Synthese (2024) 203:99
https://doi.org/10.1007/s11229-024-04526-x

ORIGINAL RESEARCH

An idealised account of mechanistic computation

Luke Kersten[1]

Received: 23 September 2023 / Accepted: 8 February 2024 / Published online: 14 March 2024

Abstract
The mechanistic account of computation offers one promising and influential theory of computational implementation. The aim of this paper is to shore up its conceptual foundations by responding to several recent challenges. After outlining and responding to a recent proposal from Kuokkanen (Synthese 200:247, 2022a), I suggest that computational description should be conceptualised as a form of idealisation (selectively attending to modified subsets of model features) rather than abstraction (selectively attending to subsets of features within a target system). I argue that this conceptualisation not only offers the best way of making sense of computational implementation, but also a way of resolving each of the outstanding challenges facing the mechanistic account. The idealisation view allows the mechanistic account to make sense of the omission process found in computational descriptions without leaving the relationship between physical and computational properties mysterious.

Keywords Mechanistic computation · Implementation · Mechanistic hierarchy · Abstraction · Idealisation

1 Introduction

The project of naturalising physical computation has been underway for some time now. From simple mapping and causal accounts (Putnam, 1975; Chalmer, 1994, 2011; Chrisley, 1995) to representational and semantic proposals (Fodor, 1981; Sprevak, 2010; Shagrir, 2006, 2020), philosophers of mind and cognitive science have long sought to articulate an objective, explanatory and taxonomically adequate conception of computation. One particularly promising and influential theory in recent years

✉ Luke Kersten
kersten@ualberta.ca

1 Department of Philosophy, University of Alberta, Edmonton T6G 1C9, Canada

is “the mechanistic account of computation” (or simply “the mechanistic account”) (Piccinini, 2007, 2015, 2020; Fresco, 2014; Miłkowski, 2013, 2015). According to the mechanistic account, the task of explaining under what conditions a physical system implements a computation (computational implementation) is best explicated within a mechanistic framework. Computational properties are mechanistic properties, computational explanation is a species of mechanistic explanation, and computational mechanisms are a special type of mechanism (e.g., ones with teleological functions).[1]

To provide a concrete example, consider Piccinini’s (2015) version of the mechanistic account. For Piccinini, concrete computation occurs wherever there is a physical system that has an organisation of spatiotemporal components such that it computes an abstract function in virtue of manipulating medium-independent vehicles. The account offers three conditions on implementation.

The first is that a physical system must be kind of functional mechanism – that is, a mechanism with teleological functions.[2] The system has to possess properties that organise in such a way so as to produce or support some behaviour – the reverse of which is that if a system fails to perform its function it must be the result of a breakdown in the organisation of the system’s component parts. For example, calculators are functional mechanisms in virtue of the fact that they carry out their particular functions (e.g., computing sums) via the interaction, organisation, and relation of their component parts (e.g., processors, memory units, input devices, and output devices).

The second is that one of the capacities of the mechanism must be the ability to compute at least one mathematical function. A system must be able to map from an input I (and possibly internal states S) to an output O. A system’s behaviour must satisfy at least one abstract description mapping of inputs to outputs, which also suffices to show that a system is following a rule. The neural network in the ocular-motor system responsible for horizontal eye movement, for example, computes at least one abstract function (an integration relation) in virtue of preserving the relationship between eye-velocity and eye-position (Leigh & Zee, 2006).[3]

[1] Complementing the question of computational implementational is the question of computational *individuation*, i.e., the task of explaining under which conditions it is true or false to say of a physical system that it implements a particular computational model rather than another, e.g., AND versus OR (Sprevak, 2018). While important, the current paper focuses only on computational implementation for matters of scope. It leaves discussion open as to which account of computational individuation best fits with the mechanistic account, though it is worth noting that authors diverge on the issue (see, e.g., Fresco, 2010, 2021; Dewhurst, 2018; Coehlo Mollo, 2018; Fresco & Milkowski, 2021; Fresco, 2021).

[2] For Piccinini (2020), a teleological function is a stable contribution towards a goal of an organism. Goals can be either biological or nonbiological. Biological goals include survival, development, reproduction or helping, whereas nonbiological goals are any other goals pursued by an organism, such as using a Tobacco pipe to hold tobacco.

[3] Relevant here is the distinction between normative teleological functions and perspectival functions. For Piccinini (2015, 2020), teleological functions are objective in the sense that they are stable causal contributions to the goals of an organism. The functions can be individuated objectively in virtue of some combination of the organism and its environment. Perspectivalism, on the other hand, which says that functional attributions are relative to observer-interests (and hence subjective), is ill-suited to account of the functions of objects (artefactual or biological) because it does not do justice to our scientific practices – some traits have multiple functions, but not in virtue of multiple perspectives, e.g., the function of the

The third is that a physical system must process *medium-independent vehicles*. For a state or variable to be medium-independent, it must possess certain structure (i.e., degrees of freedom), and be capable of being implemented in different media in virtue of that structure (Garson, 2003). If the input–output mapping is sensitive to at least some portion of the medium-independent vehicle over which it is defined, then it counts as a computation. A digit, for example, can be said to be medium-independent because it can be implemented in completely different materials (e.g., silicon chips or vacuum tubes) in virtue of possessing the right structure (i.e., where it lies along a string). For Piccinini, only functionally integrated systems that compute at least one abstract function via vehicle manipulation qualify as concrete computing systems.

Given the conditions offered for implementation, it is often suggested that computational explanations take on a dual emphasis within the mechanistic account (Piccinini & Craver, 2011; Miłkowski, 2013; Piccinini, 2015, 2020).[4] In particular, computational explanations must not only provide abstract, functional characterisations of a physical system, but also a detailed, structural descriptions of how the system's component parts are organised and operate, what are respectively referred to as the "functional" and "structural" aspects of constitutive explanation. For example, to explain horizontal eye movement, a mechanistic computational analysis not only has to describe the function being computed by the ocular-motor system, such as an integration relation, but one also has to explain how the neurons in the ocular-motor system carry out the particular function via preserving morphic-relations between eye-velocity and eye-position. To qualify as a physical computing system, the ocular-motor system must be capable of sustaining a functional description in terms of an input-output relation and a structural description in terms of the activities and organisation of its component parts. A mechanism's ability to perform computations is explained mechanistically in terms of its components, their functions, and their organisation.

The mechanistic account has several advantages as theory of implementation. One is its claim to objectivity. According to Piccinini's account, for example, only those systems which implicate functional mechanisms processing medium-independent vehicles qualify as computing systems. The mechanistic account ties computational implementation directly to specific features of the world, providing a matter of fact as to whether or not physical systems qualify as computing systems. Another is its explanatory adequacy. A theory of implementation should explain computational implementation in terms better understood than computational implementation. Since physical computing is explained relative to the mechanistic framework, computational implementation is cashed out in terms of better understood notions, such as functional mechanism or teleological function. A third is its taxonomic adequacy. Because only those functional mechanisms that process medium-independent vehicles qualify as computing systems, for instance, paradigmatic cases of computing, such as Turing machines or calculators, qualify as computing systems, while non-

heart pumping blood (see Piccinini (2015, p.103) for details. For further discussion of the perspectivalist position, see Schweizer (2019) or Lee (2021).

[4] While I cannot defend the particular take of the mechanistic account on computational explanation here, for a defense, see Piccinini (2015, p.142).

paradigmatic cases, such as digestive systems or solar systems, do not. The concept of physical computation emerging from the mechanistic account is one that is sufficiently restrictive so as to be useful for explanatory purposes yet liberal enough to cover a number of important types of computing, such as Turing machines, neural networks, and digital/analog computers.[5]

The trouble is that the mechanistic account has come under pressure recently from several directions. Some, for instance, have worried that computational processes do not involve the manipulation of abstract medium-independent properties in the way that the mechanistic account suggests (Hutto et al., 2019); others that there is no method of identifying the generality of computational phenomena within the mechanistic account (Kersten, 2020); and some further still that computational and physical properties cannot be reconciled at all within a mechanistic hierarchy (Elber-Dorozko & Shagrir, 2019). Such challenges threaten to undermine the mechanistic account's status as a workable theory of implementation, putting pressure one or more of its key assumptions.

The aim of this paper is to shore up the conceptual foundations of the mechanistic account by offering a novel extension to the view. The goal is to rescue one prominent and influential theory of implementation from a series of recent challenges, and in the process further explicate its conceptual underpinnings. I begin, in Sect. 2, by outlining three outstanding challenges facing the mechanistic account, unpacking the key assumptions targeted by each. Then, in Sect. 3, I examine a recent response from Kuokkanen (2022a), one which appeals to a distinction between 'vertical' and 'horizontal' abstraction. I argue that Kuokkanen's proposal, while promising, nonetheless introduces a further problematic gap in thinking about implementation. This leads me, in Sect. 4, to propose an alternative. I suggest that computational description is better conceptualised as a form of *idealisation* rather than abstraction within the mechanistic account. I suggest that this conceptualisation not only offers the best way of making sense of implementation, but also a way of resolving the three outstanding problems. To further cement the benefits of the shift in thinking, I conclude, in Sect. 5, by taking up a further recent challenge.

2 Three challenges

As mentioned, three recent challenges have been levied against the mechanistic account.[6] Each puts pressure on one or more of account's key assumptions, and each attempt to show that computational explanation fails to conform in one or more ways to the norms of mechanistic explanation.

[5] There are other desiderata that are sometimes offered, such as miscomputation and non-circularity, but these three broadly capture some of the features regularly offered by theorists in favour of the mechanistic account (see, e.g., Miłkowski, 2013; Fresco, 2014; Piccinini, 2015; Sprevak, 2012, 2018).

[6] For discussion of further problems that have been raised, such as the decomposition problem, see Shagrir (2022, Ch. 6).

2.1 The abstraction problem

The first challenge is the "abstraction" problem (Haimovici, 2013; Hutto et al., 2019; Maley, forthcoming). While there have been several formulations of the worry, Hutto et al. (2019) offer a particularly clear expression, writing: "The trouble is that if medium independent vehicles are defined by their abstract properties then it is unclear how such vehicles could be concretely manipulated by their abstract properties" (p. 278).[7] The concern is that while it makes sense to say that computational processes are abstract (i.e., medium-independent), it makes less sense to say that abstract, computational processes are concrete processes *of* a mechanism.[8]

To illustrate, Hutto et al. focus on Piccinini and Bahar's (2013) discussion of neural spike trains (the time-series electrical signals recorded from individual neurons). Neural spike trains are said to be medium-independent according to Piccinini and Bahar in virtue of the fact that they (i) depend on functionally relevant aspects of the neural events, such as firing rates and timing, and (ii) can be implemented in different physical media, such as silicon chips. Brains are said to perform a generic form of computation in virtue of manipulating these medium-independent vehicles (see also Piccinini, 2015, p.120-5).

The trouble, though, as Hutto et al. see it, is that Piccinini and Bahar's discussion does not explain, even in principle, how neural spike trains, as abstract, medium-*independent* processes, could be processed by concrete, medium-*dependent* properties. They write, for instance: "Understanding how neural processes can be sensitive to concrete, medium dependent properties presents no conceptual difficulty. By contrast, we have no conception of how concrete neural process could causally manipulate, abstract medium independent vehicles" (p. 278). There seems to be a tension in saying that abstract, medium-independent properties are manipulated by concrete physical processes as some proponents of the mechanistic account do. Incorporating talk of concrete mechanisms appears to introduce a difficulty in understanding in what sense computational process and vehicles remain abstract.

The abstraction problem emerges from two different elements regularly brought together within the mechanistic account. The first, as we saw, is that computational explanations are said to require some level of structural detail. To qualify as a computational explanation, a given analysis must specify how a mechanism's component parts and activities are organised and operate to support a system's capacities – recall condition two of Piccinini's account, for instance. The second, though, is that computational explanations involve preserving certain degrees of freedom in their descriptions (i.e., medium-independence) – condition three of Piccinini's account. Computational explanations are thought to be necessarily abstract. These two assumptions appear set to pull in opposite directions. In putting the abstract and concrete together under one roof, the mechanistic account is seemingly led into confusing talk of concrete, physical processes manipulating abstract, medium-independent properties.

[7] Haimovici (2013) was the first to point out the tension, albeit in a slightly different form. The focus there was on the relation between functional and structural properties, but the conclusion is similar.

[8] For additional discussion, see Kuokkanen and Rusanen (2018) or Kersten (2020).

2.2 The generality problem

The second challenge is the “generality” problem (Elber-Dorozko & Shagrir, 2019; Kersten, 2020). The generality problem worries that the mechanistic account is unable to identify the generality of computational phenomena within its mechanistic hierarchies.

Part of the issue is the assumption that generality is determined by the vertical level of mechanistic hierarchies. For example, one might say that rat navigation is a “more” general phenomenon than memory because of its position at the top of a mechanistic hierarchy. Explanatory shifts from higher-level phenomenon, such as rat navigation, to lower-level component and activities, such as rat memory, involve a reduction in the generality of the phenomenon being explained. One method for fixing or determining the generality of a phenomenon within mechanistic analysis is to appeal to cross-situational stability (Boone & Piccinini, 2016). For example, one might say that because rats often make particular types of navigational errors in water mazes this reveals something important about how spatial maps are used in memory. Going beyond specific token instances allows researchers to identify the robust, cross-situational properties that form general features of rat memory.

The trouble is that cross-situational stability will not work as a method for determining the generality of computational phenomena. This is because cross-situational stability would require that physical and computational properties stand in part-whole relations to one another, the idea being that moving from higher or lower levels of a given hierarchy would reflect a corresponding move between compositionally related higher- and lower-level phenomena. However, as many have pointed out, the relation between computational and physical properties or entities is not one of part-to-whole; it is not a relation of constitution. Rather, physical and computational properties relate via a relation of *implementation* (Piccinini, 2015; Sprevak, 2018).

For example, whereas the components of a Dell Laptop are parts of the computer, the state of a transistor is not a part of the vehicle. A transistor does not constitute a computational vehicle but implements a medium-independent vehicle such as a bit. This means that cross-situational stability is unavailable as a method for determining generality. It makes matters unclear as to whether it is the computational description or the explanandum phenomenon that is more or less general. If it is the former rather than the latter, then a phenomenon’s generality is fixed or determined by the generality of an abstract description. This threatens to loosen the mechanistic account’s grip on objectivity. There is no way of tracking the computational properties back to their implementational counterparts. If computational descriptions are detached from their implementational properties, then the mechanistic account lacks the means to match the generality of a phenomenon and its abstract descriptions. Without a method of determining the generality of its phenomenon, the mechanistic account’s ability to provide abstract, objective descriptions of computational phenomenon begins to look tenuous.

2.3 The hierarchy problem

The final challenge is the "hierarchy" problem, or what has also been called the "problem of integration of the computational and mechanistic" (Elber-Dorozko & Shagrir, 2018, 2019; Shagrir, 2022). The hierarchy problem suggests that the mechanistic account faces a dilemma: it is either unable to explain how computational and implementational descriptions relate or it makes computational explanation non-mechanistic.

Consider the first horn of the dilemma. According to the mechanist, a computational-level C1 of some mechanistic hierarchy, consisting of the component parts (e.g., registers and circuits), their function, and their organisation, can be analysed by describing the computational components of an underlying computational level C0, such as logic gates. But C1 can also be described as being implemented by medium dependent physical properties, such as voltages. Call this level P0. Notice that it is unclear how P0 and C1 are supposed to relate if computations are part of a single mechanistic hierarchy. While P0 describes physical medium dependent properties, C1 describes abstract medium-independent properties. If the properties of P0 are not parts of the properties of C1 and vice versa, then P0 cannot be at a "lower level" than C1. There appear to be two sets of properties within a single mechanistic hierarchy, one computational (C0, C1, C2…) and one implementational (P0, P1, P2…), and it is unclear how they relate. The challenge is to explain how the relevant computational and implementational properties can be brought together within the same level(s) of a mechanistic hierarchy.

One way to avoid the issue is to separate computational and implementational properties. According to this "two-hierarchy view", computational properties reside on one mechanistic hierarchy, and implementational properties reside on adjacent levels of a separate hierarchy. Each level is a complete explanation of the phenomenon at the level above. For example, computational properties, such as a CPU, are explained by lower-level properties of the control unit, registers, and logic units, whereas silicon chips are constituted by ensembles of transistors. The benefit of this move is that it avoids explaining how the two hierarchies come together. Computational explanations are both abstract and full-fledged mechanistic explanations (Coehlo Mollo, 2018). They pick out two different mechanistic hierarchies: one computational; one implementational.

The trouble is that the separation into two hierarchies reduces computational explanation to form of functional analysis. This is the second horn of the dilemma. If there are now two separate hierarchies, then computational explanations no longer involve decomposing computations into lower level implementational properties. Rather, functional decompositions simply involve decompositions from one set of computational properties to another. Implementational properties may still figure into analysis, but only via mapping to a given computational level. The separation of computational and implementational may avoid the integration issue, but only at the cost of giving up on what makes the mechanistic account unique, i.e., mechanistic explanation. The dilemma, in short, is that the mechanistic account can either choose the single-hierarchy view, in which case it sacrifices explaining the relation between computational and implementational properties, or it can adopt for the two-hierarchy view, in which case it reduces computational explanations to (non-mechanistic) functional explanations. Neither option is particularly palatable.

So, to sum up, the mechanistic account faces three outstanding challenges. The first is that computational processes do not seem to involve the causal manipulation of abstract medium-independent properties (the abstraction problem); the second is that there is no method for identifying the generality of phenomena (the generality problem); and the third is that computational and physical properties do not appear to integrate (the hierarchy problem). Notice that the last two problems are closely connected. If the generality of a phenomenon is determined by its place in a mechanistic hierarchy, then the mechanist has to show how computational and implementational properties fit together.

3 Computational description as horizontal abstraction

Kuokkanen (2022a) has recently set out to tackle the abstraction, generality, and hierarchy problems. The aim is to rescue the "objective status" of abstract descriptions within the mechanistic account by showing how computational properties can be proper parts of the world.

To do so, Kuokkanen calls on a distinction between two forms of abstraction, what are called "vertical" and "horizontal" abstraction.[9] Vertical abstraction involves omitting details from a description when moving between levels of organisation, whereas horizontal abstraction requires fixing attention to level of organisation and then abstracting away from the physical details of entities at that level. Vertical abstraction requires movement from lower to higher levels of organisation, while horizontal abstraction requires more or less general descriptions of general entities at a given level.

According to Kuokkanen, computational descriptions are a product of horizontal abstraction within the mechanistic account. To be specific, they are the product of omitting irrelevant physical details at a given mechanistic level. As Kuokkanen (2022a) describes the view: "Both vertical and horizontal abstraction are at work, but what does the crucial work in capturing the relevant properties for the computational description is the horizontal abstraction" (p. 246). According to this picture, mechanistic levels are 'wide', in the sense that descriptions at one end of a mechanistic level provide full physical detail and at the other they provide abstract, computational detail.

To illustrate, consider, as Kuokkanen enjoins, how one might describe the states of a transistor. On the one hand, one might describe how a transistor is constituted or realised in different physical media, such as electrical circuits. This would be to abstract away from the physical details between levels. On the other hand, one might describe the states of the transistor in terms of the properties of a logic gate, i.e., 1s and 0s. This would be to abstract away from physical details at the same level of organisation. One can either move vertically between levels, or horizontally between more or less abstract descriptions at the same level. Computational descriptions are

[9] The original distinction is owed to Mäki (1992).

to be found on the horizontal axes, while constitutive, part-whole relations are to be found on the vertical axes.[10]

The vertical/horizontal distinction is also said to cut across the two "roles" of abstraction (Boone & Piccinni, 2016). One role is to identify the specific complex components, subsets of causal powers, and organisational relations that produce a general phenomenon, e.g., rat navigation. This is what is known as "ontic" abstraction. Another role is to strip unnecessary details from one's descriptions, e.g., details about the aerodynamics of a toy model plane. This is known as "epistemic" abstraction. If ontic and epistemic abstraction concern the *roles* of abstraction, then vertical and horizontal abstraction concern the *directions* of abstraction. Horizontal abstraction involves moving sideways at the same level, whereas vertical abstraction involves moving up and down between levels. Horizontal and vertical abstraction can be either epistemic or ontic. However, as we will see, to successfully defend the mechanistic account, an ontic conception of the distinction is required.

The question is, how does the move to horizontal abstraction help in addressing the abstraction, generality, and hierarchy problems?

First, notice that if computational descriptions are understood as the result of ontic horizontal abstraction, then the abstraction problem no longer appears to arise. If it right to say that computational descriptions pick out general phenomena within a given mechanistic level, then there is nothing particularly mysterious in saying how abstract, medium-independent properties relate to physical medium-dependent properties. Abstract, computational vehicles are simply more general phenomena residing at a given wide mechanistic level, ones identified via horizontal descriptive abstractions. The computational properties described by horizontal abstract descriptions correspond to proper parts of the world (i.e., descriptions of complex components, subsets of causal powers, and organisational relations at a given mechanistic level). There is nothing metaphysically spooky about them. Computational descriptions are both objective and metaphysically harmless.[11]

Second, recall that the generality problem worried there was no way of tracking computational properties back to their implementational counterparts. One consequence of the vertical/horizontal distinction, though, is that there are now two kinds of generality, one in which higher-level, general phenomena are abstracted away from lower-level phenomena (vertical abstraction), and one in which the physical properties are omitted at a given level resulting in medium-independent, computational entities (horizontal abstraction). Because of this, there is no need to track computational properties back to their implementational properties. The generality of a phenomenon is determined by the amount of detail provided by a horizontal descriptive abstraction within a given mechanistic level. As Kuokkanen (2022a) puts it: "In MAC [the mechanistic account of computation], computational or mathematical descriptions are arrived at by horizontal abstraction in the same mechanistic hierarchy one uses for determining the generality of a physical phenomenon." The generality problem

[10] Kuokkanen (2022a) suggests that one can maintain the vertical-horizontal distinction, even if computational descriptions are arrived at by horizontal descriptive abstraction. The distinction is a more general feature of modelling practice.

[11] For an alternative solution to the abstraction problem, see Kersten (2020).

does not arise because computational and implementational properties reside at the same mechanistic levels of a single hierarchy.

Finally, and following on from the previous solution, if computational descriptions are a product of horizontal abstraction, then the relation between computational and physical properties can be explained (the hierarchy problem). If varying the amount of detail at a given mechanistic level allows one to move from physical to computational descriptions (horizontal descriptive abstraction), then computational and physical properties can relate at any given level of a single mechanistic hierarchy. Computational and physical properties exist alongside one another on a continuum of abstractive description within a given mechanistic level. Because mechanistic levels are wide, the question of how to relate two separate hierarchies does not arise. There are not two separate computational and implementational hierarchies in need of integration but one hierarchy involving parallel computational and physical properties residing on wide mechanistic levels.

3.1 The limits of horizontal abstraction

Kuokkanen's (2022a) proposal is an admirable one. It offers a straightforward resolution to the three challenges and clarifies several aspects of abstraction, such as its roles versus directions. The problem is that the proposal, while promising, comes up short in delivering an account of *computational* description, and this spells trouble for the proposed solutions.

To see why, let us return to the case of a logic gate. In a standard digital computer, a logic gate is made up of combinations of transistors – physical devices which carry electrical currents, act either as a switch or amplifier, and are usually built out of silicon chips. Under normal conditions, if an electric current is present, then a transistor's states can be described as either 'on' or 'off'. If the transistor's states are on, then it can be assigned a value of '1'; if they are off, then it can be assigned a value of '0'. The upshot is a description of a logic gate. A logic gate is built out components which can hold two stable but different states. Combining sets of transistors in different ways creates different types of logic gates, e.g., AND or OR.

Notice, though, that while a logic gate can be built out of transistors, abstracting away from its physical properties does not deliver a description of '0s' and '1s'. Omitting detail from a physical description does not thereby transform it into a computational description. Ignoring a transistor's weight, size, or colour, for example, does not result in a mathematical description. It simply provides a more stripped back physical description. While simple, this point is important. Regardless of how helpful horizontal abstraction might be in capturing the physical structures within a given mechanistic level, it does not appear to account for the implementational relation between physical and computational properties. If horizontal descriptive abstractions are not capable, even in principle, of delivering *computational* descriptions, then there is an important gap at the centre of Kuokkanen's proposal. There is a clear limit to what horizontal abstraction can do in explicating mechanistic computation.[12]

[12] To be clear, Kuokkanen (2022a, b) is more interested in analysing Piccinini's version of MAC (e.g., how to interprets Piccinini's talk about abstraction) and offering some theoretical clarifications than arguing for

Interestingly, Kuokkanen (2022b) is alive to this issue, even framing it as a problem for the single hierarchy view, writing: "No matter how much descriptive abstraction one performs, she still needs to decide the rules according to which the mapping or implementation is carried out between the physical properties left from the descriptive abstraction and the mathematical properties." However, despite acknowledging the issue, there are important implications for the proposal as it relates to the three problems.

First, notice that if horizontal abstractive descriptions are not sufficient for producing computational descriptions, then it is unclear how to explicate the relationship between concrete physical processes and abstract, medium-independent properties (the abstraction problem). Horizontal abstractions were supposed to explain the sense in which abstract, computational vehicles could be said to interact with concrete physical processes. If they no longer do so, then it is once again unclear in what sense computational processes are concrete processes *of* a mechanism.

Second, if horizontal abstraction does not suffice to account for the presence of computational entities, then it seems that Kuokkanen's account no longer provides a means of determining the generality of computational phenomena (the generality problem). Recall that the generality problem was said not to arise because horizontal abstractions occurred within the same mechanistic hierarchy. But, if this is no longer the case, because abstractive description within a level does not suffice to identify computational properties, then there is once again a need to explain how computational properties can be tracked back to their implementational counterparts.

Finally, if computational properties do not arise as a result of horizontal abstraction, then computational and physical properties no longer exist within a single mechanistic hierarchy (the hierarchy problem). If varying the amount of detail at a given mechanistic level no longer moves one from a physical to computational description (via horizontal descriptive abstraction), then computational and physical properties cannot be related at a given level of a single mechanistic hierarchy. This means the integration issue re-emerges. The mechanist is forced to either sacrifice the relation between computational and implementational properties or reduce computational explanation to a form (non-mechanistic) functional explanation.

But perhaps there is a way to supplement the proposal. As hinted at, Kuokkanen (2022b) suggests that a mapping rule might suffice to connect descriptive abstractions to computational descriptions, writing: "To arrive at a computational description, we make an agreement of a sort: we must agree that when the current in a certain transistor is between, say, x and y, its state is 1. This establishes the implementation relation: we set some rules according to which we map mathematical descriptions or properties onto physical descriptions." In addition to abstractive description, the suggestion is that a full description of the implementation relation also requires making a contract or decision so as to determine when the mathematical or computational entities map onto physical entities at a given level of a mechanistic hierarchy.

There is trouble looming for such a proposal, though. If horizontal descriptive abstraction is not sufficient for computational description and additional interpreta-

or defend the mechanistic account. Here I am simply extending the analysis to mechanistic computation more generally insofar as it offers a solution to the three challenges.

tive choices on the part of the observer are required, then Kuokkanen's proposal no longer appears to deliver the objectivity it once promised for the mechanistic account. Recall that part of the goal was to show that computational descriptions within the mechanistic account were *ontic* in nature – that is, that they involve identifying the specific complex components, subsets of causal powers, and organisational relations that produce a general phenomenon. Descriptions earn their ontic status when their structures successfully track or match those of the world. However, it now seems that the accuracy of a computational description depends on the choice of mapping rule. If this is right, then computational descriptions are primarily the product of descriptive choices, rather than a property of the processes being described; there is nothing that in principle constrains the mapping choice. Kuokkanen's proposal seems to be in a bind: it either, at best, comes up short with respect to explaining computational description or, at worst, undermines the mechanistic account's claim to objectivity. Neither option is particularly palatable.

To be clear, the worry is not that the vertical-horizontal distinction itself is suspect or that horizontal abstraction does not occur within mechanistic hierarchies. Rather, the concern is that in tying computational description to horizontal abstraction, the mechanistic account either has to leave the implementational relation unexplained or, in explicating the relation, it has to give up an important part of what makes the mechanistic account initially desirable as a theory of implementation: namely, objectivity. Thus, if there is a way to resolve the three problems while avoiding these outcomes, then such a proposal would be doubly appealing. It is to this task I now turn.

4 Computational description as idealisation

Part of the trouble for Kuokkanen's proposal, I think, is that it overemphasises the notion of *abstraction* in thinking about mechanistic computation. This is understandable to some extent, but it throws up barriers in coming to grips with implementation. The better approach, I want to suggest, is to think of computational description within the mechanistic account not as a form of abstraction but as a type of *idealisation*.[13]

While there are a number of ways to draw the distinction, one plausible way, and the one I adopt here following Portides (2018), is to think of abstraction as a process of omitting features of a phenomenon from a model and idealisation as a process of modifying features that are retained in a model.[14] On this reading, the difference between abstraction and idealisation is the difference between attending to certain features of a phenomenon within a model and attending to the ways in which certain model features of a phenomenon can be treated. One of the benefits of this framing is that it avoids thorny debates about the role of approximation and truth/falsehood,

[13] Schweizer (2019) articulates a somewhat related idea. However, while his "computational perspectivalism" also emphasises computational descriptions as a form of idealisation, it does not do so from within a mechanistic view, nor does it develop a conception of computation as idealisation specifically with an eye to explicating physical computation. The view is also explicitly non-objective and observer-dependent.

[14] For alternative ways of drawing the distinction, see Jones (2005), Godfrey-Smith (2009) or Morrison (2015).

while nonetheless providing a relatively straightforward way to capture the intuitive difference between the two notions.[15]

For instance, in mechanics, the material composition and internal structure of a body are often ignored in setting-up equations for motion. In such cases, the omission of certain features of a phenomenon (e.g., weight, height, etc.) results in a simplified description of a target system, e.g., a body in motion. A mechanical model is created by focusing on certain features of a target system at the expense of others. This is an instance of abstraction. In building a model of population genetics, however, mating (the explanandum phenomenon) is often treated as random, population size as infinite, and generations as non-overlapping. In this instance, certain features are omitted in the model, but the retained features are not treated as they normally appear in actual circumstances (e.g., real populations are finite, mating patterns are non-random, and grandparents often co-exist with their grandchildren). The model involves modifying certain features found in an actual system for various model-specific ends, e.g., to avoid the evolutionary effects of genetic drift. This is a case of idealisation.

The distinction can also be drawn more formally. As Portides (2018, pp. 5889-90) characterises the two notions:

Abstraction Let Σ consist of all features of an arbitrary system (not necessarily all identified) that could contribute to the parameters of theory and could be expressed by means of those parameters. Let Δ be a set of identified features, where $\Delta \subset \Sigma$. Abstraction is the act of attending to Δ as if all other members of Σ were absent.

Idealisation Let Σ consist of all logically possible ways by which perceived features (or more generally, knowledge about features) of actual systems could be modified. Let every member of Σ be expressible by means of the parameters of theory and not conflict with theoretical principles. Then Σ consists of all possible ways by which the features retained in a model description could be treated. Let Δ be a set of ways by which the different retained features are treated or considered, where $\Delta \subseteq \Sigma$. Idealisation is the act of attending to Δ.[16]

Whereas abstraction involves selectively attending to a subset of features within a target system relative to the parameters of a theory, idealisation involves selectively attending to a subset of features once they have been modified or treated according to the parameters of a background theory.[17]

As the forgoing suggests, abstraction and idealisation are similar in several respects. Both result in simplified descriptions, both involve a process of selective attention, both operate relative to background theories, and both lead to distortions in model features (not all features of a target system are included in a model). However, despite such similarities, the two processes nonetheless come apart. As Portides

[15] For further detail and defence of the view, see Portides (2018).

[16] Portides' (2018) formulation is itself adapted from an earlier version by Mäki (1992, pp.107–139).

[17] This characterisation of idealisation is similar to 'Galilean' and 'minimalist' forms, in that it also can involve distortions and false assumptions under the guidance of a representational ideal, but it does not require that its modifications, strictly speaking, have to be false (see Weisberg, 2007).

(2018, p.5891) notes, attending to certain features at the expense of others (abstraction) is a way of stressing the priority of those features in explaining some observable behaviour or phenomenon, whereas attending to certain ways features of a phenomenon can be treated (idealisation) is a way to meet the more specific aims of a model, such as screening-off casual factors, grouping particulars into a genus, or making approximations about a system.

It should be relatively clear, I hope, that Kuokkanen's proposal thinks of computational description along the lines of abstraction rather than idealisation. For instance, in describing the relationship between computational description and horizontal abstraction, he writes: "In MAC, computational descriptions are located a certain point on the horizontal axes. One arrives at the point of computational description by omitting irrelevant physical properties at level." For Kuokkanen, arriving at a computational description involving omitting sufficient physical details from one's descriptions at a given mechanistic level. The movement from physical to computational description requires attending to features of a target system and not others.

We have already seen the limits of this approach, but we are now in a position to explain why it comes up short. While it is certainly true to say that computational description involves omitting physical detail, the retained model features must also undergo a transformation during conceptual isolation. That is, computational description involves modifying or treating the retained features of a mechanistic model in accord with the parameters of computational theory. Adapting Portides' (2018, p. 5985) formulation slightly, if every member of Σ of a mechanistic model is expressible by means of the parameters of computational theory, then Σ consists of all possible ways by which the features retained in a physical description could be treated. If Δ picks out a set of ways of attending to how the retained features are treated or considered, then computational descriptions are idealisations, where $\Delta \subseteq \Sigma$. This is how physical descriptions within a mechanistic model can be turned into computational descriptions.

For example, consider the case of modelling a standard digital computer. First, a number of physical structures and properties of the target system can be described, e.g., the computer's transistors, registers, memory units, etc. – in the present terminology, these properties and structures constitute the 'retained' features of the model, as not all the features of the target system are included in the model. Then, such features can be arranged or organised into different levels, which constitutes a mechanistic hierarchy – when focusing on a specific level of this hierarchy, we identify a subset of the features within the target system, such as the physical properties of its transistors or registers.

The claim here is that to arrive at a description of computational properties we must further selectively attend to a subset of the features of the mechanistic model and modify them in accord with the assumptions of computational theory. For example, at the level of primitive computing component, a transistor can be only interpreted as a logic gate when its states are assigned the values 1 and 0. This interpretation is only possible if the transistor is able to support two stable but different states. If the transistor is not bi-stable, then it conflicts with the requirements of being a computational vehicle, i.e., having the requisite two degrees of freedom. It is in virtue of attending to a subset of the features of the transistor (i.e., its degrees of freedom), as they accord with computational theory, that the description of the physical transistor turns into a mathematical description of a device that processes 1s and 0s, i.e., a logic gate.[18]

Further cases can be provided using empirical examples. For instance, when modelling rat navigation, researchers often construct a hierarchy of levels, ranging from outward behaviour and spatial memory to hippocampal and single neuron activity (Craver, 2007, 2013). In constructing such a representation, there is a focus on a subset of physical properties within the target system which are modified in ways that do not align with biological reality, such as treating the all-or-nothing principle found in neurons as a stepwise function or long-term potentiation as form of Hebbian learning. This is done in order to explore various information processing capacities of the system, such as why rats exhibit certain rotational errors in mazes. Notice that in such cases certain features of the target phenomenon are omitted, such as a neuron's refraction rate or depolarisation, while others, such as the all or nothing-or-nothing principle, are treated in ways that align with certain background principles, e.g., information processing theory. In constructing a mechanistic model of rat navigation, computational properties emerge via a process of omission and modification.

For present purposes, what is particularly interesting about the shift from abstraction to idealisation is that it allows us to make sense of the omission process found in computational descriptions, but it does not leave the relationship between physical and computational properties unaccounted for. This is because the move from physical to computational description, on the proposed account, involves selectively attending to a subset of retained features within a mechanistic model and treating them in accord with computational theory. This process does involve omitting physical details, but it is not this fact alone which transforms a physical description into a computational one. Rather, a computational description requires focusing on the ways in which certain features of a mechanistic model can be treated in light of constraining computational assumptions; computational description requires more than abstraction.

It is quite understandable, though, why Kuokkanen (2022a) and others have come to focus on abstraction. The hierarchical nature of mechanisms naturally lends itself to talk of 'abstracting away' from physical details – mechanists, such as Piccinini (2015), for example, who's account forms the core of Kuokkanen's analysis, regularly speaks of computational descriptions as being a product of "abstracting away" from lower-level details. In wedding computational implementation to the mechanistic framework, it is not surprising that some have been tempted to think of the movement between computational and physical properties as a form of abstraction. Moreover, given that abstraction usually precedes idealisation in model building, in that one needs to attend to certain features before one can attend to the ways the features can be treated, it is understandable that the distinction between abstraction and idealisation has been largely overlooked in discussions of the mechanistic account. As we saw, the two processes are often tied up together in building a representation of a target system.

[18] In many ways, this is not a new point, but the reframe is important because it does much to bring out what is often implicit in discussion of mechanistic computation (e.g., Miłkowski, 2013; Piccinini, 2015).

Nonetheless, this does not mean that abstraction and idealisation are the same, nor that computational description should be thought as a form of abstraction. As we have seen, doing so results in an important gap in our account of implementation, whereas treating computational description as a form of idealisation helps to close this gap. Reframing computational description as a form of idealisation not only helps to explicate the implementational relation within the mechanistic account but it also accords nicely with scientific modelling practice.[19]

4.1 The abstraction, generality, and the hierarchy problems: redux

If computational description is indeed better understood as a form of idealisation rather than abstraction within the mechanistic account, then what follows? Let us return to the abstraction, generality, and hierarchy problems to see.

First, notice that if computational descriptions are idealisations, then computational processes and structures reflect a particular way of describing certain features within a mechanistic model; computational properties are modified subsets of retained model features. This provides a clue as to how abstract, computational processes can be concrete processes *of* a mechanism (i.e., the abstraction problem). Abstract, medium-independent properties relate to concrete, medium-dependent properties in virtue of the latter being the properties out of which the former are assembled. Computational vehicles are general types of phenomena, residing at a given mechanistic level, identified through conceptual isolation and modification. For example, returning to the case of Piccinini and Bahar's neural spike trains, a computational description first depends on identifying the functional features of a mechanism, such as firing rates and timing, and describing them in ways that satisfy a particular an input-output relation. Evaluating whether populations of spiking neurons are medium-independent involves assessing the degrees of freedom amongst electrochemical signals in the brain. Computational descriptions are the product of modifying or treating subsets of physical properties relative to a mechanistic model in accord with computational theory. They are objective yet metaphysically innocuous.

Second, consider that if the current proposal is right, then, relative to a mechanistic model, computational properties are fixed at a given level. As we saw, in transforming a physical description into a computational description, one needs to focus on a subset of physical properties in a given mechanistic model that accord with constraining background assumptions – for example, that the structures possess the requisite degrees of freedom to be described as computational vehicles. Another way to put this is to say that for every member of Σ of a mechanistic model that is expressible by means of the parameters of computational theory, Δ picks out a subset of ways members of Σ can be treated. This means there is no need to track computational properties back to their implementational counterparts. Each mechanistic level includes those computational entities which are formed out of the relevantly constrained

[19] To be clear, while it may be the case that idealisation is involved in computational explanation more generally, the current proposal is only meant to apply to the mechanistic account. The question of the nature of computational explanation is a complex and vexed one. I am here simply working under the assumption that the mechanistic account offers one plausible account of computational explanation.

physical properties. The generality problem does not arise because computational phenomena are tied or locked-in to each level of a mechanistic hierarchy. Similar to Kuokkanen's proposal computational and implementational properties reside at the same mechanistic levels of a single hierarchy, but, unlike Kuokkanen's proposal, they are not purely a product of descriptive abstraction. Rather, they are formed through idealising selections.

Finally, and following on from the previous solution, if computational properties are indeed products of idealisation, then there is a clear sense in which computational and implementational properties can co-exist within a single hierarchy (the hierarchy problem). For example, as mentioned, for a given computational level C1 of some mechanistic hierarchy (consisting of the component parts, their function, and their organisation), C1 can relate to an underlying computational level C0 in virtue of mapping the set of features picked out by C1 to the set of features picked out by C0 within a hierarchy. But this does not mean that C1 cannot also be described in terms of a set of medium *dependent* properties, such as voltages. In fact, it is to be expected. P describes the unmodified set of physical properties populating a mechanistic level which form the basis of idealising selections resulting in C1. The relation between C1 and P is not one of two different sets of properties (computational and implementational) but, rather, one set of properties selectively attended to in different ways within the same level. Computational and implementational properties can be brought together within the same level(s) of a mechanistic hierarchy insofar as one selectively attends to different features of the retained model features relative to different constraining background theories (e.g., functional versus computational).

The question remains, though, does treating computational description as a form of idealisation deliver the objectivity required by the mechanistic account? Recall here that a description earns its ontic status if it tracks or matches its structures with those of the world. In the present context, this means that a computational description must pick out complex components, subsets of causal powers, and organisational relations within a target phenomenon. As we saw, computational properties are the product of omitting and modifying certain features within a mechanistic model. However, mechanistic models are also representations of target physical systems, such as neurons or digital computers; mechanisms are always mechanisms *of* something (Craver, 2013; Miłkowski, 2013). This means that the model features out of which computational properties are assembled are poised to track structures in a target physical system, such as a digital computer. Of course, whether a particular target system is apt to be described along computational lines is subject to the further conditions of explanatory utility and accuracy, but the representational elements found within the mechanistic model nonetheless correspond to features of the world. Relative a given mechanistic model, it can be true or false to say of a physical system that it implements a particular computation.

It is important to be clear about the form of objectivity secured here, as one might worry that the idealisation view makes computational ascriptions agent relative in some problematic sense (i.e., non-naturalistic or pragmatic). Thankfully this is not the case. For while it is true that computational descriptions are relative to a mechanistic model according to the current proposal, this does not mean that agents can independently decide what each variable in a system represent. This is because, as mentioned, computational ascriptions are still importantly constrained by physical features of a mechanism. Recall, for example, that if a transistor is not capable of being described as bi-stable, then it conflicts with the requirements of being a computational vehicle. A computational vehicle must possess spatiotemporal parts that can take on different values and change over time. A transistor without such features fails to have the requisite degrees of freedom so as to qualify as computational; mapping choices are constrained by the world in various ways.[20] The claim is not that that one should modify the *phenomena* to match the relevant background theory, which would make computational ascriptions agent-centric. Rather, the suggestion is that ascriptions of computational processes depend on the features of the world aligning or conforming to the relevant background theory, e.g., computation. Conceptualising computational description as a form of idealisation, at least within the mechanistic framework, does secure an important form objectivity.

What is more, as argued, abstraction and idealisation are both defined relative to background theories (e.g., computation). Whereas abstraction involves selectively attending to a subset of features within a target system relative to the parameters of a theory, idealisation involves selectively attending to a subset of features once they have been modified or treated according to the parameters of a background theory. This means that the notion of objectivity secured here is the same as the one at play in Kuokkanen's account. Abstractive descriptions also operate relative to the assumptions of computational theory, a point, recall, revealed by the fact that Kuokkanen (2022b) suggests that a mapping rule might suffice to connect descriptive abstractions to computational descriptions. Thus, even if one remains sceptical of the form of objectivity secured by the current proposal, Kuokkanen's proposal fares no better in establishing a notion of objectivity via abstraction – this is in addition to the sceptical considerations offered in Sect. 3.1. There would still be reason to prefer the proposed account over Kuokkanen's even if one remained sceptical of the notion of objectivity delivered, as the current account solves the three challenges while also explicating the implementational relation.

So, to summarise the results of the section: first, computational descriptions are a form of idealisation within the mechanistic account; second, conceptualising computational description as a form of idealisation helps to explicate the implementation relation; third, thinking of computation as a type of idealisation provides a solution to the three outstanding challenges; and fourth, the suggested shift secures a form of objectivity for the mechanistic account. Taken as a whole, these considerations should begin to tip the balance, I think, in favour of the proposed account. While Kuokkanen's proposal is instructive in several respects, the current proposal helps to resolve the three problems while also further explicating implementation and retaining a sense of objectivity. It preserves the core elements of Kuokkanen's proposal – the idea that omitting physical details is involved in ontic abstraction – but does so without compromising other aspects of the mechanistic account.

[20] It is worth mentioning that Shagrir (2020, 2022) marshals this point, in his master argument, to argue for more than one possibility open for mapping. However, the argument, while important, applies more directly to account of individuation rather than implementation, which is the present concern, and so it worth setting it to one side for the moment.

5 Mechanistic computation and information processing

I want to address a final concern now, and in so doing further cement the benefits of thinking of computational description as a form idealisation.

In a recent, wide-ranging treatment, Shagrir (2022) argues that the most pressing objection to the mechanistic account is the fact that it downplays the importance of information processing and representation in the cognitive and neural sciences.[21] For example, the 'normalization equation', which can be found across the nervous system, provides a quantitative description of a cell's response. It helps to explain why parts of the nervous system exhibit certain behaviours, such as cross-orientation suppression. But the normalization equation does not make essential reference to causal structures.[22] Rather, it explains the behaviour of the nervous system via reference to a single computational principle, i.e., that neurons are able to transit more information if their individual firing rate is suppressed by the population average. If this is right, then the mechanistic account appears to offer an inadequate account of computation. As Chirimuuta (2014, p.128) elsewhere makes the point: "Information-theoretic and computational principles are central to interpretative modelling, and causal-mechanical descriptive detail and accuracy is not required." Computational explanations not only aim to describe *how* mathematical functions are implemented and performed, but also *why* the mathematical operations relate to the information processing task being performed.

Notice, though, that, as we saw, while one of the constraints on idealisation is the background theory (e.g., computational theory), another is the *aim* of a model. There are many different aims in making idealising selections, and any particular idealisation can serve more than one aim at a time; another way to make this point is to say that there can be multiple 'representational ideals' on a model (Weisberg, 2007). For example, as mentioned, when it is assumed that population size is infinite in a population genetic model, this modifies certain features of the target system. But doing so also aids in screening off complicating causal variables, such as those concerning evolutionary affects from genetic drift. Attending to the different ways features of a phenomenon can be treated (idealisation) is a way of meeting the more specific aims of a model, where that might mean screening-off casual factors, grouping particulars into a genus, or making approximations about a system.

A similar point holds in the case of mechanistic computation. While one of the aims of a computational model within a mechanistic analysis is to describe the internals of mechanism parts and their interactions (i.e., answering how-questions), another is to account for why a given mathematical function relates to its wider task environment (i.e., answering why-questions). Miłkowski (2013) calls these the *isolated* and *contextual* levels of analysis. For example, in a cash register, at the contextual level, one describes the cash register as playing a certain role in a supermarket, such as enabling

[21] For a similar worry based on interpretative models in cognitive neuroscience, see Chirimuuta (2014).

[22] Chirimuuta (2014) argues that it is efficient coding which delineates the set of counterfactual dependencies between input to the system (e.g. sensory information) and/or system requirements (e.g. task for which information is needed) and the computational properties of the system. For replies showing how efficient coding might be integrated with the mechanistic account, see Wajnerman Paz (2017) or Fuentes (2023).

the calculation of sums; one includes here a description of a bar-code scanner, a conveyor belt, etc. On the other hand, at the isolated level, a dedicated computer using special software is described, one which enables the commutativity or associativity of addition, for example.

While the isolated level describes how the parts of the mechanism are specified along with their interactions (activities or operations), the contextual explicates the function of the mechanism in its broader context (i.e., how inputs and outputs of the system connect with the surrounding context). The necessity of the contextual level stems from the fact that knowing which computing system's properties are relevant to a mechanism's inputs and outputs requires knowing how the mechanism's inputs and outputs interact with their context. Without knowing which external events cause which internal events, one cannot distinguish the functionally relevant and irrelevant properties of a mechanism (Piccinini, 2015, p.139).[23]

For example, returning the ocular-motor system case, whereas the how-question describes the mathematical function as computed by the neural integrator, the why-question describes the relation between the computed function and the physical environment. Here the analysis invokes a physical constraint: namely, the relation between eye velocity and the distance between successive eye positions. This is what helps to distinguish the function being performed as an integration relation rather than, say, multiplication or exponentiation (Bechtel & Shagrir, 2011, p.316). There is a morphism mapping relation between the function and the target domain. A complete mechanistic computational explanation requires analysis at both levels. It involves a formal specification of the information processing task being performed and a specification of relevant implementational variables, e.g., the relevant algorithms.

The importance of this point for present purposes is that there is nothing that prevents computational explanations from explicating the relationship between a mathematical function and an information processing task. It simply requires shifting attention to the different explanatory aims of the computational model – aims which are reflected in the different levels of analysis one can take during computational explanation. A mechanistically adequate model of computation must answer both why and how questions at multiple levels, each corresponding to the different aims of selective idealisations. The idealisation view provides a lens through which to view the multiple functions of computational descriptions within the mechanistic account.

Why, though, is it not enough to say that the semantic account, which does make information essential to computational implementation, is better equipped to handle such cases? How is the mechanistic account superior in dealing with cases such as the normalisation equation? This is Chirimuuta (2014) and Shagrir's (2022) take after all. This is an interesting question, but it is not, strictly speaking, the present concern. The challenge is whether the mechanistic account has the resources *in principle* to make sense of information processing. As we have seen, it does. While there may be additional reasons to favour the semantic account, the mechanistic account, particularly when interpreted along idealisation lines, does not lack the tools to make sense of the relation.[24] To be clear, the argument on offer does not establish the superiority of

[23] For an externalist reading of this point, see Kersten (2017).

[24] For further discussion of the relation, see Piccinini (2018).

the mechanistic account over rival accounts of implementation, such as the semantic account; that is a different argument for a different day. Rather, it provides additional reasons to favour the idealisation view as a right interpretation of the mechanistic account. This outcome is slightly more modest, but it is still important.

With all that said, it is perhaps understandable why the preoccupation with abstraction has led some to suggest that computational description involves a process of subtracting features from a target system (i.e., abstraction); Kuokkanen's proposal encourages such a view, for example. Nonetheless, such a conception gives the misleading the impression that computational explanations only relate to implementational details, whereas re-envisaging mechanistic computation along idealisation lines helps to further reveal the varied nature of computational descriptions.

6 Conclusion

So, what are the key takeaways? First, the mechanistic account should move away from thinking about computational description in terms of abstraction. As we have seen, this approach is useful, but it comes at a cost. Second, the mechanistic account is better served by thinking of computational description as a form of *idealisation*. As argued, this shift not only helps to explicate implementation, but it also offers a clear path to resolving the three outstanding challenges facing the mechanistic account. The idealisation view allows the mechanistic account to make sense of the omission process found in computational descriptions without leaving the relationship between physical and computational properties mysterious. And finally, shifting to the idealisation view helps to secure an important form of objectivity for the mechanistic account. As we saw, it provides a matter of fact as to whether or not a physical system qualifies as a computing system, preserving a key motivation for the mechanistic account as a theory of implementation. Thus, while the current account still requires further elaboration and defence, I think it adds a constructive proposal for how to further explicate the conceptual foundations of mechanistic computation, as well as how to respond to several outstanding criticisms.

Acknowledgements I want to thank two anonymous reviewers from the journal for their helpful feedback and comments on earlier drafts of the paper. This research was generously supported by the Killam Trusts.

Declarations

Competing Interests The author declares no competing interests.

References

Bechtel, W., & Shagrir, O. (2011). The non-redundant contributions of Marr's three levels of analysis for explaining information processing mechanisms. *Topics in Cognitive Science*, *7*(2), 312–322. https://doi.org/10.1111/tops.12141.

Boone, W., & Piccinini, G. (2016). Mechanistic abstraction. *Philosophy of Science, 83*, 686–697. https://doi.org/10.1086/687855.

Chalmers, D. (1994). On implementing a computation. *Minds and Machines*, *4*(4), 391–402. https://doi.org/10.1007/BF00974166.

Chalmers, D. (2011). A computational foundation for the study of cognition. *Journal of Cognitive Science*, *12*(1), 323–357.

Chirimuuta, M. (2014). Minimal models and canonical neural computations: The distinctness of computational explanation in neuroscience. *Synthese*, *191*(2), 127–153. https://doi.org/10.1007/s11229-013-0369-y.

Chrisley, R. (1995). Why everything doesn't realize every computation. *Minds and Machines*, *4*, 403–430.

Coelho Mollo, D. (2018). Functional individuation, mechanistic implementation: The proper way of seeing the mechanistic view of concrete computation. *Synthese*, *195*, 3477–3497. https://doi.org/10.1007/s11229-017-1380.

Craver, C. (2007). Constitutive explanatory relevance. *Journal of Philosophical Research*, *32*, 1–20. https://doi.org/10.5840/jpr_2007_4.

Craver, C. (2013). Functions and mechanisms: A perspectivalist account. In P. Huneman (Ed.), *Functions: Selections and Mechanisms* (pp. 133– 158). Dordrecht: Springer.

Dewhurst, J. (2018). Individuation without representation. *British Journal for the Philosophy of Science*, *69*(1), 103–116. https://doi.org/10.1093/bjps/axw018.

Elber-Dorozko, L., & Shagrir, O. (2018). Computation and levels in the cognitive and neural sciences. In M. Sprevak, & M. Colombo (Eds.), *The Routledge Handbook of the computational mind* (pp. 205–225). Routledge. https://doi.org/10.4324/9781315643670.

Elber-Dorozko, L., & Shagrir, O. (2019). Integrating computation into the mechanistic hierarchy in the cognitive and neural sciences. *Synthese*. https://doi.org/10.1007/s11229-019-02230-9.

Fodor, J. (1981). *Representations: Philosophical essays on the foundations of Cognitive Science*. MIT Press. A Bradford Book.

Fresco, N. (2010). Explaining computation without semantics: Keeping it simple. *Minds and Machines*, *20*, 165–181. https://doi.org/10.1007/s11023-010-9199-6.

Fresco, N. (2014). *Physical computation and Cognitive Science*. Springer.

Fresco, N. (2021). Long-arm functional individuation of computation. *Synthese*, *199*(6), 5. https://doi.org/10.1007/s11229-021-03407-x.

Fresco, N., & Miłkowski, M. (2021). Mechanistic computational individuation without biting the bullet. *British Journal for the Philosophy of Science*, *72*(2), 431–438.

Fuentes, J. (2023). Efficient mechanisms. *Philosophical Psychology*. https://doi.org/10.1080/09515089.2023.2193216.

Garson, J. (2003). The introduction of information into Neurobiology. *Philosophy of Science*, *70*(5), 926–936. https://doi.org/10.1086/377378.

Godfrey-Smith, P. (2009). Abstractions, idealizations and evolutionary biology. In A. Barberousse, M. Morange, & T. Pradue (Eds.), *Mapping the future of biology: Evolving concepts and theories. Boston studies in the philosophy of science* (pp. 47–55). Springer.

Haimovici, S. (2013). A problem for the mechanistic account of Computation. *Journal of Cognitive Science*, *14*, 151–181.

Hutto, D., Myin, E., Peeters, A., & Zahnoun, F. (2019). The cognitive basis of computation: Putting Computation back in its place. In M. Colombo, & M. Sprevak (Eds.), *The Routledge Handbook of the computational mind* (pp. 265–281). Routledge.

Jones, M. (2005). Idealization and abstraction: A framework. In M. Jones, & N. Cartwright (Eds.), *Idealization XII: Correcting the model, idealization and abstraction in the sciences* (pp. 173–217). Rodopi.

Kersten, L. (2017). A mechanistic account of wide computationalism. *Review of Philosophy and Psychology*, *8*(3), 501–517. https://doi.org/10.1007/s13164-016-0322-3.

Kersten, L. (2020). How to be concrete: Mechanistic computation and the abstraction problem. *Philosophical Explorations*, *23*(3), 251–266. https://doi.org/10.1080/13869795.2020.1799664.

Kuokkanen, J. (2022a). Vertical-horizontal distinction in resolving the abstraction, generality and hierarchy problems. *Synthese*, *200*, 247. https://doi.org/10.1007/s11229-022-03725-8.

Kuokkanen, J. (2022b). No computation without implementation? A potential problem for the single hierarchy view of physical computation. *Synthese*, *200*(370). https://doi.org/10.1007/s11229-022-03696-w.

Kuokkanen, J., & Rusanen, A. (2018). Making too many enemies: Hutto and Myin's attack on Computationalism. *Philosophical Explorations*, *21*(2), 282–294. https://doi.org/10.1080/13869795.2018.1477980.

Lee, J. (2021). Mechanisms, wide functions, and content: Towards a Computationalism pluralism. *British Journal for the Philosophy of Science*, *72*(1), 221–244.

Leigh, J., & Zee, D. (2006). *The neurology of Eye Movements*. Oxford University Press.
Maley, C. J. (forthcoming). Analogue computation and representation. *The British Journal for the Philosophy of Science*. https://doi.org/10.1086/715031.
Miłkowski, M. (2013). *Explaining the computational mind*. MIT Press.
Miłkowski, M. (2015). Computational mechanism and models of Cognition. *Philosophia Scientiae*, *18*(3), 1–14.
Mäki, U. (1992). On the method of isolation in Economics. *Poznan Studies in the Philosophy of the Sciences and the Humanities*, *26*(4), 317–351.
Morrison, M. (2015). *Reconstructing reality*. Oxford University Press.
Piccinini, G. (2007). Computing mechanisms. *Philosophy of Science*, *74*, 501–526. https://doi.org/10.1086/522851.
Piccinini, G. (2015). *Physical computation: A mechanistic account*. Oxford University Press.
Piccinini, G. (2018). Computational mechanisms. In S. Glennan, & P. Illari (Eds.), *The Routledge Handbook of mechanisms and Mechanical Philosophy* (pp. 435–446). Routledge Taylor & Francis Group.
Piccinini, G. (2020). *Neurocognitive mechanisms: Explaining Biological Cognition*. Oxford University Press. https://doi.org/10.1093/oso/9780198866282.001.0001.
Piccinini, G., & Bahar, S. (2013). Neural computation and the computational theory of Cognition. *Cognitive Science*, *37*(3), 453–488. https://doi.org/10.1111/cogs.2013.37.issue3.
Piccinini, G., & Craver, C. (2011). Integrating psychology and neuroscience: Functional analyses as mechanism sketches. *Synthese*, *183*, 283–311.
Portides, D. (2018). Idealization and abstraction in scientific modeling. *Synthese*. https://doi.org/10.1007/s11229-018-01919-7.
Putnam, H. (1975). The mental life of some machines. *Mind, language and reality, philosophical papers*, volume 2, (pp. 408–28). Cambridge University Press: Cambridge.
Schweizer, P. (2019). Computation in Physical systems: A normative mapping account. In D. Berkich, & M. Vincenzo d'Alfonso (Eds.), *On the cognitive, ethical, and scientific dimensions of Artificial Intelligence* (pp. 27–47). Philosophical Studies Series, Springer.
Shagrir, O. (2006). Why we view the brain as a computer. *Synthese*, *153*, 393–416. https://doi.org/10.1007/s11229-006-9099-8.
Shagrir, O. (2020). In defense of the semantic view of computation. *Synthese*, *197*, 4083–4108.
Shagrir, O. (2022). *The nature of physical computation*. Oxford University Press.
Sprevak, M. (2010). Computation, Individuation and the received view on representation. *Studies in the History of Philosophy of Science Part A*, *41*, 260–270.
Sprevak, M. (2012). Three challenges to Chalmers on computational implementation. *Journal of Cognitive Science*, *13*, 107–143.
Sprevak, M. (2018). Triviality Arguments About Computational Implementation. In *The Routledge Handbook of the Computational Mind*, edited by M. Sprevak, and M. Colombo, 175–191. London: Routledge.
Wajnerman Paz, A. (2017). Pluralistic mechanism. *Theoria: Revista De Teoría. Historia y Fundamentos de la Ciencia*, *32*(2), 161–175.
Weisberg, M. (2007). Three kinds of idealization. *Journal of Philosophy, 104*(12), 639–659.