed by such a universal machine are precisely the Turing computable functions.

In summary, I have suggested that 'effective computation' is currently understood, at least in computer science, in terms of finite machine computation. My contention is supported by the current formulations of CTT, which always refer to machine computation, and by the current argumentation for CTT, which is explained by the fact that CTT is nowadays understood in the form of MCTT.

I close this section by noting that the notion of computation by a finite computer is different from the notion of computation by a *physical* machine. Consequently, MCTT should be distinguished from physical versions of the Church-Turing thesis. The latter versions address the computational powers of *physical* systems, and assert that the computational powers of physical systems do not transcend Turing computability:

> **The Physical Church-Turing Thesis (PCTT)**: Any function (of positive integers) that can be computed by a *physical* system is Turing-computable.[23]

In this context, a physical system is any system, real or potential, such that (a) its states occupy a finite space, (b) its terminating dynamics are completed in finite real time, and (c) its dynamics are consistent with the laws of physics.

PCTT is primarily the concern of physicists interested in the computational bounds on physical systems. Though usually careful to distinguish PCTT from the Church-Turing Thesis (CTT), some of them mistakenly identify these theses.[24] The confusion is not surprising. On the one hand, computer scientists are interested in finite machines primarily because these are the machines that we can actually build. On the other hand, physicists are interested in physical computers that are also "finite" in the sense that they occupy a finite space and complete the computation in finite time.[25] Yet the terms 'physical computing machine' and 'finite computing machine' are not only intensionally different, but also extensionally different, referring to different classes of computing machines. There are machines that satisfy conditions

(1)-(4), yet are not physically implementable.[26] But there are also physical systems that satisfy conditions (a)-(c), but not conditions (1)-(4). For instance, there are analog systems that occupy a finite space, yet do not satisfy condition (3), since their states do not consist of finitely many types of atoms.[27] More interestingly, there are discrete physical systems that complete a computational process in a finite real time, yet do not satisfy condition (1), since these processes consist of infinitely many steps.[28]

3. Computation by Gandy machines

According to the Gandy-Sieg account, Turing analyzed human computation, not machine computation. Analyses of machine computation, advanced much later, are different: “It has been claimed frequently that Turing analyzed computations of machines. That is historically and systematically inaccurate, as my exposition should have made abundantly clear. Only in 1980 did Turing’s student Robin Gandy characterize machine computations” (Sieg 2001: 396). Sieg in fact builds on Gandy’s work as a basis for “a characterization of computations by machines that is general and convincing as that of computations by human computors given by Turing” (forthcoming: 245). Gandy, who was aware that “Turing’s arguments can be applied indifferently to men or machines”, also observed that “there are crucial steps in Turing’s analysis where he appeals to the fact that the calculation is being carried out by a human being” (1980: 124). In what follows, I briefly review Gandy’s 1980 analysis of machine computation, focusing on the conceptual framework, and omitting the many technical details.[29] I then consider the question of what kind of machines Gandy is really analyzing, and, in particular, whether the machines he analyzes are the finite machines discussed above.

Gandy explicitly seeks to provide an argument for the following thesis:

> "**Thesis M**. *What can be calculated by a machine is [Turing] computable*" (1980: 124).

Thesis M requires clarification. First, by 'machine' Gandy means a *physical* machine such as "the parts of Babbage's 'Analytical Engine', which are preserved in the Science Museum at South Kensington" (1980: 125). Next, Gandy does not consider all physical computing machines, but only discrete deterministic mechanical devices. Discrete computers "are, in a loose sense, digital computers" (p. 126). Gandy thus excludes analog computers from his analysis. A deterministic computer is one whose subsequent behavior "is uniquely determined once a complete description of its initial state is given." (p. 126). 'Mechanical device' refers to "the nineteenth century meaning" of machine (pp. 125-6). Here Gandy apparently refers to physical computing machines that we could devise, at least in principle. These include discrete deterministic machines whose calculations might not satisfy Turing's constraints on human computation, that is, calculations that do not proceed as sequences of elementary steps, but might, say, print an arbitrary number of symbols simultaneously. The more interesting machines that fall within Gandy's characterizations are parallel machines, namely, machines whose computation steps involve parallel changes in arbitrarily many overlapping parts. We will call these discrete deterministic mechanical devices *Gandy machines*. We can now formulate Gandy's thesis more accurately:

> **Gandy's thesis**: Any function that can be computed by a discrete deterministic mechanical device (a Gandy machine) is Turing computable.

Gandy's argument for the thesis deliberately reproduces the structure of Turing's argument. Gandy first argues that each physical discrete deterministic device satisfies certain constraints: principles I-IV. Gandy then proves the theorem that any function that can be computed by a device that satisfies principles I-IV is Turing computable. The argument can thus be expressed as follows:

Gandy's argument:

1. "**Thesis P.** *A discrete deterministic mechanical device satisfies principles I-IV below*." (1980: 126)
2. "**Theorem.** *What can be calculated by a device satisfying principles I-IV is [Turing] computable*." (1980: 126)
3. **Gandy's Thesis.** What can be calculated by a discrete deterministic mechanical device is [Turing] computable.

Principles I-IV approximate to these:

(I) Form of description: Any discrete deterministic mechanical device M can be described as <S,F>, where S is structural class (a subclass of the hereditarily finite sets HF over an infinite set U of atoms that is closed under isomorphic structures), and F is a structural operation from S_i to S_j. Thus, if S_0 is the initial state, then $F(S_0)$, $F(F(S_0))$,… are its subsequent states.
(II) Limitation of Hierarchy: the set theoretic rank of the states is bounded.
(III) Unique Reassembly: Any state S_i of S is assembled from parts of bounded size.
(IV) Local causation: Parts from which F(x) can be reassembled depend only on bounded parts of x.

Thesis P, then, asserts that the abstract principles I-IV are a precise mathematical formulation of the limitations on Gandy machines. Principles I-III are meant to capture the discrete and deterministic nature of Gandy machines. They provide one precise characterization of conditions (1)-(3) on finite machines, with one exception: they do not explicitly state that a terminating process must consist of finitely many steps. Principle IV (local causation) puts restrictions on the type of operations available. It says that each changed part of a state is affected by its local "neighborhood", which is bounded. The principle abstracts from two physical presuppositions: “that there is a lower bound on the linear dimensions of every atomic part of the device and that there is an upper bound (the velocity of light) on the speed of propagation of changes” (p. 126). The first is a physical constraint on discrete devices; the second arguably applies to all physical machines. The constraints imply that each changed part can be affected by a bounded number of atoms. Since the propagation of information is bounded, an atom can transmit and receive information in its bounded neighborhood (in bounded time). Since there is a lower bound on the size of the atoms, the number of atoms in this neighborhood is bounded. It thus follows that F(x) is assembled from bounded, though perhaps overlapping, parts of x.

Gandy’s Thesis P parallels Turing’s thesis in that it specifies constraints on a class of computers. While Turing’s thesis asserts that conditions D-B-L are restrictive conditions on human computers, Gandy’s Thesis P states that principles I-IV are restrictive conditions on Gandy machines. Gandy’s next step – the Theorem – also parallels the second step in Turing’s argument – Turing’s Theorem. While Turing’s theorem asserts that any function computed by an abstract device that satisfies the conditions D-B-L is Turing computable, Gandy’s Theorem states that any function computed by an abstract device that satisfies principles I-IV is Turing computable. But Gandy’s Theorem is far more comprehensive. From a formal point of view, computers that satisfy D-B-L are species of computers that satisfy I-IV (Gandy 1980: 145-146, Sieg 2001: 400-402). In computers that satisfy D-B-L, each step consists of a change in one bounded part of the symbolic configuration. In computers that satisfy I-IV, each step of the computation can consist in arbitrarily many bounded, and sometimes even overlapping, parts. Thus, like Turing, Gandy does not assume that each step of the computation is Turing computable. The Turing computability of each change derives from the finiteness constraints I-IV. These constraints arguably ensure that at each step a Gandy machine modifies arbitrarily many bounded parts, that each bounded part consists of finitely many atoms, and that there are, overall, finitely many types of local modifications of bounded parts. It thus follows that all possible changes, of which there are infinitely many, consist of finitely many types of bounded modifications, each of which is Turing computable.

Despite Gandy's careful analysis, I suggest that there is an ambiguity with regard to the types of machine he is postulating. I offer three interpretations, which differ in their scope and limitations, and finally conclude that none of them provide the basis for claiming that Gandy characterized finite machine computation.

Gandy machines as physical machines: On one interpretation, Gandy's thesis is a variant of PCTT, namely, a thesis about the computational bounds of a subclass of physical machines, namely, discrete deterministic physical devices. This interpretation is supported by the manner in which Gandy presents and argues for the thesis, and which is described above. Under this interpretation, however, Gandy's thesis seems to be fallacious. Elsewhere, Pitowsky and I provide an example of a discrete deterministic mechanical device, the dynamics of which are compatible with the principles of general relativity, that satisfies Gandy's physical presuppositions: its computational processes always terminate in finite 'local' time (though some of its terminating computations require an infinite number of computation steps).[30] Yet the device computes a function that is not Turing computable. Which premise is then false? Thesis P – which states that any discrete deterministic mechanical device satisfies principles I-IV – is consistent with General Relativity. In particular, it is consistent with our counterexample to Gandy's thesis. If anything, Gandy notes, local causation is inconsistent with Newtonian mechanics: "Principle IV does not apply to machines obeying Newtonian mechanics. In these there may be rigid rods of arbitrary lengths and messengers travelling with arbitrary large velocities, so that the distance they can travel in a single step is unbounded" (1980: 145). The second premise - (Gandy's Theorem) – which asserts that every function that can be computed by a device that satisfies principles I-IV is Turing computable – rests on the implicit assumption that the number of steps in a terminating computation is finite. Without this assumption, our device is a counterexample to the Theorem: it satisfies principles I-IV, but it computes functions that are not Turing computable.

Gandy machines as finite-physical machines: On a second interpretation, Gandy is speaking of finite-physical computing machines. According to this interpretation, Gandy meant by 'a discrete machine' finite machines, namely, machines that satisfy conditions (1)-(4). Gandy machines, then, are finite machines that are physically implementable, at least in principle, and Gandy's thesis asserts that any function computed by a finite and physical device is Turing computable. In that case, Gandy should have explicitly mentioned finite number of steps (in principle I), as this constraint is not motivated by "physical presuppositions". The two physical presuppositions rather justify the stricter constraints on the type of operations available, which guarantee that there is a bound on the number of types of operations from which any state, F(x), is assembled. This interpretation is perhaps closer to Gandy's intended meaning, given that his Theorem assumes that a finite process consists of finitely many steps, not just terminating in finite real time. This interpretation also makes it more likely that Gandy's thesis is true. The alleged counterexample to Gandy's thesis and theorem, Pitowsky and I suggest, is blocked, since one condition on finite machines is that any terminating process must consist of finitely many steps.

Gandy machines as a mathematical notion: Another way to view a Gandy machine is as a mathematical notion defined by principles I-IV. Gandy's work is viewed as an attempt to provide a mathematical definition of a machine that abstracts from the properties of a certain class of computations by finite machines. The mathematical principles I-III aim to provide a precise characterization of conditions (1)-(3), and the mathematical principle IV (local causation) aims to impose constraints on the types of operations available. This interpretation is advanced, for example, by Sieg, who does "no longer take a Gandy machine to be a dynamical system" (2001: 398), but to be "an 'abstract' mathematical definition that embodies generally accepted properties of parallel computations" (2001: 400).[31] I am not sure that this interpretation is faithful to Gandy's intended meaning. I also think that a condition about finitely many steps, which is implicitly assumed in the characterization of a Gandy machine, should be made more explicit. However, this interpretation draws attention to the true significance of Gandy's work, namely, providing a precise characterization of a large and important class of finite machines, that includes, among other machines, all the finite physical parallel machines that *we* can devise, either *de facto* or in principle.

The chief difference between this interpretation – which takes a Gandy machine to be a mathematical object characterized by principles I-IV – and the two former interpretations – which takes a Gandy machine to be a physical object – is the following. On the former interpretations, the aim of Gandy's work is to characterize a certain class of physical machines, and to prove that the functions computed by these machines are all Turing-computable. A central claim in the argument is that principles I-IV capture the basic computational properties of these physical machines (Thesis P). On the third interpretation, Thesis P is dropped as irrelevant. The goal of Gandy's work is to characterize a wide class of finite machines – physical or abstract – whose computations proceed in parallel. Principles I-IV are *motivated* by physical presuppositions, but are not *justified* by them. The main question of interest is not whether Thesis P is true, but what are the machines that satisfy principles I-IV. A physical machine that does not satisfy principles I-IV, does not refute the principles, but falls outside their scope.

We have looked at three interpretations of the notion of a Gandy machine. On one interpretation, the notion applies to physical machines including some that are not necessarily finite. On a second, the notion applies to a subclass of finite machines, namely, to finite physical machines. On the third, the notion applies to a mathematical object defined by principles I-IV. It is interesting to observe, however, that on all three interpretations, the class of Gandy machines is extensionally different from the class of finite machines. The neural net for the n-queens problem, described in section 2, is considered to be a finite machine, yet it is at odds with local causation. Each node from which a state F(x) can be reassembled depends on the values of the nodes of the state x. Thus if the states x and F(x) are large enough (each consisting of n^2 nodes), any node of F(x) will depend on a part of x that exceeds the bound. This may explain why Gandy's analysis has never become the prevailing model of machine computation, and Gandy's thesis as the machine version of CTT. Whereas the notion of effective computation has been understood in computer science in terms of finite machine computation, and CTT as a general thesis about the computational powers of finite machines, Gandy's analysis applies at best to a proper subclass of finite computers, namely, to those satisfying I-IV.[32]

Acknowledgements

I have been benefited from discussions with Eli Dresner, Paolo Mancosu, Gualtiero Piccinini, Itamar Pitowsky and Carl Posy, whose suggestions helped me to shape my arguments. I am grateful to Wilfried Sieg for detailed comments on an earlier draft of this paper. These comments contribute to the final version, which, I trust, is much improved. This research was supported by The Israel Academy of Sciences and Humanities.

Notes

[1] Though I take algorithms and effective procedures to be synonymous, algorithms are sometimes presented as abstract entities, and effective procedures as symbolic representations of algorithms.
[2] See Gandy (1988) for the historical background to the formulation, and Sieg (1994, sections 2.1 and 2.2) for Kleene's systematic development of the definition of recursive functions from Gödel's and Herbrand's 1931 notions of recursion.
[3] Church's formulation is:

> We now define the notion, already discussed, of an effectively calculable function of positive integers by identifying it with the notion of a recursive function of positive integers (or of a λ-definable function of positive integers). This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of a formal definition to correspond to an intuitive notion (1936a: 100).

Turing's is:

> No attempt has yet been made to show that the "computable" numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically (1936: 135).

Turing's definition of computability over real numbers is equivalent to the definition over functions of positive integers. Indeed, the reference to positive integers simplifies matters but is not essential. We can compute other functions whose arguments and values can be represented by a 'reasonable' symbol system. Note too that many also state the reverse inclusion. However, this side of the thesis is not addressed in this paper.

[4] See also Sieg (1994, 2001), who emphasizes Turing's analysis of a human computer. Gandy and Sieg refer to a human computer as a computor.

[5] See Sieg (1984: 72-77) for a discussion of the role of effective computation in mathematics in the years preceding 1936.

[6] A relation $R(x_1,\ldots x_k)$ of positive integers is representable in L if there is a formula $A(x_1,\ldots x_k)$ in the language of L, such that for any tuple of positive integers $<n_1,\ldots n_k>$, if $R(n_1,\ldots n_k)$ then $A(\mathbf{n_1},\ldots \mathbf{n_k})$ is provable in L, and if $\neg R(n_1,\ldots n_k)$ then $\neg A(\mathbf{n_1},\ldots \mathbf{n_k})$ is provable in L ($\mathbf{n_i}$ is the constant whose interpretation in the standard model is n_i).

[7] Kleene refers to this unlikely possibility: "In 1931, one could be uncertain whether Gödel's incompleteness theorem applies to systems quite remote in their details from *Principia Mathematica*" (1988: 43).

[8] The problem appears to have been formulated by Behmann as early as 1921 (Mancosu 1999: 320-321).

[9] See Gandy 1988: 63-64.

[10] A physical super-machine that "solves" the halting problem is presented in Shagrir and Pitowsky (forthcoming). A similar machine can also "solve" the Entscheindungsproblem.

[11] See Turing 1936, particularly sections 1 and 9. Post (1936: 291) also describes effective computation in terms of human computation.

[12] I will follow Church in using recursive functions here, instead of Turing computability.

[13] It is now known that the recursive functions can be represented even in a simple first-order theory (Boolos and Jeffrey: Chapter 14).

[14] Church also presents effectively computable functions in terms of provability in other places. In a response to an alleged counterexample to CTT, Church writes that such an example would require "an utterly new principle of logic, not only never before formulated, but also never before actually used in mathematical proof." (Sieg 1997: 168).

[15] In his argument, Turing refers to "states of mind" (1936: 136). But further on he points out that we can "avoid introducing the 'state of mind' by considering a more physical and definitive counterpart of it." (p. 139), namely, a finite set of written instructions. See also note 16.

[16] Sieg formulates these constraints by extrapolating on the ideas found in Turing, and the representation of Turing machine as Post's production systems. A mathematical definition of Turing's Computor (assuming the computer satisfies D-B-L) is given by Sieg (forthcoming: 247-249).

[17] It was later shown that the assumption of strong determinism is not essential (Sieg 1984: 95).

[18] A detailed proof is provided in Sieg (2001, forthcoming).

[19] Translated in Mancosu (1999: 321).

[20] Church's reading of Turing 1936 is criticizes by Sieg, who comments that "Church's apparent misunderstanding is rather common." On Sieg's view, "Turing analyzed … a mechanical computor, and *that* provides the basis for judging the correctness of the finiteness conditions." (1994: 94-95). But, in my view, Church did not think that Turing analyzed machine computation. Church observed, rather, that the formulation of the finiteness conditions makes the human computer less relevant. After the finiteness conditions are judged to be correct, it does not matter any more whether the computer that satisfies these conditions is a human or another machine. Church observed, in other words, that Turing's analysis provides the basis for the identification of effectiveness with a finite machine that satisfies Turing's finiteness conditions. Thus Church basically identifies effectiveness with Sieg's abstract notion of Turing Computor (Sieg 2001). Church's observation constitutes the first stage of the transition from human to machine computation, but it still falls short of identifying effectiveness with machine computation. Stretching the notion of effective computation to finite machines that do not satisfy Turing's restrictive conditions, occurred only later, with the appearance of machines that do not satisfy the conditions D-B-L, yet compute effectively.

[21] Church (1936a) mentions the argument from confluence in note 3 (p. 90). At the time, he was aware of the confluence of the definitions of the λ-definable functions and the recursive functions. Turing's second argument (1936: 138-139), where computability is identified with calculation in logic, is also

intended as an argument from confluence (Gandy 1980: 75-76). In section 10, Turing gives examples of effectively computable functions that are Turing computable. In the appendix to (1936), Turing proves the extensional equivalence of λ-definable and Turing computable sequences.

[22] This explanation partly rests on Gödel's reaction to CTT (see Sieg 1994, 1997, 2001). While Gödel's first reaction, apparently in 1934, to Church's suggestion to identify effectiveness with λ-definability, was that it is "thoroughly unsatisfactory", Godel was much more positive towards the identification in 1946. He opens his "Remarks Before the Princeton Bicentennial Conference of Problems in Mathematics" as follows: "Tarski has stressed in his lecture (and I think justly) the great importance of the concept of general recursiveness (or Turing's computability). It seems to me that this importance is largely due to the fact that with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion." (1946: 84). In the postscriptum to the lecture Gödel attributes this identification to Turing's work, writing that: "Turing's work gives an analysis of the concept of 'mechanical procedure' (alias 'algorithm' or 'computation procedure' or 'finite combinatorial procedure'). This concept is shown to be equivalent with that of a Turing machine." Gödel, however, found the other definitions of computability (recursiveness and λ-definability) "much less suitable for our purpose." Their importance is due to their equivalence with Turing-computability, but the equivalence itself is not mentioned as a reason for accepting CTT.

[23] Early versions of the physical thesis are formulated in Wolfarm 1985, Deutsch 1985, and Pitowsky 1990.

[24] "But is [CTT] true? Well, it will be if there is at least one physically Turing machine and if there are no physically possible non-Turing computers." (Hogarth, 1994: 134); "Extensive efforts have been made to prove the Church-Turing thesis, which suggests that all realizable dynamical and physical systems cannot be more powerful than classical models of computation." (Siegelmann, 1995: 545).

[25] There is still the question of why does computer science focus on finite computers, and not on physical computers. Cleland (1993, forthcoming), for example, criticizes computer science for being concerned with finite computers. She argues that the emphasis on finitude is mistaken, and the true nature of effective procedures emerges from a careful analysis of quotidian procedures, which specify a temporally organized arrangement of causal interventions in the course of nature.

[26] For example, the neural net for the n-queens problem (described above) does not satisfy Gandy's principle of local causation (see below).

[27] See for example Siegelmann (1995), though the physical reality here is questionable.

[28] See for example Pitowsky (1990), Hogarth (1994), and Shagrir and Pitowsky (forthcoming).

[29] Readers interested in these details are referred to Gandy (1980), Sieg and Byrnes (1999), and Sieg (2001, forthcoming).

[30] Shagrir and Pitowsky (forthcoming).

[31] This interpretation is also held, I believe, by other theoreticians and computer scientists. See, for example, Shepherdson (1988), where the presentation of Gandy machines depends solely on finiteness conditions and not on physical constraints.

[32] There are, however, a few computer scientists who interpret CTT as a thesis about finite-physical machines: "any algorithmic problem for which we can find an algorithm that can be programmed in some programming language, *any* language, running on some computer, *any* computer, even one that has not been built yet but *can* be built…is also solvable by a Turing machine. This statement is one version of the so-called **Church/Turing thesis**" (Harel 1992: 233). On that interpretation of CTT, Gandy's analysis does capture the essence of effective machine computation.

References

Astrachan O.L. 2000: *A Computer Science Tapestry: Exploring Programming and Computer Science with C++*. US: McGraw-Hill.

Boolos G.S. and Jeffrey R.C. 1989: *Computability and Logic*. Cambridge: Cambridge University Press.

Church A. 1936a: 'An Unsolvable Problem of Elementary Number Theory'. *American Journal of Mathematics* 58: 345-363. Reprinted in Davis 1965: 88-107.

Church A. 1936b: ‘A Note on the Entscheidungsproblem’. *Journal of Symbolic Logic* 1: 40-41. Reprinted in Davis 1965: 108-115.

Church A. 1937a: Review of Turing (1936). *Journal of Symbolic Logic* 2: 42-43.

Church A. 1937b: Review of Post (1936). *Journal of Symbolic Logic* 2: 43.

Cleland C. 1993: ‘Is the Church-Turing Thesis True?’ *Minds and Machines* 3: 283-312.

Cleland C. Forthcoming: ‘Effective Procedures and Causal Processes’. *Minds and Machines.*

Copeland B.J. 1996: The Church-Turing Thesis’. In Perry J. and Zalta E. (eds.): *The Stanford Encyclopedia of Philosophy* [http://plato.stanford.edu]

Davis M. (ed.) 1965: *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. New York: Raven.

Deutsch D. 1985: ‘Quantum Theory: The Church-Turing Principle and the Universal Quantum Computer’. *Proceedings of the Royal Society of London A* 400: 97-117.

Gandy R.O. 1980: ‘Church’s Thesis and Principles of Mechanisms’. In Barwise J., Keisler J.J. and Kunen K. (eds.): *The Kleene Symposium*. Amsterdam: North-Holland: 123-145.

Gandy R.O. 1988: ‘The Confluence of Ideas in 1936’. In Herken 1988: 55-111.

Gödel K. 1931: ‘On Formally Undecidable Propositions of Principia Mathematica and Related Systems I’. *Monatshefte für Mathematik und Physik* 38: 173-198. Translated and Reprinted in Davis 1965: 5-38.

Gödel K. 1946: ‘Remarks Before the Princeton Bicentennial Conference on Problems in Mathematics’. Reprinted in Davis 1965: 84-88.

Harel D. 1992: *Algorithmics: The Spirit of Computing* (second edition). Reading MA: Addison-Wesley.

Herken R. (ed.) 1988: *The Universal Turing Machine A Half-Century Survey*. Oxford: Oxford University Press.

Hilbert D. and Ackermann W. 1928: *Grundzuge der Theoretischen Logic*. Berlin: Springer-Verlag.

Hilbert D. and Bernays P.: 1939: *Grundlagen der Mathematik II*. Berlin: Springer-Verlag.

Hogarth M.L. 1994: ‘Non-Turing Computers and Non-Turing Computability’. *Proceedings of the Philosophy of Science Association (PSA)* 1: 126-138.

Hopcroft J.C and Ullman J.D. 1979: *Introduction to Automata Theory, Languages, and Computation*. Reading MA: Addison-Wesley.

Kleene S.C. 1936: ‘General Recursive Functions of Natural Numbers’. *Mathematische Annalen* 112: 727-742. Reprinted in Davis 1965: 236-253.

Kleene S.C. 1988: ‘Turing’s Analysis of Computability, and Major Applications of It’. In Herken 1988: 17-54.

Lewis H.R. and Papadimitriou C.H. 1981: *Elements of the Theory of Computation*. Eaglewood Cliffs, NJ: Prentice-Hall.

Mancosu P. 1999: ‘Between Russell and Hilbert: Behmann on the Foundations of Mathematics’. *Bulletin of Symbolic Logic* 5: 303-330.

Nagin P. and Impagliazzo J. 1995: *Computer Science: A Breadth-First Approach with Pascal*. New York: John Wiley & Sons.

Pitowsky I. 1990: ‘The Physical Church Thesis and Physical Computational Complexity’. *Iyuun* 39: 81-99.

Post E.L. 1936: ‘Finitary Combinatory Processes – Formulation I’. *Journal of Symbolic Logic* 1: 103-105. Reprinted in Davis 1965: 288-291.

Shagrir O. 1992: ‘A Neural Net with Self-Inhibiting Units for the N-Queens Problem’. *International Journal of Neural Systems* 3 : 249-252.

Shagrir O. 1997: ‘Two Dogmas of Computationalism’. *Minds and Machines* 7: 321-344.

Shagrir O. and Pitowsky I. Forthcoming: ‘Physical Hypercomputation and the Church–Turing Thesis’. *Minds and Machines.*

Shepherdson J.C 1988: ‘Mechanisms for Computing Over Arbitratry Structures’. In Herken 1988: 537-555.

Sieg W. 1994: ‘Mechanical Procedures and Mathematical Experience’. In George A. (ed.): *Mathematics and Mind*. Oxford: Oxford University Press: 71-117.

Sieg W. 1997: ‘Step by recursive step: Church’s analysis of effective calculability’. *Bulletin of Symbolic Logic* 2: 154-180.

Sieg W. 2001: ‘Calculation by Man and Machine: Conceptual Analysis’. In Sieg W., Sommer R. and Talcott C. (eds.), *Reflections on the Foundations of Mathematics (Essays in Honor of Solomon Feferman)*, Volume 15 of Lectures Notes in Logic. Association of Symbolic Logic: 387-406.

Sieg W. Forthcoming: ‘Calculation by Man and Machine: Mathematical Presentation’. Proceedings of the International Congress of Logic, Methodology and Philosophy of Science (Cracow 1999). Kluwer: 246-260.

Sieg W. and Byrnes J. 1999: ‘An Abstract Model for Parallel Computations: Gandy’s Thesis’. *The Monist* 82: 150-164.

Siegelmann H.T. 1995: ‘Computation Beyond Turing Limit’. *Science* 268: 545-548.

Turing A.M. 1936: ‘On Computable Numbers, with an Application to the Entscheidungsproblem’. *Proceedings of the London Mathematical Society* (2) 42: 230-265. A correction in 43 (1937): 544-546. Reprinted in Davis 1965: 115-154.

Wolfarm S. 1985: ‘Undecidability and Intractability in Theoretical Physics’. *Physical Review Letters* 54: 735-738.