Synthese (2018) 195:3477–3497
https://doi.org/10.1007/s11229-017-1380-5

Functional individuation, mechanistic implementation: the proper way of seeing the mechanistic view of concrete computation

Dimitri Coelho Mollo[1,2]

Received: 27 November 2016 / Accepted: 15 March 2017 / Published online: 21 March 2017

Abstract I examine a major objection to the mechanistic view of concrete computation, stemming from an apparent tension between the abstract nature of computational explanation and the tenets of the mechanistic framework: while computational explanation is medium-independent, the mechanistic framework insists on the importance of providing some degree of structural detail about the systems target of the explanation. I show that a common reply to the objection, i.e. that mechanistic explanation of computational systems involves only weak structural constraints, is not enough to save the standard mechanistic view of computation—it trivialises the appeal to mechanism, and thus makes the account collapse into a purely functional view. I claim, however, that the objection can be put to rest once the account is appropriately amended: computational individuation is indeed functional, while mechanistic explanation plays a role in accounting for computational implementation. Since individuation and implementation are crucial elements in a satisfying account of computation in physical systems, mechanism keeps its central importance in the theory of concrete computation. Finally, I argue that my version of the mechanistic view helps to provide a convincing reply to a powerful objection against non-semantic theories of concrete computation: the argument from the multiplicity of computations.

Keywords Philosophy of computation · New mechanism · Concrete computation · Multiplicity of computations

✉ Dimitri Coelho Mollo
dimitri.coelho_mollo@kcl.ac.uk; dimitri.coelho.mollo@hu-berlin.de

1 Department of Philosophy, King's College London, Strand, London WC2R 2LS, UK

2 Institut für Philosophie, Humboldt-Universität zu Berlin, Unter den Linden 6, 10117 Berlin, Germany

1 Introduction

The neo-mechanist approach to explanation in science has become increasingly popular in the past years, with successful applications in biology, psychology, and neuroscience. In many cases, New Mechanism has proved to be a promising framework for cashing out how explanations in the special sciences work, and what they should look like. The so far impressive track record notwithstanding, one must be careful when extending the approach to other fields of scientific endeavour, as over-extension may put at risk the internal coherence of the overall neo-mechanist picture.

Here I examine one of these risky applications of the neo-mechanistic framework: the mechanistic view of concrete computation. The attempt to use the tools provided by New Mechanism to account for computation in physical systems has been developed most forcefully by Piccinini (2007, 2015), Milkowski (2013), and Fresco (2014). The fruits to reap should this endeavour be successful are very significant: extending a successful approach to scientific explanation to a domain, computation, that has so far resisted satisfactory naturalisation and remains problematic when appealed to in scientific explanations.

However, the abstract nature of computational explanation introduces a tension in the neo-mechanistic framework, as Haimovici (2013) has pointed out. For one of the defining characteristics of New Mechanism is its insistence on the importance of providing some degree of structural detail about the mechanisms that contribute to explaining phenomena. This requirement seems to be at odds with the abstractness and medium-indepencence typical of computational explanation, in which very little, if any, structural detail is provided. Hence computational mechanists find themselves in a dilemma: either computational explanation is essentially incomplete, or, by enriching it with structural detail, its peculiar medium-independence, and the multiple realisability that falls from it, are lost.

In reply to Haimovici's dilemma, Piccinini (2015) claims that, despite the highly abstract nature of computational explanation, very weak structural constraints are nonetheless in place, thus saving the mechanistic status of computational explanation. I argue that Piccinini's reply is unsuccessful, for it trivialises the appeal to mechanism, making the account collapse into a purely functional view. The tension Haimovici points out is thereby still present, spelling trouble for the computational mechanist.

My aim in this paper is to dispel this tension. This involves amending how the mechanistic view of concrete computation is conceived of in the existing literature, especially for what regards the role played by the appeal to mechanism. While computational individuation is essentially functional, mechanisms play an important part in shedding light on computational implementation. Satisfying accounts of both computational individuation and computational implementation are crucial elements of a theory of how computations exist in the physical world. In this way, New Mechanism makes an essential contribution to our understanding of concrete computation, though for reasons other than the ones normally adduced by computational mechanists.

My approach to the mechanistic view of concrete computation has beneficial consequences for other central debates in philosophy of computation. In particular, I explore a promising account of computational individuation put forward by Dewhurst (2016), and I show that my proposal improves on it, staving off its main shortcomings.

Here is how I will proceed. In Sect. 2, I briefly introduce the neo-mechanist approach to scientific explanation, opting for one of its most general formulations. I then present the mechanistic view of concrete computation, in particular as developed by Gualtiero Piccinini, in Sect. 3, so as to bring to the fore the apparent tension that I aim to dissolve. Next, in Sect. 4, I examine Haimovici's objection to the view, which invites a ready reply by the computational mechanist. In Sect. 5 I show that the reply does not work, as it causes the mechanistic view to collapse into a purely functional view. Though this spells trouble to most computational mechanists, I argue that it is actually a welcome result, since computational individuation is indeed functional. This notwithstanding, an amended mechanistic view of concrete computation is still in the cards: I claim in Sect. 6 that the appeal to mechanism has an important role to play in an account of concrete computation. Finally, Sect. 7 tackles Dewhurst's theory of computational individuation and explores some consequences my approach has to issues regarding computational equivalence.

2 New mechanism

Piccinini (2007, 2015), Milkowski (2013), and Fresco (2014) rely on the neo-mechanist approach to explanation in order to provide an adequate account of computation in physical systems, i.e. concrete computation.

Motivation for New Mechanism comes partly from the failure of traditional approaches to explanation to do justice to the practice and explanatory aims of the special sciences, such as biology, psychology, and neuroscience. The guiding idea is that, rather than relying on laws of nature, the special sciences explain by means of breaking up a phenomenon of interest into its components, what they do, and how they are organised—i.e. by unveiling the mechanism underlying, producing, or maintaining the phenomena to be explained. These three ways in which mechanisms can be explanatory play an important role in ensuring that New Mechanism is able to accommodate the different kinds of phenomena that the special sciences tackle: not only causal chains, but also homeostatic systems, as well as systems ripe with feedback loops and complex interactions between components.

The notion of mechanism is normally left relatively vague. Narrowing it down too much risks excluding from the purview of the framework kinds of phenomena and explanations that are amenable to a broader understanding of mechanism. The notion is also ambiguous, having different meanings and usages.[1]

My employment of the term concerns exclusively the notion relevant to theories of scientific explanation. But even in this more restricted domain the notion of mechanism is understood in slightly different ways by different theorists. Often this leads to differences in the inclusiveness of the ensuing version of the neo-mechanistic framework. For the purposes of this paper, I will endorse one of the most inclusive understandings of mechanism—inasmuch as it places relatively few and vague constraints—put forward by Illari and Williamson (2012, p. 120):

[1] See Andersen (2014a, b), Levy (2013), Moss (2012).

A mechanism for a phenomenon consists of entities and activities organised in such a way that they are responsible for the phenomenon.

The bottom line of New Mechanism is the appeal to the physical components of systems, causings (their activities), and their organisation, in explaining how a target phenomenon comes about. Each of these notions—component, activity, organisation, and phenomenon—can be understood in different ways, leading to different understandings of mechanism. This can be seen as a feature of the account. The flexibility of its fundamental structure allows mechanistic explanations to be fitted to the fields or sub-fields of interest, without losing its generality.

The *explanandum* phenomenon is crucial in individuating the mechanism. Mechanisms are of, or for, a certain phenomenon. Mechanistic explanation starts with a phenomenon to be explained, and then generates the mechanism for that phenomenon. The decomposition of the mechanism into its components, activities and organisation proceeds with that in mind. What components must there be, so that the target phenomenon takes place? Which activities must those components engage in? How do their contributions come together in bringing about the phenomenon? These are the questions that scientists attempting to give a mechanistic explanation try to answer.

The components' activities have functions inside a mechanism, insofar as they make a contribution to the overall behaviour of the mechanism. Mechanisms may themselves be functional—they might have functions to perform in the context of an organism or artefact. I will refer to this notion of function, i.e. in terms of the causal roles of a component inside a system, as systemic functions (Cummins 1975).

Functional considerations play an important role in mechanistic explanation. In explaining the overall capacity of a mechanism, its decomposition proceeds by identifying the components of the mechanism as well as their systemic functions that help to bring about the overall behaviour. Structural properties are also relevant in mechanistic explanation. Roughly, while structural considerations deal with the components of a mechanism and their physical properties (such as size, shape, etc.), functional considerations deal with the activities components perform, their causal powers and how they contribute to the capacity of the whole mechanism (Piccinini and Craver 2011).

In sum, mechanistic explanation proceeds by individuating the underlying components and activities that form the mechanism, as well as their organisation, unveiling how they bring about the phenomenon to be explained. This often involves nested mechanisms—components of mechanisms are themselves mechanisms that can be decomposed into components, which might in turn also be decomposable mechanisms, until a level is reached in which mechanistic decomposition is no longer possible. This leads to the multilevel nature of mechanisms and mechanistic explanation.

For my limited purposes, a more detailed characterisation of New Mechanism as a general framework for scientific explanation is not necessary. My focus will be on one of its offshoots, the mechanistic view of concrete computation, and in particular in its perhaps more detailed version, Piccinini's (2007, 2008, 2015).

3 Computational mechanisms

According to Piccinini, computational mechanisms are a type of teleofunctional mechanism. Teleofunctional mechanisms are mechanisms that have teleological functions, that is to say, they have purposes or ends (Wimsatt 1972).[2] The purpose of an engine is to provide power, and purposes of organisms include survival and reproduction. The notion of teleological function is not to be confused with the notion of systemic function presented above. In teleofunctional mechanisms, both kinds of function are relevant. The mechanism has one or more teleological functions, and its mechanistic decomposition, in light of those teleological functions that help characterise the mechanism's capacities, will partially depend on the causal roles, the systemic functions, of its components.

The activities that components perform help determine their contribution, their function, in bringing about the overall behaviour of the mechanism, on its turn characterised by its teleological functions, its purposes—what it, as it were, is supposed to do. Internal components and processes that contribute to the performance of the teleological functions of teleofunctional mechanisms may themselves acquire teleological functions. Thereby a subset of the systemic functions of components will coincide with their teleological functions.

The appeal to teleological function makes it the case that teleofunctional mechanisms can go wrong: they may fail to perform their teleological functions due to breakage, inappropriate circumstances, and so on.

Many mechanisms are not teleofunctional. Though they have components that perform activities that explain a phenomenon, they have no end or purpose—think about planetary systems, the formation of valleys, or the water cycle. These systems can be broken down into their components and what they do in order nicely to explain how they work and why they behave the way they do. Even though their components have systemic functions, the overall mechanisms have no teleological function, and therefore cannot succeed or fail in any substantial way. Planetary systems can be altered in their workings by the intrusion of a new wandering planet, or by the effects of a supernova. Nonetheless, they do not therefore fail in performing any teleological function. They are mechanisms, but not teleofunctional ones. In what follows, I will use 'function' to refer to systemic functions, and otherwise I will use 'teleological function'.

Computational mechanisms, according to Piccinini (2015, p. 119ff.), are a subset of teleofunctional mechanisms. They are those teleofunctional mechanisms which have as one of their teleological functions that of performing concrete computations. Concrete computation, in its turn, is defined as the manipulation of vehicles according

[2] I remain neutral on what the appropriate account of teleological functions is. But note that the appeal to teleological function has the downside of making the view vulnerable to attacks against the notion of teleological function itself (Craver 2013); as well as requiring that a substantial theory of teleological function be proposed, and defended. I will not tackle these issues here, and will assume in the foregoing that teleological functions exist objectively and that there are one or more accounts of teleological function that help ground the objective existence of teleofunctional mechanisms.

to a rule sensitive only to (some of) their physical properties.[3] A rule, finally, is a mapping from inputs (and possibly internal states) to (internal states and) outputs.

Vehicles and their activities are arrived at by means of mechanistic decomposition. Given the overall capacity of the mechanism to perform computations, it is decomposed into the entities and activities so organised as to bring about that behaviour. One important property of vehicles in concrete computation is their being medium-independent—most of their physical properties are irrelevant to the computation performed. The rules that govern the changes undergone by the vehicles are sensitive only to some dimensions of variation of their physical properties, i.e. their degrees of freedom. Degrees of freedom abstract away from the physical properties themselves—consisting only of their dimensions of variation—and are thus characterised in medium-independent fashion. Physical systems made out of completely different materials, from silicon to neurons to vacuum tubes to beer tins, can perform the same computation provided they have physical properties with the appropriate degrees of freedom on which state-transition rules depend.

This understanding of concrete computation is general enough to include both digital and non-digital forms of computation—Piccinini (2015) dubs it 'generic computation'. Keeping to such a level of generality is appealing inasmuch as it allows various notions of computation to be captured without privileging any one in particular. For simplicity, I will focus exclusively on digital computation.

Digital computation is a subset of generic computation. To perform a digital computation is to manipulate digits and strings of digits according to rules sensitive only to (some of) their physical properties. Digits are realised by medium-independent vehicles characterised by the fact that they can be neatly distinguished by the computational mechanism, insofar as they are discrete and that there is a finite number of them—an alphabet. Two digits of the same type are processed equally, while two digits of different types are processed differently. In physical terms, digits are realised by equivalence classes of physical states that are treated uniformly by the system. A digit in an electronic computer is typically an interval of voltage values (e.g. 0–5 V) to which the system responds in the same way, *ceteris paribus*.

With this brief introduction to New Mechanism and to the mechanistic view of concrete computation in hand, we can start to see the tension between the two. The characterisation of computation seems to be largely functional, with no reference to structural properties of the mechanism except for its degrees of freedom, while New Mechanism seems to insist on the importance of structural detail. Haimovici (2013) explores this apparent inconsistency, and claims that the mechanistic view of computation finds itself in an uncomfortable dilemma. I will not stick closely to her argumentative line. Nevertheless, what follows in the next section is indebted to her discussion.

[3] Piccinini prefers the term 'spatiotemporal properties'. However, it is not clear how voltages, on which modern electronic computers rely, count as spatiotemporal properties. For this reason I prefer the term 'physical'. I thank Nicholas Shea for this point.

4 A dilemma

A relatively common (mis-)understanding of the neo-mechanistic approach has it that mechanistic explanation, if it is to be fully adequate, must provide structural detail about all levels of the mechanism.[4] On this view, mechanistic decomposition must proceed by individuating the systemic-functional as well as the structural properties of the mechanism that are relevant for explaining its capacity up to the point in which it reaches entities that are not further decomposable; and with the most details possible. Accordingly, full mechanistic explanation, the kind of regulative ideal we should strive for, is fully accurate and complete—it includes all detail at all levels of the mechanism.

If this is so the mechanistic view of computation is in dire straits. If good mechanistic explanations require—or at least strive for—a detailed structural description at all levels of the mechanism, computational explanations are clearly not good mechanistic explanations, and do not strive to so be. Indeed, by necessarily involving medium-independent vehicles, computational explanations will never provide enough structural detail to respect the mechanistic norms. If computational explanations should go into detail at all levels of the mechanism, they would have to include the physical details of each computational system. Mechanistic computational explanation would thereby have to include structural detail about the particular processors, memory devices, and so on, present in a particular (type of) computational mechanism. The medium-independent nature of computation, as well as its multiple realisability, would be lost.

In brief, either computational explanation is not mechanistic, or, if it is, then it loses what is most distinctive of it, i.e. medium-independence. Two claims, one about mechanistic explanation, one about computational explanation, clash, leading to the dilemma:

1. Good mechanistic explanation tends toward full structural detail at all levels of the mechanism.
2. Computational explanation is necessarily abstract, insofar as it ignores most structural detail, caring only about degrees of freedom (Piccinini 2015).

These two claims are clearly incompatible with the further claim that computational explanation is adequate mechanistic explanation.

The natural move for the computational mechanist is to reject 1. This is indeed the line followed by Piccinini (2015) in his reply to Haimovici (2013):

> I reject the view that mechanistic explanation requires the specification of structural and functional properties at all levels of organization. Instead, mechanistic explanation requires the specification of all relevant structural and functional properties at the relevant level(s) of mechanistic organization. (Piccinini 2015, pp. 124–125)

[4] Such a position is often ascribed to Machamer et al. (2000) and Craver (2007), among others, and there is space for seeing Piccinini himself as arguing for it, as some remarks in Piccinini and Craver (2011) seem to suggest. Note though that Craver, as well as Piccinini, have later denied that they subscribe to this view and, on the contrary, have argued that abstraction from detail and idealisation are vital parts of scientific explanation (Craver 2014, pp. 39–40; Piccinini 2015, pp. 124–125). The confusion may stem from an ambiguity between ontic and non-ontic understandings of explanation (see Halina forthcoming).

According to Piccinini, once claim 1. is rejected, it is clear that computational explanation counts as mechanistic. For, as all mechanistic explanations, it specifies structural properties of the system target of explanation. In the case of computational explanation, the structural properties specified are the degrees of freedom of the system. These structural properties, though quite abstract, are the ones relevant for the level of mechanism pertinent to computational explanation. Indeed, providing more structural detail does away with the medium-independence characteristic of computational explanation. Mechanistic computational explanation, therefore, stops there and refrains from providing more structural detail, on pain of missing its explanatory target.

The rejection of claim 1. is motivated by more than just saving the mechanistic view of computation. The mechanistic framework need not—and should not—pose such strict requirements on what counts as adequate scientific explanation. Requiring that full structural detail be provided is not only at odds with actual scientific practice, in which abstraction from details and idealisation play a major role; but would also be detrimental as a strategy for scientific investigation. Not every detail is relevant for explanation, quite on the contrary. An important part of scientific explanation consists in selecting what is explanatorily relevant from what is not. Often times, including irrelevant detail muddles explanation and undermines its adequacy.

This is the case across levels of mechanism as well. Most often adequate explanations need not go down all the levels of the mechanism responsible for a certain phenomenon. Explanations in biology, for instance, need not, and should not, include details at the level of physics in order to be adequate. Doing otherwise would add nothing of relevance to biological explanation. Analogously, the computational mechanist argues, computational explanation is mechanistic even though it is so remarkably abstract that it leaves out almost all structural detail (Piccinini 2015, p. 125).

Therefore, computational explanation as proposed by the neo-mechanists would be fully adequate mechanistic explanation: it delivers all relevant functional and structural detail at the explanatorily relevant levels of mechanism for computational explanation. Haimovici's dilemma would thus be avoided.

As I argue in the next section, this is too quick. The solution proposed by Piccinini exacts too high a price: it avoids the dilemma, but in so doing it gives up the mechanistic view.

5 Computational explanation is functional

Piccinini's reply to Haimovici saves the consistency of the mechanistic account, but in so doing causes it to lose its distinctiveness, making it collapse into functional explanation, as I show below. This is an unfortunate outcome for the computational mechanist, for it represents a remarkable concession to functional explanation, and it is doubtful that many proponents of New Mechanism would be willing to be this generous. Part of the motivation for the neo-mechanistic framework is indeed to supplement the putative shortcomings of functional explanation with more demanding requirements on what counts as good explanation—requirements that involve a certain amount of structural detail.

However, as computational mechanists admit, the structural constraints posed by computational explanation are extremely weak. Almost all structural detail is left out, and only constraints on the degrees of freedom of the structural components of the mechanism are left in place. This weak sort of structural constraint is on a par with the structural constraints posed by functional explanation—there as well most of the structural detail is left out, and only very weak structural constraints are in place.

To see this, consider how functional and structural decomposition would proceed when the mechanism under examination is my laptop computer. The functional decomposition, by referring exclusively to functional components such as 'processor' and 'memory register', abstracts from structural detail. Nonetheless, it does place some structural constraints: whatever plays the role of a processor must be so physically arranged as to do what a processor does, and the same goes for the other sub-capacities individuated by the functional analysis. Such structural constraints are quite abstract: the functionally-individuated processor need not even be a separate physical component of the device, but may span pieces and processes of different physical components. Analogously, the structural decomposition of my laptop, by mentioning components such as the 2 Ghz Intel Core chips, or the 500 GB solid-state Flash hard-disk, places constraints on the functional properties of the computer: it limits which functions it can perform, and how. In other words, functional and structural properties constrain each other to some extent (Piccinini and Craver 2011).

In computational explanation, the only structural constraint in place, having adequate degrees of freedom, is the one that ensures that the structural components of the computational mechanism can participate in computations, be digits, strings of digits, manipulators of digits, etc.—that is, that ensure that they can play the required functional role. This does justice to the medium-independence of concrete computation. Importantly, this is the kind of weak structural constraint that characterises functional explanation: it amounts to the trivial requirement that the physical system have structural properties able to realise the functional characterisation.

The abstraction from details that is an essential characteristic of computational explanation is comparable to the abstraction from details found in functional explanations.[5] Moreover, providing any further structural detail beyond the vague ones given by a functional analysis is fatal to the nature of computational explanation. If the computational explanation should mention structurally-individuated components, such as my 2 Ghz Intel Core chips, it would immediately foil any attempt at multiple realisability or medium-independence.

Computational explanation and functional explanation look therefore much alike. Some fundamental features of computational explanation are identical to features of functional analysis. Those same features are moreover essential to computational explanation, thereby leading to the conclusion that computational explanation is essentially a form of functional explanation.

[5] See Cummins (1975, p. 764), according to whom in functional explanation, as the functional analysis "absorbs more and more of the explanatory burden, the physical facts underlying the analyzing capacities become less and less special to the analyzed system [...] this is why it is plausible to suppose that the capacity of a person and of a machine to solve a certain problem might have substantially the same explanation …".

The computational mechanist may reply that what distinguishes computational from functional explanation is that computational explanation stops at the appropriate level of abstraction—it gives all the structural detail at the appropriate level of explanation for computational systems; while functional explanation does not—it could go on, but it just refrains from doing so. Provide more structural detail, and computational explanation is lost; provide more structural detail, and functional explanation gets better—says the computational mechanist.

Alas, this move cannot help the mechanistic view of concrete computation. For the reasons why functional analysis stops where it does are to a large extent analogous to the ones that motivate appeal to computational explanation: capturing generalisations that would otherwise be ignored, and making space for multiple realisability. It is not true that functional explanation improves with more structural detail—one of its hallmarks, multiple realisability, is lost in the process.[6] While the functional explanation of how my laptop computer works is generalisable to most other commercial laptops available today, courtesy of its abstractness, adding more structural detail makes the explanation less and less general, up to the point that it explains exclusively how my laptop works (and perhaps only at this point in time). The same goes for functional explanations of combustion engines, corkscrews, etc.

In brief, functional explanation has as much motivation for stopping where it does as does computational explanation. Functional explanation also gives all the structural detail (very little) that is relevant for its explanatory purposes—it also keeps to the appropriate level of abstraction.

Functional explanation is suitable for those *explananda*, that, by their nature, involve considerable abstraction from structural details, such as concrete computations, and perhaps psychological capacities. Even though more structural detail can be provided, thereby unveiling the workings of more levels of specific mechanisms, doing so amounts to losing multiple realisability and medium-independence. It amounts to giving up on computational and psychological explanation, inasmuch as we fail to stop at the relevant explanatory level for those kinds of phenomena.

Piccinini's claim that computational explanation is mechanistic because it involves structural properties of systems—degrees of freedom—turns out to be rather uninteresting and uninformative. Nearly all kinds of explanation, including functional explanation, place structural constraints on systems in the weak and trivial sense presented above.[7] Even the most abstract of explanations poses some extremely weak structural constraints on the systems it seeks to explain. If that is all that is needed to count as mechanistic, New Mechanism ends up being a rather trivial position. The mechanist's contribution to computational explanation would thus amount to the disappointingly unsurprising reminder that functional and structural considerations constrain each other, even in those cases, such as computational explanation, in which such constraints are very weak.

At this point, there is very little reason to stick to the label 'mechanistic' in the foregoing account of concrete computation. Indeed, calling it the 'functional view',

[6] There has been in recent years a rich debate on whether multiple realisability, at least for what regards cognitive states, is true. See Shapiro (2000).

[7] See also Shapiro (2016), Sect. 4.

as Piccinini did in the past, is perhaps fairer to its nature.[8] Concrete computation is individuated by functional properties, as described in Sect. 3, and computational explanation is a type of functional explanation in which the only structural constraint on the realising system is that it have the appropriate degrees of freedom.

Even if we accept the conclusion that computational explanation and individuation are functional, *contra* Piccinini and other computational mechanists, I believe that there is a different, and much more significant role to be played by the appeal to mechanism in a theory of concrete computation.

6 Computational systems as mechanisms

I suggest that the proper way of seeing the mechanistic view is as a hypothesis about the nature of those physical systems that are able to implement computations in a robust, non-trivial way. The role of mechanism in the mechanistic view of computation is to provide part of the connexion between abstract computation and world that a theory of concrete computation must deliver. The fact that providing some structural detail is part and parcel of mechanistic explanation—the main motivation for the worries examined above—poses no challenge to the view once it is seen in the correct light.

Computation is individuated by functional considerations—it is the capacity to go from inputs to outputs according to rules, which, as we have seen, do place structural constraints, albeit rather weak ones, on the realising physical system. Computational systems are physical systems that feature this capacity—or more precisely, that have this teleological function. What the appeal to mechanism gives us is part of the story of how computation can take place in the physical world.

The mechanistic view of concrete computation is best seen as a hypothesis about those systems in the world that actually perform computations—the hypothesis being that such systems are teleofunctional mechanisms. According to the mechanistic view, those physical systems in the world that perform computations, and that therefore can be explained computationally, are tokens of a specific type of teleofunctional mechanism.

Therefore, the amended version of the mechanistic view of concrete computation that I propose has it that computation in physical systems consists in:

1. Manipulation of medium-independent vehicles according to a rule sensitive only to their degrees of freedom.
2. The medium-independent vehicles are functional components of a teleofunctional mechanism.[9]
3. The manipulations that vehicles undergo are activities internal to a teleofunctional mechanism.
4. It is one of the teleological functions of the teleofunctional mechanism to carry out 1.

[8] See Piccinini (2007, 2008). See also Fresco (2014).

[9] There need be no one-to-one mapping between functionally-individuated vehicles and structural components.

Note that this is a functional characterisation of concrete computation, despite the appeal to mechanism. It provides very little structural detail, as functional characterisations typically do—it is silent on the physical nature of vehicles and the ways they are manipulated, preserving thereby their medium-independence. However, it makes clear the role that mechanism should play in the account. What makes computational explanation mechanistic is the suggestion that physical computational systems are mechanisms, to which, in consequence, mechanistic explanation applies most suitably. These systems can be mechanistically decomposed in light of their functionally-individuated capacity to perform computations.

There seems to be a tension in the foregoing, however. Computational explanation is both functional, as I have argued above, and mechanistic in a non-trivial way, as I have just claimed. How can this be so? To make sense of this, a further distinction must be introduced: that between computational individuation and computational implementation.

Computational systems and their states are individuated functionally, in a medium-independent way, as above. But there is the further question of how a particular computational system implements that functional architecture. The answer to this question cannot be in medium-independent terms.

Computational explanation may be seen as answering questions such as: how does system S compute function f? The answer to this question need only involve functionally-individuated states of the system—inputs, processors that follow rules of manipulation stored in memory registers, outputs, and so on. This type of explanation fully preserves multiple realisability and medium-independence. A variety of mechanisms, constituted by wildly different kinds of structural components, can be functionally decomposed in the same way—provided they have the appropriate degrees of freedom.

Computational explanation may also be seen as answering questions such as: how does system S implement the functional profile for the computation of function f? In this case, the answer will involve detail about the physical constitution and organisation of the system. How are inputs and outputs physically implemented? How does the processor manipulate inputs physically characterised? In order to explain how an electronic computer implements the mathematical function of addition, an explanation in terms of voltages, electronic gates, chips, and hard disks will be called for. Such an explanation will not be medium-independent, for the question cannot be answered in a medium-independent way.

In this latter case we are interested in computational implementation, for which an explanation in terms of physical constituents is required. The explanation of how a valve computer implements addition will be importantly different from the answer given to the same question asked about a transistor computer. This kind of explanation may not be generalisable even beyond the specific system under investigation (or perhaps rather the type of system—a Compaq Presario computer has structural components that are different from the ones present in a Lenovo ThinkPad, or an Apple MacBook computer[10]).

[10] I will not be concerned with the appropriate level of type-individuation here. The scope of implementational explanations of this kind will hinge on how we type-individuate implementations.

Computational explanation is functional because computational states and processes are functionally individuated; and it is mechanistic because those functionally-individuated states and processes happen to be realised by mechanisms, to which an implementational mechanistic explanation applies. These should not be seen as alternatives. Both kinds of explanation may deserve to be called 'computational explanation', as long as we are clear on what type of questions those explanations are answering, i.e. about individuation or implementation. Moreover, though both kinds of explanation can stand by themselves, each answering a different question, they must come together in offering the two complementary pieces of a theory of concrete computation. In order to explain how concrete computation is possible, how physical systems are able to compute, both types of explanation are needed. We need to individuate computation functionally, and we must then show how those individuated states and processes are realised in the physical world in each case. For a theory of concrete computation to be satisfying, both individuation and implementational questions must be addressed.[11] The appeal to mechanism, as we have seen, plays a role in answering both, though in different ways: as part of the individuation conditions in the former; as allowing mechanistic implementational explanations in the latter.[12]

Haimovici's (2013) dilemma does not grab a hold because the computational explanations that are purely functional are different in kind to the computational explanations that require structural detail. The latter are implementational explanations, for which the medium-independent nature of computation must be put aside. Implementational explanations are not in competition with more abstract medium-independent explanations. The dilemma she puts forward is a false one. Similarly, Piccinini's reply is misguided, as it accepts the terms set by Haimovici, and leads to the collapse of the mechanistic view into a purely functional one.

A properly understood mechanistic view insists that computational implementation involves components and activities of mechanisms that lead to and enable the capacity to perform concrete computations. Structural detail can be provided here with no risk, since this is not the dimension in which considerations about medium-independence or multiple realisability are of relevance. Implementations are not medium-independent, they must involve things such as silicon, neurons, valves, beer tins, or what have you.

Contra Piccinini, computational explanation is not mechanistic because it also places structural constraints, albeit rather weak ones, on the physical systems target of the explanatory project—this is true of many (maybe most) kinds of explanation, even at high levels of abstraction. Rather, the proper way of seeing the mechanistic view is as claiming that, since concrete computation pertains to physical systems that are types of mechanism, mechanistic explanation provides the implementational story.

In comparison with competing views of computational explanation, the mechanistic proposal places much more stringent constraints on which systems in the world count as computational. The appeal to mechanism in the mechanistic view plays an

[11] The need for an implementational story may be a special feature of endeavours that rely heavily on functional considerations. There is a worry about how to make computations, or psychological states, features of physical systems. There are no such worries in other fields, such as biology—no puzzles about making 'abstract' biology concrete.

[12] I thank an anonymous referee for prompting me to clarify this point.

analogous role to mapping relations in the simple mapping account put forward by Putnam (1988), and to causal topology in Chalmers' (2011) causal account: it provides the bridge between functional computational individuation, and implementation in physical systems. But it provides further constraints than its competitors: it includes mapping as well as causal considerations, into a richer, more constrained, picture. The mechanistic view requires that physical computational systems not only have physical states mappable onto abstract computational states, as per the simple mapping view, or that they be causal systems, as per the causal view. They must be more than that, they must be mechanisms—organised systems with relatively clear boundaries, decomposable into physical parts that play a role in bringing about the overall behaviour of the system; and they must be teleofunctional mechanisms with the teleological function of performing computations.

In sum, concrete computational systems are physical systems that fulfil the four-way functional characterisation above. Computational explanations in these medium-independent terms are perfectly adequate. However, given their abstractness—with its attached explanatory virtues—they do not go all the way in explaining how computational systems are realised in mechanisms. The medium-independent functional explanation allows for great generality and does justice to the multiple realisability of concrete computation. But the implementational explanation explains how a specific (type of) mechanism, given its particular structural components, their activities and organisation, realises the functional characterisation—thus bridging that abstract characterisation with the nitty-gritty details of its specific realisations in the world.

The foregoing picture would be lacking in motivation were there no reasons for claiming that physical computational systems happen to be teleofunctional mechanisms. Fortunately, such justification comes independently from the neo-mechanist framework. Delimiting the domain of physical computational systems to those of a specific kind of teleofunctional mechanism helps to avoid the pitfalls that have plagued competing theories of concrete computation. Pancomputationalism—the view that all or most physical systems are computational—is avoided, and the recourse to teleological functions introduces a normative dimension useful in accounting for miscomputation.[13]

In conclusion, computational explanation is mechanistic because, if the mechanistic view of concrete computation is correct, physical systems that compute are mechanisms, to which a mechanistic implementational explanation is most suitable. This does not in any way impinge on the medium-independence of computational individuation, nor on its functional nature. The arguments in Haimovici (2013), as well as the reply offered by Piccinini (2015), are beside the point. Mechanistic explanation provides structural detail about computational mechanisms because this is needed to explain how those physical systems implement computations, thereby helping to explain how computation in physical systems is possible.

[13] There are other advantages that I do not mention here. See Piccinini (2007, 2015), Milkowski (2013), and Fresco (2014).

Table 1 Device *D*'s input–output table

Input 1 (V)	Input 2 (V)	Output (V)
0–5	0–5	0–5
0–5	5–10	0–5
5–10	0–5	0–5
5–10	5–10	5–10

7 Computational individuation and the multiplicity of computations

The version of the mechanistic view of concrete computation I propose leads to other welcome results. In this final section, I argue that my view helps solve a central issue that has been at the centre of debate in philosophy of computing: the problem of the multiplicity of computations.[14]

As Shagrir (2001, 2012) and Sprevak (2010) purport to show, most of the theories of computational individuation on offer, including the mechanistic view, do not have the tools to draw distinctions central to the practices of computer science. These authors argue that semantic constraints on computational individuation are needed. The argument from the multiplicity of computations is perhaps the most powerful argument in favour of semantic views of computational individuation against non-semantic views, such as the mechanistic one.

I will focus on Sprevak's version of the argument for expository reasons, as it is considerably simpler than Shagrir's version. Consider an electronic computational device *D* that takes two voltage values as inputs and produces one voltage value as output according to the following input–output table (in terms of ranges of voltage values) (Table 1).

The device seems to be working as a paradigmatic logic gate. Logic gates are basic computational building blocks that compute logical functions such as AND, OR, NOR, NAND, etc. At first glance, *D* seems to be an unequivocal AND-gate. Take voltage range 0–5 V to stand for 'False', and voltage range 5–10 V for 'True', and we get the truth table of conjunction.

However, as Sprevak points out, if we switch semantic contents, that is, if we take the range 0–5 V to stand for 'True' and range 5–10 V for 'False', we get an OR-gate—the truth table we end up with is the one for disjunction. Without a decision on what the voltage levels stand for, or represent, so Sprevak argues, there is no way of telling whether *D* is an AND-gate or an OR-gate, or both—it seems to be simultaneously computing both logical functions. [15] Since logic gates are at the basis not only of theorising in computer science, but also in the engineering of actual computers, semantic properties seem to be required for adequate computational individuation, *contra* theories, such as the mechanistic view, that rely completely on non-semantic properties.

[14] The multiplicity of computations problem can be seen as one of the possible arguments leading to a deeper issue, which Fresco et al. (forthcoming) label the 'indeterminacy of computation' problem.

[15] This is also true of other logic gates, which, due to this property, are dubbed dual gates.

Piccinini (2008) defends the mechanistic view against the argument from the multiplicity of computations. He argues that even though *D* implements more than one computation, a wide understanding of systemic functions, reaching to the immediate context of the computational device (and of the overall mechanism), suffices to determine what the explanatorily relevant computation being performed is. Though this answer has some appeal to it, it concedes too much. It concedes that *D* implements more than one computation, that is, that there is multiplicity of computations—a concession that, I believe, should not be granted. Moreover, it is not clear that the immediate context will be able to dissolve the multiplicity that Shagrir and Sprevak point out, even when taken in terms of explanatory relevance. For instance, fully dual computational systems, i.e. in which all logic gates, as well as the whole system, can be consistently interpreted in two different ways, are possible.[16] Though improbable, these systems spell trouble for Piccinini's appeal to wide functional individuation, as in their case, this seems insufficient to get rid of multiple computations—multiplicity survives even taking the immediate context of the device and of the overall mechanism into consideration.

Dewhurst (2016) has recently put forward what I see as a more promising line of reply. In a nutshell, he accepts that whether devices like *D* compute a logical function or its dual remains indeterminate by the mechanistic view's lights. However, he claims that this should not worry the computational mechanist, for computational individuation is done at the level of the physical description of the device. Table 1 is all that is needed to individuate the computational device, no labelling nor ascription of semantic or syntactic properties is required. AND- and OR-gates are equivalent insofar as computational individuation is concerned. They compute the same function from physical inputs to physical outputs—the patterns of voltage transformation are the same. *Contra* Shagrir, Sprevak, and Piccinini, there is no indeterminacy of computational individuation caused by multiple computations being simultaneously implemented. The indeterminacy lies at a different level, the logical one, which is outside the purview of a theory of individuation proper.

This does not mean that individuation in terms of logical functions is uninteresting. Quite on the contrary, it is relevant for many applications in computer science, both in theory and in engineering. But individuation by logical function is over and above computational individuation, and may well rely on wide functions or semantic properties. Computational individuation is more basic, and non-semantic—it is done at the physical level of the mechanism. Therefore, the charge that Shagrir and Sprevak move against the mechanistic view is misguided. It is true that the mechanistic view does not distinguish AND- from OR-gates (as well as other dual gates), but this distinction is not at the level of computational individuation, for which only the physical patterns of transformation are relevant. Two devices may perform the same computation, but carry out different logical functions depending on contextual and semantic considerations. Computational individuation and logical individuation should be kept distinct. Non-semantic properties suffice for the former, while they might not suffice for the latter.

Admittedly, this picture suffers from a serious shortcoming. It makes computational equivalence impossible, thus also threatening the closely related idea that computa-

[16] I thank Oron Shagrir and Nir Fresco for pointing this out to me.

tions are multiply realisable. As Dewhurst (2016) recognises, “the physical structure of two computing mechanisms is always going to be distinct, and it is unclear whether we can draw any non-arbitrary boundary between the structures that are relevant or irrelevant to computational individuation”. It thereby follows that no two computational devices are equivalent, for there will always be physical differences between them that are difficult to rule out as computationally irrelevant in a principled way. But even if we could distinguish the structural properties that are computationally relevant from those that are not, computational equivalence would still be excessively fine-grained, for the physical description of the system is too fine-grained for computational individuation.

To illustrate, take two devices *D1* and *D2*. They work in an analogous way to device *D*, but with one important difference: for engineering reasons, they have ‘cushion’ intervals between the voltage ranges relevant for determining the output. Voltages that fall inside these cushion intervals have a ‘null’ value, and when the device has one such voltage as one of the inputs, it produces a null value, or no output at all. Suppose that *D1*’s cushion interval is 4–5 V, while *D2*’s is 5–6 V. It follows from Dewhurst’s proposal that these two devices are not computationally equivalent, for in the case of *D1* the acceptable inputs and outputs are voltages in ranges 0–4 and 5–10 V, while in *D2*’s case these are voltages in ranges 0–5 and 6–10 V.[17] The two devices have different physical descriptions, but it seems overly strong to argue that it follows from this that they are not computationally equivalent. Indeed, they have the same number of input and output types (even counting cushion ones), and the former are transformed into the latter by analogous rules of transformation—despite the fact that the processors are sensitive to different voltage ranges.

Similarly, suppose that instead of having different cushion intervals, *D1* and *D2* were identical if not for being subject to different degrees of noise. Noise makes *D1*’s behaviour unreliable when inputs fall within the range 4.5–5.5 V, say, whilst noise interferes with the functioning of *D2* when inputs fall within the 4.9–5.1 V range. Individuating computation at the physical level would have it that *D1* and *D2* are not computationally equivalent, despite their striking similarity.

In sum, the physical level is too fine-grained to make computational equivalence possible. If we want to save the notion, as we should given its explanatory importance in computer and cognitive science, we need a coarser-grained method for individuating computation (Fresco et al. 2016).[18]

[17] I am indebted to Jack Copeland, Nir Fresco, and Oron Shagrir for raising and discussing the points in this and the next paragraph.

[18] Fresco et al. (2016) propose a coarser-grained method for individuating computation that goes some way in the direction I recommend. However, there are fundamental differences between our approaches: they focus on coarse physically-individuated properties, in particular intervals of voltage values grouped into high and low voltages, instead of medium-independent functionally-individuated properties, as in my view; and they fail to draw the crucial distinction between computational and logical equivalence, which leads them to claim that even such coarser-grained individuation methods, as the one I propose, fail to solve the problem of the multiplicity of computations. As I argue, following Dewhurst (2016), once the latter distinction is properly understood, the multiplicity of computations problem becomes considerably more tractable.

Table 2 Input–output table of *D1* and *D2*'s functional equivalence classes

Input 1	Input 2	Output
EC1	EC1	EC1
EC1	EC2	EC1
EC2	EC1	EC1
EC2	EC2	EC2

The version of the mechanistic view of concrete computation that I defend in Sect. 6 has the tools to improve over Dewhurst's account in allowing for a meaningful notion of computational equivalence while keeping to the spirit of his solution to the argument from the multiplicity of computations.

In my view, the physical level of description is the wrong one to focus on in order to get adequate, determinate, computational individuation—it is too fine-grained to allow for a useful notion of computational equivalence. The physical description gives us the implementational details, but computational individuation takes place at the functional level, in which the only structural considerations at play are having appropriate degrees of freedom. There is a meaningful notion of computational equivalence available at this level of description.

Take once again *D1* and *D2* and their different cushion intervals (or noise levels). While the physical description of the two devices differ, at the functional level their description is identical. The devices respond to two distinct equivalence classes of acceptable physical inputs (voltage ranges), EC1 and EC2, and produce the same equivalence classes of physical outputs (voltage ranges) given the inputs.

The labels are fully arbitrary and introduced only for ease of exposition. How we label the equivalence classes is irrelevant to computational individuation; what matters is the overall functional profile that defines them. Equivalence classes are defined by input values that lead to uniform behaviour of the whole device—the differences in value to which the device is sensitive and which are uniformly transformed into new values. For *D1*, EC1 is the range 0–4 V and EC2 is the range 5–10 V, whilst for *D2* EC1 is the range 0–5 V and EC2 is the range 6–10 V.[19] The input–output tables of *D1* and *D2*, when put in terms of equivalence classes, are identical (Table 2).

The physical details of the two devices can be glossed over—structural details come in only when we are interested in the particular implementational details of a computational device—as it is the functional description which is of relevance for computational individuation.[20] It follows that *D1* and *D2* are computationally equiva-

[19] Alternatively, one could consider there to be three equivalence classes, including the cushion intervals as an equivalence class. I believe this would be the more precise way to go, but I am ignoring this complication for ease of exposition.

[20] In consequence, devices that differ in the number of stable states (e.g. two vs. three), as in Shagrir's (2001) version of the argument from the multiplicity of computations, are never computationally equivalent (Dewhurst 2016). They may nonetheless be logically equivalent, i.e. carry out the same logical function. That is to say, a bi-stable and a tri-stable device may carry out the same logical function, and thus be logically equivalent, despite not being computationally equivalent given their different functional profiles. Different possible groupings of the devices' stable states, as in Shagrir's argument, are irrelevant to computational individuation: given the different number of equivalence classes of physical states the two devices stabilise

lent: the functional profile from input equivalence classes of physical states to output equivalence classes of physical states is the same.

What those physical states consist in is irrelevant for computational individuation. A hydraulic computational device *D3* shares the same functional profile of *D1* and *D2* if it is sensitive to—and responds uniformly and in the same way to—the same number of equivalence classes of physical states. That those equivalence classes be of ranges of water levels in tanks is interesting when it comes to implementational details, but irrelevant for computational individuation. Thereby, computational equivalence is possible even between systems that work by means of completely different physical principles—and the multiple realisability of computation is preserved.

Indeterminacy of logical function computed still follows. Table 2 cannot determine whether the devices are AND- or OR-gates (recall that the labels are purely arbitrary and can be freely switched or changed). Computational individuation in the foregoing, as per Dewhurst's account, leaves logical individuation indeterminate. This is a welcome result, since, as Dewhurst convincingly argues, logical individuation is at least one step above computational individuation. The mechanistic view of concrete computation should not therefore worry about the arguments from multiplicity of computations put forward by Shagrir and Sprevak. What they point out is correct: the mechanistic view does not have the tools to distinguish between dual logic gates. However, such a feat is not something we should be asking of a theory of computational individuation, for computational individuation takes place below the level of logical functions.

Where I part ways with Dewhurst is on what the appropriate level for computational individuation is. He argues that it is the physical level that allows suitably to distinguish between computational devices. But he has consequently to give up any useful notion of computational equivalence. This is too high a price to pay. In contrast, I argue that computational devices can be appropriately distinguished from each other, or found to be equivalent in an informative way, by focusing on the functional level, in which it is functional, rather than physical, structure that individuates computational states and processes.

It may be objected that computational equivalence is impossible even when we focus on the functional level, rather than the physical one.[21] It may be argued that the maximal functional profiles of two physical systems will always differ, and thereby that they can never be computationally equivalent. I think that the foregoing account has the means to avoid this objection. For recall that the functional decomposition of a physical system always takes place in light of a target capacity or teleological function—in our case, the capacity to perform computations. Therefore, the functional decomposition, and the resulting functional profiles of component computational devices, do not include functional features that are irrelevant to the overall system's capacity

Footnote 20 continued
on and are differentially sensitive to, they will always be functionally distinct according to the foregoing account, and therefore not computationally equivalent. This, I take, is as it should be: given their different functional profiles, those two devices will differ in their capacity to carry out logical and mathematical functions—having a richer functional structure makes the tri-stable device considerably more versatile. I thank two anonymous referees for prompting me to clarify this point.

[21] I thank an anonymous referee and Nir Fresco for bringing this issue to my attention.

to compute. Functional differences between devices which play no role in their general computational capacities are thus excluded—e.g. because they are not relevant to the regimented input–output transformations of equivalence classes of physical states across the system. This makes so that devices that have different physical properties, such as *D1*, *D2*, and *D3* above, are computationally equivalent, insofar as their computationally-relevant functional profiles are the same.[22]

The foregoing proposal hinges on whether there are principled ways of carving the functional structure of a computational device. This is analogous to Dewhurst's (2016) worry about principled ways of carving the (computationally) relevant physical properties of a system; a worry that, he argues, runs through the whole neo-mechanist framework, and is not a problem specific to the mechanistic view of concrete computation. Dewhurst suggests, following Piccinini, that such principles can be arrived at only through choices dictated by our explanatory interests. Consequently, a degree of observer-relativity must always be in place in mechanistic explanations.

While I agree that there is a crucial worry here, to which a suitable answer must be provided, I believe that a theory of computational explanation has additional tools to deal with it in comparison to other types of explanation tackled by New Mechanism. For, as we have seen in Sects. 3 and 6, the mechanistic view of computation appeals to teleofunctional mechanisms, i.e. mechanisms with teleological functions. Teleological functions bestow, as it were, privileged, objective, capacities on teleofunctional mechanisms and their components. Hence privileged observer-independent functional and structural decompositions would be available for all teleofunctional mechanisms, computational ones included. Whether this strategy will bear any fruit depends, though, on whether there are objective teleological functions in the world—a question that lies beyond the scope of this paper.

8 Concluding remarks

I have examined a challenge to the mutual consistency of the mechanistic view of concrete computation and the overall neo-mechanistic framework to scientific explanation. I argued that the challenge dissolves once we see the mechanistic view in the proper way. The resulting theory of concrete computation fully deserves being characterised as mechanistic, as it encompasses both functional individuation and mechanistic implementational considerations. Finally, I showed that my proposal also helps solve problems related to computational indeterminacy.

Acknowledgements I am indebted to Nicholas Shea, Nir Fresco, Oron Shagrir, two anonymous referees, as well as Michael Pauen, Juan R. Loaiza, and the other participants of the Philosophy of Mind colloquium at the Berlin School of Mind and Brain for helpful discussion of this material.

[22] Once again, devices that differ in the number of their stable states do not count as computationally equivalent even though they may be able to carry out the same logical and mathematical functions. This is so because their maximal computationally-relevant functional profiles differ, since the number of equivalence classes of physical states they are sensitive to is different—regardless of whether those functional differences are exploited or else by the overall computational system in specific computations.

References

Andersen, H. (2014a). A field guide to mechanisms: Part I. *Philosophy Compass*, *9*(4), 274–283.
Andersen, H. (2014b). A field guide to mechanisms: Part II. *Philosophy Compass*, *9*(4), 284–293.
Chalmers, D. J. (2011). A computational foundation for the study of cognition. *Journal of Cognitive Science*, *12*(4), 323–357.
Craver, C. (2007). *Explaining the brain*. Oxford: Oxford University Press.
Craver, C. F. (2013). Functions and mechanisms: A perspectivalist view. In P. Huneman (Ed.), *Functions: Selection and mechanisms*. Dordrecht: Springer.
Craver, C. F. (2014). The ontic account of scientific explanation. In M. I. Kaiser, O. R. Scholz, D. Plenge, & A. Huettemann (Eds.), *Explanation in the special sciences: The case of biology and history*. Dordrecht: Springer.
Cummins, R. C. (1975). Functional analysis. *Journal of Philosophy*, *72*(20), 741–765.
Dewhurst, J. (2016). Individuation without representation. *British Journal for the Philosophy of Science*. doi:10.1093/bjps/axw018.
Fresco, N. (2014). *Physical computation and cognitive science*. Dordrecht: Springer.
Fresco, N., Wolf, M. J., & Copeland, J. B. (2016). On the indeterminacy of computation. In *Methodological issues in philosophy of computer science symposium*. The 2016 Annual Meeting of the International Association for Computing and Philosophy, University of Ferrara, Italy.
Fresco, N., Wolf, M. J., & Copeland, J. B. (forthcoming). The indeterminacy of computation: Computational explanations and neural mechanisms.
Haimovici, S. (2013). A problem for the mechanistic account of computation. *Journal of Cognitive Science*, *14*, 151–181.
Halina, M. (forthcoming). Mechanistic explanation and its limits. In S. Glennan & P. Illari (Eds.), *Routledge Handbook of the Philosophy of Mechanisms*. Routledge.
Illari, P. M., & Williamson, J. (2012). What is a mechanism? Thinking about mechanisms across the sciences. *European Journal for the Philosophy of Science*, *2*, 119–135.
Levy, A. (2013). Three kinds of new mechanism. *Biology and Philosophy*, *28*, 99–114.
Machamer, P., Darden, L., & Craver, C. (2000). Thinking about mechanisms. *Philosophy of Science*, *67*, 1–25.
Milkowski, M. (2013). *Explaining the computational mind*. Cambridge: MIT Press.
Moss, L. (2012). Is the philosophy of mechanism philosophy enough? *Studies in History and Philosophy of Biological and Biomedical Sciences*, *43*, 164–172.
Piccinini, G. (2007). Computing mechanisms. *Philosophy of Science*, *74*(4), 501–526.
Piccinini, G. (2008). Computation without representation. *Philosophical Studies*, *137*, 205–241.
Piccinini, G. (2015). *Physical computation: A mechanistic account*. Oxford: Oxford University Press.
Piccinini, G., & Craver, C. (2011). Integrating psychology and neuroscience: Functional analyses as mechanism sketches. *Synthese*, *183*(3), 283–311.
Putnam, H. (1988). *Representation and reality*. Cambridge: MIT Press.
Shagrir, O. (2001). Content, computation and externalism. *Mind*, *438*, 369–400.
Shagrir, O. (2012). Computation, implementation, cognition. *Minds & Machines*, *22*, 137–148.
Shapiro, L. A. (2000). Multiple realizations. *The Journal of Philosophy*, *97*(12), 635–654.
Shapiro, L. A. (2016). Mechanism or bust? Explanation in psychology. *British Journal for the Philosophy of Science*. doi:10.1093/bjps/axv062.
Sprevak, M. (2010). Computation, individuation, and the received view on representation. *Studies in History and Philosophy of Science*, *41*, 260–270.
Wimsatt, W. C. (1972). Teleology and the logical structure of function statements. *Studies in the History and Philosophy of Science*, *3*(1), 1–80.