Comments on *The Physical Signature of Computation*

Corey J. Maley
Purdue University
cjmaley@purdue.edu

1 Introduction

In the world of the philosophy of computation, a central problem is determining what it is that makes a physical system a computer. Like many philosophical problems, this one is simple to state, and might not seem all that difficult to solve, at least at first glance. In practice, one rarely encounters confusion about which things in, say, one's office, are computers and which are not. When checking my email or writing commentary on a newly-published philosophy monograph, I'm pretty sure I've never tried to use my coffee cup or office chair because I thought they might actually be computers, and customers at Best Buy looking for the latest computer from Apple aren't likely to be convinced that a bucket of water or a pound of ground beef can perform computations just as well as anything.

However, there *is* a problem. What most theorists—both computer scientists and philosophers—take as *definitive* of computation is not some physical configuration of circuitry, but an abstract mathematical object. Most familiar is what we now call the Turing Machine, a particular type of abstract automaton, that features centrally in computability theory and other branches of theoretical science, as well as discussions of computational functionalism in the philosophy of mind. Because they are mathematical objects (or constructs, or what have you), TMs are wholly non-physical, just as are matrices, fractions, and differential equations. Genuine computational systems of the type we all use nearly every day are, of course, physical. What is needed, then, is a precise account of how to connect, or relate, the abstracta that are definitive of computation to the concrete physical systems that constitute real computing machines. When a physical system is connected/related in the right way to a particular automaton, we say that the system *implements* that automaton; the philosophical problem is then making this connection/relation precise, such that examples of known computational systems are correctly characterized as computational, while examples of known non-computational systems are correctly characterized as non-computational.

Anderson and Piccinini (A&P) have developed a version of what is called a mapping account of computational implementation. In short, mapping accounts claim that implementing an automaton in a physical system involves mapping the elements of the automaton to the elements of the physical system. Without some restrictions on what counts as a legitimate "map," we can map the state of any automaton to any physical system, producing the result that any physical system performs any (and every) computation (Putnam, 1988). A number of mapping accounts have been offered in recent decades, each of which provides constraints on what counts as a legitimate mapping between the abstract elements of automata on the concrete elements of physical systems.

A&P's Robust Mapping Account (RMA) is easily the most sophisticated mapping account on the market, with each element of the account clearly and completely articulated, and the relations between them made very explicit. It does justice to what makes contemporary digital computers genuine implementations of automata, and thus genuine computing systems. Special attention is paid to considerations surrounding Landauer's Principle, which is about the energy costs required to change a physical element from one stable point to another—which, in the context of digital computation, is about flipping a bit.

The benefit of this attention is demonstrated in how the RMA fares better than Chalmers' account of implementation via combinatorial state automata. A&P show that Chalmers' account, while an improvement over simpler mapping account, still fails to block illegitimate computations from posing as legitimate. In brief, the benefit of the RMA is that it introduces criteria that guarantee that it's the physical computing system—rather than some clever mapping—that does the actual work of computing. This is just what we should want: to use a very simple example, we do not want a mapping of the temperature of a rock at minute 0, minute 1, minute 2, minute 3, minute 4, and so on, that corresponds to the digits 1, 4, 1, 5, 9, and so on, to justify the claim that a rock just sitting at a stable temperature is computing the digits of pi. One of the values of computing systems, after all, is that they literally compute, and if an account of computation renders a stationary rock as a genuine computer, that account must be cast into the flames.

Further benefits of the RMA are shown in discussions of pancomputationalism, which, in general, is the thesis that computation occurs in all physical systems. A&P discussion stronger and weaker varieties of pancomputationalism, as well as what they call ontic pancomputationalism, which is the view that the entire universe is, in some important sense, a computational system. In each of these cases, the RMA fares well in justifying what I take to be the most sensible views.

More positive things could be said about their account; the majority of the book is dedicated to the careful exposition of the RMA, showing where certain disagreements about the strength of the connection between an abstract formalism and a physical system can be captured as choice points in determining whether a system implements a computation. Depending on that strength, A&P classify the system as either weakly, robustly, or strongly implementing the computation in question. This flexibility, and the clarity of the decision procedure for making such determinations, are remarkably clear.

Nevertheless, criticism is probably more interesting for present purposes. So let me say briefly what my criticisms are. First, and most briefly, is a problem not just for A&P, but for all mapping accounts of computation. When computation amounts to implementing any particular automaton, and automata need not process information or representations in any rich sense, then computation ends up being about a kind of abstract specification of finite state changes in a physical system. There's nothing wrong with this kind of analysis of physical systems: it can be quite useful to see what different systems have in common with one another from a bird's-eye view. However, that also seems to miss what I take to be most interesting about computation, what separates it from other physical processes, and what makes it so useful: the processing of information and representations. My own view is that computation is, and always has been, essentially about systematically manipulating abstract objects (primarily numbers) by physically representing them, and manipulating those physical representations as a proxy for mathematically manipulating them. That's what makes the Antikythera mechanism a computer, but an arbitrary assemblage of gears that might be part of a child's toy not a computer. It's also what makes people doing long division by manipulating numbers digit by digit an instance of computation, while shuffling individual letters in some arbitrary, meaningless (but systematic!) way is not an instance of computation. However, as I said, this is not a problem specific to A&P, but a fundamental disagreement about what computation is in the first place. That argument is for another time.

The criticisms specific to A&P that I'll address here are as follows. First, despite some passing remarks to the contrary, the RMA cannot characterize analog computation. Second, it is not clear to me that the RMA has the resources to distinguish between systems that can be viewed as computational and systems that are genuinely computational. Finally, the last chapter, which is meant to connect the RMA to concerns about the place of computation in cognitive science and neuroscience, fails to clearly show how concerns about teleofunctions and medium independence cohere with the RMA. So let's go through these one-by-one.

2 Mappings and Analog Computation

Virtually everyone takes analog computation to be computation that uses continuously-valued, as opposed to discretely-valued, variables. This is sort of true, but misses cases of genuinely discrete analog computation. Moreover, it misses the more important feature of analog computation, which is that it uses physical magnitudes to represent numeric magnitudes; I've argued for these points elsewhere. But we can set this aside for now, and just consider continuous analog computational systems.

Unlike digital computation, analog computation does not involve a special computational formalism like TMs or other automata. This is a simple point, but one not appreciated because of what I call the "$N \neq 1$ Problem." This is the problem whereby digital computation is taken to be *the* paradigm of computation, and thus features of digital computation are considered to be necessary for *any* computation. However, analog computation and analog computational systems predate digital systems, even though they are no longer studied or taught in the way that digital systems are, and have as legitimate a claim to being genuinely computational as do digital systems. The result is that, because digital computation is taken to necessarily involve the implementation of automata, theorists and philosophers of computation uncritically assume that all computation must work like this, and it's a simple matter of replacing the automata of digital computation with some other formalism for analog computation.

Let's look at a real example. Consider the following initial value problem (a differential equation with initial conditions), adapted from (Peterson, 1967, p. 36):

$$y = x'' + 3x' + 16x;\ y = -80$$
$$x'(0) = -0.64;\ x(0) = 2$$

An electronic differential analyzer (EDA), one of the oft-used components in many electronic analog computers, can be set up or programmed to solve this system. Doing so requires that we connect certain electronic components in a way that reflects the mathematical structure of the system. In this example, shown in Figure 1, we have an adder (triangle with $\sum$), two integrators (triangles with $\int$), a sign inverter (triangle with $-$), and two multipliers (circles with $\times$).

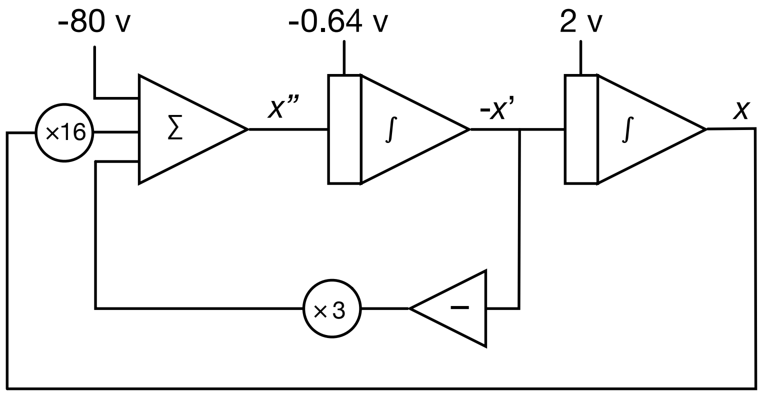

Figure 1: *Electronic analog computer schematic.*

Note that the values of the variables are not "encoded" into a digital representation; the values of the variables are simply the magnitudes of the voltages. Further, there is no algorithm here, in the usual sense. Integration, summation, and multiplication are all performed by circuit elements. There is, of course, a way that these operations are performed, which involves manipulating those voltages just mentioned. But this is a far cry from how one specifies how integration is done in a digital computation (e.g., by a Runge-Kutta algorithm, or a trapezoidal approximation).

From the schematic in Figure 1, an actual analog computer would consist of these different elements connected as suggested: the adder, for example, has three inputs (an input to the initial problem, and two inputs from two different multipliers), and one output (to the first integrator). There is no "architecture" other than the connection of these different components.

So, what exactly is the formalism that is implemented? One answer: the equations. These equations perfectly describe the physical system that is the analog computer in a way analogous to the requirements of A&P's account. So we might say that the physical system—the analog computer—implements those equations. Unfortunately, this means that lots of physical systems implement lots of these formalisms, and analog computers are everywhere. Systems described by differential equations are everywhere: this is why differential equations are ubiquitous in science and engineering. While not quite pancomputationalism, this line of thought results in taking an enormous range of physical systems as analog computers: those systems implement the equations that describe them.

So if the equations don't work as the formalism to be implemented, what else might there be? Another thought might be that the schematic shown in Figure 1 is the formalism. That might seem promising, but then the formalism in question isn't medium independent: this is a schematic for an electronic analog computer, after all, which uses the magnitude of voltages to represent the values of the variables. Medium independence is supposed to be a defining feature of computational formalisms, so this is a non-starter.

What about a modified version of this schematic, where we erase the three "v"s? Then, it would seem, we would have something medium independent. True enough, but then we also just have a different representation of the equations that we started with, and we're back at the first problem we faced.

The solution is to accept that analog computers just do not implement a medium independent formalism in the way that digital computers do. The only reason that this is hard to accept is due to the insistence that physical computation *must* involve the implementation of medium independent formalisms, which in turn is due to the $N \neq 1$ Problem.

Before ending, it is important to note that analog computers *can* be multiply realized: there are different mechanisms that can perform integration or multiplication. More dramatically, the system of equations could be solve by a mechanical analog computer that uses moving parts and magnitudes like angle or number of rotations, or length of a shaft, with integration performed by mechanical systems. Nevertheless, physical analog computation is not a matter of implementing a computational formalism.

3 Computational/Mathematical Modeling vs. Explanation

One of this books' co-authors wrote an influential article describing the difference between computational modeling and computational explanation (Piccinini, 2007). In short, while all sorts of scientific phenomena can be modeled computationally, only a few such phenomena are (putatively) explained, or explainable, by the computations they perform. Meteorologists computationally model weather systems all the time; but there are no weather-related phenomena that are the result of the computations that weather systems *literally* perform. This is in contrast with much of what happens in computational neuroscience, where particular capacities of neural systems are explained via the computations that those systems are purported to *literally* perform.

The RMA account certainly allows for all kinds of systems to be modeled computationally without those systems implementing the computations that model those systems. Given the requirements of the Principle of Computational Equivalence (PCE), there are many ways in which a particular computational system might model, or otherwise describe, some other system.

However, if there are physical systems that do implement some automaton, then they are, by the lights of A&P, computing. There are lots of candidates for such systems. For example, many artifacts implement automata, such as traffic lights, pressure-sensitive door openers, garage door openers, coin-operated turnstiles, and many more. This is a consequence of the fact that may of these systems are modeled by state diagrams that are essentially just automata.

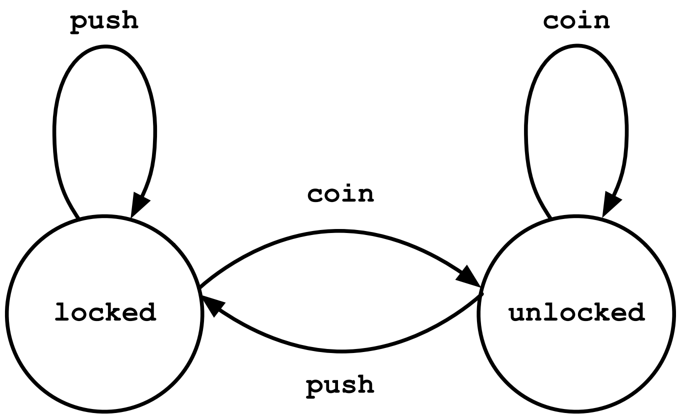

Figure 2: State diagram for a coin-operated turnstile.

There are many examples in organisms as well. DNA transcription is an example, where strings of DNA are converted into strings of RNA: this is a straightforward implementation of an automaton. Or, the vast numbers of electrophysiologically active cells, particularly non-neural cells, such as the smooth muscle cells of the heart, stomach, and pancreas.

Now, I suspect that A&P may reply that these examples are all medium *de*pendent in some way that might disqualify them. Whether medium independence is applicable or not is unclear (see the next section). However, I do not see how to thread the needle between getting the electric potentials generated by neurons to be medium *in*dependent, while these other examples are medium *de*pendent. If the argument that neural firing is medium independent is sound, despite the seeming dependence on particular ions and other chemicals (i.e., the various potassium, sodium, and calcium currents (among others), plus the particular neurotransmitters) which are responsible for neural firing, then there is no principled way to block these other examples without begging the question.

But back to the main point. If these examples are all genuine instances of the implementation of automata, then they all literally compute. But notice that this fact is completely irrelevant to the explanation of what these various examples do. DNA transcription is *not* explained by "transcription computation," or "polymerase computation." Coin operated turnstiles are not explained by the computations they perform.

A broader concern has to do with the utility of this notion of the implementation of automata. A&P use the example of harmonic oscillators as what they call a structural description of a physical system—a description that does not contain reference to the particular physical materials or components of that system. Harmonic oscillators describe spring-mass systems, certain electrical systems, acoustical systems, and many more. Finding out that a system can be described as a harmonic oscillator is useful because it tells us that something about how the system oscillates, and it provides us with mathematical tools to analyze that oscillation once the parameters of the relevant equations are determined.

Another example might well be game theoretic descriptions of social or political interactions. That a particular interaction is an instance of Prisoner's Dilemma, or Battle of the Sexes, for example, is informative. Just like harmonic oscillators, these mathematical frameworks can be applied to a wide variety of situations, with a wide variety of particular payoff matrices. Finding out that a system can be described as a Prisoner's Dilemma is useful because it tells us something about that system, and it provides us with mathematical tools to analyze that system once relevant parameters are determined.

So what is the analogous payoff for determining that a system implements an automaton? One obvious one is that when we want to *build* a system that implements a given automaton, then a successful implementation guarantees (within limits) that the physical system's behavior is fully described (at the relevant physical level) by that automaton. This is by no means a trivial feat: computer engineering is hard! But more generally, when we discover that some system is computing, what kind of inferences can we draw about another system that is also computing?

If both systems implement the same automaton, then we know that they will have the same suite of behaviors. But if they implement different automata, there does not seem to be anything interesting we can say about what they have in common. Even if we know they implement the same *type* of automaton (e.g., two different physical systems implement two different linear bounded automata), the huge variety of linear bounded automata makes it impossible to say that they have anything interesting in common (except that they both implement some automaton). This is complete unlike the situation of a harmonic oscillator, or a Prisoner's Dilemma, where these types of structural descriptions are useful for saying what different physical systems have in common: that they oscillate in specific ways, or that they have a Nash equilibrium that will result in less-than-ideal outcomes.

On this front, the generality of the behavior of automata makes them useless for comparing the behavior of systems that implement them—unless they implement identical automata. Automata can't compute everything, they can compute infinitely many things. Now, to be fair, there are some limited cases where knowing that two systems implement the same *type* of automata can be used to make some limited generalizations. Two systems that implement DFAs, for example, both cannot recognize languages more complex than regular languages. However, with large enough DFAs, this is a theoretical difference that is irrelevant to digital computational systems here on Earth. After all, *all* extant digital computers are actually DFAs, not

TMs, so they all have exactly the same limitations.

All in all, the concerns of this section have to do with what we get out of saying that a physical system implements an automata, outside of the clear utility in engineering practices. There may be things to say here, but I do not see what they are. My own view is that this is where a representational account of computation—as I like to call my own view—gets its purchase: what is useful about knowing that two systems are computational is that they both traffic in representations of numbers (or representations built out of representations of numbers), and that those representations are manipulated in systematic ways; i.e., via the manipulation of the physical properties of those representations that do the representing. However, that discussion is (again) for another time.

4 Medium Independence and Teleofunctions

The last section of the book aims to connect the RMA to issues of computation in the cognitive sciences and neuroscience. Here, medium independence plays a prominent role that it did not otherwise play in the rest of the book (although one could argue that it was implicit in much of the exposition, even though it was only explicitly mentioned once or twice). I have two main concerns: first, there is a kind of elision between the process of abstracting and the notion of abstract objects, whereby one can get to abstract objects by simply abstracting (i.e., omitting details) from physical descriptions. Of course, one can do whatever one wants! But it is not at all clear what, if any, restrictions should be in place for this process. This is important for natural systems, where we would like to know *if* they perform any computations (and if so, which ones), rather than assuming that they do from the outset. If we do not have a method in place for treating the question of whether a natural system computes as something like an empirical hypothesis, then the project of computational explanation loses all force; instead, we have computational perspectivalism, where one researcher can choose to view a system as computational (by assuming it does from the outset) and another can choose not to (by not so assuming).

My second concern is the divergence of what counts as computation in this chapter from the earlier chapters. By the lights of the rest of the book, a physical system can implement an automaton—can literally compute—without it having any functions, or even without us knowing anything about it. This is a simple consequence of the account, which A&P acknowledge. However, when it comes to organisms and artifacts, we are told in Chapter 9 that physical computation in specifically biological systems requires a mechanism with a function. The reason for this difference between computation in non-biological physical systems and biological physical systems is to account for the normative aspect of computation in artifacts and organisms. To be sure, these extra requirements seem to be consistent with the account developed earlier; however, the result is that there may be very many computations going on in an organism considered *only* as a physical system, but which then do not count as computations when the organism is considered as a biological system. Similarly, there may be very many computations going on in an artifact considered *only* as a physical system, but which then do not count as computations when the artifact is considered as an artifact. This is at least in tension with the earlier, well-motivated idea that computation is just about the right kind of mapping between physical states and computational states.

Let's look at the first concern first, having to do with medium independence and abstraction. Consider the following quotation.

> Physical computation is a process that becomes evident once the appropriate abstraction (omission of detail) is performed (cf. Kuokkanen 2022; Fuentes 2023). Abstraction is a feature of all physical descriptions, with the possible exception of some complete descriptions of elementary particles. The relevant question is which abstraction is needed to capture the physical signature of computation. The needed abstraction is the selection of a system's relevant physical states and state transitions to be mapped onto computational states and state transitions. With the right mapping in place, the property that manifests in genuine physical computing systems is physical-to-computational equivalence, which is a medium-independent property. (Anderson & Piccinini, 2024, p. 244)

The notion of a medium-independent property is very difficult to explicate, because there seem to be several conflicting definitions. On this same page, medium dependence/independence is determined by "whether the teleofunction that defines it is medium-dependent or -independent." So, playing chess is given as an example of medium-independence (or, in terms of a property, perhaps this should be "the ability to play chess," or "being a chess playing machine"). But we just saw that physical-to-computational equivalence (PCE) is a medium-independent property, so that means that the teleofunction that defines PCE must be medium independent. I cannot make sense of what a teleofunction could even be that defines PCE, especially in an organism. It would amount to something like: "X has the function of ensuring that there is a physical-computational equivalence between a computational description and a physical description." If not an outright category mistake, this is not a function that an organism could have. I suppose, in some way, the examples of *this very book* have that function: they are intended to show that these things hold. But this is totally opaque.

A few pages later, we get a different characterization of medium-independence, couched in terms of vehicles and multiple realizability.

> [P]rocesses that manipulate multiply realizable vehicles are medium-independent, and biological systems whose processes are medium-independent and manipulate vehicles in accordance with a rule are biological computing systems. (Anderson & Piccinini, 2024, p. 250)

We are then told that neurocognitive systems do just this, because neural firing rate and spike timing are the most functionally-relevant neural properties, and firing rate and timing are multiply realizable. As

I've argued elsewhere, and which is again evident in this chapter, one cannot both declare from the outset that some processes are medium dependent (what locks, for example, do) while others are medium independent (what neurons do), but then offer a *definition* of medium independence that counts the vehicles processed by supposedly medium-*de*pendent processes as multiply-realizable (the vehicles processed by locks are definitely multiply realizable, because different locking mechanisms require different types of keys), and fail to then count them as medium independent after all.

Let's dig a little deeper. Let's grant that the vehicles processed by neurons (spike frequency and timing) are medium independent. Lots of other things can generate frequencies and timings: all of the systems described by harmonic oscillators, for instance, so audio signals, and springs, etc. But none of those could actually be processed by *neurons*. Neurons actually have to use voltage changes in a very narrow range, where those voltage changes are produced by the movement of ions in and out of the cell. This seems very medium-dependent, but again, let's grant that it's not for the reason just given.

What blocks lots of other things from also not being medium independent for exactly the same reason? Declaring that locks are medium dependent, even though their vehicles are just as multiply realizable as neural spikes, is just begging the question.

What seems to be going on here is that abstracting from the particulars of neural firing to get to something like "only the frequency and timing of the firing matter, which is multiply realizable," is legitimate abstraction, but "only the pattern of the key matters, which is multiply realizable," is illegitimate abstraction.

This seems unmotivated, except of course if one has already decided that neurons compute but that locks don't. But by their own lights, lots of things are medium independent, despite what they might seem *prima facie*. It's simply begging the question if one gives a characterization of medium independence, but then does not accept the application of that characterization to cases that we might have thought were medium *de*pendent before applying the characterization.

This all seems to turn on what kind of abstraction is legitimate and what kind is not. Later, single-celled organismic information processes are given as examples of medium *de*pendent processes, because the biochemical cascades depend on particular molecules. True enough. But so does neural firing! If abstracting away from the particular molecules (potassium and sodium ions, serotonin, etc.) in neural firing is okay, then it should be okay to abstract away from the particular molecules in the single cell example. The multiply realizable property is the presence or absence of a signal. Of course, if we did actually realize this property with some other molecule, it wouldn't work in those cells. But that's true of neural firing, too. We can't switch to other molecules, and we certainly can't switch to acoustic frequencies and timings. But all that matters in the single cell case is presence or absence of a signal, which is absolutely multiply realizable.

Now, perhaps there is a way to clean this all up; I have my doubts, which I have expressed carefully in print (Maley, 2023). To be fair, A&P note this work, but then beg the question in the way I just mentioned. Nothing in this discussion of medium independence improves upon the characterization of medium independence outlined in Piccinini's earlier monograph (Piccinini, 2015), which was very carefully articulated. In fact, if anything, this chapter makes the notion murkier than it was before.

Finally, although it's only mentioned in passing, the authors claim that"Since medium-independent properties do not depend on the specific constitutive properties of a medium beyond its possession of relevant degrees of freedom and their organization, medium-independent properties can occur at *any* spatiotemporal scale," (Anderson & Piccinini, 2024, p. 246), emphasis original. However, this is absolutely not true of the examples they provide of medium independent properties, particularly in the context in which they are supposedly instantiated. Neural firing frequency and timing are supposedly medium independent, but only frequencies within a very specific range are going to count in a neural system. Further, the frequency of *anything* is fundamentally limited by physical law. Playing chess was another putative example of a medium independent property, but any physical realization of chess can be only so big or small, again, limited by physical law. Too big and a chessboard will collapse under its own weight and create a nuclear explosion; too small and there will be no particles able to actually realize it. The supposedly medium *de*pendent properties of sprinkling water and pulling corkscrews face exactly the same constraints. If we have a planet-sized cork, we could have a planet-sized corkscrew; we can have a microscopic sprinkler, as long as the nozzle is big enough to allow molecules of whatever fluid to flow out.

Enough about that: let's turn to the second concern. This concern is the addition of the teleofunctional requirements on top of the earlier requirements. Coupled with the earlier medium independence concerns, it seems that we have a completely separate—albeit compatible—account of computation in this chapter, which is basically a summary of the account developed in (Piccinini, 2015). In other words, we have the RMA account of computation, which gives objective criteria for what it takes for some physical system to compute. As mentioned earlier, this account does not require functions at all. Separately we have the mechanistic account given in Chapter 9, which does.

It seems that what the authors are doing here is providing the additional ingredients to satisfy Criterion U, thus making the computations implemented in artifacts and organisms satisfy a Strong Mapping Account of implementation, in turn making miscomputation a possibility. But then this is at odds with the Robust Mapping Account developed in Chapter 5. In particular, the authors claim that Robust Mapping "is both necessary and sufficient to qualify a physical system as a physical computing system," (Anderson & Piccinini, 2024, p. 123). But given Chapter 9, this seems to not be true if those physical systems are artifacts or organisms. I do not know what systems are supposed to be counted *in* as computing systems that are *not* organisms or artifacts in the first several chapters of the book.

This points to a broader concern: accounting for miscomputation was supposed to be one of the desiderata of an account of physical computation, but when we encounter how the RMA handles miscomputation, we are told that it needs additional resources such that "Robust" becomes "Strong." So either handling miscomputation should not be a desideratum, or the RMA is not actually an adequate theory of physical computation.

Relatedly, in Chapter 9, we are never told what particular automaton, or other formalism, is the one implemented by, say, neurons engaged in firing. What makes them computing is something entirely different than what makes physical systems computational in the earlier part of the book. Some candidates for the relevant formalism might be something like the Hodgkin Huxley equations that describe neural firing, or perhaps the purely mathematical specification of how the magnitudes of dendritic currents are added and multiplied, then if that magnitude reaches a certain threshold at the cell body, an action potential is generated, characterized by a different kind of current with its own magnitude. If this is the idea, then we face the same kinds of problems mentioned in the section earlier on analog computation: these just aren't *computational* formalisms of the kind required in the rest of the book. In any case, if that formalism *is* necessary, we are not told what it is, but we *are* told that neural systems compute despite this omission.

5 Conclusion

The primary virtue of this book is the clarity and precision of the Robust Mapping Account of Implementation and the various criteria that go into satisfying this account. As always, the downside of any precise characterization is that it makes shortcomings equally clear. As such, I do not think the account can be modified to account for analog computation because of the requirements of a particular type of computational formalism. The fix is to just leave out the suggestion that this account can apply to non-digital computation. At the same time, the notion of medium independence is unclear, except when we start with pre-given automata. Perhaps medium independence can be fixed in some way, but I do not know what that way will be. The rest of the account, however, does not seem to me to need fixing, but simply limited to digital computation. I look forward to the authors' thoughts about these comments.

References

Anderson, N., & Piccinini, G. (2024). *The physical signature of computation*. Oxford University Press.

Maley, C. J. (2023). Medium Independence and the Failure of the Mechanistic Account of Computation. *Ergo an Open Access Journal of Philosophy*, *10*(0). https://doi.org/10.3998/ergo.4658

Peterson, G. R. (1967, January 1). *Basic Analog Computation*. The Macmillan Company.

Piccinini, G. (2007). Computational modelling vs. Computational explanation: Is everything a Turing Machine, and does it matter to the philosophy of mind? *Australasian Journal of Philosophy*, *85*(1), 93–115. https://doi.org/10.1080/00048400601176494

Piccinini, G. (2015, January 1). *Physical Computation: A Mechanistic Account*. Oxford University Press.

Putnam, H. (1988). *Representation and Reality*. MIT Press.