There Is No Such Thing As Miscomputation

Joe Dewhurst

March 2020

Abstract

This paper will argue that there is no such thing as miscomputation, contrary to the received view in philosophy of computation and (possibly) computer science. There are just hardware problems on the one hand and programming errors on the other, neither of which constitute a distinct kind of *computational* malfunction. One upshot of this argument is that philosophical accounts of physical computation should not be assessed on whether they can accommodate miscomputation in the abstract, but rather on whether they can make sense of the range of different phenomenona that are commonly (and misleadingly) *described* as miscomputations.

Introduction

There has been a recent renaissance in philosophical work on physical computation (see e.g. Piccinini 2007, 2015; Milkowski 2013; Fresco 2014), but rather less has been written on the topic of *mis*computation, i.e. the circumstance under which some physical state or process counts as performing a computation, but performing it *wrong*. This is in spite of the fact that one of the leading accounts of physical computation, Gualtiero Piccinini's mechanistic account, lists miscomputation as one of six key desiderata that any account of physical computation must accommodate (2015: 13-14). Aside from Piccinini's own work, there have been articles published on (or relating to) miscomputation by Fresco & Primiero (2013), Dewhurst (2014), Floridi, Fresco, & Primiero (2015), Petricek (2017), Tucker (2018), Primiero, Solheim, & Spring (2019), and Colombo (forthcoming). In general each of these authors assume that there is a distinctive category of computational errors, but here I will argue that this is not the case, and that there is really no such thing as miscomputation *as such*. Section 1 will review existing accounts of physically computation and miscomputation, focusing primarily on mechanistic accounts. Section 2 will present the argument(s) that there is no such thing as miscomputation, but rather just various kinds of hardware malfunctions and design errors. Finally, section 3 will discuss the ramifications of this argument for existing accounts of physical computation,

and consider some future work that could be done on the topic formerly known as miscomputation.

1 Physical computation and miscomputation

I am concerned here with *physical computation*, i.e. the question of what it means to say that a physical system performs or implements an abstract computation, and hence what it would mean to say that the same system somehow *fails* in this regard, or miscomputes. Before we can discuss the latter I must first say a little about the former, although a full treatment is beyond the scope of this paper. At a first pass we can say that a physical system implements an abstract computation if there is a mapping between the physical structure of the system and the formal structure of the computation (Putnam 1960, cf. Godfrey-Smith 2009). This is a notoriously weak definition, according to which almost any physical system might implement almost any computation (Putnam 1988; see Godfrey-Smith 2009 and Sprevak 2018 for discussions of this issue), and so it is typically strengthened with additional constraints on the kinds of physical system that qualify as computational. Examples include causal constraints (e.g. Chalmers 1994), semantic constraints (e.g. Sprevak 2010), and pragmatic or perspectival constraints (e.g. Schweizer 2019), but I will focus here on a subclass of causal constraints offered by the various mechanistic accounts of physical computation, and specifically Gualtiero Piccinini's version of this account (2007, 2015; see also Milkowski 2013, Fresco 2014). My argument against the very possibility of miscomputation should mostly generalise to other accounts of physical computation, but I will note where it does not, and consider some of these other possibilities in the final section.

According to Piccinini's mechanistic account, a physical computer is a kind of mechanism whose function is to perform computations, understood as systematic transformations between medium independent digits (2015: chapter 7).[1] Digits are components whose function is to be recognised and systematically transformed (into other digits) by processors, and are medium independent insofar as they are individuated only by those physical properties that are relevant to this function. Processors are components whose function is to identify and systematically transform digits according to a rule specified by the abstract computation that the system is meant to implement (i.e. the program). Further computational component-types include input and output components that transform external stimuli into digits and *vice versa*, memory components that store strings of digits, and so on. A computing mechanism will also typically include *non*-computational components such as a power source, a cooling fan, and so on. Crucially, the core components (digits and processors) must possess a sufficiently stable causal structure to qualify as computational, thus constraining the range of physical systems that will implement a computation (although

[1]Note that this is specifically an account of *digital* computation. Piccinini does also offer related accounts of analog, generic, and *sui generis* neural computation, but for the sake of simplicity I will focus just on the digital case here.

for further discussion of some concerns about this account, see e.g. Dewhurst 2018a; Coelho Mollo 2018, 2019; Fresco & Milkowski 2019).

Piccinini gives a list of six desiderata that he thinks any account of physical computation ought to be able to fulfil, including that it "should explain how it's possible for a physical system to miscompute" (2015: 14). He defines miscomputation as cases where a system "fails to follow every step of the procedure it's supposed to follow all the way until producing the correct output" (*ibid.*), and emphasises that explaining miscomputation is important because of the important role that (avoiding it) plays in computer science. His account is able to explain miscomputation due to its functional nature: if what it means for a physical system to perform a computation is to be a mechanism whose components have the function of performing that computation, then those components could also *malfunction*, resulting in a *miscomputation* (*ibid*: 122).[2] An example of this kind of miscomputation is a hardware malfunction where e.g. a logic gate that is supposed to perform AND instead performs NAND, meaning that the digits are processed incorrectly relative to the rule specified by the abstract computation that is supposed to be implemented. Piccinini also allows for other kinds of miscomputation, including those resulting from incorrect design, implementation, or usage (*ibid*: 149-50). Here he is presenting a taxonomy that he first introduced in Piccinini (2007), which was then discussed by Fresco & Primiero (2013), and further refined for the mechanistic account by Dewhurst (2014).

Floridi, Fresco, & Primiero (2015) further develop the taxonomy of miscomputation presented by Fresco & Primiero (2013) to include software malfunctions, Petricek (2017) specifically discusses programming errors, and Primiero, Solheim, & Spring (2019) provide an additional analysis of malware classification. Tucker (2018) develops Piccinini's functional approach to miscomputation by distinguishing between a system's *proper* function and its *actual* function, and Colombo (forthcoming) applies the notion of miscomputation to computational psychiatry. I will return to each of these issues as they arise naturally in the rest of the paper.

2 There is no such thing as miscomputation

In the previous section I introduced the taxonomy of miscomputations developed by Piccinini (2007, 2015), Fresco & Primiero (2013), and Dewhurst (2014). Let us now examine that taxonomy in more detail. Piccinini (2015: 149-50) lists five notions of miscomputation, each relating to a different perspective from which we might evaluate the performance of a computational system:

1. Miscomputations relative to the designer's intentions.

[2] I have elsewhere argued against the use of proper functions in mechanistic accounts of computation (Dewhurst 2016), and proposed an alternative approach based on perspectival functions (Dewhurst 2018b), but I will not be pursuing that argument here. In the next section I will argue that, even if the mechanistic account appeals to proper functions, there is still no such thing as miscomputation.

2. Miscomputations relative to the designer's blueprint.

3. Miscomputations relative to what was actually built.

4. Miscomputations due to incorrect programming.

5. Miscomputations due to incorrect usage.

Fresco & Primiero draw a further useful distinction between "errors of functioning" and "errors of design" (2013: start of section 2), inspired by a similar distinction originally made by Turing (1950: 449). Errors of functioning, or what we might call 'operational malfunctions', occur when the system *as it was actually built* fails to function correctly, i.e. a physical component malfunctions in some way that affects the computational procedure. Piccinini's type-3 miscomputation is a clear case of operational malfunction, but all of the others seem to be of the latter type, which we might call 'design errors'. If a system is designed incorrectly (type-1), such that it cannot perform the function that it was intended for, then this is not a problem with the operation of the system itself, but rather a mistake on behalf of its designer. Similarly, if the initial design is good, but a token system is manufactured incorrectly (type-2), then this is not a case of the system itself malfunctioning, but rather a mistake on behalf of its proximal designer (the agent or system that manufactured it). A programming error (type-4) is also not a case of operational malfunction, as the physical system itself performs perfectly fine, it was just given bad instructions (at least relative to the programmer's intentions – for the system itself there is no sense in which the instructions can be good or bad). Finally, errors on behalf of the user (type-5) are clearly not operational malfunctions, as the system itself (both hardware and software) performs perfectly fine. In fact, it is not even clear that incorrect usage should be considered as a kind of design error either, but for the purpose of this analysis I will stretch the definition of 'designer' to include the end user of a computational system (who indeed might also sometimes be a programmer).

While Piccinini suggests that we should consider all five of these notions to be types of miscomputation, I will now argue that in fact *none* of them should be considered to be types of miscomputation, at least in a strict sense. The first half of this argument, that design errors are not miscomputations, is relatively uncontroversial, and so I will not spend too long on it. The second half, that operational malfunctions are not miscomputations either, will require a little more attention. After presenting this argument I will proceed in section 3 to consider some additional complications and implications that arise once we accept that there is (strictly speaking) no such thing as miscomputation.

2.1 Design errors are not miscomputations

Fresco & Primiero "argue that a computational system *can only* make an error of *functioning* (i.e. an operational malfunction)" (2013: section 1, emphasis in

original), which would presumably mean that design errors are not miscomputations. However, they also have a somewhat broader conception of operational malfunctions than that which I introduced above, including certain kinds of software errors (Piccinini's type-4 miscomputations). So why shouldn't we consider design errors, including programming errors (cf. Petricek 2017) and software malfunctions, to be kinds of miscomputation?

In Dewhurst (2014) I previously offered an argument to this effect, focusing on the mechanistic account of computation introduced above. I argued that we should not assess the behaviour of a computing mechanism relative to the designer's *intentions*, but rather relative to their actual design, as implemented in the token system that we are analysing. This is because a computing mechanism is simply a system that transforms strings of digits *according to the rules that it is given*, without any understanding of either those rules or the semantic content of the digits. So a (physically) functioning computing mechanism provided with a program to run cannot fail to correctly implement that program, even if the end result is not what the programmer intended. For this reason I do not think that we should consider programming errors of any kind to be miscomputations, at least in the sense of computation captured by the mechanistic account (I will discuss some complications introduced by other accounts of computation in the final section).

Other kinds of design error, like user errors or errors in the physical manufacture of the system, are also not miscomputations in the strict sense. Mistakes made by the end user of a computing mechanism, e.g. by issuing meaningless or misunderstood (by the user) commands, are clearly not cases of the system itself miscomputing. Manufacturing errors might qualify as malfunctions of some kind, but these cannot be *computational* malfunctions (of the end product): either the token system that is the result of the faulty manufacturing process does not function as a computing mechanism at all, or else it does function as one, but not in the way that the designer intended. The latter case is similar to programming errors, insofar as the system will continue to compute *something*, and do so 'correctly' relative to its actual physical structure, even if this is different to how the designer originally envisioned it. Such an outcome might depend on malfunctions *in the manufacturing process*, but once this process is complete, the system thereby created cannot be blamed for these malfunctions. If it functions as a computing mechanism at all, it simply computes according to the rules that it was given, i.e. those that are implicit in its actual physical structure.

Software malfunctions, as discussed by Floridi, Fresco, & Primiero (2015), are also typically a kind of design error. Some apparent software malfunctions might be due to physical limitations, such as a logic gate functioning incorrectly or a memory component failing in some way, but it is debatable whether these really qualify as *software* malfunctions, rather than physical malfunctions that affect the software (*ibid*: 18 of preprint). In any case, I will deal with the question of physical ('operational') malfunctions in the next section. Other software malfunctions might be due to badly written code (missing brackets, mistyped variables, infinite loops), but as Floridi, Fresco, & Primiero themselves

admit, malfunctions of this kind "are simply design errors for which only the designer can be deemed responsible" (*ibid*). As I argued above, a computing mechanism can only do the best it can with what it is given – as computer scientists sometime say, "garbage in, garbage out".[3]

Design errors of all kinds, including mistakes in the original specification of the computing mechanism, faulty manufacturing, bad or misguided programming, and erroneous usage, do not qualify as miscomputations, as the mistake in each case is attributable to something other than the computing mechanism itself (i.e. the machine's designer, manufacturer, programmer, or end user). In section 3 I will argue that some of the (putative) miscomputations discussed in computational psychiatry (cf. Colombo forthcoming) could also be more fruitfully understood as design errors. For now I will proceed to the second class of putative miscomputations, those due to operational malfunctions in the physical structure of the computing mechanism.

2.2 Operational malfunctions are not miscomputations

Towards the end of their analysis of miscomputation, Fresco & Primiero suggest, somewhat allusively, that what they call operational malfunctions might not strictly be computational either: "the cause of the miscomputation is often the physical substrate that is *contingent* to the computational process itself" (2013: final paragraph, emphasis added). If the physical substrate is *contingent* to the computations being performed, then it doesn't seem like operational malfunctions are an essentially computational phenomenon. Turing seemed to hold a similar view, writing that (abstract) computational machines (i.e. idealised Turing machines) are "By definition [...] incapable of errors of functioning" (1950: 449). Now, admittedly Turing was discussing abstract computations at this point, and we are concerned here with concrete (physical) computations. So should we say that a physical malfunction that causes a computing mechanism to produce the wrong result is (strictly speaking) a miscomputation?

I now that think that the answer to this second question should also be "no" (previously, in Dewhurst 2014, I suggested otherwise). To see why, we need to think a little more about what (physical) computations are, and what it might mean for them to fail to operate correctly. According to the mechanistic account (introduced in section 1), a physical computation is just the transformation of some medium-independent vehicles ('digits') according to a rule. The rule itself is specified by the physical structure of the system (in the simplest case, by the structure of a single processing component such as an AND-gate), and, crucially, none of the components of the system themselves 'understand' the rule. It is by this simple conjuring trick that physical computation is able to produce

[3]The Free On-Line Dictionary of Computation attributes this phrase to Wilf Hey (http://foldoc.org/Garbage+In,+Garbage+Out).[4] Charles Babbage expressed a similar sentiment with regard to his Difference Engine: "On two occasions I have been asked, 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' ... I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question" (1864: 67).

[4]With thanks to Keith Wilson for locating the source of this attribution.

seemingly semantic transformations from merely syntactic (or causal) processes – or as Haugeland memorably put it, "If you take care of the syntax, then the semantics will take care of itself" (1985: 106). An important consequence of this, though, is that a computational component does not "follow a rule" in the normative sense of an agent choosing to obey it, but rather it just responds causally to the physical structure of the system of which it is a part.[5] With this in mind, let us return to the question of miscomputation and operational malfunction.

There is one class of operational malfunction that we can dismiss immediately, those that render the system incapable of performing any computations at all. If the system overheats and catches fire, or if the power source stops working, then the system has not miscomputed, it is simply no longer computing. This much should hopefully be uncontroversial. The more difficult cases are those where the system still computes, i.e. we provide it with an input and it provides us with an output, but, due to a physical (operational) malfunction, it does not provide us with the 'correct' output (that which we would expect to receive, were we able to verify the abstract computation by some other means). A simple case like this might be one where the physical sensitivity of a voltage gate has changed, such that where it once computed AND it now computes OR (and even a simple case like this could of course have serious consequences for the more complex operations within which it is embedded). This might seem *prima facie* like a clear case of miscomputation due to operational malfunction: the component was meant to compute AND, but (due to a change in its physical structure) it instead computed OR. However, recall that all it means for a computational component to follow a rule is for it to respond (according to its own physical structure) to the physical structure of the system of which it is a part. Our malfunctioning AND-gate has done precisely this – it receives a pair of voltage levels as inputs and, depending on whether the sum of those voltages is above a certain threshold, it produces another voltage level as output. So it is still following the rule embodied in its physical structure, and thus computing correctly, it's just that this structure has changed. That change itself might very well qualify as a malfunction, such that we can say that the AND-gate is a malfunctioning component, but I don't think that we should say that it is *miscomputing*. This is because the component, *qua* computation, has done nothing wrong; it is 'just following the orders' given to it by its (new) physical structure (cf. Rapaport 2019: sec. 2).

There are some obvious objections and replies to this argument that I will turn to in the next section, but for now I just want to summarise where we have got to. We saw that there are two main classes of (putative) miscomputations, those due to design errors of some kind, and those due to operational malfunctions. The former do not qualify as miscomputations because the fault lies outside of the system, whether that be with the system's designer, its manufacturer, its programmer, or its end user. The latter do not qualify as miscomputations

[5]There are some concerns in this vicinity to do with Kripke's discussion of Wittgenstein's rule-following considerations, see e.g. Buechner (2011, 2018); cf. Wittgenstein 1953, Kripke 1982.

because they either prevent the system from performing any computations at all (in the case of a total breakdown), or else they change the rule that the system is meant to follow (embodied in its physical structure) such that while it computes something different from what it was originally designed to, it does not *miscompute*. Therefore there is no such thing as miscomputation.

3 The topic formerly known as miscomputation

In this final section I will attempt to address a number of outstanding issues, possible objections, and further applications of the argument developed in the previous two sections. These all have a common theme, which is that even if there is no such thing as miscomputation, we must still say something about cases where people (philosophers, cognitive scientists, and computer scientists) talk as though there was, in order to make sense of both our existing concept(s) of computation and actual scientific practice. It is worth emphasising, though, that the account presented here is not intended to be revisionary: rather than attempting to stipulate *how* researchers should talk about computation, I think this account can actually help us to explicate how researchers currently *do* talk about computation. So I am not advocating a general ban on the term 'miscomputation', but instead suggesting that we should exercise some caution in how it is understood and what it implies. (Or to put it another way, while there is strictly no such a thing as miscomputation, it might still make sense to use the term in an informal or colloquial sense, provided that it does not lead to any misunderstanding.)

3.1 Objections and replies

In the previous section I argued that an operational malfunction that changes the physical structure of a processing component, such that it now carries out a different computation, should not be classified as a miscomputation. Given that I deny that *anything* could count as a miscomputation, it seems like an obvious response would be to push back here, and argue that we should count at least *this* kind of operational malfunction as a miscomputation. After all, these are malfunctions that straightforwardly cause a component to compute something other than what it was designed for, e.g. an overly sensitive AND-gate instead computing OR, so why not just call them miscomputations? I agree that this is an intuitive thought, and that it might be a relatively harmless way of using the term miscomputation, but I still think that it implies a misunderstanding about the nature of physical computation: to compute is just to follow a rule, but the rule that a computing mechanism follows is *not* that which was intended by its designer (to which it has no access), but rather that which is embodied in its physical structure. Ideally that structure will conform to its designer's intentions, but when it does not (whether due to poor design or later malfunction) the system itself cannot be blamed for following the 'wrong' rule. What has gone wrong here is not anything computational, but rather the initial specifi-

cation of the rule itself (as embodied in the system's structure). So to call an operational malfunction a miscomputation is misleading, because the system is computing (according to the rule that it was given) just fine.

I have focused here on *mechanistic* accounts of computation, but there is another (fairly popular) class of *semantic* accounts, according to which the analysis of miscomputation might look quite different. While there has actually not been much work done on miscomputation by defenders of the semantic account, we can try and reconstruct what they might say. According to this account, physical computation is necessarily semantic; in addition to possessing the right kind of causal structure (as per the mechanistic account), a computational system must also *represent* something, and to compute is to manipulate *representational* vehicles according to a rule (see Sprevak 2010 for a basic overview). While there are many other points on which mechanistic and semantic accounts disagree (see e.g. Piccinini 2008, Dewhurst 2018a, Shagrir 2018), this does not actually change things too much when it comes to our analysis of miscomputation. I think it is clear that semantic accounts should still deny that design errors are a kind of miscomputation, for all the reasons that I discussed in section 1. The same goes for operational malfunctions, except that there is a certain kind of malfunction that might qualify as miscomputation according to the semantic account, namely *misrepresentations* (cf. Neander 1995). If computation is representational, then (at least some) misrepresentations might plausibly be understood as miscomputations, especially those caused by operational malfunctions. For example, if an operational malfunction causes a computational state to misrepresent some feature of its environment, and as a result of this it produces an erroneous output, then according to the semantic account this might qualify as a miscomputation. I will leave a full analysis of representational miscomputations for a proponent of the semantic account, but I mention this possibility here for the sake of fairness. If, like Piccinini, one thinks that allowing for the possibility of miscomputation ought to be included in the list of desiderata for our theory of physcial computation, then this might even be a reason to favour the semantic account. (Of course, this reasoning goes both ways: I don't think the semantic account works, and one consequence is that I don't think miscomputation should be on our list of desiderata.)

3.2 Applications and further issues

I have focused so far (at least implicitly) on *artificial* computers, i.e. like the one that I am typing on now, but it would also be interesting to consider the question of miscomputation in *natural* computational systems, such as (potentially) the human brain. A full analysis of this question will have to wait for future work, but I would like to briefly comment on one particularly interesting aspect of it, which is the putative role played by miscomputations in computational psychiatry. Colombo (forthcoming) has recently argued that there are several different notions of miscomputation at play in computational psychiatry, and that a semantic account of computation is required to account for all of them. I will not respond directly to that argument here, but I instead want to suggest

a different way to think about 'miscomputation' in computational psychiatry, building on Garson's (2019) work on mental disorders and malfunctions.

Garson argues that rather than being malfunctions as such, many cases of mental disorder might instead be better understood as "developmental mismatches", i.e. cases where a perfectly well-functioning system has just been placed into the wrong environment (2019: 176-8). He gives the example of how a child who has developed aggressive and violent behavioural tendencies in order to cope with an abusive environment might later be diagnosed with conduct disorder (*ibid*: 179). In their initial environment this behaviour was adaptive, but once it has become ingrained and they are removed from that environment it becomes maladaptive. Design errors, in the context of computational psychiatry, look very much like developmental mismatches: the computational system (in this case, the brain) has been designed or programmed for one purpose (whether intentionally or by accident), but is then placed in an environment within which this behaviour is maladaptive. These are neither cases of malfunction not miscomputation: the system performs perfectly adequately, *qua* computation, but this performance does not match its novel context (some of the cases that Colombo considers, such as differences in the magnitude of prediction error signalling in healthy controls and schizophrenic patients, might also be explained in this way). Operational malfunctions, on the other hand, look more like traditional 'physical' diseases, which might cause the symptoms associated with mental disorders, but should not be understood as miscomputations (for the reasons I discussed in section 2). So the neurodegenerative process that is thought to cause Alzheimer's disease might be an operational malfunction that changes how the brain performs computations, but it is not itself a miscomputation, not does it produce any. The distinction between operational malfunctions and design errors maps neatly on to Garson's loose distinction between physical diseases (often, but not always, understood as malfunctions) and mental disorders (often, but not always, understood in terms of developmental mismatches), and can help us to make sense of the role played by what might previously have been called 'miscomputations' in computational psychiatry.

Primiero, Solheim, & Spring (2019) describe malware (i.e., malicious software such as computer viruses) as a kind of miscomputation induced by a targeted attack. The taxonomy and analysis they provide is helpful, and provides some practical recommendations, but following my arguments in the previous section I think it is clear that malware does not cause miscomputations as such, but rather modifies the behaviour of a computational system in some way that is not desirable to its designer or user. The system itself either continues to compute perfectly fine (in the case of malware that hijacks or otherwise infects a system) or simply ceases to compute (in the case of malware that shuts it down entirely), but it does not miscompute. There is a further question of whether we should even think of malware as inducing (non-computational) *malfunctions* in the target computer, i.e. by rewriting the instructions (rules) enmbodied by its physical structure (see section 2.2). Here an analogy with biological viruses might be helpful: relative to the host system, a (biological or computational) virus certainly induces a malfunction, but relative to the virus itself (or its de-

signer), the system functions perfectly well, by serving to reproduce the virus. So in many cases there may not be a clear answer to the question of whether malware induces a malfunction, let alone a miscomputation.

Finally, I want to briefly discuss a useful distinction introduced by Tucker (2019), between a mechanism's *proper* and *actual* functions. He (like almost everyone else) is a realist about miscomputation, but he defines it in terms of *norms* that are individuated widely (i.e. with reference to selection history or designer intentions), whereas he defines computational *behaviour* in narrow terms (i.e., just in terms of the physical structure of the system). Widely individuated norms fix the proper function of a computing mechanism (what it's *supposed* to do) while narrowly individuated behaviour fixes its actual function (what it *actually* does), and, for Tucker, a miscomputation occurs when these two functions are misaligned. This could be because of either design error (the system was programmed badly) or operational malfunction (it doesn't work properly). I am willing to allow that mismatches of this kind explain what we usually call miscomputations, but I would argue that we should restrict talk of computation (and thus miscomputation) proper to what Tucker calls 'actual' functions (and I suspect that he might agree). A full presentation of that argument will, however, have to wait for another day.

If there is no such thing as miscomputation, then what ought we to say about the inclusion of miscomputation in Piccinini's list of desiderata for an account of physical computation? I think he is correct to say that we need to make sense of the kinds of thing people (including computer scientists, computational neuroscientists, etc.) say when they believe that a computational process has "gone wrong" in some sense, but I don't think that we need any notion of miscomputation *as such* in order to do this – and in fact, relying on such a notion might hold us back in some cases. The taxonomy provided by Fresco & Primiero (2013) makes it clear that there many different ways in which a computational process can "go wrong" that don't really have much in common with one another at all. Our diagnosis of, and solution to, a programming error is going to be very different to that which is appropriate for a manufacturing defect or operational malfunction. Each case calls for a different kind of analysis and response, and so referring to them all under the broad category "miscomputation" is misleading at best. Of course, one could choose to use the term to refer to just one of these cases (probably operational malfunctions), but for the reasons I have argued for here, I think even this limited usage could have confusing implications, and so it is better that we avoid it in favour of more precise language.

Conclusion

I first reviewed some existing accounts of physical computation and miscomputation, focusing on the version of the mechanistic account developed by Gualtiero Piccinini. I then argued that, according to this account, there can be no such thing as miscomputation, as all putative cases of miscomputation are either the result of design errors (for which the computing mechanism itself cannot be

blamed) or operational malfunctions (which are not strictly computational). Finally, I considered some possible objections to, and further implications of, this argument, including the idea that operational malfunctions should in fact qualify as miscomputations, an alternative semantic account of miscomputation, the role of miscomputation in computational psychiatry, the case of malware, and a recently proposed distinction between proper and actual functions. In future work I would like to extend these considerations to include some more general applications of the argument that there is no such thing as miscomputation, such as to debates about the neuroscience of free will, agency in artificial (and natural) systems, and what it means to follow a rule.

References

Babbage, C. 1864. *Passages from the Life of a Philosopher.* London: Longman, Green, Roberts, Longman, & Green.

Buechner, J. 2011. "Not Even Computing Machines Can Follow Rules." In Berger (ed.), *Saul Kripke.* Cambridge, UK: CUP.

Buechner, J. 2018. "Does Kripke's Argument Against Functionalism Undermine the Standard View of What Computers Are?" *Minds and Machines*, 28(3):491-513.

Chalmers, D. 1994. "On implementing a computation." *Minds and Machines*, 4(4): 391-402.

Coelho Mollo, D. 2018. "Functional individuation, mechanistic implementation." *Synthese*, 195(8): 3477-3497.

Coelho Mollo, D. 2019. "Are There Teleological Functions to Compute?" *Philosophy of Science*, 86(3): 431-452.

Colombo, M. Forthcoming. "(Mis)computation in Computational Psychiatry." In Calzavarini Viola (eds), *New Challenges in Philosophy of Neuroscience.* Springer Studies in Brain and Mind.

Dewhurst, J. 2014. "Mechanistic Miscomputation." *Philosophy & Technology*, 27(3): 495-498.

Dewhurst, J. 2016. "Review of *Physical Computation.*" *Philosophical Psychology*, 29(5):795-797.

Dewhurst, J. 2018a. "Individuation Without Representation." *The British Journal for the Philosophy of Science*, 69(1):103–116.

Dewhurst, J. 2018b. "Computing Mechanisms Without Proper Functions." *Minds and Machines*, 28(3): 569-588.

Floridi, L., Fresco, N., & Primiero, G. 2015. "On Malfunctioning Software." *Synthese*, 192:1199–1220.

Fresco, N. 2014. *Physical Computation and Cognitive Science.* Springer Netherlands.

Fresco, N. & Miłkowski, M. 2019. "Mechanistic Computational Individuation without Biting the Bullet." *The British Journal for the Philosophy of Science*, online first.

Fresco, N. & Primiero, G. 2013. "Miscomputation." *Philosophy & Technology*, 26(3):253-272.

Garson, J. 2019. *What Biological Functions Are And Why They Matter.* Cambridge, CUP.

Godfrey Smith, P. 2009. "Triviality arguments against functionalism." *Philosophical Studies*, 145:273–295.

Haugeland, J. 1985. *Artificial Intelligence: The Very Idea.* Cambridge, MA: MIT Press.

Kripke, S. 1982. *Wittgenstein on Rules and Private Language.* Oxford: Blackwell.

Milkowski, M. 2013. *Explaining the Computational Mind.* Cambridge, MA: MIT Press.

Neander, K. 1995. "Misrepresenting & malfunctioning." *Philosophical Studies*, 79:109–141.

Petricek, T. 2017. "Miscomputation in software: Learning to live with errors." *The Art, Science, and Engineering of Programming*, 1.2: 14.

Piccinini, G. 2007. "Computing Mechanisms." *Philosophy of Science*, 74(4): 501-526.

Piccinini, G. 2008. "Computation without representation." *Philosophical Studies*, 137(2):205-241.

Piccinini, G. 2015. *Physical Computation.* Oxford: OUP.

Primiero, G., Solheim, F.J., & Spring, J.M. 2019. "On malfunction, mechanisms and malware classification." *Philosophy & Technology*, 32(2), 339-362.

Putnam, H. 1960. "Minds and Machines." In Hood (ed.), *Dimensions of Mind: A Symposium*, New York: Collier.

Putnam, H. 1988. *Representation and Reality.* Cambridge, MA: MIT Press.

Rapaport, W.J.2019. "Syntax, Semantics, and Computer Programs." *Philosophy & Technology*, online first.

Schweizer, P. 2019. "Triviality Arguments Reconsidered." *Minds and Machines*, 29(2):287-308.

Shagrir, O. 2018. "In defense of the semantic view of computation." *Synthese*, online first.

Sprevak, M. 2010. "Computation, individuation, and the received view on representation." *Studies in History and Philosophy of Science Part A*, 41(3):260-270.

Sprevak, M. 2018. "Triviality arguments about computational implementation." In Sprevak & Colombo (eds.), *The Routledge Handbook of Philosophy of Computation.* Routledge.

Tucker, C. 2018. "How to Explain Miscomputation." *Philosophers' Imprint*, 18(24).

Turing, A. 1950. "Computing Machinery and Intelligence." *Mind*, 49:433-460.

Wittgenstein, L. 1953. *Philosophical Investigations.* Macmillan Publishing Company.