The Mind as Neural Software? Understanding Functionalism, Computationalism, and Computational Functionalism[1]

Gualtiero Piccinini

University of Missouri – St. Louis

This is a preprint of an article whose final and definitive form will be published in *Philosophy and Phenomenological Research*; *Philosophy and Phenomenological Research* is available online at: http://www.blackwellpublishing.com/.

Abstract. Defending or attacking either functionalism or computationalism requires clarity on what they amount to and what evidence counts for or against them. My goal here is not to evaluate their plausibility. My goal is to formulate them and their relationship clearly enough that we can determine which type of evidence is relevant to them. I aim to dispel some sources of confusion that surround functionalism and computationalism, recruit recent philosophical work on mechanisms and computation to shed light on them, and clarify how functionalism and computationalism may or may not legitimately come together.

[1] Thanks to those who commented on this paper at its various and multiple stages, especially Ken Aizawa, David Chalmers, Robert Cummins, Carl Craver, Chris Eliasmith, John Heil, Bill Lycan, Peter Machamer, Diego Marconi, Tom Polger, Oron Shagrir, and Larry Shapiro. Ancestors of this paper were presented at the 2004 Pacific APA and the 2007 SSPP. I thank the audiences and commentators—Matthias Scheutz and Charles Wallis at the APA, Whit Schonbein at the SSPP—for their feedback. This work was supported in part by a 2006 NEH Summer Seminar at Washington University in St. Louis and a University of Missouri Research Grant. The views expressed here do not necessarily reflect those of these institutions.

1 Introduction

Functionalism is forty years old, computationalism is over sixty, and philosophers often conjoin them. Yet their relationship remains obscure. With Jerry Fodor, I am struck by "the widespread failure to distinguish the computational program in psychology from the functionalist program in metaphysics" (Fodor 2000, 104). A recent paper by Paul Churchland (2005) epitomizes such a failure. Churchland argues that functionalism is false, because the brain is not a classical (i.e., more or less Turing-machine-like) computing system but a connectionist one. His argument presupposes that functionalism entails classical computationalism. But functionalism—properly understood—does *not* entail computationalism, whether classical or non-classical.

Assessing functionalism and computationalism requires clarity on what they amount to and what evidence counts for or against them. My goal here is not to evaluate their plausibility. My goal is to formulate them and their relationship clearly enough that we can determine which type of evidence is relevant to them. I aim to dispel some sources of confusion that surround functionalism and computationalism, recruit recent philosophical work on mechanisms and computation to shed light on them, and clarify how functionalism and computationalism may or may not legitimately come together.

I will frame the discussion in terms of functionalism, because functionalism is the metaphysical doctrine most closely associated (and conflated) with computationalism. But one upshot of this paper is that functionalism and computationalism need not go together. Functionalism may be combined with a non-computational theory of mind, and computationalism may be combined with a non-functionalist metaphysics. Once we understand how functionalism and computationalism mesh, we can generalize our picture

and see how metaphysical doctrines other than functionalism may be combined with computationalism as well as how theories other than computationalism may be combined with functionalism.

To a first approximation, functionalism is the view that the mind is the "functional organization" of the brain, or any other system that is functionally equivalent to the brain (cf. Putnam 1960, 149; 1967a, 200; 1967b, 32). Another formulation of functionalism is that mental states are "functional states" (Putnam 1967b, 30).[2] Putnam's main example of a description of functional organization is the machine table of a Turing machine. For him, a functional organization is a set of functional states with their functional relations, where a functional state is defined by its causal relations to inputs, outputs, and other functional states (under normal conditions). Thus, under Putnam's notion of functional organization, our two formulations of functionalism are equivalent.

Under the broader notion of functional organization that I will defend, there is more to functional organization than individual functional states and their relations. There are also aggregates of states, components bearing the states, functional properties of the components, and relations between the components and their properties. Since the first formulation of functionalism is more general than the second, I prefer the first, but nothing in what follows hinges on the difference between the two.

Stronger or weaker versions of functionalism may be formulated depending on how much of the mind is taken to be functional—how many mental states, or which

[2] In the cited references, Putnam writes "functional organization of organisms" or "functional organization of the human being" rather than "functional organization of the brain," as I wrote. Since much of the functional organization of human beings—or organisms, for that matter—is irrelevant to the mind, I replaced 'organism' and 'human being' with 'brain'. Even so, I doubt that the brain fixes the exact boundaries of the mind's realization. For present purposes, it will be convenient to understand 'brain', whenever appropriate, as referring to whatever aspects of the functional organization of organisms realize the mind according to functionalism.

aspects thereof, are functional. Are all mental states functional, or only some? Are all aspects of mental states functional, or only, say, their non-phenomenal aspects? How these questions are answered makes no difference here, because I'm not concerned with the plausibility and scope of functionalism. I'm concerned with what functionalism amounts to. One question I will address is, what is functional organization? In due course, I will examine different notions of functional organization and search for the pertinent one.

Computationalism, for present purposes, is the view that the functional organization of the brain (or any other functionally equivalent system) is computational, or that neural states are computational states. Again, stronger or weaker versions of computationalism may be formulated depending on how much of the functional organization of the brain is taken to be computational. But again, I will not assess the plausibility and scope of computationalism here.

Functionalism plus computationalism equals *computational functionalism*. In a well-known slogan, computational functionalism says that the mind is the software of the brain (or any functionally equivalent system; I will omit this qualification from now on). Taken at face value, this slogan draws an analogy between the mind and the software of ordinary, program-controlled computers. But the same slogan is often understood to suggest, more modestly, that the mind is the computational organization of the brain—or that mental states are computational states—without the implication that such a computational organization is that of a program-controlled computer. As we shall see, the ambiguity between the strong and the weak reading is one source of confusion in this area.

Computational functionalism has been popular among functionalists who are sympathetic to computationalist research programs in artificial intelligence, psychology, and neuroscience. It has also encountered ferocious resistance.[3] The present goal, however, is not to determine whether the mind is the software of the brain. It is to understand what this means and what counts as evidence for or against it.

An important caveat. The functionalism I am concerned with descends from some early writings of Hilary Putnam and Jerry Fodor (Putnam 1960, 1964, 1967a, 1967b; Fodor 1965, 1968a, 1968b). It is a metaphysics of mind—the main alternatives being dualism, behaviorism, and the type-identity theory. It accounts for minds as they are described by scientific theories and explanations. It is also known as psychofunctionalism (Block 1980).[4]

[3] The critical literature is vast. Representative examples include Block 1978, Putnam 1988, Searle 1992, and Lucas 1996.

[4] Thus, I am not directly concerned with functionalism about folk psychological theories (Lewis 1966, 1972, 1980; Armstrong 1970), analytical or conceptual truths about the mental (Shoemaker 2003b), or the *content* of mental states (e.g., Sellars 1954; Harman 1973, 1999; Block 1986).

I avoid formulating functionalism in terms of Ramsey sentences (Lewis 1972, Block 2007) because such formulations obscure the issues addressed here (cf. Gillett 2007). I will not address several other topics related to functionalism: to what extent functionalism is consistent with reductionism and the identity theory (e.g., Lewis 1969, Fodor 1975, 1997, Boyd 1980, Churchland and Churchland 1982, Lycan 1982, Enç 1983, Wilson 1984, 1993, Kim 1989, 1992, Pereboom and Kornblith 1991, Bickle 1998, Shagrir 1998, Sober 1999, Bechtel and Mundale 1999, Keeley 2000, Bechtel 2001, Prinz 2001, Pereboom 2002), whether functionalism is consistent with mental causation (Block 2003, Kim 2003, Rupert 2006), whether functionalism should be formulated in terms of roles or realizers, whether functionalism should be formulated in terms of higher level properties, higher order properties, or similarities between fundamental properties (Heil 2003, 2004), whether the mind extends into the environment (Harman 1999, Shapiro 1994, Adams and Aizawa 2001, Wilson 2004, Rupert 2004), and the correct metaphysics of realization (Kim 1998, Shapiro 2000, Shoemaker 2001, 2003a, Gillett 2002, 2003, Polger 2004, 2007, Wilson 2004).

2 The Analogy between Minds and Computers

At the origin of computational functionalism are analogies between some features of minds and some features of computers. Different analogies pull towards different versions of the view.

Putnam, the chief founder of computational functionalism, drew an analogy between the individuation conditions of mental states and those of Turing machine states (Putnam 1960, 1967a, 1967b).[5] Putnam noticed that the states of Turing machines are individuated in terms of the way they affect and are affected by other Turing machine states, inputs, and outputs. By the same token, he thought, mental states are individuated by the way they affect and are affected by other mental states, stimuli, and behavior. At first, Putnam did not conclude that mental states are Turing machine states, because—he said—the mind might not be a causally closed system (Putnam 1960). A bit later, though, Putnam reckoned that mental states can be characterized functionally, like those of Turing machines, though he added that the mind might be something "quite different and more complicated" than a Turing machine (Putnam 1967a). Finally, Putnam went all the way to computational functionalism: mental states are (probabilistic) Turing machine states (Putnam 1967b).

As Putnam's trajectory illustrates, the analogy between the individuation of mental states and that of Turing machine states does not entail computational functionalism. The latter conclusion was reached by Putnam some time after drawing his analogy, on independent grounds. What grounds?

[5] For a more detailed reconstruction and discussion of Putnam's functionalism and computational functionalism, see Piccinini 2004b and Shagrir 2005.

It's hard to know for sure. The relevant papers by Putnam contain references to the plausibility and success of computational models of mental phenomena, including Warren McCulloch and Walter Pitts's theory of the brain. In 1943, McCulloch and Pitts proposed a mathematical theory of neurons and their signals, which theory was widely interpreted to claim that, in essence, the brain is a Turing machine (without tape).[6] A few years later, John von Neumann interpreted McCulloch and Pitts's work as *proof* that "anything that can be exhaustively and unambiguously described, anything that can be exhaustively and unambiguously put into words, is ipso facto realizable by a suitable finite neural network" of the McCulloch and Pitts type (von Neumann 1951, 23).

It is now clear that von Neumann's statement isn't true, at least under its most natural interpretation. A fortiori, McCulloch and Pitts proved nothing of the sort. For one thing, the nervous system described by their theory is a simplified and idealized version of the real thing. More importantly, von Neumann's statement implicitly abuses the Church-Turing thesis. The Church-Turing thesis says that anything that is computable in an informal sense, which is intuitively familiar to mathematicians, is computable by Turing machines. From this, it doesn't follow that anything that can be exhaustively and unambiguously described is computable by Turing machines. Nor does it follow, as many would soon conclude, that everything can be simulated by Turing machines or that everything is computational. Alas, neither von Neumann nor his early readers were especially careful about these matters. After von Neumann, fallacious arguments from the Church-Turing thesis—sometimes in conjunction with McCulloch

[6] As McCulloch put it, "What we thought we were doing (and I think we succeeded pretty well) was treating the brain as a Turing machine" (quoted in von Neumann 1951, 33). For a detailed study of McCulloch and Pitts's theory, see Piccinini 2004a.

and Pitts's actual or purported results—to the conclusion that the mind is computational began to proliferate.[7]

Since McCulloch and Pitts's networks can be simulated by digital computers, von Neumann's (unwarranted) statement entails that anything that can be exhaustively and unambiguously described can be simulated by a digital computer. If you add to this a dose of faith in scientists' ability to describe phenomena—"exhaustively and unambiguously"—you obtain pancomputationalism: at a suitable level of description, everything is computational. Thus, pancomputationalism made its way into the literature. As Putnam put it, "everything is a Probabilistic Automaton [i.e., a kind of Turing machine] under some Description" (Putnam 1967b, 31). Together with Putnam's analogy between mental states and Turing machine states and the alleged plausibility of computational psychology, pancomputationalism is the most likely ground for Putnam's endorsement of computational functionalism.

For present purposes, the most important thing to notice is that the resulting version of computational functionalism is quite a weak thesis. This remains true if computational functionalism is disengaged from Putnam's appeal to Turing machine states in favor of the thesis that mental states are, more generally, computational states (Block and Fodor 1972). If mental states are computational simply because at some level, everything is computational, then computational functionalism tells us nothing specific about the mind. It is a trivial consequence of the purported general applicability of computational descriptions to the natural world. This version of computational functionalism does not tell us how the mind works or what is special about it. Such a

[7] Examples of such arguments may be found in Dennett 1978, Webb 1980, Nelson 1987, Chalmers 1996b, Simon 1996, and Baum 2004. For their refutation, see Copeland 2000 and Piccinini 2007c.

weak thesis stands in sharp contrast with others, which derive from different analogies between features of minds and features of computers.[8]

Both minds and many kinds of computing systems manipulate complex combinatorial structures. Minds produce natural language sentences and other complex sequences of actions. Computing systems manipulate complex strings of digits.[9] The structures and processes in question are complex in the sense that in the interesting cases, there are recursive rules describing the structure of the inputs and outputs as well as recursive rules describing the relationship between inputs and outputs. Furthermore, for any (universal) formalism for specifying computations (e.g., Turing machines) and any recursive function, there is a program describing a way to compute that function within that formalism. Finally, whether a program computes a function correctly (e.g., whether

[8] David Chalmers has pointed out to me that the weak thesis may be strengthened by arguing that while computation is insufficient for the instantiation of most properties, computation is sufficient for the instantiation of mental properties. Unlike most properties, mental properties might be such that they are instantiated "in virtue of the implementation of computations" (Chalmers, personal correspondence). This is a fair point, but it makes a difference only insofar as we have good evidence that computation is sufficient for mentation.

Chalmers defends a thesis of computational sufficiency along these lines as follows. He defines a notion of abstract causal organization, which involves "the patterns of interaction among the parts of the system, abstracted away from the make-up of individual parts and from the way the causal connections are implemented," and yet includes "a level fine enough to determine the causation of behavior" (Chalmers unpublished). He then argues that unlike most non-mental properties, all there is to mental properties is abstract causal organization, and abstract causal organization can be fully and explanatorily captured computationally. If Chalmers is right, then (the right kind of) computation is sufficient for mentation while being insufficient for most other properties.

Lacking space for a detailed discussion of Chalmers's argument, let me make the following brief comment. I don't see that Chalmers's argument establishes computational sufficiency for mental properties in a way that makes a difference for present purposes. Chalmers faces a dilemma. If abstract causal organization is truly fine grained enough to determine the causation of a system's behavior, then—contrary to Chalmers's intent—abstract causal organization will capture (the causal aspects of) *any* property whatsoever (including digestion, combustion, etc.). If, instead, abstract causal organization excludes enough information about a system to rule out at least certain properties (such as digestion and combustion), then—again, contrary to Chalmers's intent—there is no reason to accept that abstract causal organization will capture every aspect of mental properties. Either way, the specific connection between mentation and computation is not strengthened. Thus, Chalmers's argument does not affect our main discussion.

[9] I am using 'digit' to refer to the physical entities or states manipulated by computers, regardless of whether they represent numbers.

it computes square roots correctly) is an open question—it's "open to rational criticism" (Putnam 1960, 149), as it were.

Thanks in part to the complexity of their structure and the sensitivity of computing systems to such a structure, strings of digits may be systematically interpreted. So computers' activities are usually characterized by semantic descriptions. For example, we say that computers do arithmetic calculations, which is an activity individuated in terms of operations on numbers, which are possible referents of digits. This interpretability of the digits manipulated by computers has often been seen as part of the analogy between mental states and computational states, because mental states are also typically seen as endowed with semantic content. This, in turn, has contributed to the attractiveness of computational theories of mind.[10] But matters of mental content are controversial, and without agreement on mental content, the putative semantic analogy between mental states and computational states gives us no firm ground on which to explicate computational functionalism. In the next section, I will argue that the semantic properties of computing mechanisms make no difference for our purposes. Setting semantic properties aside, let's go back to the complexity of the structures being manipulated.

The analogy between the structures manipulated by minds and by computing mechanisms is stronger than the previous one: prima facie, most things cannot manipulate combinatorial structures of arbitrary recursive complexity according to arbitrary recursive rules. This stronger analogy is a likely source of some versions of

[10] According to Smith, "The *only* compelling reason to suppose that we (or minds or intelligence) might be computers stems from the fact that we, too, deal with representations, symbols, meanings, and the like" (1996, 11). Smith is exaggerating in calling this the *only* reason. But the semantic analogy between minds and computers does have a long and influential history. For a more detailed discussion, see Piccinini 2004c.

computationalism, and derivatively, of computational functionalism. But this analogy is still insufficient to underwrite the slogan "the mind is the software of the brain." The reason is that the analogy holds between minds and computing mechanisms in general, and many computing mechanisms (e.g., ordinary Turing machines) don't possess any software in the relevant sense. For the relevant notion of software, we need yet another analogy, which holds between certain mental capacities and certain capacities of program-controlled computers.

Program-controlled computers ("computers" from now on), unlike other computing systems, have an endless versatility in manipulating strings of digits. If they are universal, they embody in a single entity the universality of whole programming systems. Computers are versatile because they can store, manipulate, and execute programs.[11] To a first approximation, a program is a list of instructions for executing a task defined over strings of digits. An instruction is also a string of digits, which affects a computer in a special way. Most computers can execute many different (appropriately written) programs—typically, within certain limits of time and memory, they can execute *any* program. Because of this, computers can acquire any number of new capacities simply by acquiring and switching between programs. They can also refine their capacities by altering their programs. Just as minds can learn to execute a seemingly endless number of tasks, computers can execute any task, defined over strings of digits, for which they are given an appropriate program.

This special property of computers—their capacity to store and execute programs—gives rise to the special form of explanation that we employ for their behavior. How is my desktop computer letting me write this paper? By executing a

[11] For a detailed and systematic discussion of computers and their properties, see Piccinini 2008a.

word-processing program. How does it allow me to search the web? By executing an Internet browsing program. And so on for the myriad capacities of my computer. These are explanations by program execution:

> An *explanation by program execution* of a capacity *C* possessed by a system *S* is a program *P* for *C* such that *S* possesses *C* because *S* executes *P*.

Explanation by program execution applies only to systems, such as computers, that have the capacity to execute programs. Other relevant systems include certain automatic looms. Even though computers are not the only class of systems subject to explanation by program execution, computers have other interesting properties that they do not share with other program-controlled mechanisms. The main difference is that the processes generated by computer programs depend on the precise configuration of the input data (viz., the input strings of digits) for their application: to each string of input data there corresponds a different computation. Furthermore, typical computers have an internal memory in which they can store and manipulate their own data and programs, and they can compute any recursive function for as long as they have time and memory space.

These remarkable capacities of computers—to manipulate strings of digits and to store and execute programs—suggest a bold hypothesis. Perhaps brains are computers, and perhaps minds are nothing but the programs running on neural computers. If so, then we can explain the multiple capacities minds exhibit by postulating specific programs for those capacities. The versatility of minds would then be explained by assuming that brains have the same special power that computers have: the power to compute by

storing and executing programs.[12] This is the true source of the computational functionalist slogan: the mind is the software of the brain.

Compare this version of computational functionalism to the first one. Here we have a putative explanation of human behavior, based on an analogy with what explains computers' behavior. This version tells us how the mind works and what's special about it: the brain has the capacity of storing and executing different programs, and the brain's switching between programs explains its versatility. It is a strong thesis: of all the things we observe, only brains and computers exhibit such seemingly endless ability to switch between tasks and acquire new skills. Presumably, there are few if any other systems whose behavior is explained in terms of (this type of) program execution.

If we take this formulation of computational functionalism seriously, we ought to find an adequate explication of program execution. We ought to make explicit what differentiates systems that compute by executing programs from other kinds of system. For if minds are to be interestingly analogous to some aspect of computers, there must be something that minds and computers share and other systems lack—something that accounts for the versatility of minds and computers as well as the explanation of this versatility by program execution. Unfortunately, the received view of software implementation, which is behind the standard view of program execution, does not satisfy this condition of adequacy.

[12] An argument to this effect is in Fodor 1968b, which is one of the founding documents of computational functionalism and which Fodor 2000 singles out as conflating functionalism and computationalism. An argument along similar lines is in Newell 1990, 113ff. Other influential authors offered similar considerations. For the role played by program execution in Alan Turing's thinking about intelligence, see Turing 1950 and Piccinini 2003a. For the role played by program execution in von Neumann's thinking about brains, see von Neumann 1958 and Piccinini 2003b.

3 Troubles with Program Execution

Ever since Putnam (1967b) formulated computational functionalism, the received view of software implementation has been as follows. If there are two descriptions of a system, a physical description and a computational description, and if the computational description maps onto the physical description, then the system is a physical implementation of the computational description and the computational description is the system's software.[13]

The problem with this view is that it turns everything into a computer. As was mentioned in the previous section, everything can be given computational descriptions. For instance, some cosmologists study the evolution of galaxies using cellular automata. According to the received view of software implementation, this turns galaxies into hardware running the relevant cellular automata programs. If satisfying computational descriptions is sufficient for implementing them in the sense in which ordinary computers

[13] Here is a case in point:

> [A] programming language can be thought of as establishing a mapping of the physical states of a machine onto sentences of English such that the English sentence assigned to a given state expresses the instruction the machine is said to be executing when it is in that state (Fodor 1968b, 638).

Beginning in the 1970s, some authors attempted to go beyond the mapping view by imposing further constraints on implementation. Most prominently, Bill Lycan (1987) imposed a teleological constraint. Although this was a step in the right direction (more on this later), Lycan and others used 'software/hardware' and 'role/realizer' interchangeably. They offered no account specific to *software* implementation as opposed to role realization, so the conflation between functionalism and computationalism remained unaffected. When talking specifically about computation, philosophers continued to appeal to versions of the mapping view:

> [A] physical system is a computational system just in case there is an appropriate (revealing) *mapping between the system's physical states and the elements of the function computed* (Churchland and Sejnowski 1992, p. 62; emphasis added).

> [C]omputational theories construe cognitive processes as formal operations defined over symbol structures… Symbols are just functionally characterized objects whose individuation conditions are specified by *a realization function f_g which maps equivalence classes of physical features of a system to what we might call "symbolic" features*. Formal operations are just those physical operations that are differentially sensitive to the aspects of symbolic expressions that under the realization function f_g are specified as symbolic features. The mapping f_g allows a causal sequence of physical state transitions to be interpreted as a *computation* (Egan 1992, p. 446; first emphasis added).

execute their programs, then everything is a computer executing its computational descriptions. This is not only counterintuitive—it also trivializes the notion of computer as well as the analogy at the origin of computational functionalism. If the mind is the software of the brain in the sense in which certain cellular automata are the software of galaxies, then the analogy between minds and computers becomes an analogy between minds and everything else. As a consequence, the strong version of computational functionalism collapses into something very much like the weak one.

To make matters worse, the same system satisfies many computational descriptions. An indefinite number of cellular automata—using different state transition rules, different time steps, or cells that represent regions of different sizes—map onto the same physical dynamics. Furthermore, an indefinite number of formalisms different from cellular automata, such as Turing machines or C++ programs, can be used to compute the same functions computed by cellular automata. Given the received view of software implementation, it follows that galaxies are running all these programs at once.[14]

By the same token, brains implement all their indefinitely many computational descriptions. If the mind is the software of the brain, as computational functionalism maintains, then given the standard view of software implementation, we obtain either indeterminacy as to what the mind is, or that the mind is a collection of indefinitely many

[14] Matthias Scheutz has suggested an amendment to the standard explication of software implementation, according to which for a computational description to be considered relevant to software implementation, *all* its states and *all* its computational steps must map onto the system that is being described (Scheutz 2004). This proposal rules out many computational descriptions, such as computational models that employ C++ programs, as irrelevant to what software is being implemented by a system, and hence it improves on the standard view. But this proposal still leaves in place indefinitely many computational descriptions of any given system, so it doesn't solve the present problem.

pieces of software. This is not a promising metaphysics of mind, nor is it a way of explaining mental capacities in terms of program execution.[15]

The problem under discussion should not be confused with a superficially similar problem described by Putnam (1988) and Searle (1992). They argue that any physical system implements a large number of computations, or perhaps every computation, because a large number of (or perhaps all) state transitions between computational states can be freely mapped onto the state transitions between the physical states of a system. For example, I can take the state transitions my web browser is going through and map them onto the state transitions my desk is going through. As a result, my desk implements my web browser. I can establish the same mapping relation between a large number of (or perhaps all) computations and physical systems. From this, Putnam and Searle conclude that the notion of computation is observer-relative in a way that makes it useless to the philosophy of mind. Their argument is based on the received view of software implementation, and we might avoid its conclusion by abandoning the received view.

But even under the received view of software implementation, Putnam and Searle's problem is not very serious. As many authors have noted (e.g., Chrisley 1995, Copeland 1996, Chalmers 1996a, Bontly 1998, Scheutz 2001), the computational descriptions employed by Putnam and Searle are anomalous. In the case of kosher

[15] The thesis that everything has computational descriptions is more problematic than it may appear, in a way that adds a further difficulty for the standard view of software implementation. For ordinary computational descriptions are only approximate descriptions rather than exact ones, and hence a further undesirable consequence of the standard view is that programs can only approximate the behavior of the systems that are supposed to be implementing them. A full treatment of this further difficulty would take up more space than is available in this paper, so I will set it aside. For a detailed discussion of the relevance of approximation to pancomputationalism and the way pancomputationalism trivializes computationalism, see Piccinini 2007a.

computational descriptions—the kind normally used in scientific modeling[16]—the work of generating successive descriptions of a system's behavior is done by a computer running an appropriate program (e.g., a weather forecasting program), not by the mapping relation. In the sort of descriptions employed in Putnam and Searle's argument, instead, the descriptive work is done by the mapping relation.

In our example, my web browser does not generate successive descriptions of the state of my desk. If I want a genuine computational description of my desk, I have to identify states and state transitions of the desk, represent them by a computational description (thereby fixing the mapping relation between the computational description and the desk), and then use a computer to generate subsequent representations of the state of the desk, while the mapping relation stays fixed. So, Putnam and Searle's alleged problem is irrelevant to genuine computational descriptions. Still, the problem under discussion remains: everything can be given an indefinite number of *bona fide* computational descriptions.

To solve this problem, we must conclude that being described computationally is insufficient for implementing software, which is to say, we must go beyond the received view of software implementation. The same point is supported by independent considerations. The word 'software' was coined to characterize specific systems called 'computers'. Computers perform different activities from those performed by other systems such as drills or valves—let alone galaxies. We consider the invention of computers in the 1940s a major intellectual breakthrough—the discovery of something new. We have specific disciplines—computer science and computer engineering—that study the peculiar activities and characteristics of computers and only computers. For all

[16] For a systematic treatment of computational modeling in science, see Humphreys 2004.

these reasons, a good account of software implementation must draw a principled distinction between computers and other systems.

Philosophers have largely ignored this problem, and the charitable reader may legitimately wonder why. A first part of the answer is that philosophers interested in computationalism have devoted most of their attention to explaining mental phenomena, leaving computation *per se* largely unanalyzed.

A second part of the answer is that computationalist philosophers typically endorse the semantic view of computation, according to which computational states are individuated, at least in part, by their content (for a recent example, see Shagrir 2001, 2006). The semantic view appears to offer protection to the received view of software implementation, because independently of the semantic view, it is plausible that only some things, such as mental states, are individuated by their content. If computational states are individuated by their content and content is present only in few things, then explanation by program execution will apply at most to things that have content, and the trivialization of the notion of software is thereby avoided. Unfortunately, the protection offered by the semantic view is illusory. Here, there is no room for the in-depth treatment the semantic view of computation deserves. I have given such a treatment elsewhere (Piccinini 2004c, 2008b); I will only reiterate its results.

First, even the conjunction of the received view of software implementation and the semantic view of computation does not capture the notion of program execution that applies to ordinary computers. Computers (and some automatic looms, for that matter) execute programs whether or not the digits they manipulate have content, and there are mechanisms that perform computations defined over interpreted strings of digits just like

those manipulated by computers but do so without executing programs (e.g., many calculators). Second, there are computationalists who maintain that content plays no explanatory or individuative role in a computational theory of mind (Stich 1983, Egan 1992, 2003). Conjoining computationalism with the semantic view of computation begs the question of whether computational states are individuated by their content. Finally, and most seriously, the semantic view of computational states is incorrect, because computability theorists and computer designers—i.e., those to whom we should defer in individuating computational states—individuate computational states without appealing to their semantic properties. For these reasons, the semantic view of computation needs to be rejected, and cannot restore to health the received view of software implementation.

To a first approximation, the distinction between computers and other systems can be drawn in terms of program execution, where program execution is understood informally as a special kind of activity pertaining to special mechanisms (cf. Fodor 1968b, 1975; Pylyshyn 1984). Computers are among the few systems whose behavior we normally explain by invoking the programs they execute.[17] When we do so, we explain each activity of a computer by appealing to the unique program being executed. A program may be described in many different ways: instructions, subroutines, whole program in machine language, assembly language, or higher level programming language. But modulo the compositional and functional relations between programs and their components at different levels of description, a computer runs one and only one program at any given time. An expert can actually retrieve the unique program run by a computer and write it down, instruction by instruction.

[17] In the relevant sense of "program execution". Another notion is that of developmental "program," which is employed in biology. That is an independent notion of program, which would require a separate account.

True, modern computers can run more than one program "at once," but this has nothing to do with applying different computational descriptions to them at the same time. It has to do with computers' capacity to devote some time to running one program, quickly switch to another program, quickly switch back, and so forth, creating the impression that they are running several programs at the same time. (Some so-called supercomputers *can* execute many programs in parallel. This is because they have many different processors, i.e., program-executing components. Each processor executes one and only one program at any given time.)[18] All of this is in need of non-circular explication. A good account of software implementation must say why computers execute programs while most other systems don't, and hence what minds need to be like in order to be the putative software of brains. To prepare for that, it's time to clarify the relationship between functionalism and computationalism.

4 Mechanistic Functionalism

According to functionalism, the mind is the functional organization of the brain. According to computationalism, the functional organization of the brain is computational. These theses are prima facie logically independent—it should be possible to accept one while rejecting the other. But according to one construal, functional organizations are specified by computational descriptions connecting a system's inputs, internal states, and outputs (Putnam 1967b, Block and Fodor 1972). Under this construal, functional organizations are *ipso facto* computational, and hence functionalism entails

[18] It's also true that indefinitely many programs can produce the same behavior. But in any given computer processor, at any given time, there is a fact of the matter as to which program is generating the behavior in question: it's the one that the processor is executing.

computationalism. This consequence makes it impossible to reject computationalism without also rejecting functionalism, which may explain why attempts at refuting functionalism often address explicitly only its computational variety (e.g., Block 1978, Churchland 2005). The same consequence has led to Fodor's recent admission that he and others conflated functionalism and computationalism (2000, 104).

To avoid conflating functionalism and computationalism, we need a notion of functional organization that doesn't beg the question of computationalism. The broadest notion of functional organization is the purely causal one, according to which functional organization includes all causal relations between a system's internal states, inputs, and outputs. Given this notion, functionalism amounts to the thesis that the mind is the causal organization of the brain, or that mental states are individuated by their causal properties. Indeed, this is how functionalism is often formulated. The good news is, this version of functionalism is not obviously committed to computationalism, because prima facie, causal properties are not ipso facto computational. The bad news is, this version of functionalism is too weak to underwrite a theory of mind.

The causal notion of functional organization applies to all systems with inputs, outputs, and internal states. A liberal notion of input and output generates an especially broad causal notion of functional organization, which applies to all physical systems. For instance, every physical system may be said to take its state at time t_0 as input, go through a series of internal states between t_0 and t_n, and yield its state at t_n as output. A more restrictive notion of input and output generates more interesting notions of functional organization. For instance, opaque bodies may be said to take light of all wavelengths as input and yield light of only some wavelengths plus thermal radiation as output. Still, the

purely causal notion of functional organization is too vague and broad to do useful work in the philosophy of mind (and computation, for that matter). How should the notions of input and output be applied? Which of the many causal properties of a system are relevant to explaining its capacities? Does this purely causal version of functionalism entail computationalism? To answer these questions, we need to restrict our attention to the causal properties of organisms and artifacts that are relevant to explaining their specific capacities.

To fulfill this purpose, we turn to the notion of functional analysis. Functional analysis was introduced in modern philosophy of mind by Fodor (1965, 1968a). He used examples like the camshaft, whose function is to lift an engine's valve so as to let fuel into the piston. The camshaft has many causal properties, but only some of them, such as its capacity to lift valves, are functionally relevant—relevant to explaining an engine's capacity to generate motive power. Fodor argued that psychological theories are functional analyses, like our analysis of the engine's capacity in terms of the functions of its components.

When Fodor defined psychological functional analysis in general, however, he departed from his examples and assimilated psychological functional analyses to computational descriptions.[19] Several other authors developed a similar notion of functional analysis, retaining Fodor's assimilation of functional analyses to computational descriptions (Cummins 1975, 1983, 2000, Dennett 1978, Haugeland 1978, Block 1995). If functional organizations are specified by functional analyses and functional analyses are computational descriptions, then functional organizations are ipso

[19] Cf.: "the paradigmatic psychological theory is a list of instructions for producing behavior" (Fodor 1968b, 630). For a more extended discussion, see Piccinini 2004b.

facto computational. The mongrel of functional analysis and computational description is another source of the conflation between functionalism and computationalism.

To avoid this conflation, we need a notion of functional organization that has the relevant explanatory power—like Fodor et al.'s—but does not commit us to the view that every functionally organized system is computational. The recent revival of mechanisms in the philosophy of science offers us what we need. Not only do mechanisms satisfy our need; a formulation of functionalism in terms of mechanisms is also independently motivated. For an important lesson of recent philosophy of science is that (the relevant kind of) explanation in the special sciences, such as psychology and neuroscience, takes a mechanistic form[20]:

> A *mechanism M* with capacities ***C*** is a set of spatiotemporal components $A_1, \ldots, A_n$, their functions ***F***, and ***F***'s relevant causal and spatiotemporal relations ***R***, such that *M* possesses ***C*** because (i) *M* contains $A_1, \ldots, A_n$, (ii) $A_1, \ldots, A_n$ have functions ***F*** organized in way ***R,*** and (iii) ***F***, when organized in way ***R***, constitute ***C***.

A mechanism in the present sense exhibits its capacities thanks to its components, their functions, and their organization. Biologists ascribe functions to types of biological traits (e.g., the digestive function of stomachs) and engineers ascribe them to types of artifacts and their components (e.g., the cooling function of refrigerators). The functions ascribed to traits and artifacts are distinct from their accidental effects (e.g., making noise or breaking under pressure), and hence are only a subset of their causal powers. As a

[20] E.g., see Bechtel and Richardson 1993, Machamer, Darden and Craver 2000, Bechtel 2001, 2006, 2007, Craver 2001, 2005, 2006, 2007, Glennan 2002, 2005, Thagard 2003, Machamer 2004, Tabery 2004, Bogen 2005, Bechtel and Abrahamsen 2005, Darden 2006.

consequence, tokens of organs and artifacts that do not perform their functions may be said to malfunction or be defective.

As I will use the term, a mechanism's functional organization includes the states and activities of components, the spatial relations between components, the temporal relations between the components' activities, and the specific ways the components' activities affect one another. For example, the heart *pumps* blood *into* the arteries. This simple mechanistic description can begin to be unpacked as follows: (i) the mechanism includes a heart (component), arteries (components), and blood (input/output), (ii) the heart pumps the blood (activity of the heart), (iii) the heart is attached to the arteries in a certain way (spatial relation), (iv) the blood enters the arteries after it leaves the heart (temporal relation), (v) the heart's pumping causes the blood to enter the arteries (active relation). The relevant spatial relations between components may continue to hold even when the mechanism is not functioning. Not so for most temporal and active relations. Much more could be said about functional organization. The important point is that the functional organization of a mechanism is a necessary condition for the mechanism to do what it does.

Different notions of mechanism may be generated by employing different notions of function.[21] Drawing from William Wimsatt's helpful taxonomy, we find three especially pertinent notions (Wimsatt 1972, 4-5). *Perspectival* functions are causal powers that are relevant according to a view or perspective of what the system is doing. *Evaluative* functions are causal powers that contribute to a system's proper functioning.

[21] Polger 2004 follows a similar strategy in distinguishing different versions of functionalism based on which notion of function they employ. Unfortunately, in doing so he draws a false contrast between functions in the present sense and mathematical functions (cf. also Polger 2007, p. 252, for the same false contrast).

Teleological functions are causal powers that contribute to fulfilling the goal(s) of the system or its users.

These three notions are related. Fulfilling goals is one way of functioning properly, especially if proper functioning is defined as fulfilling one's goals, though in a more general sense, something may function properly without fulfilling its goals. Functioning properly may be all that is needed to fulfill one's goals, especially if one's goal is to function properly, though something may fulfill its goals without functioning properly. So evaluative and teleological functions may or may not go together. Furthermore, goals and standards of proper functioning define perspectives that we may take towards a system. Thus, as Wimsatt points out, evaluative and teleological functions are special cases of perspectival functions. But the notion of perspectival function is broader than the others: there are perspectives towards a system that have nothing to do with proper functioning or goals.

The above notions of function (as well as goals, proper functioning, and perspectives) need naturalistic explications. There is no shortage of literature devoted to that, and I cannot hope to resolve the debate on functions here.[22] For present purposes, I'll limit myself to the following caveats.

First, different authors offer slightly different accounts of mechanisms, and not all of them employ the word 'function'. But those differences are irrelevant here. All accounts of mechanisms may be subsumed under the above template by using the broad notion of perspectival function.

[22] The main competing accounts of functions in biology and engineering may be found in Allen, Bekoff, and Lauder 1998; Preston 1998; Schlosser 1998; Buller 1999; Ariew, Cummins, and Perlman 2002, Christensen and Bickhard 2002. Other contributions include Perlman 2004, Hourkes and Vermaas 2004, Cameron 2004, Johansson 2004, Schroeder 2004, Vermaas and Houkes 2006, Scheele 2006, Franssen 2006, Vermaas 2006, Houkes 2006, Houkes and Meijers 2006, Kroes 2006.

Second, there may be several legitimate notions of mechanisms, corresponding to different notions of function. Any notion of function that aspires to be relevant here, however, must be naturalistic. Which of the more precise notions of mechanism is most adequate to account for the explanatory practices that are relevant to the science and metaphysics of mind is a question that need not be resolved here.

Third, any further explication of the notion of function or mechanism cannot rely on the notion of computation in such a way as to turn all mechanisms into computing mechanisms, on pain of begging the question of computationalism again. Fortunately, computation plays no such role in current explications of these notions. As a result, appealing to mechanisms does not beg the question of computationalism.

This shows that we need not fasten together functions and computations the way Fodor and his followers did. When we appeal to the function of camshafts to explain the capacities of engines, our function ascription is part of a mechanistic explanation of the engine's capacities in terms of its components, their functions, and their organization. We do not appeal to programs executed by engines, nor do we attribute any computation to engines. In fact, most people would consider engines good examples of systems that do *not* work by executing programs (or more generally, by performing computations). The same point applies to the vast majority of mechanisms, with the notable exception of computers and other computing mechanisms (including, perhaps, brains).

Fourth and finally, we should not confuse the teleological notion of function with the etiological account of teleology. The etiological account of teleology in terms of evolutionary history is perhaps the most popular one, but it might not suit our present

purposes.[23] What matters here is that teleological functions ground a robust notion of functional organization without relying on the notion of computation. The question of how teleological functions ought to be explicated is surely important, but I can remain neutral about it.[24]

The systems studied by most special sciences—including neuroscience, psychology, and computer science—are mechanisms. Investigators in these disciplines analyze systems (e.g., trees) by breaking them down into component parts (e.g., roots) and discovering (or in engineering, designing) the functions of those parts (e.g., supporting the tree and absorbing water from the soil). Neuroscientists and psychologists elaborate their theories in the same way: they partition the brain or mind into components (e.g., the suprachiasmatic nuclei or episodic memory) and they ascribe them functions (respectively, regulating circadian rhythms and storing records of events). Mutatis mutandis, computer scientists do the same thing: they partition a computer into components (e.g., the memory and the processor) and ascribe them functions (respectively, storing data as well as instructions and executing instructions on the data).

Since mechanisms give us the notion of functional organization that is relevant to understanding theories in psychology, neuroscience, and computer science, we should adopt this notion of functional organization in our formulation of functionalism. We can

[23] Why not? First, the evolutionary account of functions does not immediately apply to artifacts, such as computers, which are not the product of evolution by natural selection. Because of this, it's unclear whether and how computational functionalism, which is based on an analogy between features of minds and features of computers, can be formulated in terms of a notion of function that relies on evolution. (This is not to say that there can't be a broader notion of selection that applies to both organisms and artifacts; cf. Wright 1973, Preston 2003.) Second, and more importantly, the evolutionary account of functions grounds any resulting theory of mind on the notions and practices of evolutionary biology rather than the empirical disciplines relevant to explaining the capacities of minds and computers—namely, psychology, neuroscience, and perhaps computer science.

[24] For example, non-etiological accounts of teleology are given by Schlosser 1998, Boorse 2002, and Wimsatt 2002.

now give a novel and improved formulation of functionalism, which does justice to the original motivations of functionalism without begging the question of computationalism. Functionalism about a system *S* should be construed as the thesis that *S* is the functional organization of the mechanism that exhibits *S*'s capacities. We can now explicate the claim that the mind is the functional organization of the brain: the brain is the relevant mechanism, and the mind is its functional organization. This *mechanistic functionalism* preserves functionalism's insight while doing justice to the relevant scientific practices.

Under this mechanistic version of functionalism, a system is individuated by its component parts, their functions, and their relevant causal and spatiotemporal relations. The functional states of the system are individuated by their role within the mechanistic explanation of the system. The states of the system are not only individuated by their relevant causal relations to other states, inputs, and outputs, but also by the component to which they belong and the function performed by that component when it is in that state. This applies to all mechanisms, including computing mechanisms. For example, pace Putnam, ordinary Turing machine states are individuated not only as having the function of generating certain outputs and other internal states on the basis of certain inputs and states, but also as being states *of* the active device (as opposed to the tape), which is a component of the Turing machine and has the functions of moving along the tape, reading the tape, and writing on it.[25]

[25] Carl Gillett (2007) has independently developed a proposal similar to what I'm calling 'mechanistic functionalism', with which he addresses other aspects of functionalism.

When mechanistic functionalism is further specified by employing teleological functions, the resulting doctrine is a close relative of teleological functionalism (Lycan 1981, 1987, Wilkes 1982, Millikan 1984, Sober 1990, Shapiro 1994, Rupert 2006). According to teleological functionalism, the mind is the teleological organization of the brain, or mental states are individuated by their teleological function. A teleological version of mechanistic functionalism adds to traditional teleological functionalism a mechanistic framework within which to specify the functional organization of the brain. Furthermore, the following two caveats apply. First, teleological functionalism is often offered as an alternative to

Mechanistic functionalism has a further great advantage, which is especially relevant to the concerns of this paper: it is based on a notion of mechanism that offers us the materials for explicating the notion of program execution, and more generally, computation.

5 Mechanisms, Computation, and Program Execution

A mechanism may or may not perform computations, and a mechanism that performs computations—a computing mechanism—may or may not do so by executing programs. To illustrate the latter distinction, consider Turing machines. Turing machines are made out of a tape of unbounded length, an active device that can take a finite number of states, letters from a finite alphabet, and functional relations (specified by a machine table) between tape, active device, states, and letters. Of course, Turing machines are usually thought of as abstract, in the same sense in which mathematically-defined triangles and circles are abstract. Like triangles and circles, Turing machines can be physically implemented. Physically implemented Turing machines and other concrete computing mechanisms operate on concrete counterparts of strings of letters, which I call 'strings of digits'. Whether abstract or concrete, Turing machines are mechanisms, subject to mechanistic explanation no more and no less than other mechanisms.[26]

---

computational functionalism (e.g., Lycan 1981, Millikan 1984, Sober 1990). But I will soon argue that mechanisms are the most adequate framework within which to explicate computation. As a consequence, computational functionalism turns out to be a special version of mechanistic functionalism. Second, many supporters of teleological functionalism endorse an etiological account of teleology. But as I already pointed out, this may not be the most suitable account of teleology for present purposes.

[26] The distinction between the abstract and the concrete easily leads to confusion on these matters. Here I have no room for a full treatment of all the relevant issues, so the following brief points will have to suffice.

Thomas Polger argues that abstract computations, being abstract, are not causally individuated (2007, 239-244). (He also expresses some second thoughts; cf. fn. 18.) Polger slides between talk of

Some Turing machines compute only one function. Other Turing machines, called 'universal', can compute any computable functions. This difference in computation power has a mechanistic explanation. Universal Turing machines, unlike non-universal ones, treat some digits on their tape as programs; they manipulate their data by appropriately responding to the programs. Because of this, universal Turing machines—unlike non-universal ones—may be said to *execute* the programs written on their tape. The behavior of all Turing machines is explained by the computations they perform on their data, but only the behavior of universal Turing machines is explained by the execution of programs. Besides Turing machines, many other mechanisms compute: finite state automata, pushdown automata, RAM machines, etc. Of these, some compute by executing programs and some don't. Of those that do, some are universal and some aren't.

Like Turing machines, the capacities of most biological systems and artifacts are mechanistically explained in terms of their components and functions (Bechtel and Richardson 1993, Craver and Darden 2001). Unlike Turing machines, the capacities of

---

abstract functions, computations, machines, states, algorithms, and programs, as if the same considerations applied to all. I take issue with that. Abstract functions are not realized in the same sense in which machines are. Functions are *computed* by machines—they are relations holding between the inputs and outputs of machines. Machines compute functions by *following* algorithms or programs. As we shall see, some special machines not only follow, but *execute* programs. Algorithms may be seen as sequences of statements or as relations holding between machine states and actions; programs may be seen as sequences of strings, sequences of statements, or relations holding between machine states and actions. Abstract functions are not causally individuated, but machines as well as their states and computations are—at least insofar as they are physically realizable. Algorithms and programs may or may not be, depending on how they are conceived of.

On a different note, it is common to see references to different levels of description of computing mechanisms, some of which are said to be more abstract than others (e.g., Newell 1980, Marr 1982). In this sense, a description is more or less abstract depending on whether it includes less or more details about a system. In this paper, I am not questioning the distinction between more abstract and more concrete levels of description and I am not focusing on the "implementation" or "physical" level at the expense of more "abstract" computational levels. Rather, I am offering a new way of understanding computational levels, in light of the fact that insofar as levels of description are relevant to the explanation of a system's capacities, they are all describing aspects of a mechanism—they are all part of a complete mechanistic explanation of the system, regardless of how abstract they are. For a detailed account of mechanistic levels, see Craver 2007.

most biological systems are not explained by appealing to putative computations they perform, let alone programs that they execute (except, of course, in the case of brains and other putative computing mechanisms).

So, explaining a capacity by program execution is not the same as explaining it computationally, which is not the same as explaining it mechanistically. Rather, explaining a (computational) capacity by program execution is a special *kind* of computational explanation, which is a special *kind* of mechanistic explanation. Computing mechanisms are subject to explanation by program execution because of their peculiar mechanistic properties. More specifically, computers are subject to computational explanation because of their peculiar mechanistic properties, some of which (though not all) they share with other computing mechanisms.

The rest of this section is devoted to explicating the above distinctions. First, I will identify the subclass of mechanisms that perform computations and whose (relevant) capacities are explained by the computations they perform.[27] Second, I will identify the subclass of computing mechanisms that execute programs and whose (relevant) activities are explained by the programs they execute. Once we have an account of these distinctions, we will have the resources to explicate computational functionalism.

Most mechanisms are partially individuated by their capacities. For instance, stomachs are things whose function is to digest food, and refrigerators are things whose function is to lower the temperature of certain regions of space. Capacities, in turn, may be analyzed in terms of inputs received from the environment and outputs delivered to the environment. Stomachs take undigested food as input and yield digested food as output; refrigerators take their inside at a certain temperature as input and deliver the same region

[27] For a more detailed formulation and defense of this account, see Piccinini 2007d.

at a lower temperature as output. Inputs and outputs may be taxonomized in many ways, which are relevant to the capacities to be explained. In our examples, foods and temperatures are taxonomized, respectively, in terms of whether and how they can be processed by stomachs and refrigerators in the relevant ways. Being a specific kind of mechanism, computing mechanisms are individuated by inputs and outputs of a specific kind and by a specific way of processing those inputs and outputs.

The inputs and outputs that are relevant to computing mechanisms are what computability theorists call strings of letters, or symbols.[28] A string of digits, as I'm using the term, is a concrete counterpart of a string of letters. What does it take for a concrete entity to be a string of digits? I will now sketch an answer in mechanistic terms. A digit is a state of a particular that belongs to one and only one of a finite number of relevant types. The digits' types are unambiguously distinguishable (and hence individuated) by the effects they have on the mechanism that manipulates them. That is, every digit of the same type affects a mechanism in the same way relative to generating the mechanism's output, and each type of digit affects the mechanism in a different way relative to generating the mechanism's output.

In other words, ceteris paribus, if Digit_1 and Digit_2 are of the same type, then substituting Digit_1 for Digit_2 results in the exact same computation (type) with the same output string (type), whereas if Digit_1 and Digit_2 are of different types, then substituting Digit_1 for Digit_2 results in a different computation, which may generate a different output string.[29] This property of digits differentiates them from many other classes of particulars, such as temperatures and bites of food, which belong to indefinitely many

[28] For the mathematical theory of strings, see Corcoran, Frank, and Maloney 1974.
[29] It is possible for two different computations to generate the same output from the same input. This simply shows that computations are individuated more finely than input-output mappings.

relevant types. (There is no well-defined functional classification of temperatures or foods such that every temperature or bite of food belongs to one among a finite number of relevant types).

A string is a list of permutable digits individuated by the digits' types, their number, and their order within the string. Every finite string has a first and a last digit member, and each digit that belongs in a string (except for the last member) has a unique successor. A digit within a string can be substituted by another digit without affecting the other digits' types, number, or position within the string. In particular, when an input string is processed by a mechanism, ceteris paribus, the digits' types, their number, and their order within the string make a difference to what output string is generated.

The fact that digits are organized into strings further differentiates strings of digits from the inputs and outputs of other functionally analyzable systems. Neither temperatures nor bites of food are organized into strings in the relevant sense. The comparison is unfair, because neither bites of food nor temperatures are digits to begin with. But let us suppose, for the sake of the argument, that we could find a way to unambiguously taxonomize bites of food into finitely many (functionally relevant) types. For instance, we could taxonomize bites of food into protein bites, fat bites, etc. If such a taxonomy were viable, it would turn bites of food into digits. Still, sequences of bites of food would not constitute *strings* of digits, because digestion—unlike computation—is not precisely sensitive to the order in which an organism bites its food.

Among systems that manipulate strings of digits, some do so in a special way: under normal conditions, they produce output strings of digits from input strings of digits in accordance with a general rule, which applies to all relevant strings and depends on the

inputs (and perhaps the internal states) for its application.[30] The rule in question specifies the function *computed* by the system. Some systems manipulate strings without performing computations over them. For instance, a genuine random number generator yields strings of digits as outputs, but not on the basis of a general rule defined over strings. (If it did, its output would not be genuinely random.) Systems that manipulate strings of digits in accordance with the relevant kind of rule deserve to be called computing mechanisms.

The activities of computing mechanisms are explained by the computations they perform. For example, if you press the buttons marked '21', ':', '7', and '=', of a (well-functioning) calculator, after a short delay it will display '3'. The explanation of this behavior includes the facts that 3 is 21 divided by 7, '21' represents 21, ':' represents division, '7' represents 7, '=' represents equality, and '3' represents 3. But most crucially, the explanation involves the fact that under those conditions, the calculator performs a specific calculation: to divide its first input datum by the second. The capacity to calculate is explained, in turn, by an appropriate mechanistic explanation. Calculators have input devices, processing units, and output devices. The function of the input devices is to deliver input data and commands from the environment to the processing units, the function of the processing units is to perform the relevant operations on the data, and the function of the output devices is to deliver the results to the environment. By iterating this explanatory strategy, we can explain the capacities of a calculator's components in terms of its components' functions and their organization.

[30] Which strings are relevant? All the strings from the relevant alphabet. For each computing mechanism, there is a relevant finite alphabet. Notice that the rule need not define an output for all input strings (and perhaps internal states) from the relevant alphabet. If some outputs are left undefined, then under those conditions the mechanism should produce no output strings of the relevant type.

Until now, I've sketched an account of computing mechanisms in general: computing mechanisms are mechanisms whose function is manipulating strings of digits according to appropriate rules; their behaviors are explained by the computations they perform. There remains to explicate the more interesting notion of a (program-controlled) computer—a mechanism that computes by executing programs.

Computers have special processing units, usually called processors. Processors are capable of performing a finite number of primitive operations on input strings (of fixed length) called 'data'. Which operation a processor performs on its data is determined by further strings of digits, called 'instructions'. Different instructions cause different operations to be performed by a processor. The performance of the relevant operation in response to an instruction is what constitutes the execution of that instruction. A list of instructions constitutes a program. The execution of a program's instructions in the relevant order constitutes the execution of the program. So, by executing a program's instructions in the relevant order, a computer processor executes the program. This is a brief mechanistic explanation of (program-controlled) computers and their capacity to execute programs in terms of their components, functions, and organization. The capacity of a processor to execute instructions can be further explained by a mechanistic explanation of the processor in terms of its components, their functions, and their organization.[31]

Only computing mechanisms of a specific kind, namely computers, have processors capable of executing programs (and memories for storing programs, data, and results). This is why only the capacities of computers, as opposed to the capacities of other computing mechanisms—let alone mechanisms that do not perform

[31] Cf. any standard textbook on computer organization and design, such as Patterson and Hennessy 1998.

computations—are explained by program execution. Computational explanation by program execution says that there are strings of digits whose function is to determine a sequence of operations to be performed by a processor on its data.

In other words, program execution requires that some states of some components of the computer *function* as a program; in an explanation by program execution, 'program' is used as a function term. The way a program determines what the computer does is cashed out in terms of the computer's mechanistic properties. A program-controlled computer is a very special kind of computing mechanism, which has the capacity to execute programs. This is why the appeal to program execution is explanatory for computers—because it postulates programs and processors *inside* computers.

As a consequence, when the behavior of ordinary computers is explained by program execution, the program is not just a description. The program is also a (stable state of a) *physical component* of the computer, whose function is to generate the relevant capacity of the computer. Programs are physically present within computers, where they have a function to perform. Somehow, this simple and straightforward point seems to have been almost entirely missed in the philosophical literature.[32]

[32] For an exception, see Moor 1978, 215. Robert Cummins, one of the few people to discuss this issue explicitly, maintained that "programs aren't causes but abstract objects or play-by-play accounts" (Cummins 1983, p. 34; see also Cummins 1977). Cummins's weaker notion of program execution is quite popular among philosophers, and is yet another side of the fuzziness surrounding functionalism and computationalism. This is because the weaker notion is not the one used in computer science and does not underwrite the strong analogy between mental capacities and computers' capacities that is behind the slogan "the mind is the software of the brain," and yet the weaker notion is often used in explicating computationalism or even functionalism. The main reason for Cummins's view of program execution seems to be the way he mixes functional analysis and computational description. Roughly, Cummins thinks that explaining a capacity by program execution is the same as giving a functional analysis of it, and therefore the program is not a part of the computer but a description of it (see Section 4 above). This leads Cummins and his followers to the paradoxical conclusion that connectionist networks compute by executing algorithms or programs (Cummins and Schwarz 1991, Roth 2005). But it should be obvious that connectionist networks do not store and execute programs in the sense I explicated in the main text, which

6 Computational Functionalism

We now have the means to explicate computational functionalism:

> *Computational functionalism*: the mind is the software of the brain.

In the broadest sense, this may be interpreted to say that the mind is (an aspect of) the computational organization of the brain, where computational organization is the functional organization of a computing mechanism and the brain is assumed to be a computing mechanism. In other words, systems that realize minds are mechanisms that manipulate strings of digits according to general rules; the mind is the collection of functional states and properties such that the mechanism manipulates those strings in accordance with those rules.

This broad interpretation cashes out the general analogy between minds and many computing mechanisms, according to which both minds and computing mechanisms manipulate complex combinatorial structures in accordance with appropriate rules. This version of computational functionalism is compatible with any nontrivial version of computationalism, including connectionist computationalism. But as I pointed out in Section 2, this analogy is neither as strong nor as explanatory as the analogy between minds and (program-controlled) computers. The more general analogy is not based on the notion of software used in computer science, which is the notion that explains the

---

is why their behavior is not as flexible as that of digital computers. For a detailed account of connectionist computationalism, including a distinction between connectionist systems that compute and those that don't, see Piccinini 2008c. I should point out that Cummins has recently agreed that "stored programs are certainly causes" (personal correspondence).

capacities of (program-controlled) computers and inspires the slogan "the mind is the software of the brain." Accordingly, in explicating computational functionalism we should give precedence to the literal notion of software. (I'll come back to this more general formulation later.)

In its strong and literal form, computational functionalism says that (i) the brain contributes to the production of behavior by storing and executing programs, in the sense sketched in the previous section, and (ii) the mind is constituted by the programs stored and executed by the brain, plus, perhaps, the states and processes generated by executing those programs. As in the broader version of computational functionalism, the mind is an aspect of the computational organization of a computing mechanism; in addition, the computing mechanism is a program-controlled computer and the mind is its programs (plus, perhaps, the states and processes generated by executing the programs). This doctrine has some interesting consequences for the study of minds and brains.

Computational functionalism licenses explanations of mental capacities by program execution. This is a kind of mechanistic explanation, which explains mental capacities by postulating a specific kind of mechanism with specific functional properties. Briefly, the postulated mechanism includes memory components, which store programs, and at least one processor, which manipulates and executes them. Together, the interaction between memories and processors determines how the system processes its data and produces its outputs. The capacities of the system are explained as the result of the processing of data performed by the processor(s) in response to the program(s).

Computational functionalism entails that minds are multiply realizable, in the sense in which different tokens of the same type of computer program can run on

different kinds of hardware. So if computational functionalism is correct, then—pace Bechtel and Mundale 1999, Shapiro 2000, Churchland 2005 and other foes of multiple realizability—mental programs can also be specified and studied independently of how they are implemented in the brain, in the same way in which one can investigate what programs are (or should be) run by digital computers without worrying about how they are physically implemented. Under the computational functionalist hypothesis, this is the task of psychological theorizing. Psychologists may speculate on which programs are executed by brains when exhibiting certain mental capacities. The programs thus postulated are part of a mechanistic explanation for those capacities.

The biggest surprise is that when interpreted literally, computational functionalism entails that the mind is a (stable state of) a component of the brain, in the same sense in which computer program tokens are (stable states of) components of computers. As a consequence, even a brain that is not processing any data—analogously to an idle computer, or even a computer that is turned off—might still have a mind, provided that its programs are still physically present. This consequence seems to offend some people's intuitions about what it means to have a mind, but it's independently plausible. It corresponds to the sense in which people who are asleep or otherwise unconscious still have minds. Their minds are "causally quiescent," as David Armstrong puts it (1981).

Computational functionalism describes the mind as a program, which means that the function of the mind is to determine which sequences of operations the brain has to perform. This presupposes a certain mechanistic explanation of the brain as a program-controlled computer, i.e., a mechanism with certain components that have certain

functions and a certain organization. Whether a system is a particular kind of mechanism is an empirical question. In this important respect, computational functionalism turns out to incorporate a strong empirical hypothesis.

Philosophers of mind have usually recognized that computationalism is an empirical hypothesis in two respects. On the one hand, there is the empirical question of whether a computer can be programmed to exhibit all of the capacities that are peculiar to minds. This is one traditional domain of artificial intelligence. On the other hand, there is the empirical question of whether all mental capacities can be explained by program execution. This is one traditional domain of cognitive psychology. As to neuroscience, computationalists have traditionally considered it irrelevant to testing their hypothesis, on the grounds that the same software can be implemented by different kinds of hardware. This attitude is unsatisfactory in two respects.

First, as we have seen, at least two important construals of functionalism are such that they entail computationalism. But if computationalism is a logical consequence of the metaphysical doctrine of functionalism, then the empirical status of computationalism is tied to that of functionalism: if functionalism is a priori true (as some philosophers believe), then computationalism should need no empirical testing; conversely, any empirical disconfirmation of computationalism should disconfirm functionalism too. An important advantage of my proposed reformulation of functionalism is that it does not entail computationalism. This leaves computationalism free to be an empirical hypothesis about the specific functional organization of the brain, which—when conjoined with functionalism—gives rise to computational functionalism.

But second, if computationalism is an empirical hypothesis to the effect that mental capacities are the result of program execution, it isn't enough to test it by programming computers and attempting to explain mental capacities by program execution. Indeed, to assume that this is enough for testing it begs the question of whether the brain is the sort of mechanism that could run mental programs at all—whether it is a (program-controlled) computer. Assuming that the mind is the software of the brain presupposes that the brain has components of the relevant kinds, with the relevant functional and organizational properties.

Whether the brain is a kind of program-controlled computer is itself an empirical question, and if the brain were not functionally organized in the right way, computational functionalism about the mind would turn out to be false. This shows computational functionalism to incorporate an empirical hypothesis that can be effectively tested only by neuroscience. Whether brains are one kind of mechanism or another can only be determined by studying brains.

This sense in which computational functionalism embodies an empirical hypothesis is more fundamental than the other two. If the brain is a (program-controlled) computer, then both classical artificial intelligence and classical cognitive psychology have a fair chance of succeeding. But if the brain is *not* a computer, then classical artificial intelligence and cognitive psychology may or may not succeed in ways that depend on the extent to which it is possible to reproduce the capacities of systems that are not computers by executing programs. It may be possible to reproduce all or many mental capacities by computational means even though the brain is not a computer, the

mind is something other than the programs running on the brain, or both. The extent to which this is possible is a difficult question, which there is no room to address here.

I have formulated and discussed computational functionalism primarily using the notion of program execution, because the analogy between minds and program-controlled computers is the motivation behind the strong version of computational functionalism. There is no question that many of those who felt the pull of the analogy between some features of minds and some features of computers—such as Alan Turing, John von Neumann, Jerry Fodor, Allen Newell, and Herbert Simon—did so in part because of the explanatory power of program execution.

But as I noticed at the beginning of this essay, computational functionalism is ambiguous between a strong and a weak reading. It is equally obvious that many other authors, who are (or were at one point) sympathetic to the analogy between some features of minds and some features of computers, such as Hilary Putnam, Robert Cummins, Paul and Patricia Churchland, Michael Devitt and Kim Sterelny (1999), and even Warren McCulloch and Walter Pitts (at least in 1943), would resist the conclusion that the brain stores and executes programs. Is there a way to cash out their view without falling into the trivial conclusion that the mind can be described computationally in the sense in which anything else can? Indeed there is. Their view is captured by the more general formulation of computational functionalism mentioned at the beginning of this section.

The account of computational explanation I sketched in Section 5 applies to all computing mechanisms, regardless of whether they are controlled by programs. In fact, computation by program execution is explicated in terms of the more general notion of computation tout court. Roughly, computation is the manipulation of data and (possibly)

internal states according to an appropriate rule. (Computation by program execution, then, is computation performed in response to instructions that encode the relevant rule.) The digital computers we use every day compute by executing programs, but non-universal Turing machines, finite state automata, and many connectionist networks perform computations without executing programs.

To cover theories that don't appeal to program execution, all we need to do is interpret computational functionalism in terms of computation tout court, without appealing to program execution. According to this generalized computational functionalism, the mind is (some aspect of) the computational organization of a (computing) mechanism, regardless of whether that mechanism is a program-controlled computer, a connectionist computing mechanism, or any other kind of computing mechanism (e.g., a finite state automaton). Given the generalized formulation, psychological explanations need not invoke the execution of programs—they can invoke either program execution or some other kind of computation (connectionist or otherwise) that is presumed to generate the behavior to be explained. This kind of explanation is still a mechanistic explanation that appeals to the manipulation of strings of digits in accordance with an appropriate rule by appropriate components with appropriate functions. Hence, this generalized formulation of computational functionalism still presupposes that the brain has the relevant mechanistic properties, which can be studied empirically by neuroscience. Given this generalization, computational functionalism is compatible with any computational theory of mind, including connectionist computationalism.

Abandoning the strong analogy between minds and computers (based on program execution), as the generalized version of computational functionalism does, produces a loss of explanatory power. The generalized version of computational functionalism still appeals to computation in explaining mental capacities, but it can no longer appeal to the flexibility that comes with the ability to acquire, store, modify, and execute different programs. Which computing mechanisms are powerful enough to explain mental capacities? We do not have room here to enter this complex debate (Macdonald and Macdonald 1995, Aizawa 2003). But by drawing attention to all functional and organizational aspects of computing mechanisms at all relevant levels, the account here proposed promises to push this debate forward.

The present account sheds light on some other old disputes too. Two mental capacities that are especially contentious are intentionality and consciousness. Several thought experiments have been proposed to show that either intentionality or consciousness cannot be explained computationally (e.g., Block 1978, Searle 1980, Maudlin 1989). The putative failure of computational explanation is then assumed to affect functionalism, presumably due to the assumption that functionalism entails computationalism. But now we have seen that properly construed, functionalism does not entail computationalism.

The only legitimate conclusion that may be drawn from these thought experiments is that computationalism is insufficient to explain intentionality or consciousness. This does not entail that computationalism explains no mental capacities, nor does it entail that intentionality and consciousness cannot be explained functionally by some process other than computation. In other words, if the intuitions behind those

thought experiments are accepted, then computationalism might still explain many mental capacities, and functionalism might still be true of the whole mind. Of course, the intuitions behind the thought experiments are themselves in dispute. They remain an uncertain basis for reaching consensus on these matters.

7 Functionalism, Computationalism, and Computational Functionalism

I have discussed three theses:

> *Functionalism*: The mind is the functional organization of the brain.
>
> *Computationalism*: The functional organization of the brain is computational.
>
> *Computational Functionalism* (generalized): The mind is the computational organization of the brain.

Computational functionalism is the conjunction of functionalism and computationalism. I have offered a mechanistic framework within which to make sense of these doctrines and exhibit some of their mutual relations.

Functionalism does not entail computationalism, and by now it should be easy to see why. There are plenty of functional organizations that do not involve program execution or any other computational process. That the mind is functionally constituted is consistent with any non-computational mechanistic explanation applying to the mind. Thus, it is a fallacy to attack functionalism by impugning some computationalist hypothesis or another (as done, e.g., by Churchland 2005).

Computationalism does not entail functionalism either. Computationalism is compatible with the mind being the computational organization of the brain, but also with the mind being some non-computational but still functional property of the brain, or even some non-functional property of the brain, such as its physical composition, the speed of its action, its color, or more plausibly, the intentional content or phenomenal qualities of its states. In short, one may be a computationalist while opposing or being neutral about functionalism, at least with respect to some aspects of the mind.

Computationalism is an empirical, mechanistic hypothesis about the brain. Even if the brain is a computing mechanism, the mind may or may not be the brain's computational organization—perhaps there are aspects of the mind that have to do with other properties, e.g., the phenomenal qualities of mental states. But if brains turn out *not* to be computing mechanisms, then computationalism (and hence computational functionalism) is false. So, regardless of whether one agrees with computational functionalism, one can still focus on whether the brain is a computing mechanism and investigate computationalism. This, of course, cannot be done by examining intuitions about imaginary scenarios (Block 1978, Searle 1980, Maudlin 1989)—it can only be done by studying the functional organization of the brain empirically.[33]

The standard formulations of computational functionalism in philosophy of mind have made it difficult to discuss computationalism as productively as it can be. They have convinced many philosophers that computationalism is an a priori thesis, to be discussed by philosophical arguments and thought experiments and assessed by the extent to which it solves philosophical problems such as the mind-body problem. This has led philosophers to ignore the fact that, in so far as it has empirical content,

[33] For some hints on the likely outcome, cf. Piccinini 2007b and 2008c.

computationalism embodies an empirical scientific hypothesis about the functional organization of the brain, which comes in several varieties that ought to be assessed by neuroscience.

References

Adams, F. and K. Aizawa (2001). "The Bounds of Cognition." Philosophical Psychology **14**(43-64).

Aizawa, K. (2003). The Systematicity Arguments. Boston, Kluwer.

Allen, C., M. Bekoff, et al., Eds. (1998). Nature's Purposes: Analysis of Function and Design in Biology. Cambridge, MA, MIT Press.

Armstrong, D. M. (1970). The Nature of Mind. The Mind/Brain Identity Thesis. C. V. Borst. London, Macmillan**:** 67-79.

Armstrong, D. M. (1981). What is Consciousness? The Nature of Mind. D. M. Armstrong. Ithaca, NY, Cornell University Press.

Ariew, A., R. Cummins, et al., Eds. (2002). Functions: New Essays in the Philosophy of Psychology and Biology. Oxford, Oxford University Press.

Baum, E. B. (2004). What is Thought? Cambridge, MA, MIT Press.

Bechtel, W. (2001). Cognitive Neuroscience: Relating Neural Mechanisms and Cognition. Theory and Method in the Neurosciences. P. Machamer, R. Grush and P. McLaughlin. Pittsburgh, PA, University of Pittsburgh Press**:** 81-111.

Bechtel, W. (2006). Discovering Cell Mechanisms: The Creation of Modern Cell Biology. New York, Cambridge University Press.

Bechtel, W. (2007). Mental mechanisms: Philosophical Perspectives on the Sciences of Cognition and the Brain. London, Routledge.

Bechtel, W. and A. Abrahamsen (2005). "Explanation: A Mechanistic Alternative." Studies in History and Philosophy of Biological and Biomedical Sciences **36**(2): 421-441.

Bechtel, W. and J. Mundale (1999). "Multiple Realizability Revisited: Linking Cognitive and Neural States." Philosophy of Science **66**: 175-207.

Bechtel, W. and R. C. Richardson (1993). Discovering Complexity: Decomposition and Localization as Scientific Research Strategies. Princeton, Princeton University Press.

Bickle, J. (1998). Psychoneural Reduction: The New Wave. Cambridge, MA, MIT Press.

Block, N. (1978). Troubles with Functionalism. Perception and Cognition: Issues in the Foundations of Psychology. C. W. Savage. Minneapolis, University of Minnesota Press. **6:** 261-325.

Block, N. (1980). Introduction: What is Functionalism? Readings in Philosophy of Psychology. N. Block. London, Methuen. **1:** 171-184.

Block, N. (1995). "The Mind as the Software of the Brain." In *An Invitation to Cognitive Science*, edited by D. Osherson, L. Gleitman, S. Kosslyn, E. Smith and S. Sternberg, MIT Press.

Block, N. (2003). "Do Causal Powers Drain Away?" Philosophy and Phenomenological Research **67**(1): 133-150.

Block, N. (2007). Consciousness, Function, and Representation: Collected Papers, Volume 1. Cambridge, MA, MIT Press.
Block, N. and J. A. Fodor (1972). "What Psychological States Are Not." Philosophical Review **81**(2): 159-181.
Bogen, J. (2005). "Regularities and Causality; Generalizations and Causal Explanations." Studies in History and Philosophy of Biological and Biomedical Sciences **36**(2): 397-420.
Bontly, T. (1998). "Individualism and the Nature of Syntactic States." British Journal for the Philosophy of Science **49**: 557-574.
Boorse, C. (2002). A Rebuttal on Functions. Functions: New Essays in the Philosophy of Psychology and Biology. A. Ariew, R. Cummins and M. Perlman. Oxford, Oxford University Press**:** 63-112.
Boyd, R. N. (1980). Materialism without Reductionism: What Physicalism Does Not Entail. Readings in the Philosophy of Psychology. N. Block. London, Methuen**:** 67-106.
Buller, D. J., Ed. (1999). Function, Selection, and Design. Albany, State University of New York Press.
Chalmers, D. J. (1996a). "Does a Rock Implement Every Finite-State Automaton?" Synthese **108**: 310-333.
Chalmers, D. J. (1996b). The Conscious Mind: In Search of a Fundamental Theory. Oxford, Oxford University Press.
Chalmers, D. J. (unpublished). "A Computational Foundation for the Study of Cognition," available at http://consc.net/papers/computation.html. References here are to the paper as downloaded on 12/15/2006.
Chrisley, R. L. (1995). "Why Everything Doesn't Realize Every Computation." Minds and Machines **4**: 403-430.
Christensen, W. D. and M. H. Bickhard (2002). "The Process Dynamics of Normative Function." The Monist **85**(1): 3-28.
Churchland, P. M. (2005). "Functionalism at Forty: A Critical Retrospective." The Journal of Philosophy: 33-50.
Churchland, P. M. and P. S. Churchland (1982). Functionalism, Qualia, and Intentionality. Mind, Brain, and Function: Essays in the Philosophy of Mind. J. I. B. a. R. W. Shahan. Norman, University of Oklahoma Press**:** 121-145.
Churchland, P. S. and T. J. Sejnowski (1992). The Computational Brain. Cambridge, MA, MIT Press.
Coffa, A. J. (1977). "Probabilities: Reasonable or True?" Philosophy of Science **44**(2): 186-198.
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." The Journal of Philosophy **XCVI**(1): 5-32.
Corcoran, J., W. Frank, and M. Maloney (1974). "String Theory." The Journal of Symbolic Logic **39**(4): 625-637.
Craver, C. (2001). "Role Functions, Mechanisms, and Hierarchy." Philosophy of Science **68**(March 2001): 53-74.

Craver, C. (2005). "Beyond Reductionism: Mechanisms, Multifield Integration and the Unity of Neuroscience." Studies in History and Philosophy of Biological and Biomedical Sciences **36**(2): 373-395.
Craver, C. F. (2006). "When Mechanistic Models Explain." Synthese **153**(3): 355-376.
Craver, C. F. (2007). Explaining the Brain. Oxford, Oxford University Press.
Craver, C. and L. Darden (2001). Discovering Mechanisms in Neurobiology. Theory and Method in the Neurosciences. P. Machamer, R. Grush and P. McLaughlin. Pittsburgh, PA, University of Pittsburgh Press**:** 112-137.
Cummins, R. (1977). "Programs in the Explanation of Behavior." Philosophy of Science **44**: 269-287.
Cummins, R. (1983). The Nature of Psychological Explanation. Cambridge, MA, MIT Press.
Cummins, R. (2000). "How does it work?" vs. "What are the laws?" Two Conceptions of Psychological Explanation. Explanation and Cognition. K. F. C. and W. R. A. Cambridge, Cambridge University Press.
Cummins, R. and G. Schwarz (1991). Connectionism, Computation, and Cognition. Connectionism and the Philosophy of Mind. T. Horgan and J. Tienson. Dordrecht, Kluwer**:** 60-73.
Darden, L. (2006). Reasoning in Biological Discoveries. New York, Cambridge University Press.
de Ridder, J. (2006). "Mechanistic Artefact Explanation." *Studies in History and Philosophy of Science* **37**(1): 81-96.
Dennett, D. C. (1978). Brainstorms. Cambridge, MA, MIT Press.
Devitt, M. and K. Sterelny (1999). Language and Reality: An Introduction to the Philosophy of Language. Cambridge, MA, MIT Press.
Egan, F. (1992). "Individualism, Computation, and Perceptual Content." Mind **101**(403): 443-459.
Egan, F. (2003). Naturalistic Inquiry: Where does Mental Representation Fit in? Chomsky and His Critics. L. M. Antony and N. Hornstein. Malden, MA, Blackwell**:** 89-104.
Enç, B. (1983). "In Defense of the Identity Theory." Journal of Philosophy **80**: 279-298.
Fodor, J. A. (1965). Explanations in Psychology. Philosophy in America. M. Black. London, Routledge and Kegan Paul.
Fodor, J. A. (1968a). Psychological Explanation. New York, Random House.
Fodor, J. A. (1968b). "The Appeal to Tacit Knowledge in Psychological Explanation." Journal of Philosophy **65**: 627-640.
Fodor, J. A. (1975). The Language of Thought. Cambridge, MA, Harvard University Press.
Fodor, J. A. (1997). Special Sciences: Still Autonomous after All These Years. Ridgeview, CA.
Fodor, J. A. (2000). The Mind Doesn't Work That Way. MIT Press, Cambridge, MA.
Gillett, C. (2002). "The Dimensions of Realization: A Critique of the Standard View." Analysis **62**: 316-323.
Gillett, C. (2003). "The Metaphysics of Realization, Multiple Realizability and the Special Sciences." The Journal of Philosophy **C**(11): 591-603.

Glennan, S. S. (2002). "Rethinking Mechanistic Explanation." *Philosophy of Science* **64**: 605-206.
Glennan, S. S. (2005). "Modeling Mechanisms." Studies in History and Philosophy of Biological and Biomedical Sciences **36**(2): 443-464.
Harman, G. (1973). Thought. Princeton, Princeton University Press.
Harman, G. (1999). Reasoning, Meaning and Mind. Oxford, Clarendon Press.
Haugeland, J. (1978). "The Nature and Plausibility of Cognitivism." Behavioral and Brain Sciences **2**: 215-260.
Heil, J. (2003). From an Ontological Point of View. Oxford, Clarendon Press.
Heil, J. (2004). Functionalism, Realism and Levels of Being. Hilary Putnam: Pragmatism and Realism. J. Conant and U. M. Zeglen. London, Routledge**:** 128-142.
Houkes, W. (2006). "Knowledge of Artefact Functions." *Studies in History and Philosophy of Science* **37**(1): 102-113.
Houkes, W. and A. Meijers (2006). "The Ontology of Artefacts: The Hard Problem." *Studies in History and Philosophy of Science* **37**(1): 118-131.
Houkes, W. and P. Vermaas (2004). "Actions versus Functions: A Plea for an Alternative Metaphysics of Artifacts." *The Monist* **87**(1): 52-71.
Humphreys, P. (1989). The Chances of Explanation: Causal Explanation in the Social, Medical, and Physical Sciences. Princeton, Princeton University Press.
Humphreys, P. (2004). Extending Ourselves: Computational Science, Empiricism, and Scientific Method. Oxford, Oxford University Press.
Keeley, B. (2000). "Shocking Lessons from Electric Fish: The Theory and Practice of Multiple Realizability." Philosophy of Science **67**: 444-465.
Kim, J. (1989). "The Myth of Nonreductive Materialism." Proceedings and Addresses of the American Philosophical Association **63**: 31-47.
Kim, J. (1992). "Multiple Realization and the Metaphysics of Reduction." Philosophy and Phenomenological Research **52**: 1-26.
Kim, J. (1998). Mind in a Physical World: An Essay on the Mind-Body Problem and Mental Causation. Cambridge, MA, MIT Press.
Kim, J. (2003). "Blocking Causal Drainage and Other Maintenance Chores with Mental Causation." Philosophy and Phenomenological Research **67**(1): 151-176.
Lewis, D. K. (1966). "An Argument for the Identity Theory." Journal of Philosophy **63**: 17-25.
Lewis, D. K. (1969). "Review of *Art, Mind, and Religion*." Journal of Philosophy **66**(22-27).
Lewis, D. K. (1970). "How to Define Theoretical Terms." Journal of Philosophy **67**: 427-446. Reprinted in D. K. Lewis (1983), Philosophical Papers, Vol. 1, pp. 78-95 [CK].
Lewis, D. K. (1972). "Psychophysical and Theoretical Identifications." Australasian Journal of Philosophy **50**: 249-258.
Lewis, D. K. (1980). Mad Pain and Martian Pain. Readings in Philosophy of Psychology, Volume 1. N. Block. Cambridge, MA, MIT Press**:** 216-222.
Lucas, J. R. (1996). "Minds, Machines, and Gödel: A Retrospect." *Machines and Thought: The Legacy of Alan Turing*. P. J. R. Millikan and A. Clark, Eds. Oxford, Clarendon.
Lycan, W. (1981). "Form, Function, and Feel." Journal of Philosophy **78**: 24-50.

Lycan, W. (1982). Psychological Laws. Mind, Brain, and Function: Essays in the Philosophy of Mind. J. I. Biro and R. W. Shahan. Norman, University of Oklahoma Press**:** 9-38.
Lycan, W. (1987). Consciousness. Cambridge, MA, MIT Press.
Macdonald, C. and G. Macdonald, Eds. (1995). Connectionism: Debates on Psychological Explanation, Volume Two. Oxford, Blackwell.
MacDonald and Macdonald 1995
Machamer, P. (2004). "Activities and Causation: The Metaphysics and Epistemology of Mechanisms." International Studies in the Philosophy of Science **18**(1): 27-39.
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.
Maudlin, T. (1989). "Computation and Consciousness." Journal of Philosophy **86**(8): 407-432.
Millikan, R. G. (1984). Language, Thought, and Other Biological Categories: New Foundations for Realism. Cambridge, MA, MIT Press.
Moor, J. H. (1978). "Three Myths of Computer Science." British Journal for the Philosophy of Science **29**: 213-222.
Nelson, R. J. (1987). "Church's Thesis and Cognitive Science." Notre Dame Journal of Formal Logic **28**(4): 581-614.
Newell, A. (1980). "Physical Symbol Systems." Cognitive Science **4**: 135-183.
Newell, A. (1990). Unified Theories of Cognition. Cambridge, MA, Harvard University Press.
Pereboom, D. and H. Kornblith (1991). "The Metaphysics of Irreducibility." Philosophical Studies **63**.
Perlman, M. (2004). "The Modern Philosophical Resurrection of Teleology." *The Monist* **87**(1): 3-51.
Piccinini, G. (2003a). "Alan Turing and the Mathematical Objection." Minds and Machines **13**(1): 23-48.
Piccinini, G. (2003b). "Review of John von Neumann's The Computer and the Brain." Minds and Machines **13**(2): 327-332.
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**(2): 175-215.
Piccinini, G. (2004b). "Functionalism, Computationalism, and Mental States." Studies in the History and Philosophy of Science **35**(4): 811-833.
Piccinini, G. (2004c). "Functionalism, Computationalism, and Mental Contents." Canadian Journal of Philosophy **34**(3): 375-410.
Piccinini, G. (2007a). "Computational Modeling 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.
Piccinini, G. (2007b). Computational Explanation and Mechanistic Explanation of Mind. Cartographies of the Mind: Philosophy and Psychology in Intersection. M. De Caro, F. Ferretti and M. Marraffa. Dordrecht, Springer**:** 23-36.
Piccinini, G. (2007c). "Computationalism, the Church-Turing Thesis, and the Church-Turing Fallacy." Synthese.

Piccinini, G. (2007d). "Computing Mechanisms." Philosophy of Science **74**(4): 501-526.
Piccinini, G. (2008a). "Computers." Pacific Philosophical Quarterly **89**(1): 32-73.
Piccinini, G. (2008b). "Computation without Representation." Philosophical Studies **137**(2).
Piccinini, G. (2008c). "Some Neural Networks Compute, Others Don't." Neural Networks **21**(2-3): 311-321.
Preston, B. (1998). "Why is a Wing Like a Spoon? A Pluralist Theory of Function." The Journal of Philosophy XCV(5): 215-254.
Preston, B. (2003). "Of Marigold Beer: A Reply to Vermaas and Houkes." British Journal for the Philosophy of Science **54**: 601-612.
Prinz, J. (2001). Functionalism, Dualism and the Neural Correlates of Consciousness. Philosophy and the Neurosciences: A Reader. W. Bechtel, P. Mandik, J. Mundale and R. Stufflebeam. Oxford, Blackwell.
Polger, T. W. (2004). Natural Minds. Cambridge, MA, MIT Press.
Polger, T. W. (2007). "Realization and the Metaphysics of Mind." Australasian Journal of Philosophy **85**(2): 233-259.
Putnam, H. (1960). Minds and Machines. Dimensions of Mind: A Symposium. S. Hook. New York, Collier**:** 138-164.
Putnam, H. (1967a). The Mental Life of Some Machines. Intentionality, Minds, and Perception. H. Castañeda. Detroit, Wayne State University Press**:** 177-200.
Putnam, H. (1967b). Psychological Predicates. Art, Philosophy, and Religion. Pittsburgh, PA, University of Pittsburgh Press.
Putnam, H. (1988). Representation and Reality. Cambridge, MA, MIT Press.
Railton, P. (1978). "A Deductive-Nomological Model of Probabilistic Explanation." Philosophy of Science **45**(2): 202-226.
Roth, M. (2005). "Program Execution in Connectionist Networks." Mind and Language **20**(4): 448-467.
Rupert, R. (2004). "Challenges to the Hypothesis of Extended Cognition." The Journal of Philosophy **CI**: 389-428.
Rupert, R. (2006). "Functionalism, Mental Causation, and the Problem of Metaphysically Necessary Effects." Noûs **40**: 256-283.
Salmon, W. C. (1984). Scientific Explanation and the Causal Structure of the World. Princeton, Princeton University Press.
Salmon, W. C. (1990). Four Decades of Scientific Explanation. Minneapolis, University of Minnesota Press.
Salmon, W. C. (1998). Causality and Explanation. New York, Oxford University Press.
Scheele, M. (2006). "Function and Use of Artefacts: Social Conditions of Function Ascription." *Studies in History and Philosophy of Science* **37**(1): 23-36.
Scheutz, M. (2001). "Causal versus Computational Complexity." Minds and Machines **11**: 534-566.
Scheutz, M. (2004). Comments presented at the 2004 Pacific APA in Pasadena, CA.
Schlosser, G. (1998). "Self-re-Production and Functionality: A Systems-Theoretical Approach to Teleological Explanation." Synthese **116**(3): 303-354.
Searle, J. R. (1980). "Minds, Brains, and Programs." The Behavioral and Brain Sciences **3**: 417-457.
Searle, J. R. (1992). The Rediscovery of the Mind. Cambridge, MA, MIT Press.

Sellars, W. (1954). "Some Reflections on Language Games." Philosophy of Science **21**: 204-228.
Shagrir, O. (1998). "Multiple Realization, Computation and the Taxonomy of Psychological States." Synthese **114**: 445-461.
Shagrir, O. (2001). "Content, Computation and Externalism." Mind **110**(438): 369-400.
Shagrir, O. (2005). The Rise and Fall of Computational Functionalism. Hilary Putnam. Y. Ben-Menahem. Cambridge, Cambridge University Press.
Shagrir, O. (2006). "What is Computing in the Brain?" Synthese.
Shapiro, L. A. (1994). "Behavior, ISO Functionalism, and Psychology." Studies in the History and Philosophy of Science **25**(2): 191-209.
Shapiro, L. A. (2000). "Multiple Realizations." The Journal of Philosophy **XCVII**(12): 635-654.
Schlosser, G. (1998). "Self-re-Production and Functionality: A Systems-Theoretical Approach to Teleological Explanation." Synthese **116**(3): 303-354.
Schroeder, T. (2004). "Functions from Regulation." *The Monist* **87**(1): 115-135.
Shoemaker, S. (2001). Realization and Mental Causation. Physicalism and Its Discontents. C. Gillett and B. Loewer. Cambridge, Cambridge University Press**:** 74-98.
Shoemaker, S. (2003a). "Realization, Micro-Realization, and Coincidence." Philosophy and Phenomenological Research **LXVII**(1): 1-23.
Shoemaker, S. (2003b). Identity, Cause and Mind, Expanded Edition. Oxford: Clarendon Press.
Simon, H. A. (1996). The Sciences of the Artificial, Third Edition. Cambridge, MA, MIT Press.
Smith, B. C. (1996). On the Origin of Objects. Cambridge, MA, MIT Press.
Sober, E. (1990). Putting the Function Back into Functionalism. Mind and Cognition. W. Lycan. Malden, MA, Blackwell**:** 63-70.
Sober, E. (1999). "The Multiple Realizability Argument against Reductionism." Philosophy of Science **66**: 542-564.
Stich, S. (1983). From Folk Psychology to Cognitive Science. Cambridge, MA, MIT Press.
Tabery, J. (2004). "Synthesizing Activities and Interactions in the Concept of a Mechanism." Philosophy of Science **71**(1): 1-15.
Thagard, P. (2003). "Pathways to Biomedical Discovery." Philosophy of Science **70**(2): 235-254.
Turing, A. M. (1950). "Computing Machinery and Intelligence." Mind **59**: 433-460.
Vermaas, P. E. (2006). "The Physical Connection: Engineering Function Ascription to Technical Artefacts and their Components." *Studies in History and Philosophy of Science* **37**(1): 62-75.
Vermaas, P. E. and W. Houkes (2006). "Technical Functions: A Drawbridge between the Intentional and Structural Natures of Technical Artefacts." *Studies in History and Philosophy of Science* **37**(1): 5-18.
von Neumann, J. (1951). The General and Logical Theory of Automata. Cerebral Mechanisms in Behavior. L. A. Jeffress. New York, Wiley**:** 1-41.
von Neumann, J. (1958). The Computer and the Brain. New Haven, Yale University Press.

Webb, J. C. (1980). Mechanism, Mentalism, and Metamathematics. Dordrecht, Reidel.
Wilkes, K. V. (1982). Functionalism, Psychology, and the Philosophy of Mind. Mind, Brain, and Function: Essays in the Philosophy of Mind. J. I. Biro and R. W. Shahan. Norman, University of Oklahoma Press**:** 147-167.
Wilson, M. (1985). "What is This Thing Called "Pain"?-The Philosophy of Science Behind the Contemporary Debate." Pacific Philosophical Quarterly **66**: 227-267.
Wilson, M. (1993). Honorable Intensions. Naturalism: A Critical Appraisal. S. J. Wagner and R. Warner. Notre Dame, Indiana, University of Indiana Press**:** 53-94.
Wilson, R. A. (2004). Boundaries of the Mind: The Individual in the Fragile Sciences. Cambridge, Cambridge University Press.
Wimsatt, W. C. (1972). "Teleology and the Logical Structure of Function Statements." Studies in History and Philosophy of Science **3**(1): 1-80.
Wimsatt, W. C. (2002). Functional Organization, Analogy, and Inference. Functions: New Essays in the Philosophy of Psychology and Biology. A. Ariew, R. Cummins and M. Perlman. Oxford, Oxford University Press**:** 173-221.
Wright, L. (1973). "Functions." Philosophical Review **82**: 139-168.