No computation without implementation? A potential problem for the single hierarchy view of physical computation

Kuokkanen, Jesse

2022-08-30

Kuokkanen , J 2022 , ' No computation without implementation? A potential problem for the single hierarchy view of physical computation ' , Synthese , vol. 200 , no. 5 , 370 . https://doi.org/10.1007/s11229-022

http://hdl.handle.net/10138/354777
https://doi.org/10.1007/s11229-022-03696-w

Synthese (2022) 200:370
https://doi.org/10.1007/s11229-022-03696-w

ORIGINAL RESEARCH

No computation without implementation? A potential problem for the single hierarchy view of physical computation

Jesse Kuokkanen[1]

Received: 1 October 2021 / Accepted: 12 April 2022 / Published online: 30 August 2022

Abstract

The so-called integration problem concerning mechanistic and computational explanation asks how they are related to each other. One approach is that a computational explanation is a species of mechanistic explanation. According to this view, computational or mathematical descriptions are mechanism sketches or macroscopic descriptions that include computationally relevant and exclude computationally irrelevant physical properties. Some suggest that this results in a so-called single hierarchy view of physical computation, where computational or mathematical properties sit together in the same mechanistic hierarchy with the implementational properties. This view can be contrasted with a separate hierarchy view, according to which computational and physical descriptions have their own hierarchies which are related to each other via a bridging implementation relation. The single hierarchy view has been criticized for downplaying the explanatory value of computational explanations and not being hospitable to multiple realization of cognitive processes. In this paper, I argue that (1) the aforementioned criticisms fail, and (2) there might be a deeper problem with the single hierarchy view, which is that the single hierarchy view might collapse into a separate hierarchy view. The kind of abstraction used by the single hierarchy view does not seem to grant mathematical or computational descriptions but only more stripped physical or implementational descriptions.

✉ Jesse Kuokkanen
jesse.kuokkanen@helsinki.fi

[1] Cognitive Science, University of Helsinki, Helsinki, Finland

1 Introduction

In cognitive sciences, researchers use both computational and mechanistic explanations (e.g., Chirimuuta 2014; Craver, 2007, 2015; Magnani & Bertolotti, 2017). Given that, traditionally, computational explanations are taken as abstract and mathematical, while mechanistic explanations are taken as physical and causal (Rusanen & Lappi, 2016), one might ask how are they related to each other. Some, including Gualtiero Piccinini, have suggested that computational explanation is, in fact, a species of mechanistic explanation and hence, there is no actual tension between computational and mechanistic explanation (e.g., Kuorikoski 2020; Piccinini, 2015, 2020; Piccinini & Craver, 2011). Mechanistic explanation is decompositional, resulting in a hierarchy of mechanistic levels (Craver, 2015). If computational explanations are mechanistic, it would seem to result in two mechanistic hierarchies: one implementational or medium-dependent and one computational or medium-independent.

Elber-Dorozko & Shagrir (2019a, 2019b) have raised a worry concerning how the two resulting mechanistic hierarchies might relate to each other. They claim that some (e.g., Rusanen & Lappi 2016; Shagrir, 2016) take Piccinini's mechanistic account of physical computation (MAC) to entail a view which Elber-Dorozko & Shagrir call *a single hierarchy view* of implementational and computational properties.[1] In the single hierarchy view, implementational and computational properties are lumped together in one mechanistic hierarchy. Elber-Dorozko & Shagrir contrast the single hierarchy view with *a separate hierarchy view*, where implementational and computational hierarchies are kept apart. According to Elber-Dorozko & Shagrir, the single hierarchy view has two main problems. First, it downplays the explanatory value of computational explanations. Second, it does not easily accommodate the multiple realizability of cognitive processes. They also argue that the separate hierarchy view avoids these problems.

In this paper, I make two suggestions. First, the arguments from explanatory value and multiple realizability are not real problems for MAC as a single hierarchy view. Second, even if the arguments from explanatory value and multiple realizability were resolved, this might be cold comfort for the single hierarchy view as there might be a deeper, more fundamental problem built into the idea of MAC as a single hierarchy view as described by Elber-Dorozko & Shagrir. The problem is that MAC seems to rely on so-called *descriptive abstraction* when it comes to the relation between implementational and computational or mathematical properties (Kuokkanen & Rusanen, 2018). Descriptive abstraction excludes implementational detail from the system's description and includes only computationally relevant properties. However, it seems that performing descriptive abstraction does not result in computational or mathematical descriptions but simply in more stripped physical descriptions.

In Sect. 2, I introduce the mechanistic account of computation. In Sect. 3, I introduce the ideas of single and separate hierarchies of physical computation. In Sect. 4, I analyze the arguments from explanatory value and multiple realizability against the

[1] Elber-Dorozko & Shagrir (2019b) also suggest that Harbecke's (2020) analysis opts for a similar structure of explanation.

single hierarchy view and present my own argument against it. Section 5 contains my concluding remarks.

2 Mechanistic account of physical computation

Computation is a well-established branch of mathematics. It studies, for example, which mathematical functions certain algorithms can solve, and how much computing resources such computations require. This work on *formal computation* leans on abstract mathematical entities such as Turing machines, and it provides the most rigorous definition we have for computation. We also speak of *physical computation*, such as neural or molecular computation or the operation of our desktop computers. The relation between formal and physical computation is tricky: formal computation does not tell us, for example, what makes something a physical computing system. This is the so-called *implementation problem* of physical computation: the problem of capturing the conditions under which a physical system implements or realizes a mathematical or abstract computation. Conditions that are too weak lead to pancomputationalism, while conditions that are too strict lead to ruling out some intuitively paradigmatic cases of physical computing systems. Traditionally, both are taken as undesirable outcomes.

Mechanistic accounts of computation have become popular in explaining physical computation and trying to provide a satisfactory answer to the implementation problem. One such account is Gualtiero Piccinini's (2015) mechanistic account of physical computation (MAC), which is the focus of this paper.[2] In Piccinini's theory, a physical computing system is a functional mechanism. It has a teleological function of manipulating medium-independent vehicles according to a rule. A standard PC is an example of a physical computing system: it is a mechanism *supposed to operate in a certain way*, which makes it a functional mechanism. Furthermore, it operates by *manipulating bits* according to certain rules. A definition of a bit does not depend on its physical fingerprint, which makes it a medium-independent vehicle. Next, leaning on Piccinini (2015), I further unpack the mechanistic account and its main components.

There are many approaches to what counts as a mechanism. According to a popular view I follow in this paper, a mechanism "is a structure performing a function in virtue of its component parts, component operations, and their organization" (Bechtel & Abrahamsen, 2005, 423). In other words, a mechanism can be explained and analyzed by breaking it down or decomposing it into its constituent components or parts, their organization, and activities. These parts can be further decomposed into their constituent components, and so on.

According to MAC, computations are mechanistic in this sense as well: computations can be decomposed into their sub-computations, which can be decomposed into their sub-computations, and so on. In physical computing systems, the operation of memory registers, for example, can be analyzed and decomposed into the operation

[2] Other mechanistic accounts of physical computation are also available (Dewhurst, 2018; Fresco & Miłkowski, 2019; Miłkowski, 2016), but they are not discussed in this paper.

of their constituent computing components, logic gates. In standard digital computers, logic gates are *primitive* computing components. They have constituent parts, but the parts are not themselves computing mechanisms. Computational mechanistic explanation, then, bottoms out at the level of primitive computing components. One can, however, continue the mechanistic analysis by decomposing the *implementation* of primitive computing components. Primitive computing components are combined with each other in a certain way to build complex computing components, such as memory registers. Complex computing components can be combined with each other to build even more complex components, and so on.[3]

One might point out that surely logic gates of standard digital computers can be further mechanistically analyzed as they can be decomposed into individual transistors, which are the constituent parts of logic gates. However, according to MAC, the analysis is not computational in the sense that individual transistors do not count as computing mechanisms. For this reason, primitive computing components can be mechanistically analyzed, but the analysis is not computational. It is important to note as well that the term *logic gate* or *memory register* does not itself refer to any specific physical realization. In standard electronic digital computers, logic gates are realized by transistor circuits. However, a logic gate does not have to be realized by transistors: logic gates can be built using domino block tiles, for example. A logic gate is an abstract, medium-independent computational term that does not depend on any specific physical medium. What matters is how it operates. Moreover, in Piccinini's account, a logic gate is the lowest one can go in *computational* decomposition. The mechanistic decomposition of the logic gate's physical realization, however, can continue further. I will have more to say on this later.

According to MAC, physical computing systems are *functional* mechanisms. Functional mechanisms have teleological functions: they are mechanisms *for* something. According to Piccinini, most mechanisms do not serve any goals or fulfill any functions: "They do what they do, and there is nothing more to it. For example, the mechanisms of chemical bonding, galaxy formation, weather, or plate tectonics do what they do without fulfilling any function in any teleological sense. Their activities are explained by the organized activities of their components, without attributing any teleological functions to the components" (2015, 100).

Certain artefacts and biological mechanisms, on the other hand, seem different. A heart is for pumping blood, a coffee maker is for making coffee, and a computing artefact is for performing computations. Each does many other things as well, but what they are *for* sets their teleological functions. In Piccinini's theory, teleological functions are stable contributions towards the goals of organisms within a population. While survival and inclusive fitness are paradigmatic objective goals, systems can also have subjective goals, attributing to which may be a function as well.[4] The fact that physical computing systems are functional mechanisms sets them apart from

[3] There are also components in physical computing systems that do not perform any computations but are nevertheless crucial components of the system. A fan might be thought of as such a component: it does not perform any computation, but without it the computing system can overheat.

[4] The plausibility of teleological functions in general is beyond the scope of this paper (see, e.g., Dewhurst 2018).

mechanisms that do not have teleological functions. Furthermore, physical computing systems are a special case of functional mechanisms in that their teleological function is to compute, not to make coffee or to pump blood.

In Piccinini's account, a physical computing system has a teleological function of computing a mathematical function *f* from inputs I (and possibly internal states S) to outputs O. The mathematical function *f* is an abstract or macroscopic description of the behavior of a physical computing system when it fulfills its teleological function. The mathematical function *f* is also the rule followed by the physical system in manipulating its vehicles. Processing or manipulation of a vehicle is any transformation of one portion of a vehicle into another. The term 'vehicle' can mean either a variable, that is, a state that can take different values and change over time, or a specific value of such a variable. A rule is a mapping from inputs, and possibly internal states, to outputs. The rule need not be represented within the system.

Physical systems, physical computing systems included, can be described at different levels of grain or abstraction. MAC takes the abstract nature of physical computational descriptions to mean abstract descriptions. Abstract descriptions omit or abstract away detail from the description of the system. Piccinini uses a Dell Latitude laptop as an example. We can choose to describe it as a 'Dell Latitude,' which omits or abstracts away many details about the system. We could, if we wanted to, also describe the laptop's components, electrical circuits, or even atoms. The same goes for, say, a pineapple. Instead of describing its molecular structure, we can use a much more economical description information-wise by simply calling it 'a pineapple.'

Computational and mathematical descriptions are similarly abstract: "Mathematical descriptions of concrete physical systems are abstract in this sense. They express certain properties … while ignoring others … [C]omputational descriptions of concrete physical systems are mathematical and thus abstract in the same sense" (Piccinini, 2015, 9). This idea has among other things been called *epistemic abstraction* (Kuokkanen & Rusanen, 2018). However, as this carries unnecessary connotations, I call it *descriptive abstraction*.[5]

According to Piccinini, when defining computations of physical systems, and the vehicles they manipulate, we only need to consider the physical properties relevant for the computation according to the rules defining the computation in question. That is, we do not need to consider all physical properties of the vehicles: we can perform descriptive abstraction. Computations and their vehicles can be described and defined independently of the physical media implementing them, which is why Piccinini calls them *medium-independent*. In a computer, the teleological function of a *logic gate*, which is a computing mechanism, is to manipulate *bits*, which are medium-independent vehicles. Neither 'logic gate' nor 'a bit' refers to the physical medium implementing them.

Summing up, a physical computing system is a functional mechanism with a teleological function of manipulating medium-independent vehicles according to a rule. Computations and their vehicles are medium-independent in the sense that they are defined according to rules which are specified on an abstract level that does not refer to physical properties of their implementing medium.

[5] This is not Piccinini's terminology.

3 MAC as a single hierarchy view of physical computation

Mechanistic explanation is hierarchical: the explanandum is decomposed into its constituent parts, which can be further decomposed into their parts, and so on. A transistor circuit is a mechanism that can be analyzed by decomposing it into its constituent transistors, wires, and the organization and activities of these parts. Those parts can be further analyzed by decomposing them into their parts and their organization and activities. This analysis in question would be implementational, not computational, as it deals with the medium-dependent mechanism.

According to MAC, computational descriptions are mechanistic and hierarchical but different from those just mentioned in the sense that computational descriptions are medium-independent. Medium-independent descriptions do not refer to any specific physical properties of the explanandum. We might describe the transistor circuit above as a 'logic gate,' which is a computational-level description: it does not describe the transistor circuit as transistors and wires. Instead, it focuses on how it manipulates inputs and outputs, which can be described, for example, with numerals such as 1 and 0.

When it comes to physical computing systems, then, it seems that we have two mechanistic hierarchies: implementational, which is medium-dependent, and computational, which is medium-independent. How are the two hierarchies related to each other?

Elber-Dorozko & Shagrir (2019a, 2019b) discuss two options: the separate hierarchy view and the single hierarchy view. In the separate hierarchy view, "there are two separate hierarchies, one computational and another implementational, which are related by the implementation relation" (Elber-Dorozko & Shagrir, 2019b, 43). In other words, the implementational hierarchy is kept apart from the computational hierarchy. Furthermore, one does not need to take a stance regarding, for example, whether the computational hierarchy is mechanistic or not. In the single hierarchy view, there is only one mechanistic hierarchy. Computational or mathematical properties sit together with the implementational or physical properties in the same mechanistic hierarchy. In other words, "the mechanistic hierarchy embeds at the same levels computational and implementational properties" (ibid.).

According to Elber-Dorozko & Shagrir, the idea of computational explanation as a mechanism sketch (Piccinini & Craver, 2011) is often associated with the single hierarchy view. The idea that computational explanations are mechanism sketches is endorsed by MAC. Furthermore, Piccinini's ideas concerning implementational and computational descriptions and abstraction, and the fact that he takes computational descriptions to be mechanistic, suggests that MAC does indeed lean on the single hierarchy view (Kuokkanen, under review).

In MAC, computational or mathematical descriptions of concrete systems abstract away or ignore medium-dependent properties of the system and express only the medium-independent properties: "[t]hey express certain properties … in an economical way while ignoring others" (Piccinini, 2015, 9). In other words, instead of mapping properties of some separate medium-independent or mathematical hierarchy to the medium-dependent hierarchy, one simply abstracts away the medium-dependent properties when providing a mathematical or computational description. According

to a recent suggestion for illustrating this approach (Kuokkanen, under review), each mechanistic level is wide: one end of the level has all the relevant medium-dependent properties in place. Moving horizontally towards the other end of the level, one starts to exclude the medium-dependent properties, gradually arriving at a computational or mathematical, medium-independent description. This kind of *horizontal abstraction* can be performed at every mechanistic level of the physical computing system.[6]

The idea of computational explanations being mechanistic is not without its critics. First, some argue that mechanistic explanations describe causal relationships, which is something that many prohibit from computational or mathematical explanations (Rusanen & Lappi, 2016). Second, Elber-Dorozko & Shagrir (2019a) argue that while some computational explanations may be decomposable, others may not be. Third, they continue that even in cases where computational explanations are decomposable, the resulting levels are not mechanistic but functional: the fact that some computational descriptions are decomposable or hierarchical does not entail that they are mechanistic.

The aforementioned criticism focuses on the claim that computational explanations are mechanistic. Another strategy is to consider what follows if computational explanation is mechanistic: even if the medium-dependent and medium-independent hierarchies did manage to integrate and form a single mechanistic hierarchy, Elber-Dorozko & Shagrir (2019b) argue that the single hierarchy view has two additional problems. First, it downplays the explanatory role of computational descriptions: once we know all the implementational properties of a specific logic gate, its computational description becomes redundant in the single hierarchy view. If there is nothing in the computational description that is not already in the implementational description, why should one bother using the computational description when it does not have any additional value? This is in tension with scientific practice, where computational descriptions have real explanatory value even when the implementational details of the system are already known.

Second, the single hierarchy has trouble in accommodating the multiple realizability of cognitive processes. Traditionally, cognitive science takes at least some of our cognitive processes to be multiply realizable. The general idea behind multiple realizability is that the same cognitive process can be realized in various systems. One motivation behind the idea is that in the cognitive sciences, it is common to build computational or mathematical models of cognitive processes that can then be implemented in different computational systems. What defines or matters the most for the cognitive process in question is the description at the mathematical or information processing level, not the fine implementational details of the realizing system. The idea behind Elber-Dorozko & Shagrir's argument is that if the computational description is not an independent description separate from the medium-dependent description, it is not clear how it is the *same* computational description that is implemented across different systems.

[6] In MAC, a physical computing system has a 'computational floor level': the floor level consists of primitive computing components. In the case of digital computers, primitive computing components are logic gates. These can be further analyzed and decomposed mechanistically, but the analysis is not computational as the parts are not themselves computational. In other words, if one wants to analyze the level of primitive computing components mechanistically, one first needs a 'horizontal de-abstraction.'

Elber-Dorozko & Shagrir argue that the shortcomings of the single hierarchy view can be avoided by separating the computational hierarchy from the implementational. This results in two hierarchies: one medium-independent or computational and one medium-dependent or implementational. An additional virtue is that in the separate hierarchy view, one does not have to make any commitments regarding whether computational explanations are mechanistic or not. In the separate hierarchy view, the two hierarchies are kept apart and related via an implementation relation, and the computational properties are mapped to, or implemented by, the implementational properties of the physical structure. As the computational and physical hierarchies are separate, it is more intuitive to take computational explanations as independent, full explanations and not merely partial implementational descriptions. In other words, the separate hierarchy view does not downplay the explanatory role of computational explanations, and computational descriptions do not become redundant once we know the implementational details of some system.

A further merit of the view, according to Elber-Dorozko & Shagrir, is that it better accommodates the multiple realizability of cognitive functions. Multiple realization about cognitive processes means that the same cognitive process can be realized across different systems. Traditionally, cognitive sciences aim at providing computational or mathematical explanations and models of cognitive phenomena. If mathematical or computational descriptions and explanations are not merely partial implementational descriptions but are independent and separate from the physical hierarchy, the idea that the *same* cognitive process is realized in different systems seems more intuitive: we have an independent mathematical model or explanation which is related to various physical structures via an implementation relation.

In the next section, I argue that neither the explanatory value of computational explanations nor the multiple realizability of cognitive processes are real problems for the single hierarchy view. However, and more importantly, I also argue that there is a more fundamental problem with the single hierarchy view: it seems to collapse into a separate hierarchy view.

4 The non-problems and *the* problem of MAC as a single hierarchy view

In the previous section, I introduced different criticisms against the mechanistic account of computation and the single hierarchy view. It is a good idea to keep these criticisms apart: even if one does not subscribe to the criticisms arguing that computational explanations are not mechanistic, one can still worry about the MAC as a single hierarchy view of physical computation. In this section, I argue that neither of the two arguments targeted against the idea of MAC as a single hierarchy view, the explanatory value of computational explanations and the multiple realizability of cognitive processes, are real problems for the single hierarchy view. However, I also argue that there is a more fundamental problem with the idea of the single hierarchy view, which is that the whole idea of a single hierarchy view might collapse into a separate hierarchy view.

Elber-Dorozko & Shagrir (2019a, 2019b) argue that the single hierarchy view downplays the explanatory value of computational explanations. In the single hierarchy view, computational descriptions are partial implementation descriptions: implementational medium-dependent properties are abstracted away so that one is left only with medium-independent or computational properties. Consider a logic gate: first, we have the relevant medium-dependent detail of the transistor circuit implementing or realizing the logic gate. To arrive at the computational or medium-independent description, one then performs descriptive abstraction, omitting properties such as color, height, and mass, which are irrelevant for its operation as a logic gate.

Once we are done with the descriptive abstraction, we are left only with the properties relevant for the computational or mathematical description: depending on the electrical current, the state of the logic gate is either on, in which case the state is 1, or the logic gate is off, in which case the state is 0. During the process, one stays within one level of mechanism or organization. Hence, the abstraction in question can be thought of as *horizontal* descriptive abstraction. Horizontal descriptive abstraction abstracts away the detail within one level of mechanism, while vertical descriptive abstraction abstracts away levels of mechanism.

Elber-Dorozko & Shagrir's argument emphasizes the fact that some who argue for the mechanistic nature of computational explanations take computational explanations to be *incomplete* mechanistic descriptions. In other words, forming a computational description is just one step on the quest toward a full mechanistic or physical description, filling in all the physical details on the horizontal axis of physical properties. According to this line of thought, we use computational descriptions because they are the best we have: once we have filled in all the details, we can dispense with the computational descriptions since they are partial and incomplete. That is, computational descriptions become redundant.

A problem with this argument is simply that the single hierarchy view does not entail that we use computational explanations due to a lack of knowledge concerning the implementational properties. On the contrary, there are many reasons why researchers use computational descriptions and explanations, and an advocate of the single hierarchy view can happily accept this. While ignorance of physical detail might be one reason for abstraction and using computational explanations in some cases, it is certainly not the only reason. Other reasons for abstraction include making the phenomenon tractable and isolating it from its surrounding structures or trying to capture a general phenomenon. As Boone & Piccinini (2016) note, there are many epistemic roles for abstraction. Depending on the situation, abstracting away physical properties can have different reasons and may result in different benefits. This means that computational explanation does not need to become redundant even if one adopts the single hierarchy view. The single hierarchy view does not entail explanatory physical chauvinism.[7]

Another alleged problem of the single hierarchy view is that it is less hospitable to the multiple realizability of cognitive processes than the separate hierarchy

[7] The question regarding the single hierarchy view and the explanatory value of computational descriptions deserves more attention and argumentation than is given here. However, a proper discussion is beyond the scope of this paper.

view.[8] In cognitive sciences, it is a popular idea that some cognitive functions can be performed by different implementational structures. There are many variants of the multiple realizability thesis. According to weaker formulations, some cognitive functions are defined on a basis that allows the same cognitive process to be realized in different human organisms. According to stronger formulations, some cognitive processes are realizable in various media. Memory, for example, can be realized not only in the human brain but also in computer chips. Here, cognitive functions are defined at the mathematical level of information processing, which does not directly refer to physical or chemical properties. For this reason, these cognitive functions are called medium-independent. What is important is how something is processed, not what processes it.

Apparently, the idea behind Elber-Dorozko & Shagrir's argument is that if the computational description is part of the same hierarchy that describes the implementational properties of the system, it is difficult to see how it is *the same* mathematical model which is realized in other systems. In other words, the mathematical description carries all the implementational detail with it, and taking a mathematical description of system x is not compatible with system y unless they are also identical in implementational detail. With the mathematical description, one also gets the full package of physical properties which the mathematical description is abstracted from. This can be avoided by keeping the mathematical description separate from the implementational mechanistic hierarchy. In this case, it is more intuitive to say that it is the *same* computational description which is implemented across different physical media.

One reply to this argument is simply to note that it is not clear how the separate hierarchy view accommodates the multiple realizability of cognitive functions better than the single hierarchy view. It seems that there is a hidden premise built into Elber-Dorozko & Shagrir's argument, which is that adopting the single hierarchy view entails that a computational description is intimately tied to the physical details it is distilled from. In other words, it is assumed that when one forms a computational description using descriptive abstraction, the resulting computational description still carries all the omitted physical details with it. This, then, entails that the computational-grain descriptions of two systems that are computationally similar but physically different, are different because their physical implementational details are incompatible with each other. However, I do not see this as a fatal problem for the single hierarchy view: simply put, the *computational-grain* description might well be the same for systems with different implementational structures.[9]

However, MAC as a single hierarchy view has a more fundamental problem. MAC seems to lean on so-called *descriptive abstraction* (Kuokkanen & Rusanen, 2018;

[8] Currently, there is much discussion on whether the mechanistic view of computational explanations is compatible with multiple realization, but unfortunately this discussion is beyond the scope of this paper. Instead, I will take a slightly different angle regarding the reasons why the multiple realization issue is not relevant for the debate in question.

[9] As with the previous argument, this question would deserve a proper and more detailed analysis. The question relates to, for instance, debates concerning how physical computations and cognitive functions are defined, which are both beyond the scope of this paper. Narrow and wide views are available for both questions.

Kuokkanen, under review), which means the omission of detail from the description of a system.[10] Piccinini (2015) uses a pineapple and a Dell Latitude laptop as examples. We can describe a pineapple as 'a pineapple' but this omits a great deal of information about that particular pineapple. If we wish, we can also choose to describe the same pineapple at a molecular level, for example. Calling it a pineapple is more economical information-wise. It is also more *abstract* in the sense that it omits more details from the description. Likewise, instead of talking about a 'Dell Latitude' laptop, we can choose to describe much of its properties like color, size, component parts or electrical circuits, and so on. In case we choose to talk about a 'Dell Latitude,' we *abstract away* numerous physical details.

According to Piccinini, "[m]athematical descriptions of concrete physical systems are abstract in this sense. They express certain properties … while ignoring others" (2015, 9). Furthermore, "computational descriptions of concrete physical systems are mathematical and thus abstract in the same sense. They express certain properties … in an economical way while ignoring others" (ibid.). In other words, when describing physical computing systems, we can choose to describe them either in medium-dependent or medium-independent terms, depending on the properties we include in and omit from their descriptions.

Consider a logic gate which belongs to the class of primitive computing components of modern digital computers. A 'logic gate' is a medium-independent, *computational-level* description. It is abstract in the sense that it does not refer to medium-dependent, implementational properties: depending on the situation, logic gates can be implemented by electronic transistors, cogwheels, domino block tiles, and so on. When describing the implementational structure in medium-dependent detail – the operation of transistors, for instance – the resulting description is not computational and medium-independent but implementational and medium-dependent: we describe the properties which depend on the realizing medium, such as voltage.

MAC is mechanistic: physical computing systems are functional mechanisms. For this reason, MAC is also hierarchical in the same sense as a mechanistic explanation is. Physical computing systems can be analyzed by decomposing them into their constituent parts, and physical computations can be analyzed by decomposing them into their sub-computations, resulting in a hierarchy of mechanistic levels. Mechanistic levels are hierarchical in the sense that they are in part-whole relations with each other.

A mechanistic hierarchy can be thought of as a *vertical* hierarchy of levels: a whole is at a higher mechanistic level than its constituting components. The relation between a computing component and its implementational structure is different as they do not stand in a part-whole relation: a transistor circuit is not a *part* of a logic gate; it *realizes* or *implements* it. As the computational description is abstract in the sense that it abstracts away the implementational details, it has been suggested that in MAC such descriptive abstraction happens in a *horizontal* direction at each mechanistic level: in horizontal abstraction, one stays within one level of mechanism and abstracts away *details within that specific level* (Kuokkanen, under review). Vertical

[10] The term is not used by Piccinini himself.

descriptive abstraction, in turn, omits or abstracts away higher or lower levels of mechanisms. In scientific practice, descriptive abstraction probably always involves both vertical and horizontal abstraction (Mäki, 1992).

MAC does not seem to have the tools to arrive at a *single* hierarchy in the first place. Insofar as the single hierarchy uses descriptive abstraction to arrive at computational descriptions, certain problems follow. The reason is that descriptive abstraction does not provide mathematical descriptions but simply more stripped physical descriptions. 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 mathematical properties. In other words, the so-called single hierarchy view seems to collapse into a separate hierarchy view.

Consider logic gates, which are primitive computing components in standard digital computers. Logic gates cannot do much on their own, but combining them in an appropriate way is basically how complex computers are made. For example, logic gates are combined in a certain way to form ALUs, which are combined to form memory registers, and so on. Basically, when it comes to the computing components of standard digital computers, it is logic gates all the way down. In standard digital computers, logic gates consist of electronic transistors. A transistor is largely made of silicon, and it can act either as a switch or as an amplifier. When it acts as an amplifier, it takes a small electronic current at its input end and produces a much bigger current at its output end. A transistor can also work as an electronic switch. This is essentially how all computer chips and logic gates work.

A logic gate is not defined through its physical properties. In standard digital computers, logic gates are made out of transistors, but one could build a logic gate using, for example, vacuum tubes or domino block tiles. What matters is that the physical stuff has an appropriate structure and can maintain stable enough states. When all the necessary criteria are met, physical stuff can be said to realize or implement a logic gate. Logic gates are built out of components that hold two stable, different states. In computational language, a logic gate works by processing 1s and 0s. In physical language, a logic gate is built out of transistors, and the 1s and 0s are determined through electrical current. If there is enough current, the transistor is 'on.' If there is not, the transistor is 'off.' When the transistor is 'on,' its state is labeled as '1.' When the transistor is 'off,' its state is '0.'

The crucial thing is that one does not get the '0' and '1' just by abstracting away the physical properties of the transistor. When we ignore the transistor's size, color, weight, and other irrelevant physical properties, we do not get mathematical descriptions. All we get is more stripped physical descriptions. To arrive at the computational description, we must 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 structures. No matter how much descriptive abstraction one applies to the physical description, it alone does not provide this mathematical mapping or implementation relation. What the descriptive abstraction *does* grant is the identification of relevant physical structures onto which the mathematical properties can be mapped.

For this reason, it seems that the descriptive abstraction does not, by itself, succeed in establishing a single hierarchy view of physical computation in the way that Elber-Dorozko & Shagrir suggest. The horizontal descriptive abstraction is crucial in capturing the relevant structures that are to be mapped onto mathematical descriptions. However, one must still have the implementation relation that relates the mathematical properties to the physical properties.

Elber-Dorozko & Shagrir (Elber-Dorozko & Shagrir, 2019a, 217) seem to hint at a similar kind of conclusion by noting that "[o]ne might argue that the medium-independent/medium-dependent distinction suffices to support the thesis that computational explanations are distinct … from the implementational level." However, this argument does not, by itself, reach its goal as one might reply by saying that medium-independence is something which is arrived at gradually by performing descriptive abstraction. In this section, I have suggested that there is a potential problem with this line of reply.

5 Conclusions

In this paper, I have argued that (1) the arguments presented against the single hierarchy view by Elber-Dorozko & Shagrir (2019a, 2019b) are not trivial and require more detailed discussion, and (2) descriptive abstraction does not, by itself, seem to result in computational or mathematical descriptions. For this reason, it seems that Piccinini's account of physical computation collapses into a separate hierarchy view if it is the presence of implementation relation between implementational and mathematical properties that makes something a single hierarchy view.

Elber-Dorozko & Shagrir have argued that some variants of the mechanistic account of computation are single hierarchy views in the sense that they take computational descriptions to be mechanism sketches formed through descriptive abstraction or the omission of physical details from their descriptions. Here, computational properties are placed at the same level of mechanism with their implementational properties, resulting in a single but wide hierarchy. Some take Gualtiero Piccinini's mechanistic account of physical computation as an example of such a single hierarchy view. In Piccinini's account, computational descriptions are descriptive abstractions or macroscopic descriptions.

Descriptive abstraction omits detail from the description of a system. This entails that descriptive abstraction results in descriptions with fewer physical details. However, arriving at a computational or mathematical description simply by omitting implementational properties seems problematic: it only results in more stripped implementational descriptions but not computational or mathematical descriptions. Descriptive abstraction helps in identifying the relevant physical structure onto which the computational or mathematical properties are supposed to be mapped. However, the implementation relation still needs to be in place. The implementation relation is a contract or decision of sorts, determining when the mathematical entities map onto physical entities.

I will briefly mention a few potential objections to my ideas. First, one might say that descriptive abstraction is not a *method* for arriving at a computational or math-

ematical description but merely a way of *exposing* the mathematical properties of the physical system that are already there. However, this does not seem to resolve the problem as one still needs to explain where the mathematical properties come from, and how they relate to the physical properties. Second, one might insist that descriptive abstraction *does* result in mathematical descriptions should one adopt some kind of a structuralist or nominalist mathematical ontology.[11] I do not have a definite stance towards this suggestion. It is, however, clearly beyond the scope of this paper and is something that should be studied in more detail in the future.

A third potential objection is that a mechanistic hierarchy does not contain only physical detail but it also has the explanandum which determines the relevant details for the mechanism. In MAC, a physical computing system is a mechanism that has a teleological function determining what it computes. In the explanations discussed in this paper, mechanisms always have functions of performing specific computations. Thus, in each case, there is a rule that also determines the relevant computational properties.[12] However, this point does not seem to affect the argument presented in this paper: even if it were the case that a function determines the computational properties of the system (a question which is outside the scope of this paper), it does not entail that it also determines the physical properties implementing the computational properties in question. A mechanistic explanandum described in physical terms might determine the physical properties, and a computational function described in mathematical terms might determine computational properties. The issue discussed in this paper deals with bridging these two seemingly different kinds of properties.

If my arguments are correct, they have certain implications. First, they resolve the juxtaposition between the so-called single and separate hierarchy views. Second, they draw attention to the role which implementation plays in the theories of physical computation. Furthermore, the role and implications of different views regarding mathematical ontology should be further studied to shed more light on the issue of physical computation and implementation.

Acknowledgements Thanks to the anonymous reviewers for their extremely valuable comments and feedback. Also, many thanks to Anna-Mari Rusanen and Otto Lappi for all the discussions, feedback, comments, ideas, and general support along the writing process.

Funding Open Access funding provided by University of Helsinki including Helsinki University Central Hospital.

[11] See, e.g., Shapiro (2005) for an overview on different positions in the philosophy of mathematics.

[12] My thanks to the anonymous referee for making this point.

References

Bechtel, W., & Abrahamsen, A. (2005). Explanation: A mechanist alternative. *Studies in History and Philosophy of Science Part C: Studies*. *History and Philosophy of Biological and Biomedical Sciences*, 36(2), 421–441. https://doi.org/10.1016/j.shpsc.2005.03.010

Boone, W., & Piccinini, G. (2016). Mechanistic abstraction. *Philosophy of Science*, 83(5), 686–697. https://doi.org/10.1086/687855

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

Craver, C. F. (2007). *Explaining the Brain*. Oxford University Press

Craver, C. F. (2015). Levels. In T. Metzinger & J. M. Windt (Eds.), *Open MIND* (Vol. 8). MIND Group. https://doi.org/10.15502/9783958570498

Dewhurst, J. (2018). Computing Mechanisms without Proper Functions. *Minds and Machines*, 28(3), 569–588. https://doi.org/10.1007/s11023-018-9474-5

Elber-Dorozko, L., & Shagrir, O. (2019a). Computation and levels in the cognitive and neural sciences. In M. Sprevak & M. Colombo (Eds.), *The Routledge Handbook of the Computational Mind* (pp. 205–222). Routledge. https://doi.org/10.4324/9781315643670-16

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

Fresco, N., & Miłkowski, M. (2019). Mechanistic Computational Individuation without Biting the Bullet. *British Journal for the Philosophy of Science*, *axz005*

Harbecke, J. (2020). The methodological role of mechanistic-computational models in cognitive science. *Synthese*. https://doi.org/10.1007/s11229-020-02568-5

Kuokkanen, J. (manuscript under review). Vertical-horizontal distinction in resolving the abstraction, hierarchy, and generality problems in mechanistic account of computation

Kuokkanen, J., & Rusanen, A. M. (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

Kuorikoski, J. (2020). There Are No Mathematical Explanations. *Philosophy of Science*. https://doi.org/10.1086/711479

Magnani, L., & Bertolotti, T. (Eds.). (2017). *Springer Handbook of Model-Based Science*. Springer, Cham. https://doi.org/0.1007/978-3-319-30526-4

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

Miłkowski, M. (2016). A Mechanistic Account of Computational Explanation in Cognitive Science and Computational Neuroscience. *Computing and Philosophy*, 191–205. https://doi.org/10.1007/978-3-319-23291-1_13

Piccinini, G. (2015). *Physical Computation: A Mechanistic Account*. Oxford University Press

Piccinini, G. (2020). Neurocognitive Mechanisms: Explaining Biological Cognition. *Journal of Chemical Information and Modeling* (Vol. 53, Issue 9). Oxford University Press

Piccinini, G., & Craver, C. (2011). Integrating psychology and neuroscience: Functional analyses as mechanism sketches. *Synthese*, 183(3), 283–311. https://doi.org/10.1007/s11229-011-9898-4

Rusanen, A. M., & Lappi, O. (2016). On computational explanations. *Synthese*, 193(12), 3931–3949. https://doi.org/10.1007/s11229-016-1101-5

Shagrir, O. (2016). Advertisement for the philosophy of the computational sciences. In P. Humphreys (Ed.), *Oxford Handbook of Philosophy of Science* (pp. 15–42). Oxford University Press

Shapiro, L. A. (Ed.). (2005). *The Oxford Handbook of Philosophy of Mathematics and Logic*. Oxford University Press