PART I

OVERVIEWS

CHAPTER 2

ADVERTISEMENT FOR THE PHILOSOPHY OF THE COMPUTATIONAL SCIENCES

ORON SHAGRIR

The term "the computational sciences" is somewhat ambiguous. At times, it refers to the extensive use of computer models and simulations in scientific investigation. Indeed, computer simulations and models are frequently used nowadays to study planetary movements, meteorological systems, social behavior, and so on. In other cases, the term implies that the subject matter of the scientific investigation or the modeled system itself is a computing system. This use of the term is usually confined to computability theory, computer science, artificial intelligence, computational brain and cognitive sciences, and other fields where the studied systems—Turing machines, desktops, neural systems, and perhaps some other biological ("information processing") systems—are perceived as computing systems. The focus of this chapter is on the latter meaning of "the computational sciences," namely, on those fields that study computing systems.

But, even after clearing up this ambiguity, the task of reviewing the philosophy of the computational sciences is a difficult one. There is nothing close to a consensus about the canonical literature, the philosophical issues, or even the relevant sciences that study computing systems. My aim here is not so much to change the situation but rather to provide a subjective look at some background for those who want to enter the field. Computers have revolutionized our life in the past few decades, and computer science, cognitive science, and computational neuroscience have become respected and central fields of study. Yet, somewhat surprisingly, the philosophy of each computational science and of the computational sciences together has been left behind in these developments. I hope that this chapter will motivate philosophers to pay more attention to an exciting field of research. This chapter is thus an advertisement for the philosophy of the computational sciences.

The chapter comprises two parts. The first part is about the subject matter of the computational sciences: namely, computing systems. My aim is to show that there are

varieties of computation and that each computational science is often about a different kind of computation. The second part focuses on three more specific issues: the ontology of computation, the nature of computational theories and explanations, and the potential relevance of computational complexity theory to theories of verification and confirmation.

1 Varieties of Computation

What is the subject matter of the computational sciences? The initial answer is computing systems. But, as it turns out, different computational sciences focus on different kinds of computers, or so I will claim. My strategy of argumentation is to first survey the Gandy-Sieg view that distinguishes between human computation and machine computation. I then provide a critique of this view. Although I commend their distinction between human and machine computation, I point out that we should further distinguish between different kinds of machine computation.

1.1 The Church-Turing Thesis

Until not long ago, people used the term "computers" to designate human computers, where by "human computers" they meant humans who calculate by mechanically following a finite procedure. This notion of procedure, now known as an *effective procedure* or an *algorithm*, has been in use for solving mathematical problems at least since Euclid. The use of effective procedures as a method of proof is found in Descartes, and its association with formal proof is emphasized by Leibnitz. Effective procedures came to the fore of modern mathematics only at the end of the nineteenth century and the first decades of the twentieth century. This development had two related sources. One is the various foundational works in mathematics, starting with Frege and reaching their peak in Hilbert's finitistic program in the 1920s.[1] The other source was the increasing number of foundational decision problems that attracted the attention of mathematicians. The most pertinent problem was the *Entscheidungsproblem* ("decision problem"), which concerns the decidability of logical systems. Hilbert and Ackermann (1928) described it as the most fundamental problem of mathematical logic.

By that time, however, the notion of effective procedure—and hence of logical calculus, formal proof, decidability, and solvability—were not well-defined. This situation was changed when Church, Kleene, Post, and Turing published four pioneering papers on computability in 1936. These papers provide a precise mathematical characterization of the class of effectively computable functions: namely, the functions whose values can

[1] See Sieg (2009) for further historical discussion.

be computed by means of an effective procedure. The four characterizations are quite different. Alonzo Church (1936*a*) characterized the effectively computable functions (over the positives) in terms of lambda-definability, a work he started in the early 1930s and which was pursued by his students Stephen Kleene and Barkley Rosser. Kleene (1936) characterized the general recursive functions based on the expansion of primitive recursiveness by Herbrand (1931) and Gödel (1934).[2] Emil Post (1936) described "finite combinatory processes" carried out by a "problem solver or worker" (p. 103). The young Alan Turing (1936) offered a precise characterization of the computability of real numbers in terms of Turing machines. Church (1936*a*, 1936*b*) and Turing (1936) also proved, independently of each other, that the *Entscheidungsproblem* is undecidable for first-order predicate calculus.

It was immediately proved that all four precise characterizations are extensionally equivalent: they identify the same class of functions. The claim that this class of functions encompasses the class of effectively computable functions is now known as the *Church-Turing thesis* (CTT). Church (1936*a*) and Turing (1936), and to some degree Post (1936), formulated versions of this thesis. It is presently stated as follows:

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

The statement connects an "intuitive" or "pre-theoretic" notion, that of effective computability, and a precise notion (e.g., that of recursive function or Turing machine computability). Arguably, due to the pre-theoretic notion, such a "thesis" is not subject to mathematical proof.[4] This thesis, together with the precise notions of a Turing machine, a recursive function, lambda-definability, and the theorems surrounding them, constitutes the foundations of computability theory in mathematics and computer science.

Although CTT determines the extension of effective computability, there is still a debate about the intension of effective computation, which relates to the process that yields computability. Robin Gandy (1980) and Wilfried Sieg (1994, 2009) argue that effective computation is essentially related to an idealized human agent (called here 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).[5] This does not mean that only human computers can effectively compute; machines can do this, too. The notion is essentially related to a

[2] Church (1936*a*), too, refers to this characterization. Later, Kleene (1938) expands the definition to partial functions. For more recent historical discussion, see Adams (2011).

[3] In some formulations, the thesis is symmetrical, also stating the "easy part," which says that every Turing-machine computable function is effectively computable.

[4] The claim that CTT is not provable has been challenged, e.g., by Mendelson (1990); but see also the discussion in Folina (1998) and Shapiro (1993).

[5] See also Sieg (1994, 2009) and Copeland (2002*a*), who writes that an effective procedure "demands no insight or ingenuity on the part of the human being carrying it out."

human computer in that a function is effectively computable *only if* a human computer can, in principle, calculate its values.

This anthropomorphic understanding of effective computation is supported by two kinds of evidence. One pertains to the epistemic role of effective computability in the foundational work that was at the background of the 1936 developments. In this context, it is hard to make sense of a formal proof, decision problem, and so on unless we understand "effectively computable" in terms of human computation. The second source of evidence is the way that some of the founders refer to effective computation. The most explicit statements appear in Post who writes that the purpose of his analysis "is not only to present a system of a certain logical potency but also, in its restricted field, of psychological fidelity" (1936, 105) and that "to mask this identification [CTT] under a definition hides the fact that a fundamental discovery in the limitations of the mathematicizing power of Homo Sapiens has been made" (105, note 8). But the one who explicitly analyzes human computation is Turing (1936), whose work has recently received more careful attention.

1.2 Turing's Analysis of Human Computation

Turing's 1936 paper is a classic for several reasons. First, Turing introduces the notion of an *automatic machine* (now known as a Turing machine). Its main novelty is in the distinction between the "program" part of the machine, which consists of a finite list of states or instructions, and a memory tape that is potentially infinite. This notion is at the heart of computability theory and automata theory. Second, Turing introduces the notion of a universal Turing machine, which is a (finite) Turing machine that can simulate the operations of any particular Turing machine (of which there are infinitely many!); it can thus compute any function that can be computed by any Turing machine. This notion has inspired the development of the general purpose digital electronic computers that now dominate almost every activity in our daily life. Third, Turing provides a highly interesting argument for CTT, one whose core is an analysis of the notion of human computation: the argument is now known as "Turing's analysis."[6]

Turing recognizes that encompassing the effectively computable functions involves the computing processes: "The real question at issue is 'What are the possible processes which can be carried out in computing a number?'" (1936, 249). He proceeds to formulate constraints on all such possible computational processes, constraints that are motivated by the limitation of a human's sensory apparatus and memory. Turing enumerates, somewhat informally, several constraints that can be summarized by the following restrictive conditions:

[6] Turing's analysis is explicated by Kleene (1952). It has been fully appreciated more recently by Gandy (1988), Sieg (1994, 2009), and Copeland (2004).

1. "The behavior of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment" (250).
2. "There is a bound *B* to the number of symbols or squares which the computer can observe at one moment" (250).
3. "The number of states of mind which need be taken into account is finite" (250).
4. "We may suppose that in a simple operation not more than one symbol is altered" (250).
5. "Each of the new observed squares is within *L* squares of an immediately previously observed square" (250).

It is often said that a Turing machine is a model of a computer, arguably, a human computer. This statement is imprecise. A Turing machine is a *letter machine*. At each point it "observes" only one square on the tape ~~(Figure 2.1)~~. "The [human] computer" might observe more than that (Figure ~~2.2~~). The other properties of a Turing machine are more restrictive, too. The next step of Turing's argument is to demonstrate that the Turing machine is no less powerful than a human computer. More precisely, Turing sketches a proof for the claim that every function whose values can be arrived at by a process constrained by conditions 1–5 is Turing-machine computable. The proof is straightforward. Conditions 1–5 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.

We can summarize Turing's argument as follows:

1. *Premise 1:* A human computer operates under the restrictive conditions 1–5.
2. *Premise 2 (Turing's theorem):* Any function that can be computed by a computer that is restricted by conditions 1–5 is Turing-machine computable.
3. *Conclusion (CTT):* Any function that can be computed by a human computer is Turing-machine computable.

Although Turing's argument is quite formidable, there are still interesting philosophical issues to be addressed: does the analysis essentially refer to humans? And what exactly is a human computer?[7] Other questions concerning computation by Turing machines include: does a Turing machine, as an abstract object, really compute (and, more generally, do abstract machines compute?), or is it a model of concrete objects that compute? We return to some of these questions later. The next step is to discuss machine computation.

[7] See Copeland and Shagrir (2013), who distinguish between a cognitive and noncognitive understanding of a human computer.

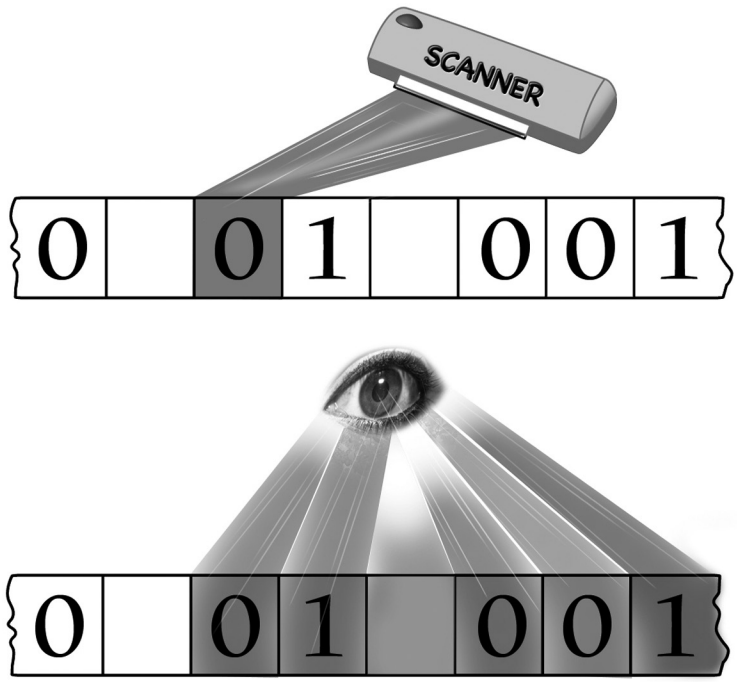

FIGURE 2.1 A human computer vs. a Turing machine. A Turing machine "observes" at each moment only one square on the tape (~~1.1~~). The number of squares observed by a human computer at each moment is also bounded, but it is not limited to a single square (~~1.2~~).

1.3 Gandy's Analysis of Machine Computation

It has been claimed that Turing characterized effective computability by analyzing human computation. The question is whether and to what extent this analysis captures the notion of machine computation in general. Gandy 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). For example, the second condition in the analysis asserts that there is a fixed bound on the number of cells "observed" at each step. In some parallel machines, however, there is no such limit. In the Game of Life, for instance, such a locality condition is satisfied by each cell: a cell "observes," as it were, only its local bounded environment; namely, itself and its eight neighboring cells. However, the condition is not satisfied by the machine as a whole because all the cells are being observed simultaneously, and there are potentially infinitely many cells.

Gandy moves forward to expand Turing's analysis and to characterize machine computation more generally. He provides an argument for the following claim:

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

Gandy confines the thesis to *deterministic discrete mechanical devices*, which are, "in a loose sense, digital computers" (1980, 126). Thus, Gandy actually argues for the following:

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

Gandy characterizes discrete deterministic mechanical devices in terms of precise axioms, called Principles I–IV. Principle I ("form of description") describes a deterministic discrete mechanical device as a pair <*S*,*F*>, where *S* is a potentially infinite set of states and *F* is a state-transition operation from S_i to S_{i+1}. Putting aside the technicalities of Gandy's presentation, the first principle can be approximated as:

> I. *Form of description:* Any discrete deterministic mechanical device *M* can be described by <*S*,*F*>, where *S* is a structural class, and *F* is a transformation from S_i to S_j. Thus, if S_o is *M*'s initial state, then $F(S_o)$, $F(F(S_o))$, . . . are its subsequent states.

Principles II and III place boundedness restrictions on *S*. They can be informally expressed as:

> II. *Limitation of hierarchy:* Each state S_i of *S* can be assembled from parts, that can be assemblages of other parts, and so on, but there is a finite bound on the complexity of this structure.
>
> III. *Unique reassembly:* Each state S_i of *S* is assembled from basic parts (of bounded size) drawn from a reservoir containing a bounded number of types of basic parts.

Principle IV, called "local causation," puts restrictions on the types of transition operations available. It says that each changed part of a state is affected by a bounded local "neighborhood":

> IV. *Local causation:* Parts from which $F(x)$ can be reassembled depend only on bounded parts of x.

The first three principles are motivated by what is called a discrete deterministic device. Principle IV is an abstraction of 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" (1980, 126). If the propagation of information is bounded, an atom can transmit and receive information in its bounded neighborhood in bounded time. If there is a lower bound on the size of atoms, the number of atoms in this neighborhood is bounded. Taking these together, each changed state, $F(x)$, is assembled from bounded, though perhaps overlapping, parts of *x*. In the Game of Life, for example, each cell impacts the state of several

(i.e., its neighboring cells). Those systems satisfying Principles I–IV are known as *Gandy machines*.

The second step in Gandy's argument is to prove a theorem asserting that any function computable by a Gandy machine is Turing-machine computable. The proof is far more complex than the proof of Turing's theorem because it refers to (Gandy) machines that work in parallel on the same regions (e.g., the same memory tape).[8]

Gandy's argument can be summarized as follows:

Premise 1 (*Thesis P*): -"A discrete deterministic mechanical device satisfies principles I–IV" (1980, 126).

Premise 2 (*Theorem*): "What can be calculated by a device satisfying principles I–IV is [Turing-machine] computable" (1980, 126).

Conclusion (Gandy's thesis): What can be calculated by a discrete deterministic mechanical device is Turing-machine computable.

Gandy's argument has the same structure as Turing's. The first premise ("Thesis P") posits axioms of computability for discrete deterministic mechanical devices (Turing formulated axioms for human computers). The second premise is a reduction theorem that shows that the computational power of Gandy machines does not exceed that of Turing machines (Turing advanced a reduction theorem with respect to machines limited by conditions 1–5). The conclusion ("Gandy's thesis") is that the computational power of discrete deterministic mechanical devices is bounded by Turing-machine computability (CTT is the claim about the scope of human computability).

The main difference between Gandy and Turing pertains to the restrictive conditions. Gandy's restrictions are weaker. They allow state transitions that result from changes in arbitrarily many bounded parts (in contrast, Turing allows changes in only one bounded part). In this way, Gandy's characterization encompasses *parallel* computation: "if we abstract from practical limitations, we can conceive of a machine which prints an arbitrary number of symbols simultaneously" (Gandy 1980, 124–125). Gandy does set a restriction on the stage transition of each part. Principle IV, of local causation, states that the transition is bounded by the local environment of this part. The Principle is rooted in presuppositions about the *physical world*. One presupposition, about the lower bound on the size of atomic parts, in fact derives from the assumption that the system is discrete. The presupposition about the speed of propagation is a basic principle of relativity. Indeed, Gandy remarked that his thesis P is inconsistent with Newtonian devices: "Principle IV does not apply to machines obeying Newtonian mechanics" (1980, 145).

Gandy's restrictions are weaker than those imposed on human computers. But are they general enough? Does Gandy capture machine computation as stated in his Thesis M? Sieg suggests that he does; he contends that Gandy provides "a characterization of computations by machines that is as general and convincing as that of computations

[8] But see Sieg (2009) for a more simplified proof of the theorem.

by human computors given by Turing" (2002, 247). But this claim has been challenged (Copeland and Shagrir 2007): Gandy's principles do not fully capture the notions of algorithmic machine computation and that of physical computation.

1.4 Algorithmic Machine Computation

In computer science, computing is often associated with *algorithms* (effective procedures). Algorithms in computer science, however, are primarily associated with *machines*, not with humans (more precisely, human computers are considered to be one kind of machines). Even CTT is formulated, in most computer science textbooks, in the broader terms of *algorithmic/effective machine computation*.[9] We can thus introduce a modern version of the Church-Turing thesis for an extended, machine version of the thesis:

> *The modern Church-Turing thesis:* Any function that can be *effectively* (algorithmically) computed by a *machine* is Turing-machine computable.

The modern thesis is assumed to be true. The main arguments for the thesis, however, do not appeal to human calculation. The two arguments for the thesis that appear in most textbooks are the argument from confluence and the argument from non-refutation. The argument from non-refutation states that the thesis has never been refuted. The argument from confluence says that many characterizations of computability, although differing in their goals, approaches, and details, encompass the same class of computable functions.[10]

The nature of algorithms is a matter of debate within computer science.[11] But the notion seems broader in scope than that of a Gandy machine. There are asynchronous algorithms that do not satisfy Principle I.[12] We can think of an asynchronous version of the Game of Life in which a fixed number of cells are simultaneously updated, but the identity of the updated cells is chosen randomly. Principle I is also violated by interactive or online (algorithmic) machines that constantly interact with the environment while computing. There might also be parallel synchronous algorithms that violate Principle IV. Gandy himself mentions Markov normal algorithms: "The process of deciding

[9] See, e.g., Hopcroft and Ullman (1979, 147), Lewis and Papadimitriou (1981, 223), and Nagin and Impagliazzo, who write: "The claim, called *Church's thesis* or the *Church-Turing thesis*, is a basis for the equivalence of algorithmic procedures and computing machines" (1995, 611).

[10] See Boolos and Jeffrey (1989, 20) and Lewis and Papadimitriou (1981, 223–224). But see also Dershowitz and Gurevich (2008) who use the Turing-Gandy axiomatic approach to argue for the truth of the thesis.

[11] The nature of algorithms is addressed by, e.g., Milner (1971), Knuth (1973, 1–9), Rogers (1987, 1–5), Odifreddi (1989, 3), Moschovakis (2001), Yanofsky (2011), Gurevich (2012), Vardi (2012), and Dean (forthcoming).

[12] See Copeland and Shagrir (2007) and Gurevich (2012) for examples.

whether a particular substitution is applicable to a given word is essentially global" (1980, 145).[13] "Global" here means that there is no bound on the length of a substituted word; hence, "locality" is not satisfied. Another example is a fully recurrent network in which the behavior of a cell is influenced by the information it gets from all other cells, yet the size of the network grows with the size of the problem.

I do not claim that computer science is concerned only with algorithmic machine computation (I return to this point later). The point is that the notion of algorithmic machine computation is broader than that of Gandy machines. This is perhaps not too surprising. As said, Gandy was concerned with deterministic (synchronous) discrete mechanical devices, and he relates *mechanical* to the "two physical suppositions" that motivate Principle IV. Let us turn then to yet another notion, that of physical computation.

1.5 Physical Computation

Physical computation is about physical systems or machines that compute. The class of physical systems includes not only actual machines but also physically possible machines (in the loose sense that their dynamics should conform to physical laws). Gandy machines, however, do not encompass all physical computation in that sense. He himself admits that his characterization excludes "devices which are *essentially* analogue" (Gandy 1980, 125). What is even more interesting is that his *Thesis M* (that limits physical computability to Turing-machine computability) might not hold for physical computing systems. There are arguably physical *hypercomputers*, namely, physical machines that compute functions that are not Turing machine computable.[14]

The claim that physical systems do not exceed the Turing limit is known as the *physical* Church-Turing thesis.[15] Gualtiero Piccinini (2011) distinguishes between bold and modest versions of the thesis. The bold thesis is about physical systems and processes in general, not necessarily the computing ones:

> *The bold physical Church-Turing thesis:* "Any physical process is Turing computable" (746).

The modest thesis concerns physical *computing* processes or systems:

> *The modest physical Church-Turing thesis:* "Any function that is physically computable is Turing computable" (746).

[13] An even earlier example is the substitution operation in the equational calculus that is generally recursive yet "essentially global" (Gödel 1934).

[14] For reviews of hypercomputation, see Copeland (2002*b*) and Syropoulos (2008).

[15] For early formulations, see Deutsch (1985), Wolfram (1985), and Earman (1986).

It turns out that most physical processes are computable in the bold sense that the input-output function that describes their behavior is real-valued Turing computable.[16] A well-known exception is presented by Pour-El and Richards (1981). Our focus is the modest thesis. A few counterexamples of physical hypercomputers have been suggested. These are highly idealized physical systems that compute functions that are not Turing machine computable. A celebrated class of counterexamples is that of *supertask machines*. These machines can complete infinitely many operations in a finite span of time. I will discuss here *relativistic machines* that are compatible with the two "physical presuppositions" that underlie Gandy's principle of local causation (the two presuppositions are an upper bound on the speed of signal propagation and a lower bound on the size of atomic parts).[17]

The construction of relativistic machines rests on the observation that there are relativistic space-times with the following property. The space-time includes a future endless curve λ with a past endpoint p, and it also includes a point q, such that the entire stretch of λ is included in the chronological past of q.[18] We can thus have a relativistic machine *RM* that consists of two communicating standard machines, T_A and T_B. The two standard machines start traveling from p. T_B travels along the endless curve λ. T_A moves along a future-directed curve that connects the beginning point p of λ with q. The time it takes T_A to travel from p to q is finite, while during that period T_B completes the infinite time trip along λ (Figure 2.2).

This physical setup permits the computation of the *halting function*, which is not Turing-machine computable. The computation proceeds as follows (Figure 2.3). T_A receives the input (m,n) and prints "0" in its designated output cell. It then sends a signal with the pertinent input to T_B. T_B is a universal machine that mimics the computation of the m^{th} Turing machine operating on input n. In other words, T_B calculates the Turing-machine computable function $f(m,n)$ that returns the output of the m^{th} Turing machine (operating on input n), if this m^{th} Turing machine halts. T_B returns no value if the simulated m^{th} Turing machine does not halt. If T_B halts, it immediately sends a signal back to T_A; if T_B never halts, it never sends a signal. Meanwhile T_A "waits" during the time it takes T_A to travel from p to q (say, one hour). If T_A has received a signal from T_B it prints "1," replacing the "0" in the designated output cell. After an hour (of T_A's time), the output cell shows the value of the halting function. It is "1" if the m^{th} machine halts on input n, and it is "0" otherwise.

How is it that *RM* can compute the halting function? The quick answer is that *RM* performs a supertask: T_B might go through infinitely many computation steps in finite time (from T_A's perspective). But this is not the whole story. A crucial additional feature is

[16] There are natural extensions of the notion of Turing computability to real-number functions. Definitions are provided by Grzegorczyk (1957); Pour-El and Richards (1989); Blum, Cucker, Shub, and Smale (1997); and others.

[17] Other examples of supertask machines are the accelerating machines (Copeland 2002*c*; Earman 1986) and the shrinking machines (Davies 2001).

[18] The first constructions of this setup were proposed by Pitowsky (1990) and Hogarth (1994).

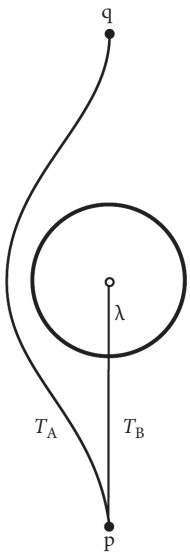

FIGURE 2.2 A setup for relativistic computation (adapted from Hogarth 1994, p. 127; with permission from the University of Chicago Press). In this relativistic Malament-Hogarth space-time the entire stretch of λ is included in the chronological past of *q*. The machines T_A and T_B start traveling together from a point p. T_A moves along a future-directed curve toward a point *q*. T_B travels along the endless curve λ. The time it takes for T_A to travel from *p* to *q* is finite, while during that period T_B completes an infinite time trip along λ.

that the communication between T_A and T_B is not deterministic in the sense familiar in algorithmic computation. In algorithmic machines, the configuration of each α + 1 stage is determined by that of the previous α stage. This condition is explicit in Turing's and in Gandy's analyses. But *RM* violates this condition. Its end stage cannot be described as a stage α + 1, whose configuration is completely determined by the preceding stage α. This is simply because there is no such preceding stage α, at least when the simulated machine never halts.[19] *RM* is certainly deterministic in *another* sense: it obeys laws that invoke no random or stochastic elements. In particular, its end state is a deterministic *limit* of previous states of T_B (and T_A).

Does *RM* violate the modest physical Church-Turing thesis? The answer largely depends on whether the machine *computes* and on whether its entire operations are *physically possible*. There is, by and large, a consensus that the relativistic machine is a genuine hypercomputer. The physical possibility of the machine is far more controversial.[20] But even if *RM* is not physically possible, we can use it to make two further points.

[19] See Copeland and Shagrir (2007) for an analysis of this point.

[20] See Earman and Norton (1993) for a skeptical note, but see also the papers by István Németi and colleagues (e.g., Andréka et al. (2009)) for a more optimistic outlook. They suggest that physical realizability can be best dealt with in setups that include huge slow rotating black holes.

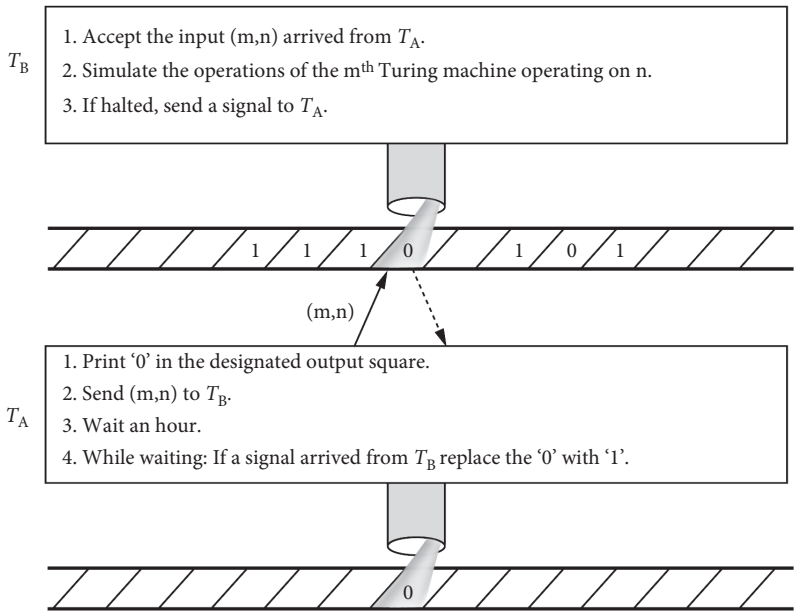

FIGURE 2.3 Computing the halting function. The relativistic machine *RM* consisting of two communicating standard (Turing) machines T_A and T_B computes the halting function. T_B is a universal machine that mimics the computation of the m^{th} Turing machine operating on input n. If it halts, T_B sends a signal back to T_A; if it never halts, T_B never sends a signal. Waiting for a signal from T_B, T_A prints "1" ("halts") if a signal is received; otherwise, it leaves the printed "0" ("never halts").

One is that there are "notional," logically possible machines that compute beyond the Turing limit; RM is one such machine, but there are many others. The second point is that the *concept* of physical computation accommodates relativistic computation (and, indeed, hypercomputation): after all, we agree that if *RM* is physically possible, then it does violate the modest Church-Turing thesis.

1.6 On the Relations Between Notions of Computation

Our survey indicates that there are varieties of computation: human, machine, effective, physical, notional, and other kinds of computation. Let us attempt to summarize the relationships among them. Our comparison is between the abstract principles that define each kind of computation ("models of computation"). The inclusion relations are depicted in Figure 2.4. We start with the Turing machine model that applies to (some) letter machines. The human computer model is broader in that it includes all machines that satisfy Turing's conditions 1–5. Gandy machines include even more machines, notably parallel machines that operate on bounded environments. Gandy machines,

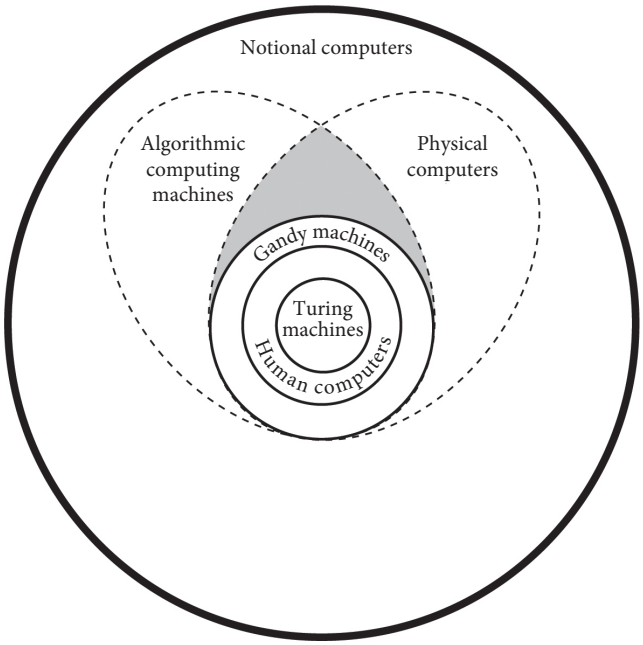

FIGURE 2.4 Relations between classes of computing systems. The relations are measured with respect to models of computation. "Turing machines" are the class of computers that are modeled by some Turing machine. "Human computers" are the class of computers that are limited by Turing's restrictive conditions 1–5. "Gandy machines" are those computers restricted by Principles I–IV. "Algorithmic computing machines" are those machines that compute by means of an algorithm (as characterized in computer science). "Physical computers" refers to computing systems that are physically realizable. "Notional computers" refers to computing systems that are logically possible.

however, do not exhaust two classes of machine computation. Algorithmic computation includes machines that are considered algorithmic in computer science yet are not Gandy machines. Physical computation includes physical machines (e.g., analogue computers) that are not Gandy machines. It seems that the classes of algorithmic computing machines and physical computing machines partially overlap. Algorithmic computation includes logically possible machines that are not physically realizable (possible). Physical computation might include hypercomputers that exceed the computational power of Turing machines (e.g., relativistic machines). Gandy machines are apparently at the intersection of algorithmic and physical computation. Whether they exhaust the intersection is an open question. Last, notional computation includes logically possible machines that are neither algorithmic nor physical; here, we may find some instances of the so-called infinite-time Turing machines.[21]

[21] See Hamkins and Lewis (2000).

1.7 What Is the Subject Matter?

What are the computational sciences about? The somewhat complex picture portrayed here indicates that there is no single subject matter covered by the computational sciences. Each "science" might be about a different kind of computation. Early computability theory and proof theory naturally focus on human computation (what can be done by human calculators). Automata theory and many other branches in theoretical computer science have focused on algorithmic machine computation. The Turing-machine model has been so central to these studies because it arguably captures the computational power of all algorithmic machines. But there are certainly some changes in current trends in computer science (and other formal sciences). One trend, which narrows the scope of computation, is to focus on models of physically possible machines.[22] Another trend, which widens the scope, is to go beyond algorithmic computation, and, in this context, I mentioned the interesting recent work on infinite-time Turing machines.

Computational neuroscience and cognitive science are about physical computation in the brain. The working assumption is that neural processes are computing physical processes, and the empirical question is about the functions computed and how they are being computed. The important lesson for philosophers here is that claims about the brain being a Turing machine or about the brain computing Turing-machine computable functions are *not* entailed from the notion of physical computation. Our survey shows that the brain and other physical systems can be computers even if they are not Turing machines and even if the functions they compute are not Turing machine computable at all. Whether or not brains are Turing machines and whether or not they compute Turing computable functions are apparently empirical questions.

2 Ontology, Epistemology, and Complexity

There are many philosophical issues involved in the computational sciences, and I cannot possibly cover all of them. I will briefly touch on three issues: the ontology of computation, the nature of computational theories and explanations, and the relevance of complexity theory to philosophy and specifically to verification and confirmation.

2.1 The Ontology of Computation

There are intriguing questions concerning the ontology of computation: what exactly are algorithms, and how are they related to programs and automata? Are algorithms,

[22] An example of this trend is given later in the section on computational complexity.

programs, and automata abstract entities? And what is meant here by abstract? Does abstract mean mathematical? Is abstract opposed to concrete? And how is abstract related to abstraction and idealization?[23] Another set of (related) questions concerns physical computation: what is the difference between a system that computes and one that does not compute? Is the set of physical computing systems a natural kind? Is there only one notion of physical computation, or are there several? And what are the realization or implementation relations between the abstract and the physical?[24] Yet another set of questions concern semantics and information in computing systems.[25]

I focus here on concrete physical computation. The central question is about the nature of physical computation. Looking at the literature, we can identify at least four approaches to physical computation: formal, mechanistic, information-processing, and modeling approaches. The four are not mutually exclusive. An account of physical computation typically includes at least two of the criteria imposed by these approaches. A formal approach proceeds in two steps. It identifies computation with one or another theoretical notion from logic or computer science—algorithm, program, procedure, rule, proof, automaton, and so on. It then associates this theoretical notion with a specific organizational, functional, or structural property of physical systems. An example is Cummins's account of computation. Cummins (1988) associates computation with program execution: "To compute a function *g* is to execute a program that gives *o* as its output on input *i* just in case $g(i) = o$. Computing reduces to program execution" (91). He then associates program execution with the functional property of *steps*: "Program execution reduces to step-satisfaction" (92).[26] I contented that both stages in the formal approach are flawed. The moral of the first part of this chapter is that physical computation is not delimited by the theoretical notions found in logic and theoretical computer science. I also suspect that the difference between computing and noncomputing physical systems is not to be found in some specific functional or structural property of physical systems (Shagrir 2006).

The mechanistic approach identifies computation with *mechanism*, as it is currently understood in the philosophy of science. In this view, computation is a specific kind of mechanistic process; it has to do with the processing of variables to obtain certain relationships among inputs, internal states, and outputs (Piccinini 2007, 2015*b*; Miłkowski 2013). The information-processing approach, as the name suggests, identifies computation with information processing (Churchland and Sejnowski 1992; Fresco 2014). The difficulty here is to spell out the notion of information in this context. Some claim that there is "no computation without representation" (Fodor 1975, 34; Pylyshyn 1984, 62).[27]

[23] See Turner's (2013) entry in the *Stanford Encyclopedia of Philosophy* for a review of some of these issues.

[24] This is ~~more the~~ focus of Piccinini's (2015*a*) entry in the *Stanford Encyclopedia of Philosophy*.

[25] See Floridi (1999) and Fresco (2014, chap. 6) for pertinent discussion.

[26] Fodor (2000) identifies computation with *syntactic properties*, associated in turn with second-order physical properties. Chalmers (2011) identifies computation with the *implementation of automata*, associated in turn with a certain causal-organizational profile. Most accounts of computation fall within this category.

[27] See Sprevak (2010) for an extensive defense of this view.

Others advocate for a nonsemantic notion of information (Miłkowski 2013). The modeling view, which I favor, associates computation with some sort of morphism. According to this view, computation preserves certain relations in the target domain. Take, for example, the oculomotor integrator, which computes the mathematical function of integration. This system preserves the (integration) relation between velocity and positions of the (target) eyes. The distance between two successive eye positions is just the mathematical integration over the eye velocity with respect to time (Shagrir 2010).

A related question is whether computation is a natural kind. This question is important in determining whether the computational sciences are part of the natural sciences or not. This question was raised in the context of the cognitive sciences by Putnam (1988) and Searle (1992). Searle and Putnam argue (roughly) that every physical object can execute any computer program (Searle) or implement any finite state automaton (Putnam). Searle infers from this that computation is not objective, in the sense that it is observer-relative rather than an intrinsic feature of the physical world. He further concludes that "There is no way that computational cognitive science could ever be a natural science, because computation is not an intrinsic feature of the world. It is assigned relative to observers" (212). The argument, if sound, applies to other computational sciences as well.[28]

I would like to make three comments about the argument. First, the argument assumes a formal approach to computation because it identifies computation with programs or automata, although it has some relevance to other approaches too. Second, it is, by and large, agreed that both Putnam and Searle assume a much too liberal notion of program execution and/or implementation of automata. Adding constraints related to counterfactual scenarios, groupings of states and others dramatically narrows the scope of the result.[29] And yet, even when these constraints are in place, almost every physical system executes at least one program and/or implements at least one automaton. What is ruled out is that the system executes/implements all programs/automata. Third, the universal realizability claim (at least under its weaker version, that every physical object executes/implements at least one program/automaton) does not entail that computation is not objective. The fact that every physical object has a mass does not entail that mass is not intrinsic to physics. Similarly, it might well be that executing a program and/or implementing an automaton is objective.

One could still insist that universal realizability does undermine the computational sciences. The assumption underlying these sciences is that we describe some physical systems as computing and others as not. But if every physical system implements an automaton (granted that this relation is objective), then there must be yet another feature of computing systems that distinguishes them from the noncomputing systems. And the worry is that this additional feature is nonobjective. Chalmers (2011) replies to

[28] Motivated by other considerations Brian Smith argues that "C*omputation is not a subject matter*" (1996, 73).

[29] See Chrisley (1994), Chalmers (1996), Copeland (1996), Melnyk (1996), Scheutz (2001), and Godfrey-Smith (2009).

this worry by noting that universal realizability does not entail that there must be this additional feature and/or that this feature must be nonobjective. He thinks that computation is no more than the objective feature of implementing an automaton. Whether we refer to this computation depends on the phenomena we want to explain. Thus, digestion might be computation, but, presumably, performing this computation is irrelevant to digestion. "With cognition, by contrast, the claim is that it is *in virtue* of implementing some computation that a system is cognitive. That is, there is a certain class of computations such that *any* system implementing that computation is cognitive" (332–333).

My view is that there is some interest-relative element in the notion of computation. This, however, does not entail that the computational sciences do not seek to discover objective facts. These sciences might aim to discover the automata realized by certain physical objects. Moreover, the class of computing systems might have a proper subclass of entirely objective computations. Let us assume that the missing ingredient in the definition of computation is representation. As we know, some people distinguish between derivative and natural representational systems (e.g., Dretske 1988). Partly objective computers are derivative representational systems that (objectively) realize automata. Fully objective computers are natural representational systems that realize automata. Computer science might well be about the first class of systems. Computational cognitive and brain sciences, however, might well be about fully objective computers. One way or another, we cannot infer from the universal realizability result that that cognitive and brain sciences are not natural sciences. The brain might well be a fully objective computer.

2.2 Computational Theories and Explanations

The computational sciences posit computational theories and explanations. What is the nature of these theories and explanations? Again, we do not focus here on the role of computer models and simulations in the scientific investigation. Our query is about the nature of scientific theories and explanations (including models) whose goal is to theorize and explain physical computing systems.

The two main themes that concern philosophers are the nature of computational explanations and the relations between computational explanations of a physical system and other ("more physical") explanations of this system. Robert Cummins (2000) takes computational explanations to be a species of *functional analysis*: "Functional analysis consists in analyzing a disposition into a number of less problematic dispositions such that programmed manifestation of these analyzing dispositions amounts to a manifestation of the analyzed disposition. By 'programmed' here, I simply mean organized in a way that could be specified in a program or flowchart" (125). In some cases, the subcapacities/dispositions may be assigned to the components of the system, but, in other cases, they are assigned to the whole system. An example of the latter is a Turing machine: "Turing machine capacities analyze into other Turing machine capacities" (125). Another example is some of our calculative abilities: "My capacity to multiply

27 times 32 analyzes into the capacity to multiply 2 times 7, to add 5 and 1, and so on, but these capacities are not (so far as is known) capacities of my components" (126).[30]

As for the relations between computational explanations and "lower level" more physical explanations, Cummins and others take the former to be "autonomous" from the latter ones. One reason for the autonomy is the multiple realization of computational properties. Whereas computational properties ("software") are realized in physical properties ("hardware"), the same computational property can be realized in very different physical substrates.[31] Thus, the (computational) explanation referring to computational properties has some generality that cannot be captured by each physical description of the realizers.[32] Another (and somewhat related) reason is that there is a gulf between computational explanations that specify function (or program) and the lower level explanations that specify mechanisms—specifically, components and their structural properties. These are arguably distinct kinds of explanation.[33]

This picture is in tension with the mechanistic philosophy of science. Some mechanists argue that scientific explanations, at least in biology and neuroscience, are successful to the extent that they are faithful to the norms of mechanistic explanations (Kaplan 2011). However, computational explanations are, seemingly, not mechanistic. Moreover, mechanists characterize levels of explanations (at least in biology and neuroscience) in terms of levels of mechanisms, whereas components at different levels are described in terms of a whole–part relationship (Craver 2007). However, it is not clear how to integrate computational explanations into this picture because computational explanations often describe the same components described by lower level mechanistic explanations.

In defending the mechanistic picture, Piccinini and Craver (2011) argue that computational explanations are *sketches* of mechanisms: they "constrain the range of components that can be in play and are constrained in turn by the available components" (303). Computational descriptions are placeholders for structural components or subcapacities in a mechanism. Once the missing aspects are filled in, the description turns into "a full-blown mechanistic explanation"; the sketches themselves can thus be seen as "elliptical or incomplete mechanistic explanations" (284). They are often a guide toward the structural components that constitute the full-blown mechanistic explanations.

Another influential view is provided by David Marr (1982). Marr (1982) argues that a complete cognitive explanation consists of a tri-level framework of computational, algorithmic, and implementational levels. The computational level delineates the

[30] As we see, Cummins invokes terminology ("program") and examples of computing systems to explicate the idea of functional analysis. Computations, however, are by no means the only examples of functional analysis. Another example Cummins invokes is the cook's capacity to bake a cake (2000, 125).

[31] Ned Block writes that "it is difficult to see how there could be a non-trivial first-order physical property in common to all and only the possible physical realizations of a given Turing-machine state" (1990, 270–271).

[32] See Putnam (1973) and Fodor (1975). There are many responses to this argument; see, e.g., Polger and Shapiro (2016) for discussion.

[33] See Fodor (1968) and Cummins (2000). But see also Lycan (1987) who argues that the "mechanistic" levels are often also functional.

phenomenon, which is an information-processing task. The algorithmic level characterizes the system of representations that is being used (e.g., decimal vs. binary) and the algorithm employed for transforming representations of inputs into those of outputs. The implementation level specifies how the representations and algorithm are physically realized.

Philosophers disagree, however, about the role Marr assigns to the computational level. Many have argued that the computational level aims at stating the ~~visual~~ task to be explained; the explanation itself is then provided at the algorithmic and implementation levels (Ramsey 2007). Piccinini and Craver (2011) argue that Marr's computational and algorithmic levels are sketches of mechanisms. Yet others have associated the computational level with an idealized *competence* and the algorithmic and implementation levels with actual performance (Craver 2007; Rusanen and Lappi 2007). Egan (2010) associates the computational level with an explanatory formal theory, which mainly specifies the computed mathematical function. Bechtel and Shagrir (2015) emphasize the role of the environment in Marr's notion of computational analysis. Although we agree with Egan and others that one role of the computational level is to specify the computed mathematical function, we argue that another role is to demonstrate the basis of the computed function in the physical world (Marr 1977). This role is fulfilled when it is demonstrated that the computed mathematical function mirrors or preserves certain relations in the visual field. This explanatory feature is another aspect of the modeling approach discussed above.

2.3 Complexity: the Next Generation

Theoretical computer science has focused in the past decades not so much on computability theory but on computational complexity theory. They distinguish between problems that are tractable or can be solved efficiently, and those that are not. To make sure, all these functions ("problems") are Turing machine computable. The question is whether there is an efficient algorithm for solving this problem. So far, philosophers have pretty much ignored this development. I believe that, within a decade or so, the exciting results in computational complexity will find their way into the philosophy of science. In "Why Philosophers Should Care About Computational Complexity," Scott Aaronson (2013) points to the relevance of computational complexity to philosophy. In this last section, I echo his message. After saying something about complexity, I will discuss its relevance to scientific verification and confirmation.

An algorithm solves a problem efficiently if the number of computational steps can be upper-bounded by a polynomial relative to the problem size n. The familiar elementary school algorithm for multiplication is efficient because it takes about n^2 steps ($\sim n^2$); n here can refer to the number of symbols in the input. An algorithm is inefficient if the number of computational steps can be lower-bounded by an exponential relative to n. The truth table method for finding whether a formula (in propositional logic) is satisfiable or not is inefficient: it ~~takes~~ $\sim 2^n$ steps (as we recall, the table consists of 2^n rows,

where n is the number of different propositional variables in the formula). The difference between efficient and inefficient reflects a gulf in the growth of the computation time it takes to complete the computation "in practice." Let us assume that it takes a machine 10^{-6} seconds to complete one computation step. Using the elementary school algorithm, it would take the machine about 4×10^{-6} seconds to complete the multiplication of two single-digit numbers, and about 1/100 second to complete the multiplication of two fifty-digit numbers. Using the truth table method, it would also take the machine about 4×10^{-6} seconds to decide the satisfiability of a formula consisting of two propositional variables (assuming that n is the number of propositional variables). However, it ~~would~~ take the machine about 10^{14} (more than 100 trillion) years to decide the satisfiability of an arbitrary formula consisting of 100 propositional variables! Not surprisingly, this difference is of immense importance for computer scientists.

A problem is tractable if it can be solved by a polynomial-time algorithm; it is intractable otherwise. It is an open question whether all problems are tractable. The conjecture is that many problems are intractable. Theoreticians often present the question in terms of a "P = NP?" question. P is the class of (tractable) problems for which there is a polynomial-time algorithm; we know multiplication belongs to this class. NP is the class of problems for which a solution can be *verified* in polynomial time.[34] The satisfiability problem (SAT) belongs to NP: we can verify in polynomial time whether a proposed row in the truth table satisfies the formula or not. It is clear that P ⊆ NP. But it is conjectured that there are problems in NP that are not in P. SAT has no known polynomial-time solution, and it is assumed that it is not in P. SAT belongs to a class of *NP-complete* problems, which is a class of very hard problems. If any of these NP-complete problems turns out to belong to P, then P = NP. Our assumption in what follows is that P ≠ NP.[35]

Much of the significance of the P = NP question depends on the ~~strong~~ *Church-Turing thesis*. The thesis asserts that the time complexities of any two general and reasonable models of (sequential) computation are polynomially related. In particular, a problem that has time complexity t on some general and reasonable model of computation has time complexity poly(t) in a single-tape Turing machine (Goldreich 2008, 33)[36]; poly(t) means that the differences in time complexities is polynomial. We can think of the thesis in terms of invariance: much as the Turing machine is a general model of algorithmic computability (which is what the Church-Turing thesis is about), the Turing machine is also a general model of time complexity (which is what the ~~strong~~ thesis is about). The computational complexity of a "reasonable" model of computation can be very different from that of a Turing machine, but only up to a point: the time complexities must be polynomially related. What is a *reasonable* model of computation? Bernstein

[34] Formally speaking, P stands for polynomial time and NP for nondeterministic polynomial time. NP includes those problems that can be computed in polynomial time by a nondeterministic Turing machine.

[35] The question of whether P = NP is traced back to a letter Gödel sent to von Neumann in 1956; see Aaronson (2013) and Copeland and Shagrir (2013) for discussion.

[36] The most familiar formulation is by Bernstein and Vazirani (1997, 1411).

and Vazirani take "reasonable to mean in principle physically realizable" (1997, 1411). Aharonov and Vazirani (2013, 331) talk about a "physically reasonable computational model." As noted in Section 1, it is interesting to see the tendency of computer scientists today to associate computation with physical computation.

I believe that computational complexity theory has far-reaching implications for the way philosophers analyze proofs, knowledge, induction, verification, and other philosophical problems. I briefly discuss here the relevance of complexity to scientific verification and confirmation (Aharonov, Ben-Or, and Eban 2008; Yaari 2011). So, here goes: quantum computing has attracted much attention in theoretical computer science due to its potentially dramatic implications for the ~~strong~~ Church-Turing thesis. Bernstein and Vazirani (1997) give formal evidence that a *quantum Turing machine* violates the ~~strong~~ Church-Turing thesis; another renowned example from quantum computation that might violate the ~~strong~~ thesis is Shor's factoring algorithm (Shor 1994).[37] More generally, quantum computing theorists hypothesize that quantum machines can compute in polynomial time problems that are assumed to be outside of P (e.g., factorization) and even outside of NP.

This result seems to raise a problem for scientific verification.[38] Take a quantum system (machine) *QM*. Given our best (quantum) theory and various tests, our hypothesis is that *QM* computes a function *f* that is outside NP; yet *QM* computes *f* in polynomial time. But it now seems that we have no way to test our hypothesis. If *n* is even moderately large, we have no efficient way to test whether *QM* computes *f(n)* or not. *QM* proceeds through $\sim n^c$ steps in computing *f(n)*; *c* is some constant. Yet the scientists use standard methods of verification—"standard" in the sense that they cannot compute problems that are outside P in polynomial time. Scientists calculate the results with paper and pencil[39] or, more realistically, use a standard (Turing) machine *TM* in testing *QM*. This method is inefficient because *TM* will complete its computation in $\sim c^n$ steps, long after the universe ceases to exist. If it turns out that many of the physical (quantum) systems compute functions that are outside NP, the task of their verification and confirmation is seemingly hopeless.[40]

Aharonov, Ben-Or, and Eban (2008) show how to bypass this problem. They have developed a method of scientific experimentation that makes it possible to test non-P hypotheses in polynomial time. The technical details cannot be reviewed here. I will just mention that at the heart of the method is a central notion in computational complexity theory called *interactive proof* (Goldwasser, Micali, and Rackoff 1985). In an interactive

[37] Factorization of an integer is the problem of finding the primes whose multiplication together makes the integer. It is assumed that factorization is not in P, although it is surely in NP: given a proposed solution, we can multiply the primes to verify that they make the integer.

[38] I follow Yaari's (2011) presentation.

[39] The (reasonable) assumption is that human computation does not reach problems outside P in polynomial time (Aaronson 2013).

[40] Given that *f* is not in NP, the verification by *TM* requires $\sim c^n$ steps. But the same problem arises for problems in NP (Aharonov and Vazirani 2013). Assuming that *QM* provides only a yes/no answer for SAT, our TM will require $\sim c^n$ steps to verify the hypothesis that *QM* solves SAT.

proof system, a verifier aims to confirm an assertion made by a prover; the verifier has only polynomial time resources, whereas the prover is computationally powerful. The verifier can confirm the correctness of the assertion by using a protocol of questions by which she challenges the prover (this notion of proof is itself revolutionary because the protocol is interactive and probabilistic.[41]). Aharonov, Ben-Or, and Eban (2008) import the notion to the domain of scientific investigation, in which (roughly) the verifier is the computationally weak scientist and the prover is Mother Nature. Yaari (2011) takes this result a philosophical step further in developing a novel theory of scientific confirmation, called *interactive confirmation*. One obvious advantage of this theory over other theories (e.g., Bayesian) is that interactive confirmation can account for *QM* and similar quantum systems.

Acknowledgements

I am grateful to Ilan Finkelstein, Nir Fresco, Paul Humphreys, Mark Sprevak, and Jonathan Yaari for comments, suggestions, and corrections. This research was supported by the Israel Science Foundation, grant 1509/11.

References

Aaronson, Scott (2013). "Why Philosophers Should Care about Computational Complexity." In B.J. Copeland, C.J. Posy, and O. Shagrir, *Computability: Turing, Gödel, Church, and Beyond* (Cambridge MA: MIT Press), 261–327.

Adams, Rod (2011). *An Early History of Recursive Functions and Computability from Gödel to Turing* (Boston: Docent Press).

Aharonov, Dorit, Ben-Or, Michael, and Eban, Elad (2008). "Interactive Proofs for Quantum Computations." *Proceedings of Innovations in Computer Science (ICS 2010)*: 453–469.

Aharonov, Dorit, and Vazirani, Umesh V. (2013). "Is Quantum Mechanics Falsifiable? A Computational Perspective on the Foundations of Quantum Mechanics." In B.J. Copeland, C.J. Posy, and O. Shagrir, *Computability: Turing, Gödel, Church, and Beyond* (Cambridge MA: MIT Press), 329–349.

Andréka, Hajnal, Németi, Istvan, and Németi, Péter (2009). "General Relativistic Hypercomputing and Foundation of Mathematics." *Natural Computing* 8: 499–516.

Bechtel, William, and Shagrir, Oron (2015). "The Non-Redundant Contributions of Marr's Three Levels of Analysis for Explaining Information Processing Mechanisms." *Topics in Cognitive Science* 7: 312–322.

Bernstein, Ethan, and Vazirani, Umesh (1997). "Quantum Complexity Theory." *SIAM Journal on Computing* 26: 1411–1473.

Block, Ned (1990). "Can the Mind Change the World?" In G. S. Boolos (ed.), *Method: Essays in Honor of Hilary Putnam* (New York: Cambridge University Press), 137–170.

[41] See Aaronson (2013). See also Shieber (2007) who relates interactive proofs to the Turing test.

Blum, Lenore, Cucker, Felipe, Shub, Michael, and Smale, Steve (1997). *Complexity and Real Computation* (New York: Springer-Verlag).

Boolos, George S., and Jeffrey, Richard C. (1989). *Computability and Logic*, 3rd edition (Cambridge: Cambridge University Press).

Chalmers, David J. (1996). "Does a Rock Implement Every Finite-State Automaton?" *Synthese* 108: 309–333.

Chalmers, David J. (2011). "A Computational Foundation for the Study of Cognition." *Journal of Cognitive Science* 12: 323–357.

Chrisley, Ronald L. (1994). "Why Everything Doesn't Realize Every Computation." *Minds and Machines* 4: 403–420.

Church, Alonzo (1936*a*). "An Unsolvable Problem of Elementary Number Theory." *American Journal of Mathematics* 58: 345–363.

Church, Alonzo (1936*b*). "A Note on the Entscheidungproblem." *Journal of Symbolic Logic* 1: 40–41.

Churchland, Patricia S., and Sejnowski, Terrence (1992). *The Computational Brain* (Cambridge, MA: MIT Press).

Copeland, B. Jack (1996). "What Is Computation?" *Synthese* 108: 335–359.

Copeland, B. Jack (2002*a*). "The Church-Turing Thesis." In E. N. Zalta (ed.)., *The Stanford Encyclopedia of Philosophy* (http://plato.stanford.edu/archives/sum2015/entries/church-turing/).

Copeland, B. Jack (2002*b*). "Hypercomputation." *Minds and Machines* 12: 461–502.

Copeland, B. Jack (2002*c*). "Accelerating Turing Machines." *Minds and Machines* 12: 281–301.

Copeland, B. Jack (2004). "Computable Numbers: A Guide." In B. J. Copeland (ed.), *The Essential Turing* (New York: Oxford University Press), 5–57.

Copeland, B. Jack, Posy, Carl, J., and Shagrir, Oron, eds. (2013). *Computability: Turing, Gödel, Church, and Beyond* (Cambridge MA: MIT Press).

Copeland, B. Jack, and Shagrir, Oron (2007). "Physical Computation: How General are Gandy's Principles for Mechanisms?" *Minds and Machines* 17: 217–231.

Copeland, B. Jack, and Shagrir, Oron (2013). "Turing versus Gödel on Computability and the Mind." In B. J. Copeland, C.J. Posy, and O. Shagrir (eds.), *Computability: Turing, Gödel, Church, and Beyond* (Cambridge MA: MIT Press), 1–33.

Craver, Carl F. (2007). *Explaining the Brain: Mechanisms and the Mosaic Unity of Neuroscience* (New York: Oxford University Press).

Cummins, Robert C. (1988). *Meaning and Mental Representation* (Cambridge MA: MIT Press).

Cummins, Robert C. (2000). " 'How Does It Work?' vs. 'What Are the Laws?' Two Conceptions of Psychological Explanation." In F. Keil and R. Wilson (eds.), *Explanation and Cognition* (Cambridge MA: MIT Press), 117–144.

Davies, Brian E. (2001). "Building Infinite Machines." *British Journal for the Philosophy of Science* 52: 671–682.

Dean, Walter (forthcoming). "Algorithms and the Mathematical Foundations of Computer Science." In L. Horsten and P. Welch (eds.), *The Limits and Scope of Mathematical Knowledge* (Oxford University Press).

Dershowitz, Nachum, and Gurevich, Yuri (2008). "A Natural Axiomatization of Computability and Proof of Church's Thesis." *Bulletin of Symbolic Logic* 14: 299–350.

Deutsch, David (1985). "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer." *Proceedings of the Royal Society of London A* 400: 97–117.

Dretske, Fred (1988). *Explaining Behavior: Reasons in a World of Causes* (Cambridge, MA: MIT Press).

Earman, John (1986). *A Primer on Determinism* (Dordrecht: Reidel).

Earman, John, and Norton, John D. (1993). "Forever Is a Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes." *Philosophy of Science* 60: 22–42.

Egan, Frances (2010). "Computational Models: A Modest Role for Content." *Studies in History and Philosophy of Science* 41: 253–259.

Floridi, Luciano (1999). *Philosophy and Computing: An Introduction* (London and New York: Routledge).

Fodor, Jerry A. (1968). *Psychological Explanation* (New York: Random House).

Fodor, Jerry A. (1975). *The Language of Thought* (Cambridge, MA: Harvard University Press).

Fodor, Jerry A. (2000). *The Mind Doesn't Work That Way: The Scope and Limits of Computational Psychology* (Cambridge, MA: MIT Press).

Folina, Janet (1998). "Church's Thesis: Prelude to a Proof." *Philosophia Mathematica* 6: 302–323.

Fresco, Nir (2014). *Physical Computation and Cognitive Science* (London: Springer-Verlag).

Gandy, Robin O. (1980). "Church's Thesis and Principles of Mechanisms." In S. C. Kleene, J. Barwise, H. J. Keisler, and K. Kunen (eds.), *The Kleene Symposium* (Amsterdam: North-Holland), 123–148.

Gandy, Robin O. (1988). "The Confluence of Ideas in 1936." In R. Herken (ed.), *The Universal Turing Machine* (New York: Oxford University Press), 51–102.

Gödel, Kurt (1934). "On Undecidable Propositions of Formal Mathematical Systems." In S. Feferman, J. W. Dawson, Jr., S. C. Kleene, G. H. Moore, R. M. Solovay, and J. van Heijenoort (eds.), *Collected Works I: Publications 1929–1936* (Oxford: Oxford University Press), 346–369.

Gödel, Kurt (1956). "Letter to John von Neumann, March 20th, 1956." In S. Feferman, J. W. Dawson, Jr., S. C. Kleene, G. H. Moore, R. M. Solovay, and J. van Heijenoort, (eds.), *Collected Works V: Correspondence, H–Z* (Oxford: Oxford University Press), 372–377.

Godfrey-Smith, Peter (2009). "Triviality Arguments Against Functionalism." *Philosophical Studies* 145: 273–295.

Goldreich, Oded (2008). *Computational Complexity: A Conceptual Perspective* (New York: Cambridge University Press).

Goldwasser, Shafi, Micali, Silvio, and Rackoff, Charles (1985). "The Knowledge Complexity of Interactive Proof Systems." *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC '85)*: 291–304.

Grzegorczyk, Andrzej (1957). "On the Definitions of Computable Real Continuous Functions." *Fundamenta Mathematicae* 44: 61–71.

Gurevich, Yuri (2012). "What Is an Algorithm?" In M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán (eds.), *SOFSEM: Theory and Practice of Computer Science, LNCS 7147* (Berlin: Springer-Verlag), 31–42.

Hamkins, Joel D., and Lewis, Andy (2000). "Infinite Time Turing Machines." *Journal of Symbolic Logic* 65: 567–604.

Herbrand, Jacques (1931). "On the Consistency of Arithmetic." In W.D. Goldfarb (ed.), *Jacques Herbrand Logical Writings* (Cambridge, MA: Harvard University Press), 282–298.

Hilbert, David, and Ackermann, Wilhelm F. (1928). *Grundzuge der Theoretischen Logik* (Berlin: Springer-Verlag).

Hogarth, Mark L. (1994). "Non-Turing Computers and Non-Turing Computability." *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association* 1: 126–138.

Hopcroft, John E., and Ullman, Jeffery D. (1979). *Introduction to Automata Theory, Languages, and Computation* (Reading, MA: Addison-Wesley).

Kaplan, David M. (2011). "Explanation and Description in Computational Neuroscience." *Synthese* 183: 339–373.

Kleene, Stephen C. (1936). "General Recursive Functions of Natural Numbers." *Mathematische Annalen* 112: 727–742.

Kleene, Stephen C. (1938). "On Notation for Ordinal Numbers." *Journal of Symbolic Logic* 3: 150–155.

Kleene, Stephen C. (1952). *Introduction to Metamathematics* (Amsterdam: North-Holland).

Knuth, Donald E. (1973). *The Art of Computer Programming: Fundamental Algorithms, Vol. 1* (Reading, MA: Addison Wesley).

Lewis, Harry R., and Papadimitriou, Christos H. (1981). *Elements of the Theory of Computation* (Englewood Cliffs, NJ: Prentice-Hall).

Lycan, William (1987). *Consciousness* (Cambridge, MA: MIT Press).

Marr, David C. (1977). "Artificial Intelligence—Personal View." *Artificial Intelligence* 9: 37–48.

Marr, David C. (1982). *Vision: A Computation Investigation into the Human Representational System and Processing of Visual Information* (San Francisco: Freeman).

Melnyk, Andrew (1996). "Searle's Abstract Argument against Strong AI." *Synthese* 108: 391–419.

Mendelson, Elliott. (1990). "Second Thoughts about Church's Thesis and Mathematical Proofs." *Journal of Philosophy* 87: 225–233.

Miłkowski, Marcin (2013). *Explaining the Computational Mind* (Cambridge, MA: MIT Press).

Milner, Robin (1971). "An Algebraic Definition of Simulation between Programs." In D. C. Cooper (ed.), *Proceeding of the Second International Joint Conference on Artificial Intelligence* (London: The British Computer Society), 481–489.

Moschovakis, Yiannis N. (2001). "What is an Algorithm?" In B. Engquist and W. Schmid (eds.), *Mathematics Unlimited—2001 and Beyond* (New York: Springer-Verlag), 919–936.

Nagin, Paul, and Impagliazzo, John (1995). *Computer Science: A Breadth-First Approach with Pascal* (New York: John Wiley & Sons).

Odifreddi, Piergiorgio (1989). *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers* (Amsterdam: North-Holland Publishing).

Piccinini, Gualtiero (2007). "Computing Mechanisms." *Philosophy of Science* 74: 501–526.

Piccinini, Gualtiero (2011). "The Physical Church-Turing Thesis: Modest or Bold?" *British Journal for the Philosophy of Science* 62: 733–769.

Piccinini, Gualtiero (2015*a*). "Computation in Physical Systems." In E. N. Zalta (ed.), *Stanford Encyclopedia of Philosophy* (http://plato.stanford.edu/archives/sum2015/entries/computation-physicalsystems/).

Piccinini, Gualtiero (2015*b*). *The Nature of Computation: A Mechanistic Account* (New York: Oxford University Press).

Piccinini, Gualtiero, and Craver, Carl F. (2011). "Integrating Psychology and Neuroscience: Functional Analyses as Mechanism Sketches." *Synthese* 183: 283–311.

Pitowsky, Itamar (1990). "The Physical Church Thesis and Physical Computational Complexity." *Iyyun* 39: 81–99.

Polger, Thomas W., and Shapiro, Lawrence A. (2016). *The Multiple Realization Book* (Oxford: Oxford University Press).

Post, Emil L. (1936). "Finite Combinatory Processes—Formulation I." *Journal of Symbolic Logic* 1: 103–105.

Pour-El, Marian B., and Richards, Ian J. (1981). "The Wave Equation with Computable Initial Data Such that Its Unique Solution Is Not Computable." *Advances in Mathematics* 39: 215–239.

Pour-El, Marian B., and Richards, Ian J. (1989). *Computability in Analysis and Physics* (Berlin: Springer-Verlag).

Putnam, Hilary (1973). "Reductionism and the Nature of Psychology." *Cognition* 2: 131–149.

Putnam, Hilary (1988). *Representation and Reality* (Cambridge, MA: MIT Press).

Pylyshyn, Zenon W. (1984). *Computation and Cognition: Toward a Foundation for Cognitive Science* (Cambridge, MA: MIT Press).

Ramsey, William M. (2007). *Representation Reconsidered* (New York: Cambridge University Press).

Rogers, Hartley (1987). *Theory of Recursive Functions and Effective Computability*, 2nd edition (Cambridge, MA: MIT Press).

Rusanen, Anna-Mari, and Lappi, Otto (2007). "The Limits of Mechanistic Explanation in Neurocognitive Sciences." In S. Vosniadou, D. Kayser, and A. Protopapas, (eds.), *Proceedings of European Cognitive Science Conference 2007* (Hove, UK: Lawrence Erlbaum), 284–289.

Scheutz, Matthias (2001). "Causal vs. Computational Complexity?" *Minds and Machines* 11: 534–566.

Searle, John R. (1992). *The Rediscovery of the Mind* (Cambridge MA: MIT Press).

Shagrir, Oron (2006). "Why We View the Brain as A Computer." *Synthese* 153: 393–416.

Shagrir, Oron (2010). "Computation, San Diego Style." *Philosophy of Science* 77: 862–874.

Shapiro, Stewart (1993). "Understanding Church's Thesis, Again." *Acta Analytica* 11: 59–77.

Shieber, Stuart M. (2007). "The Turing Test as Interactive Proof." *Noûs*, 41: 686–713.

Shor, Peter W. (1994). "Algorithms for Quantum Computation: Discrete Logarithms and Factoring." In S. Goldwasser (ed.), *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (Los Alamitos, CA: IEEE Computer Society Press), 124–134.

Sieg, Wilfried (1994). "Mechanical Procedures and Mathematical Experience." In A. George (ed.), *Mathematics and Mind* (Oxford: Oxford University Press), 71–117.

Sieg, Wilfried (2002). "Calculations by Man and Machine: Mathematical Presentation." In P. Gärdenfors, J. Wolenski, and K. Kijania-Placek (eds.), *The Scope of Logic, Methodology and Philosophy of Science, Volume I* (Dordrecht: Kluwer Academic Publishers), 247–262.

Sieg, Wilfried (2009). "On Computability." In A. Irvine (ed.), *Handbook of the Philosophy of Mathematics* (Amsterdam: Elsevier), 535–630.

Smith, Brian C. (1996). *On the Origin of Objects* (Cambridge, MA: MIT Press).

Sprevak, Mark (2010). "Computation, Individuation, and the Received View on Representation." *Studies in History and Philosophy of Science* 41: 260–270.

Syropoulos, Apostolos (2008). *Hypercomputation: Computing Beyond the Church-Turing Barrier* (New York: Springer-Verlag).

Turing, Alan M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society, Series 2* 42: 230–265.

Turner, Raymond (2013). "Philosophy of Computer Science." In E. N. Zalta (ed.), *Stanford Encyclopedia of Philosophy* (http://plato.stanford.edu/archives/win2014/entries/computer-science/).

Vardi, Moshe Y. (2012). "What is an Algorithm?" *Communications of the ACM*, 55: 5.

Wolfram, Stephen (1985). "Undecidability and Intractability in Theoretical Physics." *Physical Review Letters* 54: 735–738.

Yaari, J. (2011). *Interactive Proofs as a Theory of Confirmation*. PhD thesis (The Hebrew University of Jerusalem).

Yanofsky, Noson S. (2011). "Towards a Definition of Algorithms." *Journal of Logic and Computation* 21: 253–286.