

The Functional Account of Computing Mechanisms¹

Gualtiero Piccinini

10/23/2003

On the one hand, we have a very elegant set of mathematical results ranging from Turing's theorem to Church's thesis to recursive function theory. On the other hand, we have an impressive set of electronic devices that we use every day. Since we have such advanced mathematics and such good electronics, we assume that somehow somebody must have done the basic philosophical work of connecting the mathematics to the electronics. But as far as I can tell, that is not the case. On the contrary, we are in a peculiar situation where there is little theoretical agreement among the practitioners on such absolutely fundamental questions as, What exactly is a digital computer? What exactly is a symbol? What exactly is an algorithm? What exactly is a computational process? Under what physical conditions exactly are two systems implementing the same program? (Searle 1992, p. 205.)

This paper offers an account of what it is for a physical system to be a computing mechanism, namely a mechanism that performs computations. This account, which I call the *functional account of computing mechanisms*, can be used to individuate computing mechanisms and the functions they compute and to taxonomize computing mechanisms based on their different computing power. This makes it ideal for grounding the comparison and assessment of computational theories of mind and brain. The functional account stems from two main moves.

First, I use non-semantic and *a fortiori* non-intentional language to individuate computing mechanisms and the functions they compute; i.e., I keep the question of whether something is a computing mechanism and what it computes separate from the question of whether something has semantic content. I defended this move at length in Piccinini 2003e and 2003f, where I showed that this move rules out most of the existing accounts of computing mechanisms in the philosophical literature.

¹ Many thanks to Peter Machamer for his comments on earlier drafts.

Second, I explicate the notion of computing mechanism using the tool of functional analysis. I construe functional analysis as in engineering and biology, where a functional analysis of a mechanism is a partition of a mechanism into component parts and an assignment of functions to those parts. Given this construal, analyzing a mechanism functionally is different from (though compatible with) giving a computational description of that mechanism. I defended this move in Piccinini 2003c, 2003d, and 2003g. Again, this move rules out almost all existing accounts of computing mechanisms in the relevant philosophical literature, where giving a functional analysis is construed as being the same as giving a computational description.

The functional account of computing mechanisms flows naturally from these two moves. Computing mechanisms, such as calculators and computers, are analyzed in terms of their component parts (processors, memory units, input and output devices) and their functions. Those components are also analyzed in terms of their component parts (e.g., registers and circuits) and their functions. Those, in turn, are analyzed in terms of primitive computing components (logic gates) and their functions. Primitive computing components can be further analyzed but their analysis does not illuminate the notion of computing mechanism. In contrast to extant philosophical accounts of computing mechanisms, the present account explicates very naturally both our ordinary language about computing mechanisms and the language and practices of computer scientists and engineers.

Although I believe I have motivated the above moves sufficiently well in the relevant papers, in this occasion I'm going to defend the functional account of computing mechanisms on independent grounds. I will do so by first stating six desiderata that an

account of computing mechanisms should satisfy, and then comparing and contrasting the main existing accounts of computing mechanisms *vis a vis* those desiderata. I will point out the difficulties of existing accounts in satisfying those desiderata and the advantages of the functional account of computing mechanisms as independently as possible of the above moves.

1 Desiderata for an Account of Computing Mechanisms

An optimal account of computing mechanisms should satisfy the following six desiderata:

1. *Paradigmatic computing mechanisms compute.* A good account of computing mechanisms should entail that all paradigmatic examples of computing mechanisms, such as digital computers, calculators, both universal and non-universal Turing Machines, finite state automata, and humans who perform mathematical calculations, compute.

2. *Paradigmatic non-computing systems don't compute.* A good account of computing mechanisms should entail that all paradigmatic examples of non-computing mechanisms and systems, such as planetary systems, hurricanes, and stomachs, don't compute.

In the literature, computers and Turing Machines are usually taken to be paradigmatic examples of computing mechanisms, whereas planetary systems, the weather, and stomachs are taken to be paradigmatic examples of systems that don't perform computations.² When we have strong pre-theoretical intuitions about clear cases of things that belong to a class and things that don't, as in the case of computing

² For evidence, see Fodor 1968b, p. 632; Fodor 1975, p. 74; Dreyfus 1979, pp. 68, 101-102; Searle 1980, pp. 37-38; Searle 1992, p. 208. .

mechanisms, a general requirement of a good philosophical theory is that it fit those clear cases. By satisfying desiderata 1 and 2, a good account of computing mechanisms would draw a principled distinction between the class of computing mechanisms and the class of other things, and it would draw this line in a place that fits our pre-theoretical intuitions as much as possible.

3. *Computation is observer-independent.* A good account of computing mechanisms should entail that the computations performed by a mechanism can be identified independently of which observer is studying the mechanism. This is a desideratum for two reasons. First, it underwrites ordinary computer science and computer engineering, where different observers agree on which computations are performed by which mechanisms. Second, it underwrites a genuinely scientific computational theory of the brain (or mind), according to which the computations performed by the brain (or mind) can be empirically discovered.³

4. *Computations can go wrong.* A mechanism m miscomputes just in case m is computing function f on input i , $f(i) = o_1$, m outputs o_2 , and $o_2 \neq o_1$.⁴ A good account of computing mechanisms should explain what it means for a computing mechanism to miscompute. This is a desideratum because miscomputation, or more informally making “mistakes,” is an important aspect of the intuitive notion of computing as well as the practice of using machines to perform computations. Those who design and use computing mechanisms devote a large portion of their efforts to avoid miscomputations.

³ This desideratum is one instance of the general desideratum that scientific concepts and methods be inter-subjective, or public. For an extended clarification and defense of this desideratum in the case of scientific methods, see Piccinini 2003l.

⁴ Here o_1 and o_2 represent any possible outcome of a computation, including the possibility that the function is undefined for a given input, which corresponds to a non-halting computation. Miscomputation is analogous to misrepresentation (Dretske 1986), but it's not the same. Miscomputation is possible whether or not the inputs and outputs of a mechanism represent anything. Misrepresentation is possible whether or not the representations involved are computational inputs, outputs, or internal states.

To the extent that an account of computing mechanisms makes no sense of that effort, it is unsatisfactory.

5. *Some computing mechanisms are not computers.* A good account of computing mechanisms should explain how the machines that are ordinarily called computers, such as our desktops and laptops, are different from other computing mechanisms, such as calculators and finite state automata. This is a desideratum because the term “computer” in ordinary language is not used for all computing mechanisms but rather it is reserved for some special ones. A good account of computing mechanisms should explain why the term is used in this restrictive way and what is special about computers.

6. *Program execution is explanatory.* Ordinary digital computers are said to execute programs, and their behavior is normally explained by appealing to the programs they execute. In Piccinini 2003g, I showed that the literature on computational theories of mind and brain contains appeals to explanation by program execution, and more generally explanation by appeal to the computations performed by a mechanism, as the form of explanation appropriate for psychological capacities. So a good account of computing mechanisms should say how appeals to program execution, and more generally to computation, explain the behavior of computing mechanisms. It should also say how program execution relates to the general notion of computation: whether they are the same or whether program execution is a species of computation, and if so, what is distinctive about it.

With these desiderata as landmarks, we can proceed to formulate the functional account of computing mechanisms.

2 The Functional Account of Computing Mechanisms

The central idea is to analyze computing mechanisms using functional analysis. By functional analysis of a system X , I mean a description of X in terms of spatiotemporal components of X and their functions.⁵ To identify the functions of a system, I defer to the relevant community of scientists. Biologists ascribe functions to types of organs (e.g., the pumping function of hearts) and engineers ascribe them to types of artifacts (e.g., the cooling function of refrigerators). Scientists differentiate between functions and accidental effects (e.g., making noise or breaking under pressure). Tokens of organs and artifacts that do not perform their functions are said to malfunction or be defective.

The philosophical analysis of function ascription in biology and engineering is a controversial matter on which I remain neutral.⁶ For present purposes, suffice it that biologists and engineers who analyze systems functionally agree on the functions of the system and its components, and use the functions and malfunctions of components to explain the activities of the containing systems. To distinguish functionally analyzable systems in the present sense from other systems, I will say that a functionally analyzable system is a mechanism.

This notion of functional analysis applies to what are called computers in ordinary language in a way that matches the language and practices of computer scientists and

⁵ For a similar notion of functional analysis, see Bechtel and Richardson 1993. For a sophisticated version of the same notion, with emphasis on neural mechanisms, see Craver 2001. For an early application to psychological and neural mechanisms, see Deutsch 1960. The classic sources of the notion of functional analysis in philosophy of psychology, Fodor 1968a and Cummins 1975, 1983, have a lot in common with the notion used here but are potentially misleading for present purposes because, as I argued in Piccinini 2003d, those authors conflate giving a functional analysis of a mechanism with giving a computational description of that mechanism.

⁶ The main competing accounts can be found in Allen, Bekoff, and Lauder 1998; Preston 1998; Schlosser 1998; Buller 1999; Ariew, Cummins, and Perlman 2002, Christensen and Bickhard 2002.

engineers.⁷ Computing mechanisms, including computers, are mechanisms whose function is computing. Functionally analyzed systems, including computing mechanisms and their components, perform the activities we ascribe them *ceteris paribus*, as a matter of their function. In the rest of our discussion, we will mostly focus on their normal operation, but it is important to keep in mind that they can malfunction, break, or be malformed or defective. This will help satisfy desideratum 4 (Computations can go wrong).

I will now propose a way to single out computations from other activities of mechanisms, thereby differentiating between computing mechanisms and other mechanisms. To do so, I will assume that the relevant community of scientists can identify the functionally relevant causal powers of a mechanism and its components. I will suggest what criteria must be met by the functionally relevant causal powers of a mechanism for it to count as performing computations (in a nontrivial sense), and hence as being a computing mechanism.

A computation is the production of certain output strings of tokens from certain input strings of tokens (and perhaps internal states), according to a general rule that applies to all inputs and outputs. A token is a particular that belongs to a (usually finite) number of types. Upon entering a mechanism in the relevant way, a token affects the input-output behavior of the mechanism in a way that identifies it as belonging to a type. Every token of the same type has the same effect on a mechanism relative to generating the mechanism's output, and each type of token has a different effect on the mechanism relative to generating output. That is, *ceteris paribus*, substituting a token of type T_2 for a

⁷ For a standard introduction to computer organization and design, from which I took some of the technical terminology and details, see Patterson and Hennessy 1998.

token of type T_1 in a string may result in a different computation process, which may generate a different output string, if and only if $T_2 \neq T_1$.

A string is a sequence of permutable tokens identified by the tokens' types, their number, and their order within the string. Every finite string has a first and a last token member and each token member (except for the last member) has a unique successor. A token within a string can be substituted by another token without affecting the other tokens' types, number, or positions within the string. In particular, when an input string is processed by a mechanism, *ceteris paribus*, the tokens' types, their number, and their order within the string make a difference to what output string is generated.⁸

Any mechanism whose functional analysis ascribes it the function of generating outputs strings from input strings in accordance with a general rule that applies to all strings is a computing mechanism. The mechanism's ability to perform computations is explained by its functional analysis in terms of its components and the functions they perform. Elsewhere, I provided a detailed functional analysis of computing mechanisms' components (Piccinini 2003i) and of some important kinds of computing mechanisms, including computers (Piccinini 2003j). Here, there is only room for a brief summary of the results.

Computers and calculators can be analyzed in terms of four main kinds of components: (1) memories, whose function is to store instructions, data, and intermediate results, (2) processors, whose function is to perform operations on data, (3) input devices, whose function is to take input strings (either data or instructions) from the environment and deliver them to memory components, and (4) output devices, whose

⁸ For the mathematical theory of strings, see Corcoran, Frank, and Maloney 1974.

function is to take output strings from memory components and deliver them to the external environment. Memories and processors, in turn, can be analyzed into various kinds of components, such as arithmetic-logic units, whose function is to perform a finite number of arithmetic or logic operations on strings of fixed (finite) length, and memory registers, whose function is to store strings of fixed (finite) length.

All components of computing mechanisms can be ultimately analyzed into primitive computing components and primitive non-computing components. Primitive computing components have the function of performing computations on atomic strings, that is, strings made of only one token. Primitive non-computing components perform various functions that are necessary for the proper functioning of computing mechanisms; for instance, an important primitive non-computing component of ordinary computing mechanism is a clock, i.e. a component whose function is to generate a signal at fixed time intervals.

The right combination of primitive computing components and primitive non-computing components explains how complex components perform their functions, and the right combination of complex components explains how different computing mechanisms perform their computations, what functions they compute, and what computation power they have. By performing a functional analysis of computing mechanisms, it is thus possible to individuate computing mechanisms, the functions they compute, and their computing power, and to explain how they perform their computations.

3 Comparison With Previous Accounts of Computing Mechanisms

In this section, I will compare and contrast the main existing proposals about computing mechanisms in light of the desiderata of an account of computing mechanisms. The desiderata are: (1) Paradigmatic computing mechanisms compute, (2) Paradigmatic non-computing systems don't compute, (3) Computation is observer-independent, (4) Computations can go wrong, (5) Some computing mechanisms are not computers, and (6) Program execution is explanatory. Each proposal discussed below rejects at least one of the two moves on which my account is based, and in that respect I've already argued elsewhere that it should be rejected. Nevertheless, comparing different proposals *vis a vis* the six desiderata further highlights the advantages of the functional account. The results of the comparison are summarized in Table 1.

	Putnam's	Cummins's	Fodor's	Functional
(1) Computing mechanisms compute	Yes	In part/Yes	In part	Yes
(2) Non-computing systems don't compute	No	Yes/No	Yes	Yes
(3) Computation is observer-independent	Yes	No	Yes	Yes
(4) Computations can go wrong	No	No	No	Yes
(5) Some computing mechanisms are not computers	No	No	No	Yes
(6) Program execution is explanatory	No	Yes	Yes	Yes

Table 1. Desiderata satisfied by different accounts of computing mechanisms.

3.1 Putnam

Hilary Putnam's proposal, and its historical importance, was analyzed in detail in Piccinini 2003d and 2003g. According to Putnam (1960, 1967, 1988), a system is a computing mechanism if and only if there is a mapping between a computational description and a physical description of the system. By computational description, Putnam means a formal description of the kind used in computability theory, such as a Turing Machine or a finite state automaton. Putnam puts no constraints on how to find the mapping between the computational and the physical description, allowing any computationally identified state to map onto any physically identified state.

Putnam's account easily satisfies desideratum 1, because computational descriptions can be mapped onto physical descriptions of paradigmatic computing mechanisms. But Putnam's account fails to satisfy all the remaining desiderata. As to desideratum 2, it fails because computational descriptions can also be (approximately) mapped onto physical descriptions of paradigmatic non-computing systems (as the popularity of computational models in many sciences testifies), which according to Putnam turns them into computing mechanisms.⁹ As to desideratum 3, it fails because given Putnam's liberalism about mapping, different observers can devise mappings between the same behavior of the same physical system and different computational descriptions. It is well known that Putnam's account entails that most physical systems implement most computations. This consequence of Putnam's proposal has been explicitly derived by Putnam (1988, pp. 95-96, 121-125) and John Searle (1992, chap. 9).¹⁰ As to desideratum 4, it fails because most behaviors of most physical systems, although computational under Putnam's proposal, are not subject to normative evaluation the way ordinary computations are. As to desideratum 5, it fails because the computational power of a physical system depends only on what computational description maps onto it, and given Putnam's proposal, there is no fact of the matter as to which of the many computational descriptions that map onto it is the right one. As to desideratum 6, it fails because in order for program execution to carry nontrivial explanatory force, program execution must be a specific kind of process that is exhibited

⁹ For a more extended discussion of how this feature of Putnam's account trivializes the notion of computing mechanism and computational theories of mind and brain, see Piccinini 2003c and 2003g.

¹⁰ Both Putnam and Searle take this to apply to computational descriptions in general, independently of Putnam's account of computing mechanisms. Both use this to argue that computationalism—the view that the brain is a computing mechanism—is a trivial thesis of no explanatory value. Here, I simply take this to be a reason against Putnam's account of computing mechanisms.

by specific mechanisms.¹¹ Since Putnam's account turns every behavior of every mechanism into the execution of a large number of programs, program execution has no explanatory force.

Some authors have attempted to improve Putnam's proposal so as to satisfy desideratum 3. They introduce restrictions on the mapping between computational descriptions and physical descriptions of a system. According to David Chalmers (1996) and Jack Copeland (1996), for a physical description to count as an implementation of a computational description, the state transitions identified by that physical description must support counterfactuals of the following form: if the system were in state s_1 , then it would change into state s_2 ; if the system were not in state s_1 , then it would not change into state s_2 (where s_1 and s_2 are states identified by the computational description). Chalmers and Copeland argue that the state transitions employed by Putnam and Searle in their arguments do not obey this constraint, and because of this Chalmers and Copeland reject Putnam and Searle's conclusion that computation is observer-relative.¹² Along similar lines, Matthias Scheutz (1999) proposes to start with physical descriptions of a system in terms of electrical circuit theory, and then abstract some of the physical properties away until a unique computational description of the system is left.

These authors also introduce some considerations pertaining to the inputs, outputs, and components of computing mechanisms. For example, Chalmers argues that the only appropriate models of cognitive mechanisms are automata with inputs and outputs, whose inputs and outputs must map onto the inputs and outputs of cognitive systems. Chalmers also adds that physical implementations of computational

¹¹ For an argument to this effect, see Piccinini 2003g.

¹² Further criticisms to Putnam and Searle's arguments, similar to those of Copeland and Chalmers, can be found in Chrisley 1995.

descriptions must decompose into parts and there must be a mapping between the states of those parts and suitably defined sub-states identified by the computational description. That is, the system must have a “fine-grained causal structure” that maps onto the computational structure identified by the computational description. These considerations might be seen as implicit steps towards the functional account of computing mechanisms that is here defended.

To the extent that these refinements of Putnam’s account satisfy desideratum 3, they constitute an improvement over Putnam’s account of computing mechanisms. There is no room here to analyze these proposals in detail to see if they solve the problem of observer-relativity of computation in a satisfactory manner, and fortunately there is no need to do so. This is because even if it is granted that these refined proposals satisfy desiderata 1 and 3, as they stand they still fail to satisfy the other desiderata. As to desideratum 2 (paradigmatic non-computing systems don’t compute), Chalmers and Scheutz explicitly point out that under their proposal, everything implements some computation or other (Chalmers 1996, p. 331; Scheutz 1999, p. 191). This makes it hard to see how program execution, or computation in general, can play the specific explanatory role that it plays in our explanations of computing mechanisms’ behavior, so that these proposals fail to satisfy desideratum 6 (program execution is explanatory). Copeland’s proposal does not appear to be different in this respect. With respect to desideratum 5 (some computing mechanisms are not computers), it seems that these proposals do not have the resources to differentiate between the computing powers of different mechanisms, because they use the same computational formalism to represent the behavior of every physical system.

3.2 Cummins

According to Robert Cummins (1977, 1983, 1989, esp. pp. 91-92), a system is a computing mechanism if and only if it has inputs and outputs that can be given a systematic semantic interpretation and the system's input-output behavior is the "execution" of a program that yields the interpretation of the system's outputs given an interpretation of the system's inputs. For Cummins, a process is the "execution" of a program if and only if it is made up of sub-processes each of which instantiates one of the steps described by the program.

It is hard to determine the degree to which Cummins's account satisfies our first two desiderata. Many paradigmatic examples of computing mechanisms have inputs and outputs that can be given systematic interpretations, and their processes can be divided into sub-processes each of which instantiates the steps of a relevant program. Cummins's account counts all of those as computing mechanisms, which goes in the direction of satisfying desideratum 1. But there are two kinds of exceptions. One kind includes Turing Machines like the one described by J. Buntrock and H. Marxen in 1989 (cited by Wells 1998), which performs 23,554,764 steps before halting when it's started on a blank tape. It's unclear what if any interpretation can be assigned to this machine, and yet it is a Turing Machine—a paradigmatic example of computing mechanism. The other kind of exception includes many connectionist computing mechanisms, whose processes are not analyzable into sub-processes corresponding to steps defined by a program in any obvious way. In fact, connectionist computing mechanisms have been used as examples that not all computation is algorithmic (Shagrir 1997). In order to include these

exceptions within the class of computing mechanisms, thereby fully satisfying desideratum 1, Cummins might liberalize the constraints on how to assign interpretations and algorithmic descriptions to systems.¹³ But then it becomes hard to rule any system out of the class of computing mechanisms, so that desideratum 2 fails to be satisfied.

Cummins satisfies desideratum 6 because for him, executing a program is the same as satisfying a certain computational description, and satisfying a certain computational description is the same as being functionally analyzed. According to Cummins, functional analysis provides a distinct mode of explanation, i.e. functional explanation (Cummins 1975, 1983). Hence, program execution is explanatory in the sense that it provides a functional explanation. Cummins says he doesn't know how to satisfy desideratum 3 (computation is observer-independent),¹⁴ has not addressed desideratum 4 (computations can go wrong), and his account fails to satisfy desideratum 5 (some computing mechanisms are not computers), because it turns every computing mechanism into a program-executing mechanism, thereby trivializing the difference between different computing powers of different computing mechanisms.¹⁵

¹³ This seems to be implicitly done by Cummins when he attempts to apply his account of computing mechanisms to connectionist systems (Cummins and Schwartz 1991).

¹⁴ Cf.:

[D]irect interpretation seem[s] like a rather subjective and relativistic affair. Nevertheless, it appears to me to be absolutely central to the notion of function instantiation (and hence computation), so I'm simply going to assume it, leaving to someone else the task of filling this hole or widening it enough to dink the ship (Cummins 1989, p. 105).

¹⁵ Accounts of computing mechanisms similar to Cummins's account have been offered by Dennett (1978a), Haugeland (1978), and Churchland and Sejnowski (1992). I discussed Cummins's account because it is the most detailed and systematic in this family of accounts.

3.3 Fodor

According to Jerry Fodor (1968b, 1975, 1998, pp. 10-11), a system is a computing mechanism if and only if (i) it operates on symbols by responding to internal representations of rules and (ii) the system's operations "respect" the semantic content of the symbols. For Fodor, a symbol is a semantically individuated token that can be manipulated by virtue of its formal (non-semantic) properties. At least in the case of natural systems like brains, for Fodor the content of symbols is a natural property that can be discovered in a way that is observer-independent (Fodor 1990, 1998).

Fodor's account improves over both Putnam's and Cummins's accounts. It satisfies desiderata 2, 3, and 6 well. As to 2 (paradigmatic non-computing systems don't compute), non-computing systems are ruled out by their lack of internal representations of rules. As to 3 (computation is observer-independent), computation is made observer-independent by Fodor's naturalistic theory of content, which underwrites his notion of computation. As to 6 (program execution is explanatory), Fodor's account explicates the notion of program execution in terms of the system's internal representation of rules together with the fact that the system's processes are caused by those internal representations. However, Fodor's account still has problems with desideratum 1, and it satisfies neither 4 nor 5. As to 1 (paradigmatic computing mechanisms compute), computing mechanisms that are not obviously interpretable as operating on contentful symbols cannot be accommodated within Fodor's account. Perhaps more seriously, there are plenty of paradigmatic computing mechanisms, starting with ordinary (non-universal) Turing Machines, which do not possess internal representations of rules. Fodor's account legislates that these mechanisms don't compute. As to 4 (computations can go wrong), perhaps Fodor's account has the resources to satisfy it, but Fodor has been silent on this.

Desideratum 5 (some computing mechanisms are not computers) remains unsatisfied, because for Fodor all genuinely computing mechanisms are essentially stored-program computers, which leaves little room for differentiating between their computing power and the computing power of other mechanisms that are ordinarily seen as computing.¹⁶

3.4 The Functional Account and the Six Desiderata

None of the existing accounts satisfies all six desiderata. None of them satisfies desiderata 4 and 5. I will now show how the functional account satisfies the six desiderata in a natural way.

1. *Paradigmatic computing mechanisms compute.* All paradigmatic examples of computing mechanisms, such as digital computers, calculators, Turing Machines, finite state automata, and computing humans, have the function (goal in the case of humans) of generating certain output strings from certain input strings according to a general rule that applies to all strings. According to the functional account, this is necessary and sufficient for them to be ascribed computations. Accordingly, the functional account properly counts all paradigmatic examples of computing mechanisms as such.

2. *Paradigmatic non-computing systems don't compute.* The functional account of computing mechanisms explains why paradigmatic examples of non-computing

¹⁶ An account similar to Fodor's is that of Zenon Pylyshyn (1984). Like Fodor, Pylyshyn individuates computational states, inputs, and outputs by their semantic properties. The main difference between the two is Pylyshyn's notion of functional architecture. A suitably de-semanticized version of Pylyshyn's notion of functional architecture can be seen as the application of the functional account of computing mechanisms to stored-program computers. (However, there remains an important difference between Pylyshyn's functional architecture and the functional account of computing mechanisms: Pylyshyn speaks interchangeably of the hardware and of the "virtual architecture" of a computing mechanism as being their functional architecture, whereas the functional account draws a sharp distinction between the two and accounts for the latter in terms of the former. See Piccinini 2003j for details.) Because of Pylyshyn's restrictive use of his notion of functional architecture and his reliance on semantic properties to individuate computational states, though, his account does not seem to improve on Fodor's account in satisfying our desiderata.

systems don't compute by invoking their functional analysis (or lack thereof), which is different from that of computing mechanisms. Planetary systems, hurricanes, stomachs, etc., even when they are functionally analyzed and have the function of yielding certain outputs in response to certain inputs (as in the case of stomachs), do not manipulate strings of tokens, i.e. inputs and outputs with the functional properties relevant to computing. They also lack the relevant kinds of components (e.g., input devices, output devices, memories, and processors) functionally organized in the appropriate way. Accordingly, the functional account properly counts all paradigmatic examples of non-computing systems as such.¹⁷

3. *Computation is observer-independent.* The functional account of computing mechanisms does not rely on “interpreting” systems in ways that are potentially observer-relative. Either something has a certain function or it doesn't, depending on what functional analysis applies to it. For example, either something is a memory register or not, an arithmetic-logic unit or not, etc., depending on what it contributes to its containing mechanism. The functional analysis of a computing mechanism is no less observer-independent than any other functional analysis in biology or engineering. It can be discovered in biological organisms or in artifacts independently of the observer.

4. *Computations can go wrong.* The functional account of computing mechanisms explains what it means for a computing mechanism to miscompute, or make a mistake in a computation. Miscomputations are a kind of malfunction, i.e. events in which a functional system fails to fulfill its function. In the case of computing

¹⁷ Non-computing systems may be said to be “computational” in some looser senses than genuine computing mechanisms. I offered a taxonomy of those looser senses in Piccinini 2003c.

mechanisms, whose function is to compute, functional failure results in a mistake in the computation.

There are many kinds of miscomputations. The most obvious kind is hardware failure, i.e. failure of a hardware component to perform its function (specified by the functional analysis of the mechanism). Hardware failure may be due to the failure of a computing component, such as a logic gate, or of a non-computing component, such as a clock. Another kind of miscomputation, which applies to artifacts, may be due to a mistake in computer design, so that the designed mechanism does not in fact compute the function it was intended to compute. Again, the design mistake may be due to a computing component that does not compute what it was intended to compute or to a non-computing component that does not fulfill its function (e.g., a clock with a too short cycle time). Another kind of miscomputation may be due to a programming error, whereby instructions are either mistakenly written (and hence cannot be executed) or do not play their intended role within the program. Yet another kind may be due to the accumulation of round-off errors in the finite precision arithmetic that computer processors employ. Finally, miscomputations may be due to faulty hardware-software interaction. A familiar example of this last type occurs when the execution of a program requires more memory than the computer has physically available. When there is no more memory available, the computer “freezes” without being able to complete the computation.¹⁸

5. *Some computing mechanisms are not computers.* The functional account of computing mechanisms explains why only some computing mechanisms are computers

¹⁸ For an early discussion of several kinds of computing mistakes by computing mechanisms, see Goldstine and von Neumann 1946. For a modern treatment, see Patterson and Hennessy 1998.

properly so called: only genuine computers are programmable. Computing mechanisms that are not programmable deserve other names, such as calculators, arithmetic-logic units, etc., and can still be differentiated from one another based on their computing power, which is determined by their functional organization.¹⁹

6. *Program execution is explanatory.* The functional account of computing mechanisms explicates the notion of explanation by program execution. It is a special kind of functional explanation that relies on the special kind of functional analysis that applies to soft programmable computers, namely, computers with processors that perform different operations in response to different strings of tokens of the appropriate kind (i.e., instructions). Program execution is a process by which a certain part of the mechanism, the program, affects a certain other part of the mechanism, the processor, so that the processor performs appropriate operations on a certain other part of the mechanism, the data. A mechanism must be functionally analyzable in this way to be subject to explanation by program execution. Explanation by program execution is the most interesting genus of the species of explanation by appeal to the computation performed by a mechanism. Appealing to the computation performed by a mechanism is explanatory in so far as the mechanism is a computing mechanism, i.e. a mechanism subject to the kind of functional analysis described in section 2.

4 Conclusion

The functional account of computing mechanisms is a viable account of what it means for a mechanism to compute. It satisfies the desiderata of an account of computing

¹⁹ For detailed comparisons of the different computing power of different computing mechanisms based on their functional analysis, see Piccinini 2003i and 2003j.

mechanisms. It allows us to formulate the question of whether a mechanism computes as an empirical hypothesis, to be decided by looking at the functional organization of the mechanism. It allows us to formulate a clear and useful taxonomy of computing mechanisms and compare their computing power (Piccinini 2003i and 2003j). I submit that the functional account of computing mechanisms can be used profitably in discussing computational theories of mind and brain, and that it constitutes an improvement over existing accounts of computing mechanisms that is valuable in its own right.

References

- Allen, C., M. Bekoff, et al., Eds. (1998). Nature's Purposes: Analysis of Function and Design in Biology. Cambridge, MA, MIT Press.
- Ariew, A., R. Cummins, et al., Eds. (2002). Functions: New Essays in the Philosophy of Psychology and Biology. Oxford, Oxford University Press.
- Buller, D. J., Ed. (1999). Function, Selection, and Design. Albany, State University of New York Press.
- Bechtel, W. and R. C. Richardson (1993). Discovering Complexity: Decomposition and Localization as Scientific Research Strategies. Princeton, Princeton University Press.
- Chalmers, D. J. (1996b). "Does a Rock Implement Every Finite-State Automaton?" Synthese **108**: 310-333.
- 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. S. and T. J. Sejnowski (1992). The Computational Brain. Cambridge, MA, MIT Press.
- Copeland, B. J. (1996). "What is Computation?" Synthese **108**: 224-359.
- Corcoran, J., W. Frank, et al. (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.
- Cummins, R. (1975). "Functional Analysis." Journal of Philosophy **72**(20): 741-765.
- 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. (1989). Meaning and Mental Representation. Cambridge, MA, MIT 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.
- Deutsch, J. A. (1960). The Structural Basis of Behavior. Chicago, University of Chicago Press.
- Dreyfus, H. L. (1979). What Computers Can't Do. New York, Harper & Row.
- 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. (1990). A Theory of Content and Other Essays. Cambridge, MA, MIT Press.
- Fodor, J. A. (1998). Concepts. Oxford, Clarendon Press.
- Goldstine, H. H. and J. von Neumann (1946). On the Principles of Large Scale Computing Machines. Princeton, Institute for Advanced Studies.
- Haugeland, J. (1978). "The Nature and Plausibility of Cognitivism." Behavioral and Brain Sciences **2**: 215-260.
- Patterson, D. A. and J. L. Hennessy (1998). Computer Organization and Design: The Hardware/Software Interface. San Francisco, Morgan Kaufman.
- Piccinini, G. (2003c). "Is Everything a Turing Machine, and Does It Matter to the Philosophy of Mind?" Manuscript submitted for publication.
- Piccinini, G. (2003d). "Functionalism and Computationalism 1: Mental States." Manuscript submitted for publication.
- Piccinini, G. (2003e). "Functionalism and Computationalism 2: Mental Contents." Manuscript submitted for publication.
- Piccinini, G. (2003f). "What Is a Computational State?" Manuscript submitted for publication.
- Piccinini, G. (2003g). "The Mind as Neural Software: Functionalism, Computationalism, and Computational Functionalism." Manuscript submitted for publication.
- Piccinini, G. (2003i). "Components of Computing Mechanisms and their Peculiar Functions." Manuscript submitted for publication.
- Piccinini, G. (2003j). "Computers." Manuscript submitted for publication.
- Piccinini, G. (2003l). "Epistemic Divergence and the Publicity of Scientific Methods." Studies in the History and Philosophy of Science **34**(3): 597-612.
- Preston, B. (1998). "Why is a Wing Like a Spoon? A Pluralist Theory of Function." The Journal of Philosophy **XCV**(5): 215-254.
- Putnam, H. (1960). Minds and Machines. Dimensions of Mind: A Symposium. S. Hook. New York, Collier: 138-164.
- Putnam, H. (1967). Psychological Predicates. Art, Philosophy, and Religion. Pittsburgh, PA, University of Pittsburgh Press.
- Putnam, H. (1988). Representation and Reality. Cambridge, MA, MIT Press.
- Pylyshyn, Z. W. (1984). Computation and Cognition. Cambridge, MA, MIT Press.
- Scheutz, M. (1999). "When Physical Systems Realize Functions ..." Minds and Machines **9**: 161-196.
- 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.
- Shagrir, O. (1997). "Two Dogmas of Computationalism." Minds and Machines **7**(3): 321-344.
- Wells, A. J. (1998). "Turing's Analysis of Computation and Theories of Cognitive Architecture." Cognitive Science **22**(3): 269-294.