Australasian Journal of Philosophy
Vol. 85, No. 1, pp. 93 – 115; March 2007

COMPUTATIONAL MODELLING VS. COMPUTATIONAL EXPLANATION: IS EVERYTHING A TURING MACHINE, AND DOES IT MATTER TO THE PHILOSOPHY OF MIND?[1]

Gualtiero Piccinini

According to *pancomputationalism*, everything is a computing system. In this paper, I distinguish between different varieties of pancomputationalism. I find that although some varieties are more plausible than others, only the strongest variety is relevant to the philosophy of mind, but only the most trivial varieties are true. As a side effect of this exercise, I offer a clarified distinction between computational modelling and computational explanation.

I. Pancomputationalism and the Computational Theory of Mind

The main target of this paper is *pancomputationalism*, according to which everything is a computing system. I have encountered two peculiar responses to pancomputationalism: some philosophers find it obviously false, too silly to be worth refuting; others find it obviously true, too trivial to require a defence. Neither camp sees the need for this paper. But neither camp seems aware of the other camp. The existence of both camps, together with continuing appeals to pancomputationalism in the literature, compel me to analyse the matter more closely. In this paper, I distinguish between different varieties of pancomputationalism. I find that although some are more plausible than others, only the strongest variety is relevant to the philosophy of mind, but only the most trivial varieties are true. As a side effect of this exercise, I offer a clarified distinction between computational modelling and computational explanation.

The canonical formulation of pancomputationalism is due to Hilary Putnam: 'everything is a Probabilistic Automaton under some Description' [Putnam 1999: 31; 'probabilistic automaton' is Putnam's term for

[1]This paper was prompted by a remark by Diego Marconi, which convinced me that the issue of whether everything is a Turing Machine needed to be sorted out. A version was presented at the 2002 APA Eastern Division in Philadelphia. I am grateful to the audience and commentator Julie Yoo for their feedback. Thanks to John Norton, Sam Scott, Oron Shagrir, Brandon Towl, the referees, and especially Peter Machamer for comments on previous versions of this paper.

Australasian Journal of Philosophy
ISSN 0004-8402 print/ISSN 1471-6828 online
http://www.tandf.co.uk/journals DOI: 10.1080/00048400601176494

probabilistic Turing Machine].[2] Pancomputationalism is typically stated without evidence or argument.[3] But that is not my main concern. My main concern is that there is a tension between pancomputationalism and the Computational Theory of Mind (CTM).

According to CTM, minds are computing systems. Different varieties of CTM can be formulated by picking appropriate classes of computing systems. For instance, there are versions of CTM according to which minds are a kind of Turing machine [Putnam 1999], digital computer [Fodor 1975; Pylyshyn 1984], connectionist computing system [Rumelhart and McClelland 1986], or even hypercomputer [Copeland 2000]. All versions of CTM have in common that they purport to explain mental phenomena by appeal to putative mental computations. CTM claims not only that minds are a special kind of computing system among others, but also that computation is a special kind of process—a process with characteristics that are relevant to explaining mental phenomena.

Supporters of CTM have offered different reasons for why computation should explain mentation. Warren McCulloch, the principal founder of modern CTM, thought that the ability to acquire and use knowledge can be reduced to the ability to draw logical inferences, and the ability to draw logical inferences can be explained by hypothesizing that the brain is a certain kind of computing system [McCulloch 1965].[4] In philosophy, Putnam argued that minds and Turing machines have at least two things in common: they can describe themselves and are 'open to rational criticism' [Putnam 1960: 149]. Another early proponent of the link between mentation and computation, Jerry Fodor, argued that both minds and computing systems exhibit their capacities by executing instructions, and hence the appeal to instruction execution is the standard form of psychological explanation [Fodor 1968a].

The putative commonalities between computing systems and minds—that they draw logical inferences, are open to rational criticism, execute instructions, etc.—are an important motivation for the view that CTM explicates the sense in which minds are genuine rule-following, as opposed to merely rule-governed, systems. For present purposes, we need not settle what feature of computing systems allows them to explain what minds do. It is enough that according to CTM, computation has this explanatory role to play.

[2]Cf. also: 'A [physical symbol] system always contains the potential for being any other system if so instructed' [Newell 1980: 161]; 'For any object there is some description of that object such that under that description the object is a digital computer' [Searle 1992: 208]; 'everything can be conceived as a computer' [Shagrir 2006]. Similar views are expressed by Block and Fodor [1972: 250], Churchland and Sejnowski [1992], Chalmers [1996b: 331], Scheutz [1999: 191], and Smith [2002: 53].

Pancomputationalism should not be confused with the stronger claim that everything implements every computation [Putnam 1988; Searle 1992; Ludlow 2003]. For a rebuttal to this stronger claim, see Copeland [1996], Chalmers [1996b], Bontly [1998], and Scheutz [2001].

[3]Sometimes, the thesis that everything is a Turing machine is said to be equivalent to, or to follow from, the Church-Turing thesis, i.e., the thesis that everything effectively calculable is computable by a Turing machine. But this is based either on a misunderstanding of the Church-Turing thesis [Copeland 2000; 2002] or on fallacious arguments [Piccinini 2007].

[4]For a detailed study of McCulloch's CTM, and a defence of the statement that McCulloch was the principal founder of modern CTM, see Piccinini [2004a].

If pancomputationalism is true, so that everything is a computing system, then minds are computing systems too. But at the same time, computation ceases to be a specific kind of process among others. If the fact that minds are computing systems follows trivially from the fact that everything is, it is unclear how computation could explain how minds exhibit their peculiarly mental characteristics. In other words, if everything is a computing system, it is unclear how computation could be interestingly related to inference, rationality, executing instructions, following rules, or anything else specific to explaining mental phenomena.[5]

The problem becomes more striking when CTM and pancomputationalism are combined with a third assertion, which may be found in the writings of some of the same authors. That is, the assertion that some things are *not* computing systems: 'the solar system is not a computational system, but you and I, for all we now know, may be' [Fodor 1975: 74, n15; see also Fodor 1968: 632; Dreyfus 1972: 68, 101–2; Searle 1980: 37–8; Searle 1992: 208]. Besides planetary systems, stomachs and the weather are some of the most often cited paradigmatic examples of systems that do not perform computations. The view that some things are not computing systems is intuitively plausible, and it is compatible with CTM's claim to explain what minds do, but it flatly contradicts pancomputationalism. To resolve this contradiction, something needs to be done.

A way out of this conundrum would be to distinguish different kinds of computational descriptions. Some computational descriptions might be relevant to explaining the behaviour of things by appeal to their computations, others might not. The former would be relevant to CTM, the latter would not. If, in the relevant sense, not everything is a computing system, then the explanatory import of CTM might be restored. Some authors have suggested something along these lines. For example, Block and Fodor write that 'there are many ways in which it could turn out that organisms are automata [i.e., probabilistic Turing machines] in some sense more interesting than the sense in which everything is an automaton under some description' [Block and Fodor 1972: 250].[6] No one, however, has spelled out in a satisfactory way the different kinds of computational descriptions and their implications for pancomputationalism and the philosophy of mind. In the rest of this paper, I propose to do that.

To a first approximation, the distinction we need is that between using a computational description to *model* the behaviour of a system—such as when meteorologists predict the weather using computers—and using it to *explain* the behaviour of a system—such as when computer scientists explain what computers do by appealing to the programs they execute. The two kinds of computational descriptions have different ontological implications about whether the behaviour being described is a computation.

[5]As Fodor puts it, 'suggesting ... that *every* causal process is a kind of computation [trivializes the] nice idea that *thought* is' [Fodor 1998: 12].

[6]Here is another example: 'we wanted to know how the brain works, specifically how it produces mental phenomena. And it would not answer that question to be told that the brain is a digital computer *in the sense* that stomach, liver, heart, solar system, and the state of Kansas are all digital computers' [Searle 1992: 208; emphasis added].

In *computational modelling* (as I'm using the term), the outputs of a computing system *C* are used to describe some behaviour of another system *S* under some conditions. The explanation for *S*'s behaviour has to do with *S*'s properties, not with the computation performed by the model. *C* performs computations in order to generate subsequent descriptions of *S*. The situation is fully analogous to other cases of modelling: just as a system may be modelled by a diagram or equation without being a diagram or equation in any interesting sense, a system may be modelled by a computing system without being a computing system in any interesting sense.

In *computational explanation*, by contrast, some behaviour of a system *S* is explained by a particular kind of process internal to *S*—a computation—and by the properties of that computation. For instance, suppose we have a calculator in working order (i.e., it has power and is functioning properly). Shortly after we press certain buttons on the calculator in a certain sequence—say, the buttons marked '5', '√', and '='—a certain string of symbols, i.e., '2.236...', appears on the calculator's display. We explain the calculator's output by pointing to the inputs we inserted into the calculator, the fact that the string '2.236...' represents the number 2.236..., the fact that 2.236... is the square root of 5, and most crucially for present purposes, the specific activity performed *ceteris paribus* by the calculator; namely, the computation of square roots. Whether we use another computing system to describe our calculator's behaviour is independent of whether the explanation for that behaviour appeals to a computation performed by the calculator. If we do use a computing system *C* distinct from our calculator to describe the calculator's *ceteris paribus* behaviour, then there will be two different computations: the calculator's and *C*'s. Nonetheless, the behaviour of the calculator is explained by the fact that *it* performs a square root computation.

In the rest of the paper, I provide a more explicit and precise taxonomy of legitimate senses in which something may be described computationally. I discuss first ordinary computational models based on differential equations, then computational models based on discrete formalisms, and finally computational explanation. In each case, I formulate a precise version of pancomputationalism, evaluate it, and draw the relevant consequences for CTM.

II. Computational Modelling (1)

In one type of computational description, the states of a system *S* are represented by the outputs of a computing system *C*, and *C* computes representations of *S*'s state at different times. In order for *C* to compute representations of *S*, *C* must be given two sorts of inputs: (i) an input specifying *S*'s state at some initial time t_0, and (ii) an input specifying *S*'s dynamical evolution (i.e., how *S*'s state evolves over time).

Trivially, *S*'s dynamical evolution may be specified by representations of *S*'s states at subsequent times, which may be obtained by measuring the relevant variables of *S* at subsequent times. Such a trivial specification

would then constitute a look-up table of *S*'s dynamical evolution. In the presence of such a table, *C*'s job reduces to retrieving the appropriate item from the look-up table. Less trivially, and more commonly, *S*'s dynamical evolution is given by a mathematical description *A*—typically, a system of differential equations—which specifies how *S*'s variables vary as a function of *S*'s state.

If *A* is solvable analytically (and if the solution is known), then *C* may be given a representation of *A*'s solutions as inputs, and *C* may use that input (together with an input specifying *S*'s initial state) to compute a representation of *S*'s state at any given time. As is well known, however, most systems of differential equations are not solvable analytically, and this is where the present type of computational modelling proves most helpful. Mathematicians have devised numerical methods for approximating the values of a system *S*'s variables directly from *S*'s dynamical description *A*, without needing to rely on *A*'s analytic solutions. In such cases, *C* may be given a representation of *A* as input, and *C* may apply numerical methods to those inputs (together with an input specifying *S*'s initial state) to compute a representation of *S*'s state at any given time. This is the most common type of computational modelling, which has become ubiquitous in many sciences [Rohrlich 1990; Humphreys 2004].

The versatility of computational models and their popularity in many quarters of science may be part of the motivation behind pancomputationalism. Given how many systems are routinely given computational descriptions by scientists in the most disparate disciplines, ranging from physics to biology to the social sciences, it is tempting to conclude that everything can be described as a computing system in the present sense. In fact, sometimes pancomputationalism is formulated as the claim that everything can be 'simulated' by a computing system.[7] Even in this sense, however, careful examination of computational modelling undermines pancomputationalism.

Most scientific descriptions are not exact but approximate. At the very least, measurements can be performed only within a margin of error, and the values of a system's variables can be specified only with finite precision. The kind of computational descriptions under discussion are not only approximate in these standard manners, but also in more significant ways. First, the mathematical description *A* that specifies the dynamical evolution of a system *S* only represents what is known about the dynamical evolution of *S*. Some factors that influence *S*'s dynamical evolution might be unknown, and since *A* might not capture them, the dynamical evolution specified by *A* might differ from *S*'s actual dynamical evolution. Second, to include everything that is known about *S* in *A* may make the mathematics analytically or computationally intractable. Typical dynamical descriptions within the sciences embody idealizations and simplifications relative to what

[7]For early claims to this effect, see von Neumann [1951] and Putnam [1964]. Some recent examples are the following: 'a standard digital computer . . . can display any pattern of responses to the environment whatsoever' [Churchland and Churchland 1990]; 'the laws of physics, at least as currently understood, are computable, and . . . human behavior is a consequence of physical laws. If so, then it follows that a computational system can simulate human behavior' [Chalmers 1996a: 329].

is known about a system, and these idealizations and simplifications may generate a difference between what the descriptions say and what the system does. Third, the numerical methods for computing the state of a system from its dynamical equations are only approximate, introducing a further discrepancy between the outputs of the computational model and the behaviour of the modelled system. Fourth, computational accuracy requires computational resources, such as memory and time. Typically, the more accuracy is required, the more computational resources need to be invested, but computational resources are always finite. Fifth, most deterministic dynamical systems are nonlinear, and most nonlinear deterministic dynamical systems have dynamics that are very sensitive to the system's initial conditions. As a consequence, many systems' dynamical evolution diverges exponentially from any representation of their dynamical evolution based on a finite specification of their initial condition. (A finite specification, of course, is all that scientists can generate in practice.) Finally, many systems are non-deterministic, so that their model can predict one of their possible behaviours, but not their actual one. Because of these factors, computational models generate descriptions that only approximate the behaviour of a system to some degree.

If we don't care how good our approximations are, i.e., if we allow the approximations generated by our computational descriptions to be arbitrarily distant from the dynamical evolution of the system being approximated, then the thesis that everything can be described as a computing system in the present sense becomes trivially true. But one virtue of scientific descriptions is accuracy, and one goal of scientists when building computational descriptions is to generate relatively accurate representations of a system's dynamical evolution. If we do care how good our approximations are, then the thesis that everything can be described as a computing system becomes too fuzzy to be significant. For whether something can be described as a computing system becomes a matter of degree, which depends on whether it can be computationally approximated to the degree of accuracy that is desired in any given case. The answer varies from case to case, and it depends at least on the dynamical properties of the system, how much is known about them, what idealizations and simplifications are adopted in the model, what numerical methods are used in the computation, and what computational resources are available. Building computational models that are relatively accurate and knowing in what ways and to what degree they are accurate takes a lot of hard work.[8]

The statement that something can be described as a computing system in the present sense applies equally well to paradigmatic computing systems (e.g., digital computers can be approximated by other computers) and to paradigmatic non-computing systems (e.g., the weather can be

[8]Some authors have argued that some physical systems have dynamical evolutions whose state transitions are not computable by ordinary computing systems, such as digital computers [e.g., Penrose 1994]. If one is strict about approximation and there are systems whose dynamical evolution involves state transitions that are not computable by ordinary computing systems, then the thesis that everything is an (ordinary) computing system in the present sense becomes strictly false.

approximated by meteorological computer programs). What explains the system's behaviour has to do with the properties of the system, which may or may not be computational, not with the computation performed by the model.

In the present sense, '*S* is a computing system' means that *S* can be described as a computing system to some degree of approximation for some modelling purpose. This can be done in an indefinite number of ways using a variety of assumptions, algorithms, notations, programming languages, and architectures. None of the resulting computational descriptions constitute computations performed by the modelled system. The computational descriptions play a modelling role fully analogous to the role played by differential equations, diagrams, and other modelling tools. Just as the same equation can describe systems that are physically very different, in the present sense the same computational description can describe systems that are physically very different. Just as the same system can be described by many different equations, some of which may approximate its behaviour better than others, the same system can be described by many different computational descriptions, some of which may approximate its behaviour better than others. Just as being described by a system of equations does not entail being a system of equations in any interesting sense, being described as a computing system in the present sense does not entail being a computing system. So, computational descriptions in the present sense say nothing about whether something literally computes. They are not the basis for a CTM.

III. Computational Modelling (2)

In a second type of computational description, the states of a system *S* are represented directly by the discrete states of an ordinary computing system *C* (such as a Turing machine or a cellular automaton), and *C*'s state transitions represent *S*'s state transitions. If *S* is analysed as a system with inputs and outputs, then *C*'s inputs and outputs represent *S*'s inputs and outputs, and given any inputs and outputs of *C* (representing any inputs and outputs of *S*), *C* goes into internal states and generates outputs that represent the states that *S* goes into and the outputs it generates.

Prima facie, not everything is describable as a computing system in this sense. For most things do not seem to have (discrete) inputs, internal states, and outputs like ordinary computing systems do, so it is not obvious how to compare their behaviour to the behaviour of a computing system to determine whether they are the same.

A natural suggestion might be that, for any system *S*, there is a computing system whose state transitions map onto *S*'s state transitions under its ordinary dynamical description. But this will not work. In modern science, dynamical descriptions are usually given not by means of computing systems but by systems of differential equations, which determine a continuous state space, which assigns an uncountable number of possible

states and state space trajectories.[9] But ordinary computing systems, such as Turing Machines (TM), can only take a finite number of states. Even if we combine the internal states of a TM with the content of the machine's tape to increase the number of possible states, the total number of states that a TM can be in is only countably infinite. Moreover, TMs can only follow a countable number of state space trajectories. The same point applies to any ordinary computing system of the kinds used in scientific modelling. So ordinary computational descriptions do not have a cardinality of states and state space trajectories that is sufficient for them to map onto ordinary mathematical descriptions of natural systems. Thus, from the point of view of strict mathematical description, the thesis that everything is a computing system in this second sense cannot be supported.[10]

This second sense in which things can be described as computing systems may be loosened by allowing the computational description C of a system S to approximate, rather than strictly map onto, the states and behaviour of S. This kind of approximation is behind the increasingly popular use of cellular automata as a modelling tool [Rohrlich 1990; Hughes 1999]. As with any model, the computational model of a system S only represents what is known about S. More importantly, discrete computational models require that S be discretized, namely, they require the partitioning of S's states into discrete states, of S's state transitions into discrete state transitions, and (in the case of cellular automata models) of S's spatial regions into discrete spatial regions. This can be done in an indefinite number of ways using an indefinite variety of formalisms, some of which may be more accurate than others for some modelling purposes.[11] Finally, computational accuracy still requires computational resources, which are always finite.

Once approximation is allowed, the caveat discussed in the previous section applies. If we don't care how good our approximations are, then the thesis that everything can be described as a computing system becomes trivially true in the present sense. Otherwise, whether something can be described as a computing system in the present sense depends on whether it can be computationally approximated to the degree of accuracy that is desired in any given case.

In any case, this second type of computational description is irrelevant to CTM, because it applies to anything depending merely on how discrete it is at the relevant level of description, i.e., on whether it has discrete inputs,

[9]Any real number within a relevant interval specifies a different initial condition of a dynamical system. For any initial condition, there is a separate state space trajectory. And within any real interval, there are uncountably many real numbers. Therefore, there are uncountably many state space trajectories. This is true not only in physics but also biology, including neuroscience (for an introduction to theoretical neuroscience, see Dayan and Abbott [2001]).

[10]The same argument applies, of course, to the kind of computational modelling described in the first section, where we reached the same conclusion by a different route.

[11]Some proponents of this kind of modelling have suggested that the universe may be fundamentally discrete in the relevant senses, so that it is possible to build exact computational models of the universe [e.g., Vichniac 1984; Toffoli 1984; Wolfram 2002]. Even if true, however, this would make these models *exact* only when the most fundamental physical variables are represented at the most fundamental orders of magnitude. All other computational modelling would remain approximate. As far as I can tell, this includes all modelling done to date. For no one knows what the most fundamental physical level is. Furthermore, there is no independent evidence that the universe is fundamentally discrete. The view that the universe is fundamentally discrete appears to be motivated by the desire for *exact* discrete computational models rather than by empirical evidence or independent theoretical considerations.

outputs, internal states, and state transitions, and perhaps on one's criteria for acceptable approximations. For example, few people would count hard bodies as such as computing systems. Yet, at a high level of abstraction hard bodies can be in either of two states, whole or broken, depending on how much pressure is applied to their extremities. A simple two-input, two-state TM, or a simple cellular automaton, can approximate the transition of a hard body from one state to the other. Nevertheless, there seems to be no useful sense in which this turns every hard body into a computing system. If you hit an ordinary desktop computer sufficiently hard, you will break it. The resulting state transition of the computer, far from constituting a computation, will prevent the computer from performing computations in the future. So, this type of computational description says nothing about whether something computes. It cannot be the notion employed by CTM.

It may be replied that at the appropriate level of description, even a computer that breaks is performing a computation, albeit an uninteresting one. According to this line of thought, the same system can perform different computations at different levels of description, and the breaking of a computer is just one computation at one level of description among many. Some authors like to call any activity of any system a computation [e.g., Wolfram 2002]. Ascribing computations in this way, though, does not make this kind of computational description relevant to the philosophy of mind. There are two reasons for this.

First, computation in this sense plays little if any explanatory role. What explains the breaking of a hard body is the physical properties of the body, the amount of pressure applied to it, and the relevant physical laws. Different bodies of different shapes and hardness break under different pressures applied in different ways—the indefinitely many computational descriptions that are common to them are *post hoc* and give no information about when something will break. Unlike this example, cellular automata and other computational formalisms can have a nontrivial modelling role, but the important point still applies. The explanation for the system's behaviour is given by properties and initial conditions of the system and relevant laws and principles, not by the models' computations. Since this kind of computational description would not play an explanatory role in a theory of mind (or of computing systems, for that matter), it's not what CTM should appeal to.

The second reason is that computations ascribed in the present sense (as well as in the previous one), unlike computations properly so called, cannot go wrong. When a computer or person who is computing function f gives the wrong output for a given input, we say that a mistake was made (e.g., because of distraction in the case of a person, or component failure in the case of a machine), and the resulting event may be called *miscomputation*. The possibility of specifying the function to be computed independently of the performance during execution, so that one can point to mistakes in the computation, is an important reason why computation is used as an ingredient in theories of mind. But there is no sense in which something that breaks under pressure can fail to generate the appropriate output: it is simply a law of physics that it will break—the system can't do anything

different. Even if we weaken or strengthen a system so that it won't break under certain conditions, there is no useful sense in which the modified system is doing something wrong. To the extent that the philosophy of mind requires a notion of computation such that mistakes can be made during computations, the present type of computational description is irrelevant to the philosophy of mind. Again, this is not what people who are interested in CTM should be concerned with.

IV. Computations, Representations, and Functions

To obtain a more robust notion of computational description, with some explanatory purchase to be employed in CTM, philosophers have explored two routes: representation and function. This section briefly discusses their pros and cons, paving the way for an improved understanding of computational explanation.

If we assume that some things are representations and some aren't, and if we define genuine computations as manipulations of representations, perhaps we obtain the notion of computation that we need for CTM. According to this line of thought, which I call the *semantic view of computation*, only processes defined over representations count as genuine computations, so that only systems that manipulate representations count as genuine computing systems [e.g., Peacocke 1999]. There is consensus that most systems, including most systems that are paradigmatic examples of non-computing systems—such as the weather, stomachs, and planetary systems—do not manipulate representations. So, according to the semantic view of computation, most systems do not count as genuine computing systems. This strategy has the great virtue of tailoring a robust notion of computation to the common assumption that mental states are representations. If minds manipulate representations, then they qualify for being members of the class of genuine computing systems. If they are computing systems in this sense, then their behaviour is explained by the computations they perform.[12] I have argued at length elsewhere that in spite of its virtues, the semantic view of computation should be replaced by a functional (non-semantic) notion of computation [Piccinini 2004b; see also Egan 1999; 2003]. Here, I only have room for sketching a few considerations against the semantic view.

For starters, the semantic view of computation trades one difficult problem for another. There is no consensus on what content is, or how to find genuine representations in the world, or even whether the notions of content and representation are sufficiently clear to make these questions

[12]Since representations can misrepresent [Dretske 1986], it may also appear that the semantic view of computation offers us a way to explicate the notion of miscomputation—namely, the making of mistakes in computing—in terms of misrepresentation. But miscomputation is orthogonal to misrepresentation: a computation may be correct or incorrect regardless of whether its inputs, outputs, and internal states represent correctly or incorrectly, or even whether they represent anything at all (see below); similarly, a representation may represent correctly or incorrectly regardless of whether it is the input, output, or internal state of a correct or incorrect computation, or even whether it is the input, output, or internal state of a computation at all.

answerable [Gauker 2003: 27]. Nevertheless, many believe that these problems can be solved by a naturalistic theory of content, and I will leave concerns over content aside. There remains the problem of specifying which representations, among the general class of representations, can support genuine computations, and which processes, among those defined over representations, count as genuine computations. Few would maintain that all representations, including paintings and facial expressions, are equally suited to have computations defined over them. By the same token, many processes defined over representations, such as drawing a diagram or playing a record, do not appear to be computations. So far, supporters of the semantic view have yet to specify in detail how to build a robust notion of computation from a suitable notion of representation.

Furthermore, there are plenty of paradigmatic computing systems that lack representational content. Examples include parsers, compilers, and assemblers.[13] For this reason among others, the relevant experts—computability theorists and computer designers—normally individuate computations without presupposing any appeal to content. For instance, computability theorists individuate strings of symbols by their formal properties (i.e., regardless of their content), and then define computations as manipulations of strings based on the strings' formal properties. This is the biggest problem faced by the semantic view of computation: it conflicts with the standard way of individuating computations by the relevant community of experts. Because of this, the semantic view is untenable.

Oron Shagrir has independently argued, based on similar considerations, that in order to include all paradigmatic examples of computation, the semantic view must be formulated with a weak notion of representation, according to which everything is a representation [Shagrir 2006]. He takes that as a reason to accept pancomputationalism. I take it as a reason against the semantic view of computation.

The second route explored by philosophers in search of a robust notion of computation is *function*. Roughly speaking, a functional explanation (or analysis) of a system is an explanation of the capacities of a system in terms of its sub-capacities [Cummins 1983; 2002]. In many cases, a system's sub-capacities are assigned to its components as their functions. A standard example is the circulatory system of organisms, which may be partitioned into the heart, the arteries, and the veins, each of which are assigned specific functions. The heart is assigned the function to pump blood from the veins into the arteries, and the capacity of the circulatory system to circulate the blood under normal conditions is explained by the functions performed by the heart, the arteries, and the veins.

There is a controversy over the correct account of functional explanation [Allen, Bekoff, and Lauder 1998; Preston 1998; Schlosser 1998; Buller 1999; Ariew, Cummins, and Perlman 2002; Christensen and Bickhard 2002]. According to some, functional explanation may be entirely explicated in terms of the contribution made to some capacity of a system by its components or sub-capacities. According to others, functional explanation

[13]Thanks to Ken Aizawa for suggesting parsers.

requires appealing to the evolutionary history of a system. According to a third group, there is a legitimate distinction between two styles of functional explanation, an 'engineering' style, which makes no appeal to evolutionary history, and a 'teleological' style, which does. In the case of artefacts, there is a debate as to whether functional explanation requires reference to human intentions or whether an account in terms of replication and use will do. And these are only the most popular among existing accounts of functional explanation.

Fortunately, for present purposes we need not choose among accounts of functional explanation. What matters here is that functional explanation is a distinctive explanatory style that applies only to some systems among others. And there is consensus that successful functional explanation has nontrivial ontological implications for the system being described: it gives functional significance to the behaviour of the system and its components; namely, it attributes to the system and its components the function of acting in certain ways under certain conditions. In fact, only artefacts and biological systems are usually said to have functions, which are then invoked in explaining their behaviour.[14]

Some philosophers have proposed to formulate theories of mind directly in terms of functional explanation, without the notion of computation.[15] Along these lines, someone might suggest that just as a functional explanation explains the behaviour of a system by appealing to functions, CTM attempts to explain mental phenomena by appealing to functions. But the notion of function employed by CTM, namely the mathematical notion of function (defined over strings or natural numbers), is not suited for a theory of mind. In formulating a theory of mind, she would propose to replace the mathematical notion of function with the notion of function employed in functional explanation. This proposal is based on a false opposition between two notions of function. Within a description of a system, the mathematical notion of function and the notion of function employed in functional explanation need not be in competition.

For instance, both notions of function play legitimate roles within computational models of biological systems and artefacts, where functional explanation is fruitfully combined with computational modelling. This can be done with either of the two kinds of computational models discussed above.

In the first kind of computational modelling, the mathematical description of a system's dynamical evolution may embody assumptions about the system's functional explanation. For instance, standard equations for the action potential of neurons, such as the classical Hodgkin-Huxley equation,

[14]Supporters of a teleological notion of function often doubt that the distinction between genuinely functional systems—such as biological systems and artefacts—and other systems can be drawn without appealing to the evolutionary history of the systems (or perhaps human intentions in the case of artefacts). I cannot pursue these doubts here, except to note that they have not persuaded supporters of alternative accounts of functional explanation.

[15]The most explicit is perhaps Elliott Sober: 'criticisms [to functionalism] are defused when that doctrine is understood in terms of a teleological (rather than Turing Machine) notion of function' [Sober 1985: 166]. For other teleological formulations of functionalism, see Lycan [1987], Millikan [1984], Shapiro [1994]. Unlike Sober, however, these authors do not explicitly reject computational formalisms as an ingredient in their theory.

include terms corresponding to several electric currents. The different currents are assumed to be the effects of different components and properties of a neuron's membrane (such as various ion channels) under normal conditions, so the different terms in the equations embody these assumptions about the functional explanation of the neuron. When computer programs are built to compute representations of action potentials based on the relevant equations, they implicitly rely on the functional analysis embodied in the equations.

Mutatis mutandis, the same point applies to the use of discrete computational models such as cellular automata. The only difference is that now the functional explanation of a system is embodied directly in the topological and dynamical structure of the model. Typically, different regions of a finite automaton represent different regions of the modelled system, and the transition rules between states of the finite automaton represent the dynamical properties of the regions of the system. If the system is functionally explained, then different regions of a finite automaton may correspond to different components of the modelled system, and the transition rules between states of the finite automaton may represent the functions performed by those components under normal conditions. In all these cases, functional explanation and the computation of mathematical functions coexist within the same description of a system. There is no reason to see one as the replacement for the other. This, however, does not yet give us the robust notion of computation that we are looking for.

Computational models that embody functional explanations explain the capacities of a system in terms of its sub-capacities. But this explanation is given by the assumptions embodied in the model, not by the computations performed by the model on the grounds of the assumptions. In the case of standard computational models, the functional explanation is embodied in the dynamical equations that describe the dynamical evolution of the system or in the assumptions behind the topology and transition rules of a cellular automaton. The situation is analogous to other computational models, where the purpose of the computation is to generate successive representations of the state of a system on the basis of independent assumptions about the system's properties. What does the explanatory job is the set of assumptions about the system's properties—in this case, the functional explanation that is the basis for the model.

The same point may be made by comparing the normativity inherent in functional explanation to the normativity inherent in computation. In the case of computational models that embody a functional explanation, the two sets of norms are independent, in the sense that they may be broken independently. The system may fail to perform its functions (e.g., a heart may cease to pump) in a variety of ways under a variety of circumstances. This may or may not be represented by a computational model of the system; to the extent that the model is accurate, it should represent malfunctions and functional failures of the modelled system under the relevant circumstances. This has nothing to do with miscomputation.

The computations that generate representations of the modelled system within a computational model may also go wrong in a variety of ways under

a variety of circumstances. For instance, the system may run out of memory, or a component of the hardware may break down. But if the computation goes wrong, the result is not a representation of a malfunction in the modelled system. Rather, it is simply a misrepresentation of the behaviour of the modelled system, or more likely, the failure to generate a representation of the modelled system. This shows that in this kind of modelling, the normativity of the functional explanation is independent of the normativity of the computation. The system is supposed to do what its functional explanation says; the model is supposed to compute what the equations (or other relevant assumptions) say. So in this case, still, computation is not explaining the behaviour of the system.

At this point, a tempting way to assign explanatory force to computation is to simply assert that functional explanations are themselves computational explanations. This move was implicitly made by the two original philosophical proponents of CTM [Fodor 1968b; Putnam 1999]. Since then, the mongrel of functional and computational explanation has become entrenched in the philosophy of psychology and neuroscience literature, where it can be found in many places to various degrees [e.g., Cummins 1975; 1983; Dennett 1978; Haugeland 1978; Marr 1982; Churchland and Sejnowski 1992; Eliasmith 2003].[16] According to this view, the functions performed by a system's components according to its functional explanation are also functions computed by that system. Hence, one can explain the behaviour of the system by appealing to the computations it performs. The virtue of this move is that it assigns computation (and hence CTM) explanatory force. The vice is that it assigns explanatory force by definitional fiat.

Aside from the desire to assign explanatory force to computation, there is no independent motivation for calling all functional explanations computational, and all activities described by functional explanations computations. Functional explanation applies to many kinds of systems engaged in many kinds of activities, ranging from pumping blood to generating electricity to digesting food. As we'll see in the next section, functional explanation (or better, mechanistic explanation) applies to what are ordinarily called computing systems too, for ordinary computing systems engage in a certain type of activity—computation—in virtue of the functions performed by their components under normal conditions. For instance, Turing machines perform their computations in virtue of the activities performed by their active device (whose function is to write and erase symbols in appropriate ways) on their tape (whose function is to store symbols). But if we take computation in anything like the sense in which it is used in computability theory and computer science, we must conclude that most functional explanations do not ascribe computations to the systems they analyse.

If we like, we may start calling every function of every system a computation. As a result, computation will acquire explanatory force by piggybacking on the explanatory force of functional explanation. But this also turns every system that is subject to functional explanation into a

[16]For a study of how this came about and a more detailed criticism, see Piccinini [2004c].

computing system. Functional explanation applies to artefacts, organisms, and their components, including stomachs and other paradigmatic examples of non-computing systems. This way of ascribing computations is too liberal to be directly relevant to the philosophy of mind in the sense of using genuine computation (as opposed to a system's function) to explain behaviour. To be relevant to the philosophy of mind, the notion of computational explanation must be restricted further.[17]

The proponents of the view that functional explanations are computational explanations have gotten the relation between functional explanation and computational explanation backwards. I will now argue that rather than functional explanation being always computational, computational explanation is a special form of functional (or better, mechanistic) explanation, which applies only to systems with special functional properties.[18]

Instead of functional explanation, I will speak of *mechanistic* explanation. The reason is two-fold: first, I wish to avoid the common assumption that functional explanation is by definition computational; second, and more importantly, I am looking for an explanatory style that appeals unequivocally not only to the capacities and sub-capacities of a system but also to its components, their functions, and their organization. As I briefly hinted above, the notion of functional explanation is usually formulated so as to blur the distinction between explanation in terms of sub-capacities of a whole system and explanation in terms of the capacities of a system's components [Cummins 2002]. By contrast, I will understand a mechanistic explanation as an explanation that appeals to a system's components, their functions, and their organization.[19]

V. Computational Explanation

Some systems manipulate inputs and outputs of a special sort, which may be called strings of digits. A *digit* is a particular or a discrete state of a particular, discrete in the sense that it belongs to one (and only one) of a finite number of types. Types of digits are individuated by their different effects on the system, that is, the system performs different functionally relevant operations in response to different types of digits. A *string* of digits is a concatenation of digits, namely, a structure that is individuated by the types of digits that compose it, their number, and their ordering (i.e., which digit token is first, which is its successor, and so on).

[17]To solve this problem, Bontly proposes to conjoin the functional approach to computational explanation with the semantic view of computation [1998: 570]. His proposal is ingenious but suffers from the problems with the semantic view, which I outlined above.

[18]For a detailed account of the individuation of computational states along these lines, see Piccinini [forthcoming].

[19]A thorough and adequate discussion of the relationship between functional and mechanistic explanation would take us too far afield. I plan to offer such a discussion elsewhere. Recent work on mechanistic explanation includes Bechtel and Richardson [1993], Machamer, Darden, and Craver [2000], Craver [2001], and Glennan [2002]. The most detailed and adequate account of mechanistic explanation that I know of is in Craver [forthcoming].

A string of digits may or may not be interpreted, that is, assigned content. If it is interpreted, it may be called a symbol or representation. But a string's interpretation is not part of its individuation, so strings of digits are not necessarily representations. Strings of digits in this sense are, to a first approximation, a physical realization of the mathematical notion of string.[20]

Among systems that manipulate strings of digits in this sense, some of them do so in a special way. Under normal conditions, these systems perform sequences of operations that depend on which strings of digits are present within the system and on the internal state of the system, so that by performing those operations, the system may generate different output strings of digits in response to different input strings of digits and different internal states.

Suppose that one can specify a general rule *R*, which applies to any string entering a system and depends on the inputs (and perhaps internal state) of a system for its application. And suppose that *R* specifies which strings the system outputs under normal conditions (for example, *R* might say that the outputs should contain all the digits present in the input arranged in alphabetical order). Then, there is a robust sense in which *R* specifies the computation that the system has the function to perform. This rule carries explanatory force. When the system generates an output given a certain input under normal conditions, and the output bears the relevant relation to the input (e.g., it contains all the digits in the input arranged in alphabetical order), *R* may be invoked to explain this fact. The system has generated that output given that input because (i) that output contains all the digits in the input arranged in alphabetical order and (ii) the function of the system is to arrange the input digits in alphabetical order.

The mathematical theory of how to generate output strings from input strings in accordance with general rules that apply to all input strings and depend on the inputs (and sometimes internal states) for their application is called computability theory. Within computability theory, the activity of manipulating strings of digits in this way is called computation. Any system that performs this kind of activity is a computing system properly so called. Of course, there remains the task of explaining how it is that a system is capable of performing a certain computation. This will be done by a mechanistic explanation of the system, which explains how the system performs its computations by pointing at the functions performed by its components under normal conditions and the way the components are organized together. For instance, in the case of a Turing machine, the mechanistic explanation will include a partition of the system into its two main components (i.e., the tape and the active device), the assignment of functions and organization to those components (the active device moves along the tape, etc.), and the specification of the machine table that expresses the algorithm the machine follows under normal conditions.

It might be helpful to briefly locate the present view in relation to the 'seven primary construals of computation' listed by Brian Smith [2002: 28].[21] Smith's 'construals' are the following: formal symbol manipulation,

[20]For the mathematical notion of string, see Corcoran, Frank, and Maloney [1974].
[21]Thanks to a referee for suggesting this.

effective computability, execution of an algorithm (or rule-following), calculation of a function, digital state machine, information processing, and physical symbol systems (as defined by Newell and Simon [1976]).

Of these, information processing is the least pertinent. Although the terms 'computation' and 'information processing' are often loosely used interchangeably, for present purposes the two notions must be understood independently. I'm employing the notion of computation that is pertinent to CTM. This is the notion that originates in computer science, and before that, in logic and computability theory (see below). Computation in this sense has to do with the generation of certain discrete outputs from certain inputs, and the present account is an attempt to be more specific about the relevant kinds of input, output, and relations between the two. This notion is independent of the notion of information, whose main use originates in communication engineering. In its most common technical sense, information has to do with the reliable correlation between variables (which need not be discrete).[22] There are many interesting connections between information and computation, and it would be philosophically useful to explore them. But they are different notions, with different origins and analyses. A computation may be used to process information (in more than one sense of the term), but it need not be. And the processing of information may be done by means other than computing (e.g., by processes defined over continuous variables).

The notion of digital state machine is independent of the notion of computing system too. Although digits and strings thereof are digital, the internal states of a computing system need not be digital. A relevant example are many types of connectionist system, which do not possess internal digital states in any interesting sense and yet generate output strings of digits from input strings in accordance with a fixed rule defined over the strings [e.g., Hopfield 1982; Rumelhart and McClelland 1986], and hence qualify as computing systems in the present sense. Conversely, many digital state machines perform computations, but others—such as soda machines—do not.[23]

A symbol in Newell and Simon's sense is similar to a string of digits in my sense. But Newell and Simon's physical symbol systems are best treated as a *subset* of computing systems. As Newell stresses, physical symbol systems are computationally universal, namely, they can compute any function computable by TMs [Newell 1980]. By contrast, most computing systems are not computationally universal. Furthermore, some authors have speculated on the possibility of hypercomputers—systems that compute functions uncomputable by TMs. Whether hypercomputers are possible is a difficult issue [cf. Copeland 2002b; Cotogno 2003], which should not be settled by definitional fiat by an account of computational explanation.

[22]There is no room to get into a detailed analysis of the notion of information here. For some of the relevant background, see Aspray [1985] and Adams [2003]. Although Aspray does not distinguish sharply between information and computation, even in his account it is easy to see that the two notions are distinct.

[23]Some philosophers have used soda machines as an example of computing systems [e.g., Block 1980], but this waters down the notion of computing system unnecessarily. Soda machines, unlike genuine computing systems, are insensitive to the ordering of their inputs. So they cannot be properly said to manipulate *strings* of digits.

Accordingly, the notion of computing system developed here is designed to cover universal computing systems as well as non-universal computing systems and hypercomputers. It may be seen as a generalization of Newell and Simon's notion.[24]

The remaining 'construals' from Smith's list are formal symbol manipulation, effective computability, execution of an algorithm (or rule-following), and calculation of a function. These are some of the informal, intuitive notions employed in logic and mathematics to characterize computation up to the 1930s. What changed in the 1930s is that some logicians began to explicate these informal notions in terms of a number of precise formalisms. The formalisms include Turing Machines, recursive functions, Markov algorithms, RAM machines, and many others. All these formalisms are extensionally equivalent, in the sense that any function computable by one of them is computable by any of them. They form the core of classical computability theory as well as theoretical computer science.[25]

The formalisms of computability theory include ways of specifying rules for manipulating strings of (abstract) letters from a finite alphabet (Turing Machine tables, recursive equations, programs for RAM machines, etc.). These are the type of rule invoked by the present account. In addition, some of the formalisms define a class of (abstract) computing systems that follow the rules (Turing Machines, RAM machines, etc.). But the formalisms are mathematical descriptions of abstract entities. Before they can be applied to explain—as opposed to model—the behaviour of concrete physical systems (such as brains and computers), we need an account of the relationship between the formalisms and the physical world.

The most prominent account of such a relationship, the semantic view of computation, was discussed in the previous section. According to the semantic view, only (some) entities that are essentially representational count as genuinely computational states. As we saw, however, being essentially representational is neither necessary nor sufficient for a system's behaviour to be explained computationally. A more promising idea, also discussed in the previous section, is that for a system to be explained computationally, it must be functionally organized. Being functionally organized is necessary but not sufficient. What is also needed is that the functional organization be of a specific kind—involving the relevant kind of processing of the relevant kind of discrete inputs and outputs. One of this section's contributions has been to sketch, in a general form, the kind of functional organization that is required for a system's behaviour to be explained in terms of rules defined over strings of letters—the strings and rules defined by the formalisms of computability theory.

[24]Newell and Simon [1976] define physical symbol systems in terms of the notions of 'designation' and 'interpretation', which may suggest that they hold a semantic view of computation. But their notions of designation and interpretation are defined in terms of functional relations between symbols and the internal processes that operate on them.

[25]The widely held thesis that these formalisms are adequate formal counterparts of the informal notions is known as the Church-Turing thesis. For an introduction to computability theory, see Davis et al. [1994]. For some of the relevant history and philosophy, see Piccinini [2003] and references therein.

Each particular computing system has its own specific mechanistic explanation, but all mechanistic explanations of computing systems include an alphabet out of which strings of digits can be constructed, a set of components that are functionally organized to manipulate strings, and a rule that specifies which output strings must be produced in a way that depends on the relevant input strings (and perhaps internal states). Digits can be labelled by letters, so that the rules defined within the formalisms of computability theory can be used to describe the processes of genuine computing mechanisms. In summary, in order to have a genuine computational explanation, we need to combine a certain mechanistic explanation of a system—in terms of strings of digits and components that manipulate strings—with a certain kind of description of the process by which the strings are manipulated—in terms of rules that apply to all input strings and depend on the inputs for their application.

We now have a robust notion of computation, which captures an important sense in which a system may be said to follow rules rather than merely being rule-governed. A computing system ought to follow the rule that specifies the relevant relation between its input and output strings. If it doesn't, a miscomputation occurs. So, this notion of computation captures the relevant sense in which computing systems may miscompute. Computing systems are functional systems, and as such they can malfunction. Since their function is to perform a certain computation, their malfunction results in a miscomputation. For instance, if the tape of a (physically implemented) Turing machine breaks, the machine may fail to complete its computation correctly. Miscomputation, then, is the specific kind of malfunction exhibited by computing systems.

A final observation pertains to the relation between computational modelling and computational explanation. Just as a computational model can embody the mechanistic explanation of any functional system, a computational model can embody the mechanistic explanation of a computing system. If this is done, one computing system is being used to model the behaviour of another computing system. Since the activity of the modelled system is a computation, in this case the model is computing representations of a computation. Since the modelled system can miscompute, and this miscomputation may be captured by the mechanistic explanation of the system that is embodied in the model, under appropriate conditions the model may be computing the miscomputation performed by the modelled system. The modelling of computing systems by other computing systems is a standard practice in the discipline of computer design, where it is used to test the properties (and crucially, the possible miscomputations) of a computing system before building concrete machines. The model itself, of course, may malfunction too. In such a case, the result is not a representation of a miscomputation by the modelled system, but rather a misrepresentation of (or a failure to represent) the behaviour of the modelled system.

It should go without saying that when we employ computational explanation in the present sense, pancomputationalism is false. To begin with, most systems, such as planetary systems and the weather, are not

functional systems—they are not subject to mechanistic explanation in terms of the functions performed by their components. Computational explanation does not apply to any of those, because computational explanation applies only to a subclass of functional systems. Within the class of functional systems, most systems, such as stomachs, do not manipulate entities of the relevant class, i.e., strings. A sequence of food bites is not a string of digits, if for nothing else, because the ordering of the bites makes little if any functionally significant difference to the process of digestion. Finally, even within the class of systems that do manipulate strings, many do so without following a rule of the appropriate kind. For instance, genuine random number generators do generate output strings from input strings, but there is no specifiable rule that tells which outputs ought to be produced from which inputs (otherwise, the outputs would not be genuinely random). Whether the brain is a computing system in the present sense remains open; it depends on whether it manipulates strings of digits in accordance with appropriate rules. But we know that in this sense, most systems are not computing systems.

VI. The Computational Theory of Mind without Pancomputationalism

There are many ways to describe a system computationally. Different computational descriptions carry different ontological implications about whether the system itself performs computations—whether it is a genuinely rule-governed system. And the assertion that everything is describable as a computing system takes a different significance depending on which kind of computational description is at stake.

Most computational descriptions are forms of computational modelling, in which the computations are performed by the model in order to generate representations of the modelled system at subsequent times. Computational models need not ascribe computations to the systems they model, and a fortiori they need not explain their behaviour by postulating computations internal to the systems they model. Even so, the claim that everything is describable by a computational model is only true in the most trivial sense. For computational models are approximate. If we care how good our approximations are—and if our models are to serve our scientific purposes, we'd better care—then whether something has a computational description depends on whether it can be computationally approximated to the degree of accuracy that is desired in a given case.

Some computational descriptions, however, are forms of mechanistic explanations. These mechanistic explanations individuate systems with special sorts of inputs and outputs called strings, and assign them the function of generating output strings from input strings in accordance with a general rule that applies to all strings and depends on the inputs (and perhaps internal states) for its application. These kinds of description ascribe computations to the systems themselves and are explanatory. They describe systems as rule-following, rather than merely rule-governed, because they explain the behaviour of the systems by appealing to the rule

they normally follow in generating their outputs from their inputs (and perhaps their internal states). This kind of computational description is suited to formulating genuinely explanatory theories of rule-following systems, as CTM is designed to do, and it applies only to very select systems, which manipulate special sorts of input and output in special sorts of way. It is far from true that everything is a computing system in this sense.

Given all this, different sources of evidence need to be used to support different versions of the claim that something is a computing system, and the version that is relevant to CTM turns out to be empirical in nature. This is an important consequence of the present analysis, because naturalistically inclined philosophers should prefer to settle CTM empirically rather than a priori. Whether something is a computing system properly so called turns out to depend on whether it has certain functional properties, so that determining whether the mind is a computing system is a matter of determining whether it has the relevant functional properties. This is consistent with the original formulation of CTM by McCulloch and others, who discussed it as a hypothesis about the functional properties of the mind [McCulloch and Pitts 1943; Wiener 1948; von Neumann 1958; Fodor 1975; Newell and Simon 1976; Pylyshyn 1984].

As to the thesis that everything is a computing system, it turns out to be motivated by a superficial understanding of the role of computation in modelling. Once the thesis is made more precise, it loses both plausibility and relevance to the philosophy of mind.

University of Missouri, St Louis

Received: October 2003
Revised: June 2004

References

Adams, F. 2003. The Informational Turn in Philosophy, *Minds and Machines* 13: 471–501.
Allen, C., M. Bekoff, and G., Lauder, eds. 1998. *Nature's Purposes: Analysis of Function and Design in Biology*, Cambridge MA: MIT Press.
Ariew, A., R. Cummins, and M. Perlman, eds. 2002. *Functions: New Essays in the Philosophy of Psychology and Biology*, New York: Oxford University Press.
Aspray, W. 1985. The Scientific Conceptualization of Information: A Survey, *Annals of the History of Computing* 7: 117–40.
Bechtel, W. and R. C. Richardson 1993. *Discovering Complexity: Decomposition and Localization as Scientific Research Strategies*, Princeton: Princeton University Press.
Buller, D. J., ed. 1999. *Function, Selection, and Design*, Albany: State University of New York Press.
Block, N. and J. A. Fodor 1972. What Psychological States Are Not, *Philosophical Review* 81: 159–81.
Block, N. 1980. Introduction: What is Functionalism?, in *Readings in Philosophy of Psychology I*, ed. N. Block, London: Methuen: 171–84.
Bontly, T. 1998. Individualism and the Nature of Syntactic States, *British Journal for the Philosophy of Science* 49: 557–74.
Chalmers, D. J. 1996a. *The Conscious Mind: In Search of a Fundamental Theory*, New York: Oxford University Press.
Chalmers, D. J. 1996b. Does a Rock Implement Every Finite-State Automaton?, *Synthese* 108: 310–33.
Christensen, W. D. and M. H. Bickhard 2002. The Process Dynamics of Normative Function, *Monist* 85: 3–28.
Churchland, P. M. and P. S. Churchland 1990. Could a Machine Think?, *Scientific American* 262: 26–31.
Churchland, P. S. and T. J. Sejnowski 1992. *The Computational Brain*, Cambridge MA: MIT Press.
Copeland, B. J. 1996. What is Computation?, *Synthese* 108: 224–359.
Copeland, B. J. 2000. Narrow Versus Wide Mechanism: Including a Re-Examination of Turing's Views on the Mind-Machine Issue, *Journal of Philosophy* 96: 5–32.
Copeland, B. J. 2002a. The Church-Turing Thesis, in *The Stanford Encyclopedia of Philosophy (Fall 2002 edn)*, ed. E. N. Zalta, URL=<http://plato.stanford.edu/archives/fall2002/entries/church-turing/>

Copeland, B. J. 2002b. Hypercomputation, *Minds and Machines* 12: 461–502.
Corcoran, J., W. Frank, and M. Maloney 1974. String Theory, *Journal of Symbolic Logic* 39: 625–37.
Cotogno, P. 2003. Hypercomputation and the Physical Church-Turing Thesis, *British Journal for the Philosophy of Science* 54: 181–223.
Craver, C. 2001. Role Functions, Mechanisms, and Hierarchy, *Philosophy of Science* 68: 53–74.
Craver, C. F. forthcoming. *Explaining the Brain: What a Science of the Mind/Brain Could Be*.
Cummins, R. 1975. Functional Analysis, *Journal of Philosophy* 72: 741–65.
Cummins, R. 1983. *The Nature of Psychological Explanation*, Cambridge MA, MIT Press.
Cummins, R. 2002. Neo-teleology, in *Functions: New Essays in the Philosophy of Psychology and Biology*, ed. A. Ariew, R. Cummins, and M. Perlman, New York: Oxford University Press: 157–72.
Davis, M., R. Sigal, and E. J. Weyuker 1994. *Computability, Complexity, and Languages*, Boston: Academic.
Dayan, P. and L. F. Abbott 2001. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, Cambridge MA: MIT Press.
Dennett, D. C. 1978. *Brainstorms*, Cambridge MA: MIT Press.
Dretske, F. I. 1986. Misrepresentation, in *Belief: Form, Content, and Function*, ed. R. Bogdan, New York: Oxford University Press: 17–36.
Dreyfus, H. L. 1972. *What Computers Can't Do*, New York: Harper & Row.
Egan, F. 1999. In Defence of Narrow Mindedness, *Mind and Language* 14: 177–94.
Egan, F. 2003. Naturalistic Inquiry: Where does Mental Representation Fit in?, in *Chomsky and His Critics*, ed. L. M. Antony and N. Hornstein, Malden MA: Blackwell: 89–104.
Eliasmith, C. 2003. Moving Beyond Metaphors: Understanding the Mind for What It Is, *Journal of Philosophy* 100: 493–520.
Fodor, J. A. 1968a. The Appeal to Tacit Knowledge in Psychological Explanation, *Journal of Philosophy* 65: 627–40.
Fodor, J. A. 1968b. *Psychological Explanation*, New York: Random House.
Fodor, J. A. 1975. *The Language of Thought*, Cambridge MA: Harvard University Press.
Fodor, J. A. 1998. *Concepts*, Oxford: Clarendon Press.
Gauker, C. 2003. *Words without Meaning*, Cambridge MA: MIT Press.
Glennan, S. S. 2002. Rethinking Mechanistic Explanation, *Philosophy of Science* 69: S342–53.
Haugeland, J. 1978. The Nature and Plausibility of Cognitivism, *Behavioral and Brain Sciences* 2: 215–60.
Hopfield, J. J. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proceedings of the National Academy of Sciences* 79: 2554–8.
Hughes, R. I. G. 1999. The Ising Model, Computer Simulation, and Universal Physics, in *Models as Mediators*, ed. M. S. Morgan and M. Morrison, Cambridge: Cambridge University Press: 97–145.
Humphreys, P. 2004. *Extending Ourselves: Computational Science, Empiricism, and Scientific Method*, New York: Oxford University Press.
Ludlow, P. 2003. Externalism, Logical Form, and Linguistic Intentions. *Epistemology of Language*, ed. A. Barber, New York, Oxford University Press: 399–414.
Lycan, W. 1987. *Consciousness*, Cambridge MA: MIT Press.
Machamer, P. K., L. Darden, and C. Craver 2000. Thinking About Mechanisms, *Philosophy of Science* 67: 1–25.
Marr, D. 1982. *Vision*, New York: Freeman.
McCulloch, W. S. 1965. *Embodiments of Mind*, Cambridge MA: MIT Press.
McCulloch, W. S. and W. H. Pitts 1943. A Logical Calculus of the Ideas Immanent in Nervous Nets, *Bulletin of Mathematical Biophysics* 7: 115–33.
Newell, A. 1980. Physical Symbol Systems, *Cognitive Science* 4: 135–83.
Newell, A. and H. A. Simon 1976. Computer Science as an Empirical Enquiry: Symbols and Search, *Communications of the ACM* 19: 113–26.
Peacocke, C. 1999. Computation as Involving Content: A Response to Egan, *Mind and Language* 14: 195–202.
Penrose, R. 1994. *Shadows of the Mind*, New York: Oxford University Press.
Piccinini, G. 2003. Alan Turing and the Mathematical Objection, *Minds and Machines* 13: 23–48.
Piccinini, G. 2004a. The First Computational Theory of Mind and Brain: A Close Look at McCulloch and Pitts's 'Logical Calculus of Ideas Immanent in Nervous Activity', *Synthese* 141: 175–215.
Piccinini, G. 2004b. Functionalism, Computationalism, and Mental Contents, *Canadian Journal of Philosophy* 34: 375–410.
Piccinini, G. 2004c. Functionalism, Computationalism, and Mental States, *Studies in the History and Philosophy of Science* 35: 811–33.
Piccinini, G. 2007. Computationalism, the Church-Turing Thesis, and the Church-Turing Fallacy, *Synthese* 154: 97–120.
Piccinini, G. forthcoming. Computation without Representation, *Philosophical Studies*.
Preston, B. 1998. Why is a Wing Like a Spoon? A Pluralist Theory of Function, *Journal of Philosophy* 95: 215–54.
Putnam, H. 1960. Minds and Machines, in *Dimensions of Mind: A Symposium*, ed. S. Hook, New York: Collier: 138–64.
Putnam, H. 1964. Robots: Machines or Artificially Created Life?, *Journal of Philosophy* 61: 668–91.
Putnam, H. 1999 (1967). The Nature of Mental States, in *Mind and Cognition: An Anthology* 2nd edn, ed. W. Lycan, Malden MA: Blackwell: 27–34.
Putnam, H. 1988. *Representation and Reality*, Cambridge MA: MIT Press.
Pylyshyn, Z. W. 1984. *Computation and Cognition*, Cambridge MA: MIT Press.
Rohrlich, F. 1990. Computer Simulation in the Physical Sciences, *PSA 1990*, 2: 507–18.
Rumelhart, D. E. and J. M. McClelland 1986. *Parallel Distributed Processing*, Cambridge MA: MIT Press.

Scheutz, M. 2001. Causal versus Computational Complexity, *Minds and Machines* 11: 534–66.
Schlosser, G. 1998. Self-re-Production and Functionality: A Systems-Theoretical Approach to Teleological Explanation, *Synthese* 116: 303–54.
Searle, J. R. 1980. Minds, Brains, and Programs, *Behavioral and Brain Sciences* 3: 417–57.
Searle, J. R. 1992. *The Rediscovery of the Mind*, Cambridge MA: MIT Press.
Shagrir, O. 2006. Why We View the Brain as a Computer, *Synthese* 153: 393–416.
Shapiro, L. A. 1994. Behavior, ISO Functionalism, and Psychology, *Studies in the History and Philosophy of Science* 25: 191–209.
Smith, B. C. 2002. The Foundations of Computing, in *Computationalism: New Directions*, ed. M. Scheutz, Cambridge MA: MIT Press: 23–58.
Sober, E. 1985. Panglossian Functionalism and the Philosophy of Mind, *Synthese* 64: 165–93.
Toffoli, T. 1984. Cellular Automata as an Alternative to (rather than an Approximation of) Differential Equations in Modeling Physics, *Physica* 10D: 117–27.
Vichniac, G. Y. 1984. Simulating Physics with Cellular Automata, *Physica*, 10D: 96–116.
von Neumann, J. 1951. The General and Logical Theory of Automata, in *Cerebral Mechanisms in Behavior*, ed. L. A. Jeffress, New York: Wiley: 1–41.
von Neumann, J. 1958. *The Computer and the Brain*, New Haven: Yale University Press.
Wiener, N. 1948. *Cybernetics or Control and Communication in the Animal and the Machine*, Cambridge MA: MIT Press.
Wolfram, S. 2002. *A New Kind of Science*, Champaign: Wolfram Media.