Computational Mechanisms and Models of Computation

Marcin Miłkowski
Institute of Philosophy and Sociology,
Polish Academy of Sciences (Poland)

Résumé : to be added

Abstract: In most accounts of realization of computational processes by physical mechanisms, it is presupposed that there is one-to-one correspondence between the causally active states of the physical process and the states of the computation. Yet such proposals either stipulate that only one model of computation is implemented, or they do not reflect upon the variety of models that could be implemented physically. In this paper, I claim that mechanistic accounts of computation should allow for a broad variation of models of computation. In particular, some non-standard models should not be excluded *a priori*. The relationship between mathematical models of computation and mechanistically adequate models is studied in more detail.

In this paper, I analyze the relationship between computational mechanisms—physically instantiated computers—and models of computation. Models of computation are used in various fields, including, but not limited to, computer science, information technology, and computational modeling in cognitive science. They are used to analyze various relationships between algorithms, to determine computational capabilities of various machines, to prove theorems about computational complexity of algorithms, and so forth.

I distinguish a special class of models of computation, namely *mechanistically interpretable* models, and defend the claim that only some of the models usually perused in computer science can be mechanistically adequate models of physical computations; most of them need to be accompanied by additional specifications of the mechanism, which I call *instantiation blueprints*. It is plausible that both are needed in most computational explanations of cognitive phenomena.

The structure of the paper is as follows. In the first section, I introduce the notion of a model of computation and sketch some requirements that a satisfactory theory of implementation should meet. Next, the modeling relationship between the model of the computational mechanism and its physical instantiation is analyzed in terms of weak and strong equivalence. In the third section, I argue that a mechanistically adequate model is required for strong equivalence to obtain. At the same time, I admit that most models, even if mechanistically adequate, are not complete models of mechanisms, and for this reason, they are accompanied by background considerations, or instantiation blueprints. The fourth section gives a short case study of a specific model of computation, namely a Kolmogorov-Uspensky Machine, and shows how the machine was implemented using a biological substrate—slime mold. I conclude by pointing to other, less exotic examples studied in cognitive science.

1 Models of computation

Computer science abounds with talk of models of computation. For example, entering the query "model of computation" in the online database of papers in computer science, CiteSeerX [`http://citeseerx.psu.edu`], hosted by the University of Pennsylvania, returns around 28 thousand documents, while "realization of computation" returns around 800 results. In most cases, a model of computation is a description of a computation in abstract terms. The term "model of computation" is usually used without any definition specifying its intension [Savage 1998]; [Fernández 2009]. I will also introduce the term with a definition by enumeration: the conventional models of computation include Turing machines, lambda calculus, Markov algorithms, or finite state machines. These are conventional, as they have been shown to be equivalent with regard to all the functions computable by entities implementing such models (another name for the notion of equivalence in question is 'Turing-equivalence').

Notably, there are also unconventional models; some of them compute functions incomputable for a universal Turing machine. Note that they compute such functions only in principle; the question whether they can do so physically is still undecided. Some unconventional models simply rely on a fairly non-standard method of implementation (chemical computation, wetware computation, DNA computation), some are not digital (various analog computers). In these unconventional models, quantum computation seems to be quite prominent; one of the reasons being that certain quantum algorithms are of interest to security applications. That is, they seem to outperform all conventional machines in certain respects; for example, Shor's prime factorization algorithm for quantum computers [Shor 1997] could be detrimental to current encryption algorithms that rely on the assumption that prime factorization is computationally very expensive.

Models of computation studied in computability theory (I refer to them below as “models of computation in the proper sense”) are to be distinguished from computational models, for example those used to study climate changes or the weather. The latter are simply used to model a given phenomenon using computational means [Humphreys 2003], [Winsberg 2010]. They do not describe a computational process; rather they are used to describe something computationally. In this paper, I refer to the first kind of models of computation, not to computational models in the latter sense (though, admittedly, some models of computation will be at the same time computational models, for example in computational neuroscience or psychology, e.g., [Lewandowsky & Farrell 2011]).

Models of computation in the proper sense are mostly formal (see also [Miłkowski 2011]). This means that they do not usually describe the physical realization that they would require to be physically implemented. For example, it would be quite peculiar to find that a paper about Turing machines delves into engineering questions, such as which physical materials would serve the role of the potentially infinite tape best. Note that this is less true of unconventional computation such as DNA computation. Seminal papers about DNA computation include both the mathematical description of the model (what elementary operations are possible, how binary strings will be represented, etc.), as well as specification of the biological, or molecular, basis for the physical computation [Adleman 1994], [Boneh, Dunworth *et al.* 1996]. However, a model of computation might well be physically or at least technologically impossible. For example, an analogue computer that computes over real numbers (for example, as a neural network, see [Siegelmann & Sontag 1994]) might be physically impossible, because infinite resolution of measurement would be required. Similarly, nobody would actually try to build a computer near a rotating black hole simply to prove that hypercomputation is physically possible [Etesi & Nemeti 2002].

Surprisingly, however, most philosophical accounts of realization (or implementation) of computational processes by physical entities seem to ignore the fact that models of computation are not only described in a mathematical, formal manner. There are several types of philosophical accounts of computational realization [Piccinini 2010]; in this paper, a structural or mechanistic conception [Chalmers 2011], [Piccinini 2007], [Miłkowski 2013] will be assumed. Yet other accounts, such as the formal syntactic account [Pylyshyn 1984] or semantic conception [O’Brien & Opie 2009] usually similarly ignore the non-formal part of realization. In addition, most defenders of the account of implementation via mechanisms or causal structures seem to presuppose that there is one-to-one correspondence between the causally-active states of the physical process and the states of the computation, as described by its model.

Such proposals either stipulate that there be only one model of computation for implementation (for example, Chalmers stipulates that a combinatorial finite state machine should be used as it could be easily matched to physical states of any system), or they do not reflect upon the possible variety of models of computation being implemented physically. Piccinini, for example, relies on string-rewriting models of computation, which seem to exclude all state-transition models. But the mechanistic account of computation should also allow for a broad variety of models of computation. In particular, non-standard models should not be excluded *a priori*. For example, it should not be stipulated that only models equivalent (in terms of the set of functions computed) to a universal Turing machine, or a less powerful mechanism, can be implemented physically. Why should a philosopher decide *a priori* that the field of unconventional or hyper-Turing computation is pseudoscientific? In this respect, philosophers should adhere to the principle of *transparent computationalism* [Chrisley 2000]. In section three, I will show that this requirement is easy to fulfill when one takes into account the duality of descriptions of models of computation in the field of unconventional computation.

Another important requirement for an adequate theory of implementation is that it should appeal to aspects of what is called "implementation" in science. This might sound trivial, but take an example from cognitive science—the notable three-level account of explanation of computers developed by David Marr [Marr 1982]. In this account, the lowest level is called "implementation". Broadbent, in his criticism of connectionist models, claims that these models are only relevant to the implementation level [Broadbent 1985]: connectionist networks are mere realizations of computations best described classically, not as quasi-neural process in artificial neural networks. In response, Rumelhart & McClelland claim that they too are only interested in algorithms implemented, while they do not yet know about implementation in the brain [Rumelhart & McClelland 1985]. I don't want to question here whether connectionism is right or not. The point is that both parties in the controversy seem to agree that implementation is (1) required for the physical computation to occur; and (2) *not* specified by algorithms *alone*. Broadbent simply assumes that there are no algorithms, or abstract accounts of computation, that would make connectionist models different from classical models. Whether he is right or not is a matter for another paper.

Let me summarize. By referring to how the term 'model of computation' is used in various fields dealing with computers, I have implied that there are two aspects to such models: the formal account of computable functions, and the non-formal account of physical realizations. Both aspects, I have claimed, should be explicated by a descriptively accurate account of computational implementation in philosophy of science. In addition, the account should not presuppose that there is a single preferred mathematical account of computation, and thus should remain neutral in this regard.

2 Modeling relationship

The models of computation introduced in the previous section have a clearly representational function: they are descriptions of computations. Depending on the purpose of the model, they might describe a computational system in terms of functions computable by the system; they might also give faithful descriptions of how the function is being computed. In the first instance, the model is *weakly equivalent* to the system in question [Fodor 1968]: the model computes the same function, i.e., it has the same set of inputs and outputs as the modeled system. In the latter case, it is *strongly equivalent*: it also describes how inputs are transformed into outputs.

Weakly equivalent models can be useful, for example in proving that different types of machines can compute the same set of functions. For this reason, a universal Turing machine can be used to describe the operation of other models of computation; if one is successful in describing a machine as weakly equivalent to a Turing machine, the result constitutes a proof of Turing-computability of its functions.

Note that non-formal aspects of models are discarded when assessing the relation of weak equivalence. Functions of the standard desktop computer will be Turing-computable without there being anything that literally corresponds to the potentially infinite tape, writing head, or the elementary operations of the Turing machine. Of course, the work of the desktop PC is in many respects analogous to that of the Turing machine, but there are parts of the desktop PC that do not have to correspond to the latter without thereby violating the condition of weak equivalence. For example, there is no CPU in the Turing machine, not to mention expansion cards or graphics boards.

Strongly equivalent models are more significant for cognitive science, because cognitive science models do not merely describe the function being computed (which would correspond, roughly, to what's meant by "competence" in cognitive research) but also the way it is computed [Miłkowski 2013]. In addition, empirical evidence needed to distinguish between strongly and weakly equivalent models arguably has to refer to non-formal properties of models of computation, i.e., to physical features of the realization that influence, for example, the speed of processing or the amount of available working memory. Without this reference, empirical evidence such as response times, used in cognitive science and neuroscience to support hypotheses about the organization of computational architecture and complexity of algorithms being implemented [Sternberg 1969, 2011], [Posner 2005], would be useless in determining which of the proposed accounts of cognitive processing is correct. The practice of using such evidence gives support to the mechanistic account of computational explanation [Miłkowski 2011].

Note however that the distinction between weakly and strongly equivalent models does not rely on one's adherence to neo-mechanistic philosophy of science (see next section for details on the mechanistic account), or on

Fodor's functionalist views on psychology. It can be argued that other theorists also posited similar types of modeling relationships; for example, Rosen distinguished between *simulation* (which corresponds to weak equivalence) and modeling proper (strong equivalence), while being strongly opposed to mechanism [Rosen 1991, chap. 7].

3 Mechanistically adequate models

One of the most widely endorsed views in the philosophy of special sciences is neo-mechanism [Machamer, Darden *et al.* 2000], [Craver 2007], [William 2008]. According to this view, to explain a phenomenon is to elucidate the underlying mechanism. Mechanistic explanation is a species of causal explanation, and explaining a mechanism involves describing its causal structure. While mechanisms are defined in various ways by different authors, the core idea is that they are organized systems, comprising causally relevant component parts and operations (or activities) thereof. Components of the mechanism interact and their orchestrated operation contributes to the capacity of the mechanism.

This neo-mechanistic framework has also been applied to computation [Piccinini 2007], [Miłkowski 2011]. Piccinini focuses on digital effective computation and has only recently admitted the need to accommodate unconventional models, all under the umbrella of "generic computation" [Piccinini & Scarantino 2010]. His account of computational models is based on abstract string rewriting: computation is construed as rewriting strings of digits [Piccinini 2007, 501]. But this violates the principle of transparent computationalism. In addition, Piccinini has not developed any detailed account of how "generic computation" is to be understood in mechanistic terms.

There are several mechanistic norms of explanation that are particularly important in explaining computation. These are: the requirement of completeness; the requirement of specifying the capacity of the mechanism; and the requirement that the model contain only causally relevant entities. I will explicate these below. But first, let me elaborate on how computation can be conceived in a mechanistic manner.

Computation is generally equated with information-processing, and this is why the notion of information is crucial in models of computation for the account of implementation: a computational process is one that transforms the stream of information it has as input into a stream of information for output. During the transformation, the process may also appeal to information that is part of the very same process (internal states of the computational process). Information may be, although need not be, digital—that is, there is only a finite, denumerable set of states that the information vehicle can have and that the computational process is able to recognize, as well as produce as output. (In analogue computing, the range of values in question is restricted, but continuous, i.e., infinite.) By "information" I mean quantitative structural-information-content in MacKay's sense of the term: the physical vehicle must be capable of taking at least two different states to be counted as information-bearing (for a detailed explication of the notion of structural-information-content and its relation to selective-information, i.e., Shannon information, see [MacKay 1969]).

Computational explanations, according to the mechanistic account, are constitutive mechanistic explanations: they explain how the computational capacity of a mechanism is generated by the orchestrated operation of its component parts. To say that a mechanism implements a computation is to claim that the causal organization of the mechanism is such that the input and output information streams are causally linked and that this link, along with the specific structure of information processing, is completely described. Importantly, the link can be cyclical and as complex as one could wish.

Understanding computation as information-processing offers several advantages over more traditional accounts, especially because it furnishes us with criteria for strong equivalence. Namely, the strongly equivalent model of computation C is best understood as the mechanistically adequate model of C. To describe information-processing one usually employs models of computation used in computer science, mentioned in section 1 of this paper. To be explanatorily relevant and descriptively accurate about a given physical computational mechanism, the model chosen has to be mechanistically adequate. Note that this requirement for descriptive accuracy and explanatory relevance does not mean that all models in computer science have to be mechanistically adequate. Rather, wherever weak equivalence is enough for one's purposes, mechanistic explanation might be spurious. For example, a causal mechanistic explanation is redundant and uninformative when one wants to explain why Markov algorithms are equivalent to Turing machines. In such a case, formal modeling is enough.

If physical instantiation is relevant (for example, in neuroscience or DNA computing) then we use the mechanistic explanation. The description of a *mechanistically adequate model* of computation comprises *two* parts: (1) an abstract specification of a computation, which should include all the causally relevant variables; (2) a complete blueprint of the mechanism on three levels of its organization. I will call the first part the *formal model of the mechanism* and the second the *instantiation blueprint* of the mechanism, for lack of a better term. While it should be clear that a formal model is required, it is probably less evident why the instantiation blueprint is also part of the mechanistically adequate model. One of the norms of the mechanistic explanation is that descriptions of mechanisms be complete [Craver 2007]. Of course, this does not mean that one has to include every possible piece of information in the description of the mechanism (also called the model of mechanism, cf. [Glennan 2005]). It must be complete as a causal model, i.e., all causally relevant parts and operations should be specified without gaps or placeholder terms.

But formal models cannot function as complete causal models of computers. For example, it is not enough to know that my laptop is an instantiation of a von Neumann machine, or even that it runs Linux on an Intel x86 family of processors. To explain why it executes the computation of one million digits of π in such-and-such a time, one needs to know the details of the hardware, such as the frequency of the CPU's clock. Only an appeal to the non-formal part of the model can supply such data. As I pointed out in section 1, unconventional models of computation might furnish us with parts of instantiation blueprints at least, by specifying that DNA computation is used to "apply a sequence of operations to a set of strands in a test tube" [Boneh, Dunworth *et al.* 1996, 85], that operations include the melting of double strands of DNA to dissolve them, that the double strands of DNA are used to store information because single ones are fragile, and so forth.

The model of the physical computational process, at the level of detail required to support the claim that it is strongly equivalent, typically includes an instantiation blueprint and an abstract model of computation. Of course, it is possible to create a formalism of computation that would include timing and *other* implementation factors so that it would, by itself, comprise anything that common engineering blueprints typically contain. However, scientists usually describe the properties of the abstract mathematical model separately from physical machines, even if both are included in the same paper.

Another norm of the mechanistic explanation is that the model of the mechanism should identify the capacity of the mechanism that is explained. There are no mechanisms *as such*; there are only mechanisms *of* something—and here that something is the formal model of computation. By providing the instantiation blueprint of the system, we explain the physical exercise of its capacity, or competence, abstractly specified in the formal model. In other words, the completeness norm requires that we include the instantiation blueprint in our model, and the specification norm tells us to specify the formal model of the computation. But, as stated above, the model has to be causally relevant as well. This leads to the observation that only mechanistically interpretable models of computation might be used in the mechanistically adequate descriptions of mechanisms.

Let me elaborate. Take a standard programming language, such as C++, used to write various programs. In my usage of the term 'model of computation', C++ definitely qualifies as one such model. But it is not a mechanistically interpretable model for compiled C++ programs executed on my desktop machine, because a compiler is needed to translate C++ to machine code. In other words, the operation of my PC cannot be described in terms of the primitive operations given by C++ specification. In particular, even some operations specified in the source code might have no counterpart in the compiled code. Some inefficient code might be optimized by a compiler, and even some variables that seem to contain values (for the programmer) might not be identifiable in the binary version of the code [Scheutz 1999]. One could retort that it is the instantiation blueprint, in this case, that includes the specification of the compiler; in particular, specification in the way it is supposed to interpret C++ instructions in terms of the machine code. But this would be counterintuitive: this kind of specification is still fairly abstract, and no physical instantiation is ever mentioned when describing the relationship between machine instructions and higher-level languages.

For this reason, I propose to call models such as C++ programs, without their compilers, *mechanistically incomplete.* To completely describe how they work on a particular machine, we would also need to know the supporting formal model of the compiler, interpreter, or some other tool that relates the higher-level abstraction and the mechanistically interpretable formal model of the computation. For example, on Intel x86 only the assembly language is mechanistically interpretable; all other languages are mechanistically incomplete models that should be accompanied by a description that shows the relationship between them and the assembly language. Notice that even code developers do not usually know this relationship in any great detail (unless they have written or read the code of the compiler, which can but does not have to be written in the assembler as well). In everyday speech, we talk about higher-level programs being executed, while strictly speaking this is only an idealization (albeit a useful one). Literally, only a compiled or interpreted program is executed.

This concludes my survey of the mechanistic theory of implementation. Let me turn to a case study.

4 Physarum machine

In this section, I will analyze how slime mold was used by Adamatzky to implement a Kolmogorov-Uspensky Machine (KUM). KUM was introduced by Kolmogorov and developed later with his student, Uspensky, and is named after its creators. Notice that KUM is (weakly) equivalent to recursive functions [Kolmogorov & Shiryaev 1993]. The main reason why KUM was used is that interpreting physical processes that do not contain fixed structures or cellular-automaton structures is troublesome [Adamatzky 2007, 455]. In other words, Adamatzky seems to presuppose that a Turing machine would not have been mechanistically adequate in this case; whereas KUM, as a formal model, is a mechanistically adequate model in the sense used in this paper.

KUM is a prominent model of real-life computation. According to Adamatzky, the operation of the machine is as follows:

KUMs are defined on a colored/labeled undirected graph with bounded degrees of nodes and bounded number of colors/labels. KUMs operate, modifying their storage, as follows:

(1) Select an active node in the storage graph;

(2) Specify local active zone, i.e., the node's neighborhood;

(3) Modify the active zone by adding a new node with the pair of edges, connecting the new node with the active node;

(4) Delete a node with a pair of incident edges;

(5) Add/delete the edge between the nodes.

A program for KUM specifies how to replace the neighborhood of an active node with a new neighborhood, depending on the labels of edges connected to the active node and the labels of the nodes in proximity of the active node [Adamatzky 2007, 456].

The instantiation blueprint contains the slime mold *Physarum polycephalum*. Note that this machine was the first physical instantiation of KUM (or at least the first documented instantiation). The slime mold was used in a vegetative stage, as plasmodium. This is a single cell, visible to the naked eye, and it propagates and searches for nutrients when placed on appropriate nutrients (such as agar gel). It has previously been shown that this simple organism has the ability to find the minimum-length solution between two points in a labyrinth [Nakagaki, Yamada *et al.* 2000]. Adamatzky built a *Physarum machine* that contained wet filter paper with colored oat flakes. The machine had two kinds of nodes: stationary (oat flakes) and dynamic (origins of protoplasmic veins); the edge of the machine is a strand or vein of protoplasm connecting any of the nodes. The data and program are represented by the spatial configuration of stationary nodes, while results are provided by the configuration of dynamical nodes and edges. (In KUMs, the whole graph structure is the result of the computation.)

Plasmodium, as we would expect, does not stop operating when the required result has been attained; it continues until the nutrients are depleted. For this reason, Adamatzky supposes that the Physarum machine halts when all data-nodes are utilized. In addition, there is an active zone that is simply a growing node in plasmodium. The graphs of the Physarum machine have bounded connectivity. However, one property of KUM is not satisfied: not all nodes are uniquely addressable (coloring the oat flakes was supposed to help with this).

Adamatzky goes to detail the basic operations of the Physarum machine (INPUT, OUTPUT, GO, HALT); but, in his 2007 paper, no single algorithm is actually shown to have run on the Physarum machine. For this reason, there is no causal model of the plasmodium that would correspond to the Physarum machine; and, more importantly, the capacity of the mechanism has not been specified in detail. We know that the Physarum machine is supposed to be a realization of KUM, but no information about *what* is computed is actually given. In other words, the description of the mechanism is incomplete and violates mechanistic norms; it is just a sketch of the mechanism in the article cited. However, in his later book [Adamatzky 2010], Adamatzky uses Physarum machines to solve very simple tasks and explains how to program them with light.

All in all, it is evident that the description of the implementation of KUM seems to be of the form that I introduced in the previous section: a formal model accompanied by an instantiation blueprint, which is biological in this case.

5 Conclusion

In this paper, I analyzed the relationships of weak and strong equivalence between models of computation. I claimed that mechanistic explanations require that mechanistically adequate models of computation include two parts: formal models of the computation in question, and instantiation blueprints. In contrast to earlier work on mechanisms, my account does not violate the principles of transparent computationalism and yet avoids being excessively liberal. Notice that the 2007 description of the Physarum machine, from the mechanistic point of view, is not a satisfactory mechanistic model. It does not specify the computation, and it was not shown that a causal model of the way in which plasmodium behaves could be used to predict the results of computation in KUM, or that results of the manipulation of plasmodium could be predicted by using our knowledge of KUM initial configuration. For this reason, the Physarum machine remains a bold hypothesis, which is only later made plausible by showing how it can be programmed.

But let me put unconventional models to the side. The reason I introduced them is that they make it clear that implementation is not a matter of formal models only. Rather, hardware is very important in the functioning of computers. There is a lot of evidence that models of computation in neuroscience [Piccinini & Bahar 2013] and cognitive science [Miłkowski 2013] include some specifications of the constitutive parts of the mechanism, which I called instantiation blueprints. There are reasons, therefore, to believe that the mechanistic theory of implementation of computation, here only roughly depicted, is both normatively and descriptively adequate.

Acknowledgments

Work on this paper was financed by a Polish Ministry of Science Habilitation Grant # N N101 138039, and the National Science Centre OPUS Grant under the decision DEC-2011/03/B/HS1/04563. The author wishes to thank the audience of the CLMPS in Nancy in 2011, where a shorter version of this paper was presented.

Bibliography

Adamatzky, Andrew [2007], Physarum machine: Implementation of a Kolmogorov-Uspensky machine on a biological substrate, *Parallel Processing Letters*, 17(04), 455–467, doi:10.1142/S0129626407003150.

—— [2010], *Physarum Machines: Computers from Slime Mould*, World Scientific series on nonlinear science., Series A, Monographs and treatises, vol. 74, Singapore: World Scientific.

Adleman, Leonard M. [1994], Molecular computation of solutions to combinatorial problems, *Science*, 266(5187), 1021–1024, doi:10.1126/science.7973651.

Boneh, Dan, Dunworth, Christopher, Lipton, Richard J., & Sgall, Jiří [1996], On the computational power of DNA, *Discrete Applied Mathematics*, 71(1–3), 79–94, doi:10.1016/S0166-218X(96)00058-3.

Broadbent, Donald [1985], A question of levels: Comment on McClelland and Rumelhart, *Journal of Experimental Psychology: General*, 114(2), 189–190, doi:10.1037/0096-3445.114.2.189.

Chalmers, David J. [2011], A computational foundation for the study of cognition, *Journal of Cognitive Science*, 12, 325–359.

Chrisley, Ronald [2000], Transparent computationalism, in: *New Computationalism*, edited by M. Scheutz, Sankt Augustin: Academia Verlag, Conceptus-Studien, vol. 14, 105–121.

Craver, Carl F. [2007], *Explaining the Brain. Mechanisms and the mosaic unity of neuroscience*, New York: Oxford University Press.

Etesi, Gabor & Nemeti, Istvan [2002], Non-Turing computations via Malament-Hogarth space-times: General relativity and quantum cosmology, *International Journal of Theoretical Physics*, 41(2), 341–370, doi:10.1023/A:1014019225365.

Fernández, Maribel [2009], *Models of Computation: An Introduction to Computability Theory*, Undergraduate Topics in Computer Science, London: Springer, doi:10.1007/978-1-84882-434-8.

Fodor, Jerry A. [1968], *Psychological Explanation: An Introduction to the Philosophy of Psychology*, New York: Random House.

Glennan, Stuart S. [2005], Modeling mechanisms, *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences*, 36(2), 443–464, doi:10.1016/j.shpsc.2005.03.011.

HUMPHREYS, Paul [2003], Computational models, *Philosophy of Science*, 69, 1–11.

KOLMOGOROV, Andreĭ N. & SHIRYAEV, Albert N. [1993], *Selected Works of A.N. Kolmogorov. Vol. III: Information Theory and the Theory of Algorithms*, Mathematics and its Applications, vol. 27, Dordrecht: Kluwer.

LEWANDOWSKY, Stephan & FARRELL, Simon [2011], *Computational Modeling in Cognition: Principles and practice*, Thousand Oaks: Sage Publications.

MACHAMER, Peter, DARDEN, Lindley, & CRAVER, Carl F. [2000], Thinking about mechanisms, *Philosophy of Science*, 67(1), 1–25.

MACKAY, Donald M. [1969], *Information, Mechanism and Meaning*, Cambridge, MA: MIT Press.

MARR, David [1982], *Vision. A Computational Investigation into the Human Representation and Processing of Visual Information*, New York: Freeman.

MIŁKOWSKI, Marcin [2011], Beyond formal structure: A mechanistic perspective on computation and implementation, *Journal of Cognitive Science*, 12(4), 359–379.

—— [2013], *Explaining the Computational Mind*, Cambridge, MA: MIT Press.

NAKAGAKI, Toshiyuki, YAMADA, Hiroyasu, & TÓTH, Ágota [2000], Intelligence: Maze-solving by an amoeboid organism, *Nature*, 407(6803), 470, doi:10.1038/35035159.

O'BRIEN, Gerard & OPIE, Jon [2009], The role of representation in computation, *Cognitive Processing*, 10(1), 53–62, doi:10.1007/s10339-008-0227-x.

PICCININI, Gualtiero [2007], Computing mechanisms, *Philosophy of Science*, 74(4), 501–526, doi:10.1086/522851.

—— [2010], Computation in physical systems, in: *The Stanford Encyclopedia of Philosophy*, edited by E. N. Zalta, fall 2012 edn., URL `plato.stanford.edu/archives/fall2012/entries/computation-physicalsystems/`.

PICCININI, Gualtiero & BAHAR, Sonya [2013], Neural computation and the computational theory of cognition, *Cognitive Science*, 37(3), 453–488, doi:10.1111/cogs.12012.

PICCININI, Gualtiero & SCARANTINO, Andrea [2010], Computation vs. information processing: Why their difference matters to cognitive science, *Studies in History and Philosophy of Science Part A*, 41(3), 237–246, doi:10.1016/j.shpsa.2010.07.012, computation and cognitive science.

POSNER, Michael I. [2005], Timing the brain: Mental chronometry as a tool in neuroscience, *PLoS biology*, 3(2), e51, doi:10.1371/journal.pbio.0030051.

Pylyshyn, Zenon W. [1984], *Computation and cognition: Toward a foundation for cognitive science*, Cambridge, MA: MI.

Rosen, Robert [1991], *Life Itself: A comprehensive inquiry into the nature, origin, and fabrication of life*, New York: Columbia University Press.

Rumelhart, David E. & McClelland, James L. [1985], Levels indeed! A response to Broadbent, *Journal of Experimental Psychology: General*, 114(2), 193–197, doi:10.1037/0096-3445.114.2.193.

Savage, John E. [1998], *Models of Computation: Exploring the power of computing*, Reading, MA: Addison Wesley.

Scheutz, Matthias [1999], The ontological status of representations, in: *Understanding Representation in the Cognitive Sciences*, edited by A. Riegler, M. Peschl, & A. von Stein, Springer, 33–38, doi:10.1007/978-0-585-29605-0_4.

Shor, Peter W. [1997], Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Journal on Computing*, 26(5), 1484–1509, doi:10.1137/S0097539795293172.

Siegelmann, Hava T. & Sontag, Eduardo D. [1994], Analog computation via neural networks, *Theoretical Computer Science*, 131(2), 331–360, doi:10.1016/0304-3975(94)90178-3.

Sternberg, Saul [1969], The discovery of processing stages: Extensions of donders' method, *Acta Psychologica*, 30(0), 276–315, doi:10.1016/0001-6918(69)90055-9.

—— [2011], Modular processes in mind and brain, *Cognitive neuropsychology*, 28(3–4), 156–208, doi:10.1080/02643294.2011.557231.

William, Bechtel [2008], *Mental Mechanisms*, New York: Routledge.

Winsberg, Eric B. [2010], *Science In the Age of Computer Simulation*, Chicago; London: University of Chicago Press.