

Computation, hypercomputation, and physical science*

Konstantine Arkoudas

Abstract

Copeland and others have argued that the Church-Turing thesis (CTT) has been widely misunderstood by philosophers and cognitive scientists. In particular, they have claimed that CTT is in principle compatible with the existence of machines that compute functions above the “Turing limit,” and that empirical investigation is needed to determine the “exact membership” of the set of functions that are physically computable. I argue for the following points: (a) It is highly doubtful that philosophers and cognitive scientists have widely misunderstood CTT as alleged.¹ In fact, by and large, computability theorists and mathematical logicians understand CTT in the exact same way. (b) That understanding most likely coincides with what Turing and Church had in mind. Even if it does not, an accurate exegesis of Turing and Church need not dictate how today’s working scientists understand the thesis. (c) Even if we grant Copeland’s reading of CTT, an orthodox stronger version of it which he rejects (Gandy’s thesis) follows readily if we only accept a highly plausible necessary condition for what constitutes a deterministic digital computer. Finally, (d) regardless of whether we accept this condition, the prospects for a scientific theory of hypercomputation are exceedingly poor because physical science does not have the wherewithal to investigate computability or to discover its ultimate “limit.”

1 Introduction

In a series of publications over the last decade, the philosopher Jack Copeland has ignited a debate concerning the proper reading of the Church-Turing thesis (CTT), charging many prominent cognitive scientists and philosophers with serious misunderstandings. His contention is that the thesis, properly understood, says nothing about machines in general. Rather, it speaks only about what idealized *human* computists can and cannot accomplish when working by rote with paper and pencil, namely, that such computists can only compute functions that lie below the so-called “Turing limit.” More precisely, let us refer to Copeland’s version of CTT as CTTATC (or the “Church-Turing Thesis According To Copeland”):

CTTATC If a function f can be computed by an idealized human clerk working by rote with pencil and paper, then f is Turing-computable.

If one believes that CTTATC is the proper construal of CTT, then one may endorse CTT while simultaneously countenancing the possibility that certain types of machines might be capable of computing functions that are not Turing-computable. Copeland (2003, p. 11) writes:

As has already been emphasized, CTT concerns the extent of effective methods. Putting this another way (and ignoring contingencies such as boredom, death, or insufficiency of paper), the thesis concerns what a *human being* can achieve when working by rote with paper and pencil. The thesis carries no implications concerning the extent of what *machines* are capable of achieving (even digital machines acting in accordance with “explicitly stated rules”). For among a machine’s repertoire of basic operations, there may be those that no human working by rote with paper and pencil can perform.

*I wish to thank Selmer Bringsjord, Jim Fahey, Bram van Heuveln, and the anonymous referees for their helpful feedback and insights.

¹I am not claiming that philosophers and cognitive scientists have an impeccable understanding of CTT, or that they have never misused CTT in their arguments. What I am claiming is that the particular misunderstanding that Copeland and others have alleged is no misunderstanding at all.

Accordingly, he differentiates sharply between what he considers to be “the Church-Turing thesis properly so-called,” i.e., CTTATC, to which he appears to subscribe, and what he regards as quite stronger statements that are often mistakenly conflated with it and which he does not affirm. He singles out, for instance, what he calls the maximality thesis, or “thesis M”:

Thesis M If a function f can be computed by a machine (working on finite input in accordance with a finite program of instructions), then f is Turing-computable.

Of course, thesis M is often understood precisely as the interesting half of CTT. Copeland’s view is that, in fact, thesis M says much more than CTT.

Gandy (1980) made similar distinctions long before Copeland.² One difference is that whereas thesis M pertains to completely arbitrary machines, Gandy focused on deterministic digital machines.³ And, more importantly, unlike Copeland, Gandy believed that no such machine could out-compute the universal Turing machine. Specifically, Gandy, who was a student of Turing, set out to *prove* that any function that can be computed by a deterministic digital machine adhering to a few minimally demanding principles can also be computed by a Turing machine. This is sometimes referred to as Gandy’s thesis; it can be understood as a somewhat restricted version of thesis M:

Thesis G If a function f can be computed by a digital deterministic machine (working on finite input in accordance with a finite program of instructions), then f is Turing-computable.

In Copeland’s opinion, confusions between “CTT properly so-called” and more general statements along the lines of theses G or M are quite deleterious, and he seldom misses an opportunity to decry them in print. For instance, Copeland (2003, p.10) states:

A myth has arisen concerning Turing’s work, namely that he gave a treatment of the limits of mechanism, and established a fundamental result to the effect that the UTM [the Universal Turing Machine] can simulate the behavior of *any* machine. The myth has passed into the philosophy of mind, theoretical psychology, cognitive science, Artificial Intelligence, and Artificial Life, generally to pernicious effect.

Copeland maintains that this pernicious “myth” has led many thinkers astray, causing them to fall prey to various serious fallacies, which Copeland has deftly labeled for easy identification: the “Church-Turing fallacy,” the “simulation fallacy,” the “equivalence fallacy,” etc. The list of misguided authors and publications that Copeland singles out for rebuke is not short. It includes Daniel Dennett, the Oxford Companion to the Mind, Paul and Patricia Churchland, Andrew Hodges (Turing’s biographer), Alan Newell, Jerry Fodor, John Searle, and several others (Copeland 2003, Copeland and Proudfoot 1999, Copeland 2000, Copeland 2002b).

2 On mechanical computability and exegetical analyses

A key claim of Copeland concerns the meaning of the term “mechanical computability” in logic and theoretical computer science. He contends that the term means computability by “an ideal *human* clerk” working by rote with pencil and paper, and that “this is the technical meaning of ‘mechanical’” in logic. This is a claim that he makes repeatedly, insisting that in logic the term ‘mechanical’ does not “carry its everyday sense” because it has nothing to do with machines, pertaining only to idealized human computists;⁴ e.g., see Copeland (2002a, p. 487) or Copeland (2004, p. 42). The reason why Copeland puts so much weight on this point is simple: Logicians often express CTT by saying that every “mechanically computable” function is Turing-computable, and this formulation, if taken literally, is tantamount to thesis M, which Copeland emphatically denies. Hence Copeland’s insistent claim that in logic the term “mechanical” has nothing to do with machines per se, but is instead tied to human computists. If true, this terminological peculiarity would also serve as an error theory for Copeland, i.e., it would help to explain how the alleged misunderstanding arises. The explanation would go as follows: Logicians and computability theorists habitually formulate CTT as the claim that every “mechanically computable” function is Turing computable; unsuspecting cognitive scientists and philosophers come across such

²The same goes for Sieg (1994), who also predated Copeland in this respect.

³Some basic terminology: A machine is *deterministic* iff any given state of it has at most one successor state. It is *digital* iff it uses discrete (non-continuous) values to represent information. An *analog* machine is one that is not digital.

⁴I am using the term “computist” as a synonym for what Sieg (1994) and others call “computer,” namely, a human computer.

formulations, and, unaware that in logic the term “mechanically computable” has a special meaning that tethers it to human computists, they mistakenly come to believe much stronger propositions, such as theses G or M. The whole unfortunate situation would thus be little more than a terminological confusion. If only everybody understood what “mechanically computable” really means in logic, the misunderstandings could be averted.

However, Copeland’s claim is not true. In logic, “mechanically computable” is used synonymously with *computable by way of an algorithm*, where the term “algorithm” has no connotations involving idealized human computists. In fact, virtually all scholars in the field describe algorithms as procedures that are carried out by unspecified arbitrary “machines” or “digital computers,” *not* human computists.⁵ Indeed, it is widely regarded as entirely immaterial whether an algorithm is carried out by an organic life form such as a human, by silicon, or by some other type of hardware device. Algorithms could just as well be executed by properly trained pigeons. To keep insisting, as Copeland does, that in logic mechanical computability is tantamount to computability by idealized *humans* is to impute a degree of anthropocentrism and psychologism to the field that is not warranted by the facts. Of course, Copeland could claim that logicians and theoretical computer scientists have also misinterpreted the notion of an algorithm, and that they, too, have fallen for the same “myths” surrounding CTT as everybody else. But that claim would have little *prima facie* plausibility, as it would entail that the misunderstandings that he alleges are not endemic to cognitive science and the philosophy of mind, but rather border on a collective delusion of sorts that cuts across all relevant fields, including those which comprise professional computability theorists. In any event, that is not the claim that he makes. The claim that he does make is that in logic “mechanically computable” is understood as computable by an idealized *human* computist, and, as a descriptive statement of fact, that claim is false.

Historically, it is true that the fundamental constraints on algorithms (e.g., that every step should manipulate a finite amount of information, terminate in a finite amount of time, etc.) can be traced to corresponding human limitations, and that Turing explicitly referred to human computers as a means of analogy when he first introduced Turing machines (e.g., comparing the state of the machine to a human’s “state of mind,” etc.), and in some of his later writings as well. However, the constraints in question are extremely meager, and a moment’s reflection will suggest that they are applicable not only to humans but to any type of *finitary* system or device that purports to compute symbolically,⁶ that is, any type of computer in which all of the following quantities are finite: number of components, duration of converging computations, memory used by converging computations, number of internal states (or program size); symbol-recognition sensitivity; and precision of symbolic output. There is nothing intrinsically human about these constraints. In fact the one attribute that *is* intrinsically human, namely intelligent insight, is explicitly barred from algorithmic computation, which is required to proceed without any recourse to ingenuity. That is why algorithmic computation is also called “mechanical,” because the agent executing the algorithm exhibits no incentive or originality, carrying on like a *machine* instead of a thinking, creative human being. Contra Copeland, that is a perfectly faithful reflection of the everyday sense of the term “mechanical.”

The fact that Turing’s original analysis (1936) made explicit references to human computists is an interesting piece of historical information, and important for getting our history of ideas right, but its overall significance for contemporary interpretations of CTT is dubious, even if we accept Copeland’s reading of Turing (and certainly there are eminent Turing scholars, including Turing’s biographer, who do not accept that reading; see the references below). Consider Boolean algebra as an analogy. Its origins are mired in psychologism—George Boole wrote that his aim was “to investigate the fundamental laws of those operations of the mind by which reasoning is performed” (1958, p. 1) and to formulate “a science of the intellectual powers” (p. 3). Yet we do not claim on those grounds that Boolean algebra is an analysis of human thought, or that Boole’s postulates are—or *should*—be understood as expressing facts about mental operations. Ideas rarely remain static. As time goes by they evolve, acquiring new shades of meaning and connections to other ideas, and in the process they often end up deviating substantially from what their originators had in mind. Even if they do not coincide, what Turing and Church actually meant and what

⁵For some prominent and quite representative examples, see Lewis and Papadimitriou (1997, p. 222), Enderton (1972, p. 201), Davis (1982 [1958], p. xv), Machtey and Young (1978, p. 1), and Hinman (1978, p. 27). An exception is the recursion theorist Robert Soare (1996).

⁶Turing himself opened his landmark paper on computability with the following sentence:

The “computable” numbers may be described briefly as the real numbers whose expressions as a decimal are calculable *by finite means* (Turing 1936, my italics).

If by “calculable” Turing had meant calculable by a *human* computer, then the qualification “by finite means” would have been superfluous.

CTT signifies today *need not* coincide.

They most probably *do* coincide, however. Because even for strictly exegetical purposes, the really important question is not whether Turing or Church made references to human computists, but whether this was a point that they thought of as essential or whether they would have been receptive to formulations of their thesis that spoke of arbitrary computing machines instead. *That* is much more debatable. Indeed, Hodges (2004, 2006) has made a very cogent case to the effect that certainly Church, and very probably Turing as well, did not attach much significance to Copeland's vehement differentiation of algorithmic computability from general mechanical computability—just as most contemporary thinkers do not. Piccinini (2003, p. 28) agrees with this assessment, writing that Turing “thought that his machines delimited the computing power of *any* machine” (my emphasis).

Interestingly, Gandy admitted that most thinkers are not receptive to distinctions between his thesis and CTT. He wrote:

The reader may (like several of those with whom I have talked) feel that, once Turing's ideas have been grasped, [thesis G] is so unproblematic as to make arguments for it uninteresting or even unnecessary. (Gandy 1980, p. 125)

Gandy went on to offer some brief hypotheses about the reasons behind such attitudes. The main reason, in his view, is that most of the actual computing machines in use are digital computers whose design was greatly influenced by Turing's ideas. Perhaps this tends to lock people into a certain Turing-machine mindset which predisposes them to believe that any type of computation whatsoever can be performed by Turing machines. But this is not a plausible explanation. Theoretical computer scientists and logicians had been well aware for a long time of computational models that appeared radically different from Turing machines and yet turned out to be no more powerful than them. For instance, as a model of computation, the λ -calculus has very little to do at first glance with Turing machines. It is, as Turing himself recognized, “very differently defined” from the Turing model of computation (1936, p. 231), which, incidentally, is why the efficient compilation of higher-order functional languages into Turing-like machine languages is notoriously more difficult than it is for imperative programming languages. There are only two operations, β - and η -reduction, and there are no tapes, no scanning heads, etc. Yet it turns out to capture the same class of functions as Turing machines. Likewise for Gödel's recursive equations. And by the time Gandy's paper was written, many other diverse types of software and hardware computing paradigms had appeared, including logic programming (Prolog), message-passing models, dataflow computer architectures—such as the MIT Tagged Token machine—featuring content-addressable memories with significant differences from the classic Turing-von Neumann model, and so on. But they all turned out to capture the same class of functions.⁷ So it is not for lack of diversity or for want of alternative ways of computing that computer scientists identify CTT with theses G or M. If anything, the fact that all these seemingly very different alternatives give rise to the same class of functions is often taken as a sort of inductive evidence for CTT. I will argue, however, that this equivalence is more properly understood as a *consequence* of CTT, not as evidence for it.

3 Prediction and explanation for deterministic digital computers

At any rate, it is not so much the distinction between algorithmic computability and computability by arbitrary machines that is of significance here. Gandy was the first to call attention to the distinction, but if we accept his analysis then no harm is done by conflating the two notions because he comes to the conclusion that the two are coextensional, at least for the types of computing machines that he considered. Blurring the distinction becomes significant—a lamentable “myth”—only if we believe it likely that the former notion of computability, the algorithmic kind, is properly contained inside the latter. Gandy thought that such an inclusion did not obtain, although he believed that this required argumentation over and above Turing's analysis. And the vast majority of computer scientists, cognitive scientists, philosophers of mind, etc., also do not believe that such an inclusion

⁷Copeland attempts to sidestep this diversity by claiming that *all* well-known computing paradigms were developed in order to analyze *human* computability. He writes that the equivalence of such a diverse collection of computing models “is nothing more than a confusion . . . The analyses under discussion are all analyses of the notion of an effective method. Each seeks to characterize the processes that are mechanical in the sense that they can be carried out by a human computer” (2002a, p. 488). No evidence is adduced to support the claim that all these different models were specifically seeking to analyze “human computers.” In fact the claim is certainly false for more recent computing models such as logic programming or cellular automata.

obtains. I am with them, although I do not believe that the arguments of Gandy (ingenious as they were) are necessary in order to establish his thesis. I think there is a simpler connection between algorithmic computability and computability by arbitrary deterministic digital computers, a connection which ensures that even if we accept Copeland's interpretation of CTT, thesis G follows from it. Specifically, in what follows I will argue that thesis G is entailed by CTTATC in tandem with a quite plausible necessary condition for being a deterministic digital computer.⁸

The condition is this: systematic predictability of observable behavior. Any class \mathcal{C} of deterministic digital computers ought to be amenable to systematic analysis. We should be able to construct mathematical idealizations of the elements of \mathcal{C} such that given *any* mathematical description of a machine $C \in \mathcal{C}$ and *any* description of an appropriate input x , we are able to *predict the course of the execution of C on input x* . That is, we should be able to make precise statements about what will happen at any given point in the future once we start running C on x . In general, of course, prediction is an essential aspect of all forms of science and engineering: We construct a mathematical model of a class of systems (e.g., pendulums or airplanes), usually by solving an appropriate system of differential equations, and use it to make predictions about the future behavior of any particular system, given some initial conditions. It is my contention that if no such mathematical theory is available and no such predictions can be made systematically, then we are not dealing with deterministic computing machines in any reasonable sense of the term. Loosely put, if the observable behavior of a device is inherently unpredictable, then that device is not a deterministic digital computer.

That certainly accords both with ordinary practice and with our pretheoretical intuitions about this concept. Abstractly, a digital deterministic machine M is normally understood as a device such that, given a description of its operational semantics and an initial state of it, we can effectively predict the exact state in which M will be after an arbitrary number of execution steps.⁹ This property is considered to be an essential ingredient of what it means to be a digital machine in an abstract sense. For instance, it is an essential feature of the sense in which the term "abstract machine" is used in programming language theory (witness the SECD machine (Landin 1964), the Algol machine (Randell and Russell 1964), the Warren abstract machine (Warren 1983), the P-machine (Ammann 1981), etc.), as well as the sense in which it is used in specification and analysis of arbitrary dynamic systems (software or hardware), as in the "abstract machines" of the B methodology (Abrial 1996).

But what does it mean exactly for the behavior of a deterministic computer to be systematically predictable? Intuitively, what we mean when we say that we can predict the behavior of a machine is that given an abstract model of such a machine and *any* input to it, we can sit down, roll up our sleeves, and calculate the state of the

⁸I would actually go further and claim that thesis M follows from CTTATC in tandem with this condition. First, non-determinism is theoretically dispensable, in that it only provides greater economy of expression (and potentially efficiency), but not greater computability power. Moreover, the arguments given here about systematic prediction and explanation apply to non-deterministic machines as well. In particular, there exists a robust notion of systematic predictability for conventional non-deterministic computers that is not shared by non-deterministic hypercomputers. (Of course, prediction in the presence of non-determinism is probabilistic, as it is in scientific theories such as statistical mechanics.) The same is true for explanation, although the situation there is more subtle, as intuitions vary regarding what it means to explain improbable events. Similar considerations apply to analog computers. Certainly any analog computer that has ever been hitherto used could be simulated with perfect precision by classical digital means. It is the received view that such simulation is *always* possible, unless one starts making highly questionable assumptions, e.g., about the precision of physical measurements. Some of these points will be touched upon in the sequel, but in the interest of focus I will be content here to concentrate on Gandy's thesis, i.e., on digital deterministic computers. That hardly limits the scope of the discussion, since many hypercomputational models have been deterministic and digital, and proponents of hypercomputation routinely deny Gandy's thesis. The remarks in the following section will apply to hypercomputation in general, whether analog or digital, deterministic or not.

⁹By the *state* of a machine (at some particular time instant) I mean a complete description of all those components of the machine that are essential for computational purposes. At a minimum, this will include all the information that is necessary in order to *continue* the computation from that point forward. The said components can be mathematically represented by variables ranging over certain domains, and then a machine state can be understood more precisely as an ascription of particular values (drawn from the corresponding domains) to these variables. For instance, the state of a Turing machine will depict the contents of a finite portion of the tape (the infinite remainder is always assumed to be blank), the position of the scanning head, and the "internal state" of the machine. (Some authors use the term "state" for what I have called "internal state," while the term "configuration," or sometimes "snapshot," is used in place of what I have called the state of the machine. Note that a Turing machine has a finite and fixed number of internal states, whereas the number of configurations in which it can find itself is—in principle—infinite.) Mathematically, all this information can be represented by a quadruple of the form (q, u, a, w) , where q denotes the internal state (this could be a special halting state, indicating that the machine has terminated its operation), u denotes the string inscribed on the tape to the left of the current square, a denotes the symbol written on the current square, and w denotes the string to the right of the current square. As another example, the state of a P-machine (Ammann 1981) will include the contents of the registers, the stack, the heap, the code, and the value of the program counter.

machine at any given point in time after execution has commenced. And since we are dealing with mathematical idealizations, we assume unbounded paper, time, and patience. If we are asked in what state a deterministic finite automaton will be after reading a string of 10^{80} characters, we can, in principle, oblige with the answer.¹⁰ But in order to be independently reproducible by engineers, such pencil-and-paper calculations must be effective; that is, they must be routine, proceeding by rote and without any guesswork or brilliant flashes of insight. The calculations, in other words, must be *algorithmic*, precisely in Copeland's sense. An idealized human computist given sufficient paper, ink, time, and patience, should be able to carry out such calculations. This is not an ad hoc stricture on such computing machines; it is motivated by straightforward engineering and epistemological considerations. Making a prediction about a future state of a computing machine for a given input should not be an open research problem on the level of Goldbach's conjecture. The mere logical possibility of obtaining an answer based on luck or genius at some indeterminate point in the future is not enough. Algorithms must be available that engineers could use to carry out the predictions in a systematic manner.

The algorithms in question can usually be understood simply as the interpreters for the corresponding operational semantics. For instance, in the case of Postscript the algorithm evaluates programs by carrying out stack operations; in the case of the λ -calculus, the algorithm might be a graph-reduction strategy; in the case of cellular automata such as Conway's Artificial Life, the algorithm uses the evolution rules to update each cell in accordance with the states of its neighbors; and so on. I will refer to predictions obtained by such algorithms as *functional* predictions; the reasons for this terminology—as well as the logical structure of such predictions—will be elaborated shortly.

In any case, therein lies the intimate link between general mechanical computability and algorithmic computability. A class of physical objects can serve as deterministic digital computers only insofar as humans can understand how these objects work, construct decidable mathematical theories of their operation, and use the associated algorithms in order to predict—and as I will discuss next, explain—their observable behavior. That is why, as I remarked earlier, the fact that alternative models of computation turn out to be coextensional with Turing machines should be viewed as a consequence of CTTATC, rather than as evidence for it. Take a machine M_λ , for instance, that implements the λ -calculus. By virtue of being a machine, its observable behavior will be systematically predictable, and hence there will be an algorithm capable of predicting the state of M_λ at any point in time after a computation has started. Therefore, by CTTATC, there exists a Turing machine that can simulate M_λ . Such results should not be surprising. Indeed, once CTTATC has been accepted on independent grounds (say, owing to Turing's original cogent analysis), then such equivalence results ought to be expected in view of the systematic predictability criterion.

For digital deterministic computers, the flip side of prediction is explanation. One pertains to future machine states and the other to states that have already occurred, but both have the same underlying logical structure; this symmetry is often referred to as the “structural identity” of explanation and prediction.¹¹ At this point it will be helpful to say a few things about the logical structure of predictions and explanations of machine behavior.

Typically, engineers explain the operation of computing machines (indeed, of mechanical artifacts in general, including cars, radios, watches, etc.) by providing what I will call *functional explanations*, which are given from what Dennett calls the “design stance” (Dennett 1987).¹² Such explanations derive from knowledge of the functional organization of the machine coupled with knowledge about some initial state of it. In the case of deterministic machines, the logical structure of a functional explanation is inferential. It deduces the explanandum (that is, a statement describing the machine state to be explained) from the following explanans: (a) a set of formally expressed constraints representing the operating principles of the machine at some appropriate level of detail,

¹⁰Of course in practice we often enlist the help of high-speed digital computers to make predictions about the behavior of other computers, e.g. as we do when we run PCSpm to emulate a MIPS R3000 machine on a Pentium processor, or when we run an implementation of the Java abstract machine on a Macintosh. But, by the (easy) converse of CTTATC and the equivalence of Turing machines and extant digital computers, such use is inessential.

¹¹I am not endorsing this identity thesis tout court, e.g., including explanation and prediction in the social sciences. Again, the present discussion concerns the behavior of deterministic digital computing machines.

¹²The functional level of explanation and prediction which I develop here should not be conflated with either the computational or the algorithmic level of Marr (1982), or with similar distinctions made by Newell and others, although there are some similarities. The subject matter and intended scope of these distinctions are different from what I am discussing, and at any rate Marr's methodology would beg the question at issue here.

typically its operational semantics; and (b) a set of statements that express an initial state of the machine.

An example of a general principle of operational semantics in the case of Turing machines is the following:

If a Turing machine M with transition function δ_M is in a state of the form $(q, u, a_1, a_2 \cdot w)$ and if $\delta_M(q, a_1) = (q', R)$, then the next state of M will be $(q', u \cdot a_1, a_2, w)$.

(See footnote 9 for an explanation of this notation.) Or, more concisely:

If $\delta_M(q, a_1) = (q', R)$ then $(q, u, a_1, a_2 \cdot w) \rightsquigarrow_M (q, u \cdot a_1, a_2, w)$,

where all variables are assumed to be universally quantified; $a \cdot w$ (or $w \cdot a$) denotes the string obtained by prepending (respectively, appending) the symbol a to the string w ; $\delta_M(q, a) = (q', R)$ means that if the internal state is q and the current square contains the symbol a then the machine is to enter internal state q' and move one square to the right; and the symbol \rightsquigarrow_M denotes state transitions for M . In tandem with initial conditions expressing the starting configuration of the machine, such principles allow us to infer deductively the state of the machine at some future point. Of course, in practice such explanations are rarely given as actual deductions in full detail. They are usually expressed enthymematically, as skeletal deduction *sketches*. But they could be filled in completely if sufficient patience and resources were available. Indeed, that is a crucial component of their explanatory power and utility; engineers often untangle complicated behaviors by starting out with a high-level explanation and then gradually expanding it with increasing amounts of detail.

The formal specification of the functional structure of the machine can be understood as providing an implicit definition of the various terms that occur in it. In general, the meaning of a term t figuring in a specification (e.g., the term “blank symbol” in the case of Turing machines) can be understood as anything that behaves in accordance with the constraints placed upon t by the specification.

Superficially, functional explanations seem to conform to the deductive-nomological (D-N) model of explanation introduced in the seminal analysis of Hempel and Oppenheim (1948). But there is one important difference. In a D-N explanation, at least some of the general principles must be nomological, i.e., they must express empirical laws.¹³ By contrast, oftentimes all of the “laws” to which a functional explanation appeals are devoid of empirical content. They are *uninterpreted* operational specifications expressed in some appropriate logico-mathematical formalism.¹⁴ Because the specifications are uninterpreted at the functional level, the question of whether or not they are true does not arise. It makes little sense to ask whether the operational semantics of a Turing machine are true; they are true in a rather uninteresting sense, by stipulation. A more substantive question is that of logical consistency. As long as the specified theory is consistent, there will be some abstract models of it (in the set-theoretic sense of Tarskian semantics), at least one of which will be singled out as standard, or prototypical in some sense. The crucial empirical question will then be whether there also exists a class of physical systems whose structure and behavior is sufficiently similar to that of the prototypical abstract model. If so, we may say that the members of that class are physical instantiations of the abstract model, albeit with some degree of inevitable imperfection, given that the abstract models will always embody certain idealizations. We can then view the formal specification as a fairly accurate description of the behavior of the corresponding physical systems, and use it to explain and/or predict the latter.¹⁵

Accordingly, when functional explanations and predictions are issued for a specific physical system, empirical import is achieved because the consumers of the explanations (or predictions) make the key assumption that the physical system at hand satisfies the general specification. That is, the consumers of a functional explanation provide a concrete empirical interpretation of both the abstract operating principles and the initial conditions, by taking it as a hypothesis, justified by a certain body of evidence, that the physical system under consideration

¹³Or at least they must be law-like statements with counterfactual force. They need not be completely exceptionless.

¹⁴This is not to deny that those who laid down the specification did not have at least a partial concrete interpretation—often an empirical one—in mind during the design process; only that a more abstract treatment is seen to afford greater generality, applicability, etc., not unlike the mathematical definition of rings, which might be motivated from the concrete case of the integers but is nevertheless given in a more abstract setting.

¹⁵This functional analysis of deterministic machines has many similarities to the so-called semantic conception of scientific theories in general (Suppe 1989), where, for instance, a “Newtonian system” might be logically *defined* as any collection of objects that adhere to Newton’s equations. A substantive empirical theory is then obtained by a hypothesis to the effect that a given collection of physical objects is a Newtonian system.

satisfies both. That is what tacks down the abstract concepts to concrete physical materials, and what allows the formal operating principles to serve as an instrument of empirical explanation and prediction. But the functional explanations and predictions themselves are not tied to any particular physical system. That decoupling provides a significant conceptual advantage. The purely logico-mathematical nature of prediction and explanation at the functional level lies at the heart of multiple realizability. As the well-worn example goes, a heart does not have to be made of human tissue in order to perform its function. An artificial heart could work just as well, as long as it plays the right *role*, i.e., as long as it operates in accordance with its specification. Therefore, the very same proof (sketch) could be adduced to explain the operation of an artificial heart or the operation of a “real” heart, without regard for the physical substrate that constitutes the causal occupier of that role. Likewise, the very same proof could explain the behavior of a Turing machine implemented by bottles and buckets or that of an electronically implemented Turing machine.¹⁶

By their nature, functional explanations are only capable of explaining the operation of a physical machine on the assumption that the latter is a correct implementation of the specification, *and* on the assumption that the underlying physics has cooperated. That is, they can only explain the behavior of a system on the assumption that the physical components of the system have performed in accordance with their functional roles. (Conversely, a functional prediction is only capable of predicting the operation of a physical machine on the assumption that the components of the machine will perform in accordance with their functional roles.) Physical malfunctions, in particular, cannot be accounted for at the functional level. To account for such phenomena we need to descend to the physical level—which is where the D-N model enters the picture.¹⁷ Physical explanations of machine behavior, then, on the D-N model, are deductions—or deduction sketches—of a machine state from (a) general empirical laws (along with some appropriate mathematics) and (b) concrete statements expressing an initial state of the physical system. Physical explanations of machine behavior, however, are extremely rare, not so much because constructing the required deduction would be a formidable task (again, one could abbreviate heavily, use derived laws, and so on), but simply because in practice physical malfunctions are uncommon, and when they do occur it is fairly obvious that the glitch was “physical” (e.g., a power plug was accidentally pulled) and we do not care to belabor the physics of the situation, as long as it was a fluke.

Of course, neither functional nor physical explanations by themselves can give a thorough account of the operation of a machine. To achieve that we also need a teleological explanation, i.e., an explanation of the purpose of the artifact, of what it was designed to do. And formalism here has little to offer. An account of what my tax-filing program does, for instance, would need to appeal to an extremely intricate network of propositional content, including mathematical truths, evaluative political judgments, social and economic notions and conventions such as marriage, property, and money, and so on. We will return to this point later.

With this background, we can pose the following question: We have a concrete deterministic computing machine M of some sort, a member of a family of physical systems that are known to instantiate a corresponding family of logically specified abstract machines; we feed an input x to it and M starts its operation; we carry out the computation for an arbitrary duration and we then pause and record the state of M . The question is: *why is M in that state?* Putting aside the issue of physical malfunctions, the appropriate explanation would be functional; it would take the form of a deduction (sketch) demonstrating how that state was obtained from an initial configuration encoding the input x in tandem with the abstract operational semantics—an explanation from the design stance. This problem, to which I will refer as “the functional explanation problem” for a class of machines, can be made mathematically precise, assuming that the abstract functional specification was expressed in a formalism in which machines, inputs, machine states, computations, proofs, etc., can be described symbolically (a classical first-order

¹⁶Note that at the functional level we are exclusively concerned with the explanation and prediction of *behavior* or *operation*. It is not the point of functional explanations as I have conceived them here to answer questions such as “Why does a Turing machine have a tape?” or, more generally, to account for the presence of a certain component or feature. That type of teleological consideration has nothing to do with the notion of functional explanation developed here (in contrast to the biological sciences, where the presence of a trait is sometimes “functionally explained” in a forward-looking manner by appeal to its future effects).

¹⁷It is not actually necessary to subscribe to the D-N model. Explanations of machine behavior at the physical level can be obtained in accordance with the semantic conception of scientific theories simply by switching our view of the system at hand and regarding it as an instantiation of *another* abstract model, one that specifies the material rather than the functional structure of the system (provided, of course, that a set of such models has been formally defined). We would then use inference from the principles of that abstract model and the corresponding initial conditions in order to explain the physical state of the system. But this point is not too important for our present purposes.

set theory such as ZF would suffice for almost all purposes). We can then ask whether the functional explanation problem for the corresponding class of abstract machines is effectively solvable.

The answer to that question ought to be affirmative for any class of putative computing machines that operate deterministically. There must be an algorithm for producing a satisfactory functional explanation to any question of the form “why is this computer in such-and-such state?” Ideally there ought to be an algorithm for producing explanations at the physical level as well, and in fact such algorithms do invariably exist, at least in theory, because all known physical systems at present exhibit Turing-computable dynamic evolutions.¹⁸ But this is not terribly important, given that physical behavioral explanations are uncommon. By contrast, the existence of an algorithm for producing functional explanations is indispensable owing to the prevalence and importance of functional explanations. Indeed, suppose that no such algorithm exists, so that the problem of explaining the observable behavior of a certain class of computers is at least as unsolvable as the problem of determining theoremhood in number theory. This means that in trying to explain why a given machine entered a certain state at some point during a computation, we could find ourselves in the same epistemic position as in trying to determine whether Goldbach’s conjecture is true. Entire generations of mathematicians for centuries on end could try to explain why the machine acted as it did, without any guarantee of success. That would be a manifestly absurd situation for any engineering artifact, particularly for a device that was presumably designed by humans in order to exhibit highly systematic behavior in a deterministic manner. But if we accept that the explanation problem for any deterministic computing machine M must be systematically solvable, then, by virtue of the fact that an explanation for such a machine could have served as a prediction, it follows that there is an algorithm for predicting the operation of M . Therefore, by CTTATC, there is a Turing machine that can simulate the behavior of M , and so any function computable by M is also computable by a Turing machine.¹⁹

But the observable behavior of hypercomputers is not systematically predictable, and hence, in the case of deterministic machines, it is not systematically explainable either. To see this more concretely, suppose we are presented with an o-machine, that is, a Turing machine equipped with an oracle for answering queries of the form $x \in S$, where S is a set that is not Turing-computable. Suppose further that we are given a particular input, say 7. How can we determine the state of the o-machine after, say, 40 execution steps or 2035 execution steps, if one of those steps happens to be a call to the oracle? We can certainly not rely on the usual theory of Turing machines and try to simulate the machine by hand, using pencil and paper. For once we arrive at the oracle query we are stuck, since, *ex hypothesi*, the set S is not Turing-computable and therefore, by CTTATC, there is no algorithm for deciding membership in S .²⁰ Therefore, insofar the behavior of any deterministic computer ought to be systematically predictable and explainable, such machines are not, strictly speaking, computers.

The argument of this section can be summarized as follows:

- Premise 1: *The behavior of deterministic computing machines must be systematically predictable and explainable.*

¹⁸Mathematical results such as that of Pour-El and Richards (1981), who demonstrated that the wave equation does not preserve Turing-computability even when starting from Turing-computable initial conditions, do not constitute a counter-example for the simple reason that we do not know of any concrete physical systems that are correctly characterized by the required initial conditions. If we did, we would actually know that there exists a potential hypercomputer in nature (on the crucial assumption that we could somehow harness the relevant process). In other words, constructions such as that of Pour-El and Richards (1981) remain *abstract* models.

¹⁹Note that I am not requiring the existence of an algorithm for explaining and predicting any true statement whatsoever that can be made about the operation of a computing machine. For instance, I do not require that divergence (infinite loops) should be systematically predictable or explainable. Algorithms are only required for explaining and predicting statements that describe *observable machine states*, attainable in a finite amount of time after the beginning of a computation. Divergence is not an event that can be said to occur at some specific point in time, and cannot be identified with any particular machine state. The difference can be clarified by the logical form of each explanandum—an atomic sentence vs. a universally quantified formula. But the point can be better made epistemologically in terms of the difference between justifying an assertion and explaining it, which in turn hinges on the medieval distinction between *ratio essendi* (reason for being) and *ratio cognoscendi* (reason for knowing). One could be justified in asserting that a computation has some observable property at some point in time (e.g., that register R_3 contains a certain bit pattern) simply by claiming to have made the relevant observation, though one might not be able to explain why that is the case. By contrast, no such distinction obtains for assertions expressing non-observable properties of computations, such as failure to terminate; one can only justify such a claim if one can explain (i.e., prove) the claim.

²⁰Bear in mind that predictability is required for arbitrary inputs: We must be able to predict the course of the computation for any given input. While for some particular inputs to uncomputable problems a finite amount of paper-and-pencil computation could produce the answer, that is provably impossible for arbitrary inputs (as long as one accepts CTTATC).

Certainly a system could be deterministic although its behavior is neither systematically predictable nor systematically explainable. For all we know, the universe itself might be a deterministic system composed of a huge number of tiny particles, with every state completely determined by its predecessors. But its behavior at that level of detail and complexity might well fail to be systematically predictable or explainable. Under those circumstances, however, we would not say that such a deterministic system is a *computer*. Of course we might still refer to it as a computer in a loose or metaphorical sense, but not as a computer in the literal sense, i.e., not as a device that we could actually *use* to perform arbitrary computations. In the strict sense, the systematic ability to issue predictions and explanations is part and parcel of what it means to be a deterministic computing machine.

There is, of course, a tacit but crucial universal quantification in this premise: We are talking about all behaviors, or more precisely, about the prediction and explanation of an *arbitrary* behavior drawn from an infinite set of possible behaviors—not about any one particular behavior. That is an essential feature of systematicity, after all—the ability to predict and explain *any* machine behavior (in principle).

- Premise 2: *Algorithms are the only instruments there are for systematizing prediction and explanation over an infinite range of behaviors.* At least they are the only intersubjective and transparent instruments there are, amenable to collective mathematical analysis and scrutiny, and conducive to uniform and genuine understanding. Some could claim that Zeus is systematically issuing predictions or explanations on their behalf, but even in the miraculous case that observation corroborated their claims, our ignorance of the *process* that generated the predictions and explanations would rob us of the necessary understanding. Mere regularity is not sufficient for systematization; understanding is needed. Moreover, in the Zeus scenario we would have no guarantee that the verdicts will continue to be correct in the future. That is, we would have no guarantee that we can predict and explain *arbitrary* machine behaviors, and hence no guarantee that our ability is systematic after all. By contrast, we can deductively prove the scope of an algorithm's predictive and explanatory power.
- Corollary: *The behavior of deterministic computers must be algorithmically predictable and explainable.* This follows from the first two premises.
- Conclusion: *Any deterministic computer can be simulated by a Turing machine.* If the behavior of a deterministic computer is algorithmically predictable, then, by CTTATC, there is a Turing machine that can carry out these predictions at every given step, thereby simulating the computer.

I believe that the above corollary captures a constitutive aspect of deterministic digital computation.²¹ Admittedly, however, intuitions are known to vary widely. What one considers to be an essential property of *X* might seem like a pointless restriction to another, and after a while it is silly to quibble about what can and cannot properly be called '*X*'. The aim of conceptual analysis is not terminological legislation, but the probing of certain intuitions and the investigation of their origins and consequences. In my view, the thought experiment that pictures entire generations of engineers struggling unsuccessfully over periods of centuries to explain the operation of their own artifact, a single deterministic computer, suggests that something is severely awry with the concept of hypercomputation. In fact I view that scenario as a *reductio* of hypercomputation. Perhaps someone else's computing intuitions are not in conflict with such scenarios. Very well, but there are still prescriptive issues to be faced. As a matter of engineering principle, I hold that the explanation and prediction of the behavior of a deterministic computer *should not* require talents of creativity, originality, and imagination. It must be routine. It should never be a breakthrough worthy of publication when we manage to issue a successful prediction or explanation of the behavior of our own computing artifacts. That is a normative claim that deserves to be evaluated apart from any conceptual issues.

²¹My contention that algorithmic predictability and explainability are necessary features of deterministic digital computers is qualitatively similar to Davidson's claim that compositionality is a necessary feature of theories of meaning for natural languages. He wrote:

I propose what seems to me clearly to be necessary feature of a learnable language: it must be possible to give a constructive account of the meaning of the sentences in the language. Such an account I call a theory of meaning for the language, and I suggest that a theory of meaning that conflicts with this condition, whether put forward by philosopher, linguist, or psychologist, cannot be a theory of a natural language; and if it ignores this condition, it fails to deal with something central to the concept of a language. (Davidson 2001 [1966], p. 3)

Likewise, I am arguing that any machine whose behavior is not algorithmically predictable and explainable cannot be a deterministic digital computer.

4 Computation and physical science

The conclusion that oracle machines are not computing machines in anything other than a figurative sense should not be surprising to anyone familiar with their theoretical history. Oracle machines were never meant to be considered as computing machines per se, let alone as potential models of actual physical computers, in the same way that, in infinitary logic, formulas of uncountably infinite length were never meant as types that could be physically instantiated. The principal use of oracle machines has been for the strictly abstract purpose of *relativizing* computation, which induces an extremely rich and interesting partial-order structure on the class of uncomputable sets. But hypercomputation advocates often selectively exhibit quotes from Turing's work in ways that do not necessarily reflect Turing's intentions. For instance, in an article entitled "Alan Turing's Forgotten Ideas in Computer Science," Copeland and Proudfoot (1999, p. 102) state:

In his 1938 doctoral thesis at Princeton University, [Turing] described "a new kind of machine," the "O-machine."

It has been protested that quotations such as the above are designed to capitalize on the authority of Turing in order to lend credibility to the hypercomputation cause. The logician Martin Davis, for instance, had the following to say about the above paper by Copeland and Proudfoot:

It is perfectly plain in the context of Turing's dissertation, that O-machines were introduced simply to solve a specific technical problem about definability of sets of natural numbers. There is not the faintest hint that Turing was making a proposal about a machine to be built . . . It makes no sense to imagine that he was thinking about actual machines to compute the uncomputable. Turing advisedly used the term "oracle", a word redolent of the supernatural, as though to underline the purely abstract nature of his conception. Yet Copeland and Proudfoot, referring to O-machines insist that "Even among experts, Turing's pioneering theoretical concept of a hypermachine has largely been forgotten." This co-option of Turing to the fold of hypercomputation on the basis of these O-machines is without the slightest justification (Davis 2004, p. 205).

Unlike Turing, Copeland claims that, mathematically speaking, as abstract objects, oracle machines are bona fide deterministic digital computers,²² and hence, at least mathematically if not physically, thesis G is clearly false (Copeland 2003, p. 12), and therefore so is thesis M (since M entails G). I grant that there is little harm in calling these deterministic digital computing machines, as long as it is understood that the terminology is used in a loose and metaphorical sense. Naming is arbitrary, after all—anyone can attach any label they choose to any object they want. But that certainly does not mean that the mere act of dubbing these "machines" or "computers" automatically results in a refutation of thesis G, even if we only consider oracle machines as abstract mathematical entities. For Copeland has not laid down any precise criteria for when an abstract object may properly be said to be a machine.

What could such criteria be? Indeed, if we accept oracle machines as genuine abstract computing machines, then it is very difficult to see what we would not accept as such. For any number-theoretic function f , for instance, one can introduce a new type of "machine" M_f which receives an input x and then simply produces the output $f(x)$ in one step of "computation." That is perfectly formal, rigorous, deterministic, and digital. But simply defining such machines into existence does not demonstrate, in any meaningful or interesting way, that theses G or M are mathematically false.

In fact, under the suggested definition, it is trivial that for every function there is an abstract machine that computes it. Hence, every problem, modeled as a function, is "mechanically solvable" by some abstract machine, and we are thus happily reduced to abstract pancomputability.²³ After all, why accept oracle machines as genuine abstract machines and not these? What exactly are the conditions that determine whether or not an abstract object is a machine? Surely it can't be mathematical rigor alone, or the step-wise application of a fixed number of primitive operations, etc., as the foregoing trivial machines satisfy these constraints—they are rigorously defined and they "work" in steps (in one step, to be precise) by applying a fixed number of primitive operations (one operation, to be exact, i.e., producing the appropriate output result for a given input).

²²He calls them "digital computing machines" (Copeland 1998, p. 128).

²³I use the term *abstract pancomputability* to signify the thesis that every (number-theoretic) function is computable by some abstract machine. As I argue in the text, this is a rather vacuous proposition. *Physical pancomputability*, by contrast, will refer to the thesis that every function is physically computable. Neither should be confused with *pancomputationalism*, which is usually understood as the thesis that everything is a computing system, or more precisely that every physical system is a computing system; see Piccinini (2007) for an in-depth critical discussion of pancomputationalism. Note that physical pancomputability neither implies nor is implied by pancomputationalism.

The point is simple but deserves to be pressed: Anyone can define any type of abstract machine they like. Simply calling a certain type of set-theoretic object a “machine” does not automatically confer upon it the honorific distinction of breaking any “limits” and certainly does not result in any type of refutation of theses G or M other than entirely trivial ones. To say that theses M and G are false from a mathematical or abstract viewpoint is meaningless. Such a claim cannot be evaluated unless one specifies precise criteria for what is to count as an abstract machine in general. In the absence of such criteria, theses G and M can indeed be falsified definitionally, with one stroke of the pen, but such refutations are of no interest.

Copeland (2002a, p. 489) tries to anticipate and rebut the objection that hypercomputation leads to pancomputability. He formulates the objection as follows:

It seems that according to hypercomputationalists, every function is computable (or generatable by some machine). Each number-theoretic function is computable by a machine accessing an infinite tape on which are listed all the arguments of the function and the corresponding values. ETMs (Section 1.6) even permit an entire real number to be stored on a single square of the machine’s tape. And there is no reason to stop there—additional fantasy brings additional computable functions. On the new way of speaking, “computable function” means simply “function”. Hypercomputationalism comes down to this: the term “computable” is redundant.

He proceeds to offer the following rejoinder (reprinted here in its entirety):

Hypercomputationalists believe that statements concerning computability are explicitly indexed to a set of capacities and resources. When classicists say that some functions are absolutely uncomputable, what they mean is that some functions are not computable relative to the capacities and resources of a standard Turing machine. That particular index is of paramount interest when the topic is computation by effective procedures. In the wider study of computability, other indices are of importance. As the objection indicates, some indexed statements of computability are entirely trivial—for example, the statement that each number-theoretic function is computable relative to itself. This is not generally so, however. Mathematical theorems of the form “ f is computable relative to r ” are often hard-won. Questions about which functions are computable relative to certain physical theories are seldom trivial. The question of which functions are computable relative to the theories that characterize the real world is of outstanding interest.

Note that here Copeland focuses on computability “relative to certain physical theories,” without any defense of his claim that theses G or M are falsified from a purely mathematical viewpoint (a defense which, again, would require precise criteria for determining when a mathematical object qualifies as a machine). Let us concentrate on physical computability then, since, in view of the above discussion, that is the most interesting sense in which the veracity of theses G and M can be debated.

First, however, it should be noted that the belief that “statements concerning computability are explicitly indexed to a set of capacities and resources” is not peculiar to hypercomputationalists, nor did it originate with them. Its origins lie in the work of “classicist” logicians and computer scientists, who have traditionally studied resource-bounded hierarchies of computability; witness the Chomsky hierarchy, the various complexity classes, unsolvability degrees, etc. The point of the classicists is that Turing-machine computations form a maximal class of physically realizable computations. If hypercomputationalists claim that the Turing-computable functions do *not* form a maximal set of physically computable functions, then it behooves them to point out which set of functions does. There are three possibilities:

1. There is no line drawn anywhere. *All* functions are physically computable (“relative to the theories that characterize the real world”).
2. The line is drawn at the Turing limit (or perhaps even below), in accordance with something like thesis G or M.
3. The line is drawn somewhere properly in between the above two points.

The first alternative is physical pancomputability, which is highly implausible and reduces computability to triviality. The second alternative reflects the beliefs of the classicists. One would thus conclude that hypercomputationalists would advocate the third alternative. Accordingly, what we need from them is a theory that makes a properly scientific claim to the effect that: (a) *These* are the functions over and above the Turing-computable functions that can be physically computed, and here is exactly why their computability is consistent with the principles of \mathcal{T} (where \mathcal{T} is the preeminent physical theory of our time); and (b) no other functions can be physically computed,

because that would provably violate the principles of \mathcal{T} . Note that a purely mathematical theory—such as the theory of oracle machines, coupled or accelerating Turing machines, super-Turing neural networks, etc.—will not fit the bill. We must be told which physical laws (subject to empirical falsification) manage to draw a line in the set of all number-theoretic functions, and exactly where that line lies. In fact, achieving that sort of demarcation is a key aspiration of the hypercomputationalists. Copeland (2002a, p. 12), for instance, writes:

One set of functions (or numbers) is of special interest: the functions (or numbers) that are in principle computable in the real world. The exact membership of the set is an open question.

Presumably, this is a substantive open question to be settled on a posteriori grounds.

However, I will argue that there are very good reasons to doubt that this is a viable—or even meaningful—project. The root of the problem lies in viewing computers as natural rather than artifactual kinds, as illustrated in the preceding passage from Copeland, or in the following:

Computers are physical objects and computations are physical processes. What computers can or cannot compute is *determined by the laws of physics alone*, and not by pure mathematics (Deutsch 1998, p. 98, my italics).

Likewise, Hogarth (1994, p. 134) writes:

The physically possible computing limit does not “hold sway above the flux,” like the concepts of pure mathematics, but is firmly tied to some contingent and as yet unknown facts about the world.

The view of computation as a natural phenomenon that will “ultimately” be accounted for by physics is typical of the reductionist materialism that reached its heyday with the positivists and has since then been repeatedly discredited. Reductionism is admittedly a natural impulse for a materialist. It certainly seems, after all, that if material stuff is all there is, and if everything—including ourselves—is made up of such stuff, then physical science should be able to account for everything. There are no phenomena outside its purview, since all events, from the movement of heavenly bodies to a country’s elections and to computation and cognition, are ultimately physical events and hence subject to physical laws. Ultimately, therefore, once a “complete” physics has been obtained, every special science will become dispensable in that every true theory of these sciences will be entailed by the complete physical theory of our universe.

In the cognitive sciences and the philosophy of mind, this type of reductive materialism was killed a while back by a number of developments, from Davidson’s arguments for anomalous monism (Davidson 1980 [1970]) to Fodor’s arguments for the methodological autonomy of psychology (Fodor 1975) and the realization that appropriate bridge laws will not be forthcoming.²⁴ Almost twenty years ago Kim (1989, p. 32) was already noting that being a reductionist “is a bit like being a logical positivist or a member of the Old Left: an aura of doctrinaire naiveté hangs over such a person.” Very few believe that physics is capable of discovering laws pertaining to interesting psychological phenomena, or indeed that such laws even exist, as “it is practically received wisdom among philosophers of mind that psychological properties (including content properties) are not identical to neurophysiological or other physical properties” (LePore and Loewer 1989, p. 179). Indeed, cognition has been thought to be unamenable to physical science precisely because it appears that cognitive phenomena can be genuinely explained only by adverting to intentional, contentful computational states. It is the computational and representational aspects of cognition that are widely thought to make it an inauspicious subject of study for physical science (Bickle 1998, p. 2). Yet somehow when it comes to computation proper, these points are being increasingly disregarded.

Apart from the arguments for anomalous monism and various considerations having to do with scientific methodology and progress, perhaps the most influential argument against the relevance of physics to psychology was the argument from multiple realizability. Quoting Kim (1992), Fodor (1997, p. 149) writes that “the

²⁴Let \mathcal{P} be the set of statements of a complete physical science and let \mathcal{S} be the set of fundamental laws of a special science, say economics. It is clear that \mathcal{P} cannot possibly entail \mathcal{S} directly because the statements of \mathcal{P} and those of \mathcal{S} will be expressed in different vocabularies (e.g., a term such as “marginal utility” will not occur in a physical theory). Therefore, *bridge laws* are required to link the terms that occur only in \mathcal{S} to terms that occur in \mathcal{P} , by postulating “suitable relations” between whatever is signified by such terms in \mathcal{S} and concepts already present in the physical theory (Nagel 1961, p. 355). A classic example of a bridge law is the principle asserting that temperature is proportional to mean kinetic energy, stated in the context of reducing the Boyle-Charles law to the kinetic theory of gases (where the term “temperature” does not occur in the latter). Although the exact metaphysical and epistemological status of bridge laws has been a matter of some controversy, such laws are typically taken to be synthetic *identities* between predicates of two theories. No bridge laws have been established between psychological predicates and neurophysical predicates.

conventional wisdom in philosophy of mind is that psychological states are multiply realized and that this fact refutes psychophysical reductionism once and for all.” (Fodor himself enthusiastically approves of this conventional wisdom, which he helped to bring about.) The same, of course, can be said a fortiori about computational states and computations, which are the paradigms of multiple realizability. Any given computation could be realized on a seemingly endless array of radically different physical platforms, ranging from abacuses and Turing machines implemented with bottles and buckets to DNA molecules and the population of China. Any interesting generalizations that we might want to state about computations will be purely mathematical statements quantifying over abstract domains, since the physical descriptions of such computations will have absolutely nothing in common. There is no natural law that applies to physical events or sequences of physical events by virtue of the fact that they are executions of Euclid’s algorithm. Computers are not natural kinds and computational properties are not natural properties. Of course it could turn out, by some staggering coincidence, that there exists a humongous disjunctive physical predicate that happens to apply to all and only those physical events that constitute executions of Euclid’s algorithm. But it is even more extremely unlikely that such a physical predicate would be lawful, i.e., that it would figure in counterfactual-supporting generalizations, let alone generalizations discoverable by empirical investigation.

But the worries do not end there. Indeed, the point is not simply that computers are not natural kinds, so that their physical properties do not carve nature at its joints, i.e., they are not empirically projectable properties admitting of reliable extrapolation from the investigation of a limited sample. Rather, it seems reasonable to think that the property of carrying out a certain computation is not a physical property *at all*, i.e., it is not one that holds regardless of what anyone might think, know, or believe about it.²⁵ Philosophers with otherwise widely divergent views have agreed on this point. For instance, Churchland and Sejnowski (1992, p. 65) have written:

There is no intrinsic property necessary and sufficient for all computers, just the interest-relative property that someone sees value in interpreting a system’s states as representing states of some other system, and the properties of the system support such an interpretation.

Likewise, Searle (1993, p. 318) writes:

Absolutely essential, then, to understanding the nature of the natural sciences is the distinction between those features of reality that are intrinsic and those that are observer-relative. Gravitational attraction is intrinsic. Being a five dollar bill is observer-relative. . . . Computation does not name an intrinsic feature of reality but is observer-relative and this is because computation is defined in terms of symbol manipulation, but the notion of a ‘symbol’ is not a notion of physics or chemistry. Something is a symbol only if it is used, treated or regarded as a symbol.

And a little later:

Nothing is intrinsically computational. Computation exists only relative to some agent or observer who imposes a computational interpretation on some phenomenon. This is an obvious point. I should have seen it ten years ago but I did not (Searle 1993, p. 319).

Searle has also made stronger claims to the effect that any physical system could be interpreted as implementing any computer program whatsoever (1992, p. 209); similar positions have been expressed by Putnam (1988, pp. 120-125). These claims are incorrect, as has been pointed out by Chalmers (1996), Block (2002), and others, precluded by counterfactual constraints on the notion of implementation. Nevertheless, the two issues are orthogonal. It is certainly not the case that every physical system can be interpreted as computing every function, but that does not rule out the very real possibility that there is no fact of the matter intrinsic to the physics of a system—and hence discoverable by empirical inquiry—that makes it compute a certain function. If the latter is correct, as common sense would seem to suggest, and computational properties are not even supervenient on physical properties, then computation in general is not a proper subject of investigation by the methods of physical science.

In addition, it should be kept in mind that computation is an inherently normative notion (a point that was duly emphasized by Wittgenstein). Every terminating computation produces a result that is either right or wrong. Moreover, it might accidentally produce a right result for the wrong reasons, in the same way that a broken clock might be occasionally correct. Correctness and justification are therefore of paramount importance for computation, not afterthoughts. But physical science can only tell us what *is*, not what *ought to be*. There is no right and wrong

²⁵To see the distinction, consider a predicate such as *weighs between 28 and 30 grams*. This might not pick out a natural kind, but at least it determines a physical property, one that obtains or fails to obtain regardless of what anyone thinks.

in the language of physics. We do not castigate molecules for incorrect behavior. All of the above considerations suggest very strongly that physical science simply does not have the necessary conceptual resources to account for computation.

A more direct argument undermining naturalistic attempts to determine “the exact membership” of the set of physically computable functions can be given as follows. Clearly, for a physical theory to make such a determination, it must be able to tell us which functions are *not* physically computable. Now, any reasonable scientific formulation of the predicate *physically computable* should ensure that if the dynamic behavior of a given physical system can be consistently interpreted as computing a function f , then f is physically computable. It follows that in order to show that a function f is not physically computable, a scientific theory would need to demonstrate that there is no physical system whose dynamic evolution can be interpreted as computing f . That is, the theory would need to establish that for all physical systems S and for all computational interpretations I of S , it is not the case that S computes f under I .²⁶ And this no physical theory can do, for the term “interpretation” and its cognates are not physical predicates. This is not only because “interpretation” simply happens at present to not figure in the theoretical vocabulary of any physical theory, or because it does not denote a quantitative concept (unlike, say, entropy, mass, velocity, etc.). Rather, it is because it denotes an altogether different type of concept, which makes it a sort of category error to suggest that it could be a bona fide theoretical term of a physical theory. “Interpretation” is like “experiment.” The term “experiment” is a metatheoretical term used in discourse about physical theories; it is not itself a theoretical term of any particular physical theory. It could not be, for there are no physical laws prescribing what is and what is not an experiment, or in what relations experiments enter with other scientific concepts. There are certain guidelines, to be sure, but these are the province of the methodology, philosophy, and sociology of science, and perhaps of certain mathematical disciplines such as statistics, but not of any particular physical theory. No one has ever carried out any experiments in the lab in order to discover physical laws about experiments; there are no such laws. The same points apply a fortiori to computational interpretations, only in this case the concept is even more vague and subjective, i.e., context- and observer-dependent.²⁷ So no physical theory could meaningfully quantify over all interpretations, in the same way that no physical theory could meaningfully quantify over all experiments. Accordingly, no physical theory will ever tell us that a certain function is not physically computable, and hence no physical theory will be able to fix the extension of the predicate *physically computable*.

None of the above is to say that physical science cannot *enable* the construction of new and more efficient computing platforms. Clearly, solid-state physics has done just that for digital computers, and quantum physics has the potential to go even further (although it faces very serious practical challenges). But that is engineering, not science. It is not the formulation of new scientific theories, and certainly not the discovery of new physical laws about which number-theoretic functions are physically computable and which are not. Rather, it is the application of already existing scientific theories toward the goal of building more efficient computers.

The distinction between science and engineering raises the following point: Even if we agree that a physical science of computation or hypercomputation is a pipe dream, as argued above, might it not be possible nevertheless to show that this or that particular Turing-uncomputable function is physically computable? Might engineers not be able to deploy some particular physical mechanism in order to compute a Turing-uncomputable function, even if their device does not amount to a deterministic digital computer? And is it not sheer prejudice to keep a closed mind about such a potentially revolutionary breakthrough? I concede that it is conceivable (though severely dubious²⁸) that engineers might be able to harness a physical process that can be consistently interpreted as computing a Turing-uncomputable function. And certainly one should always keep an open mind (though not quite so open as to become scatter-brained, as the old saying goes). However, it should be a truism that engineering proposals ought to be judged by engineering standards. In particular, they must meet what is commonly known as the “put

²⁶A computational interpretation I of a physical system S must, at the very least, single out a set of observable properties of S in terms of which one can specify procedures for supplying inputs to S ; determining whether a computation of S has produced a result; and extracting such results by appropriate measurements.

²⁷This is not to say that the concept is vacuous and that any system can be interpreted as computing any function, as I remarked earlier, only that the concept of a computational interpretation of a physical system is anomalous—it does not figure in physical laws.

²⁸For various reasons, some of which are elaborated by Davis (2004). Further, our inability to systematically predict, explain, and verify the behavior of such a device would hinder its usefulness dramatically, as it would not be able to meet the intersubjective normative epistemological standards that are part and parcel of computation.

up or shut up” challenge. Mere thought experiments demonstrating that some hypothetical hypercomputing device or other is vaguely compatible with the principles of, say, general relativity, will not cut any ice—although they might be interesting as “playthings of philosophers, able to survive only in the hothouse atmosphere of philosophy journals” (Earman and Norton 1990, p. 40).

In fact, even a perfectly rigorous mathematical result *proving* that some proposed hypercomputer is consistent with the principles of some physical theory \mathcal{T} —and it is telling that no such rigorous results have been proved—will be of no practical relevance unless and until such a device (or at least a prototype thereof) is built and successfully tested, the reason being that this is, after all, empirical science. For all we know, some of the scientific principles on which such a proof would be based could turn out to be false. Falsification has been the fate of many a scientific theory, and there is no \mathcal{T} that could ever be an a priori exception to that fate. Second, we have as of yet no physical theory of everything, and we will probably never attain one. There is no single theory \mathcal{T} that can predict and explain everything about the universe. That is an important point to keep in mind, because compatibility arguments with *one* theory \mathcal{T}_1 might run afoul of some *other* theory \mathcal{T}_2 . That is, assumptions which do not conflict with \mathcal{T}_1 might turn out to be problematic with respect to \mathcal{T}_2 . For instance, thought experiments purporting to show the compatibility of some supertask with general relativity might violate physical constraints obtained by analyses in the context of quantum mechanics or thermodynamics (Lloyd 2000, Lloyd 2002).²⁹ Finally, all scientific theories make idealizations, and it is far from clear whether such idealizations might be pertinent to the construction and operation of devices as exotic and delicate as those claiming to be hypercomputers. The only way to demonstrate the physical plausibility of a theoretically controversial computing device is to build a prototype.

In conclusion, the idea that physical science will be able to discover fundamental computability limits is untenable. No empirical theory can establish that a function is not physically computable in principle. Thus, any scientific theory that proposes such a demarcation is bound to be ad hoc; no matter where the border is drawn, it will be an arbitrary “limit” perpetually subject to shifting. By contrast, *if* we accept CTTATC, *then* thesis G can be justified on grounds that are conceptually and epistemologically compelling, as well as pragmatically motivated: If we assume that there is a deterministic digital computer that goes beyond Turing computability, then the observable behavior of that machine would be neither systematically predictable nor systematically explainable. That would run counter to deep-seated intuitions about what it means to be a deterministic computer, and would present formidable obstacles to the hypothetical prospect of putting such machines to practical use.

References

- Abrial, J.-R.: 1996, *The B-Book, assigning programs to meaning*, Cambridge University Press.
- Ammann, U.: 1981, Code generation of a Pascal compiler, in D. W. Barron (ed.), *Pascal—The Language and its Implementation*, Wiley.
- Bickle, J.: 1998, *Psychoneural Reduction*, MIT Press.
- Block, N.: 2002, Searle’s Arguments Against Cognitive Science, in J. Preston and M. Bishop (eds), *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence*, Oxford University Press.
- Boole, G.: 1958, *An Investigation of the Laws of Thought*, Dover.
- Chalmers, D. J.: 1996, Does a Rock Implement Every Finite-State Automaton?, *Synthese* **108**, 309–333.
- Churchland, P. S. and Sejnowski, T.: 1992, *The Computational Brain*, MIT Press.

²⁹In fact it is not even clear that the “supertasks” that are needed for many hypercomputational models (typically requiring an infinite amount of work in a finite amount of time) are *logically* coherent, despite a frequently quoted passage from Russell to the effect that enumerating all the digits of π is a mere “medical” rather than logical impossibility. It might very well involve serious conceptual difficulties—for what can it possibly mean to finish a task that has no end? Comparisons to Zeno and to movement through space on the basis of purely mathematical facts such as the convergence of infinite series are glib. The paradoxical nature of supertasks has been carefully considered by Ray (1990) in the context of Thomson’s lamp. Ray concludes that “we should follow Thomson’s advice and rule out such tasks in principle” (p. 74).

- Copeland, B. J.: 1998, Turing's O-machines, Searle, Penrose, and the brain, *Analysis* **58**(2), 128–138.
- Copeland, B. J.: 2000, Narrow versus Wide Mechanism: Including a Re-Examination of Turing's Views on the Mind-Machine Issue, *Journal of Philosophy* **97**(1), 5–32.
- Copeland, B. J.: 2002a, Hypercomputation, *Minds and Machines* **12**, 461–502.
- Copeland, B. J.: 2002b, The Church-Turing Thesis, in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, Stanford University. <http://plato.stanford.edu/entries/church-turing/>.
- Copeland, B. J.: 2003, Computation, in L. Floridi (ed.), *The Blackwell Guide to the Philosophy of Computing and Information*, Blackwell, pp. 3–17.
- Copeland, B. J.: 2004, Computable Numbers: A Guide, in B. J. Copeland (ed.), *The Essential Turing*, Oxford University Press.
- Copeland, B. J. and Proudfoot, D.: 1999, Alan Turing's Forgotten Ideas in Computer Science, *Scientific American* **253**(4), 98–103.
- Davidson, D.: 1980 [1970], Mental Events, *Essays on Actions and Events*, Clarendon Press. First published in 1970.
- Davidson, D.: 2001 [1966], Theories of Meaning and Learnable Languages, *Inquiries into Truth and Interpretation*, Clarendon Press, pp. 3–15.
- Davis, M.: 2004, The Myth of Hypercomputation, in C. Teuscher (ed.), *Alan Turing: Life and Legacy of a Great Thinker*, Springer.
- Davis, M. D.: 1982 [1958], *Computability and Unsolvability*, 2nd edn, Dover.
- Dennett, D.: 1987, *The Intentional Stance*, Bradford Books.
- Deutsch, D.: 1998, *The Fabric of Reality*, Allen Lane.
- Earman, J. and Norton, J. D.: 1990, Forever is A Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes, *Philosophy of Science* **60**, 22–42.
- Enderton, H. B.: 1972, *A Mathematical Introduction to Logic*, Academic Press, New York.
- Fodor, J.: 1997, Special Sciences: Still Autonomous After All These Years, *Noûs* **31**, 149–163. Supplement: Philosophical Perspectives, 11, Mind, Causation, and World.
- Fodor, J. A.: 1975, *The Language of Thought*, Thomas Crowell.
- Gandy, R.: 1980, Church's Thesis and Principles for Mechanisms, in J. Barwise, H. Kreisler and K. Kunen (eds), *The Kleene Symposium*, North-Holland, Amsterdam, pp. 123–148.
- Hempel, C. G. and Oppenheim, P.: 1948, Studies in the Logic of Explanation, *Philosophy of Science* **15**, 135–175.
- Hinman, P. G.: 1978, *Recursion-theoretic hierarchies*, Springer, Berlin, Germany.
- Hodges, A.: 2004, What Would Alan Turing Have Done After 1954?, in C. Teuscher (ed.), *Alan Turing: Life and Legacy of a Great Thinker*, Springer.
- Hodges, A.: 2006, Did Turing have a thesis about machines?, in A. Olszewski, J. Woleński and R. Janusz (eds), *Church's Thesis after 70 years*, Ontos Verlag. Forthcoming.
- Hogarth, M.: 1994, Non-Turing Computers and Non-Turing Computability, *Proceedings of the Biennial Meeting of the Philosophy of Science Association* pp. 126–138.

- Kim, J.: 1989, The myth of nonreductive materialism, *Proceedings and Addresses of the American Philosophical Association*, Vol. 63, pp. 31–47.
- Kim, J.: 1992, Multiple realization and the metaphysics of reduction, *Philosophy and Phenomenological Research* **52**, 1–26.
- Landin, P. J.: 1964, The mechanical evaluation of expressions, *Computer Journal* **6**, 308–320.
- LePore, E. and Loewer, B.: 1989, More on making mind matter, *Philosophical Topics* **17**, 175–191.
- Lewis, H. R. and Papadimitriou, C. H.: 1997, *Elements of the Theory of Computation*, Prentice Hall.
- Lloyd, S.: 2000, Ultimate Physical Limits to Computation, *Nature* **406**, 1047–1054.
- Lloyd, S.: 2002, Computational Capacity of the Universe, *Physical Review Letters* **88**(23), 237901–3.
- Machtey, M. and Young, P.: 1978, *An Introduction to the General Theory of Algorithms*, North-Holland.
- Marr, D.: 1982, *Vision*, Freeman.
- Nagel, E.: 1961, *The Structure of Science*, Harcourt, Brace, and World.
- Piccinini, G.: 2003, Alan Turing and the Mathematical Objection, *Minds and Machines* **13**, 23–48.
- Piccinini, G.: 2007, Computational modelling vs. Computational explanation: is everything a Turing machine, and does it matter to the philosophy of mind?, *Australasian Journal of Philosophy* **85**(1), 93–115.
- Pour-El, M. B. and Richards, I.: 1981, The Wave Equation with Computable Initial Data such that its Unique Solution is not Computable, *Advances in Mathematics* **39**, 215–239.
- Putnam, H.: 1988, *Representation and Reality*, MIT Press.
- Randell, B. and Russell, L. J.: 1964, *Algol 60 Implementation*, Academic Press.
- Ray, C.: 1990, Paradoxical Tasks, *Analysis* **50**(2), 71–74.
- Searle, J.: 1992, *The Rediscovery of the Mind*, MIT Press.
- Searle, J.: 1993, The problem of consciousness, *Consciousness and Cognition* **2**, 310–319.
- Sieg, W.: 1994, Mechanical Procedures and Mathematical Experience, in A. George (ed.), *Mathematics and Mind*, Oxford University Press, pp. 71–117.
- Soare, R. I.: 1996, Computability and Recursion, *The Bulletin of Symbolic Logic* **2**(3), 284–321.
- Suppe, F.: 1989, *The Semantic Conception of Theories and Scientific Realism*, University of Illinois Press.
- Turing, A. M.: 1936, On Computable Numbers with Applications to the Entscheidungsproblem, *Proceedings of the London Mathematical Society* **42**, 230–265.
- Warren, D. H. D.: 1983, An abstract PROLOG instruction set, *Technical Report 309*, SRI International, Menlo Park, California.