The ~~In~~Determinacy of Computation

André Curtis-Trudel
The Ohio State University
curtistrudel.1@osu.edu
www.aetrudel.net

Dec 27, 2021

Forthcoming in *Synthese*

Abstract

A skeptical worry known as 'the indeterminacy of computation' animates much recent philosophical reflection on the computational identity of physical systems. On the one hand, computational explanation seems to require that physical computing systems fall under a single, unique computational description at a time. On the other, if a physical system falls under any computational description, it seems to fall under many simultaneously. Absent some principled reason to take just one of these descriptions in particular as relevant for computational explanation, widespread failure of computational explanation would appear to follow. This paper advances a new solution to the indeterminacy of computation. Very roughly, I argue that the computational identity of a physical system is determinate relative to a contextually specified way of regarding that system computationally --- known as a labelling scheme. When a system simultaneously implements multiple computations, it does so relative to different labelling schemes. But relative to a fixed labelling scheme, a physical system has a unique computational identity. I argue that this relativistic conception of computational identity vindicates computational explanation and comports well with computational practice.

1. Introduction[1]

A skeptical worry known as 'the indeterminacy of computation' animates much recent philosophical reflection on the computational identity of physical systems.[2] The worry runs roughly as follows. On the one hand, computational explanation seems to require physical computing systems to fall under a single, unique computational description at a time. On the other, if a physical system falls under any computational description, it seems to fall under many simultaneously. Absent some principled reason for taking just one of these descriptions in particular as relevant for computational explanation, widespread failure of computational explanation seems to follow.

Standard solutions attempt to isolate properties bearing a privileged relationship to computational description and thereby to a system's computational identity. These efforts have played out along two axes. One concerns semantic vs. non-semantic properties, while the other concerns intrinsic vs. relational properties. These axes cut across each other, and the following combinations reflect the most prominent views in the contemporary literature:

- **Non-Semantic individualism** holds that a system's computational identity is wholly determined by its intrinsic, non-semantic properties (Chalmers 1996b; Dewhurst 2018b; Coelho Mollo 2018; Egan 1995).
- **Non-semantic anti-individualism** holds that a system's computational identity is always determined, in part, by its non-semantic relational properties in addition to its intrinsic non-semantic properties (Piccinini 2008, 2015, 2020a; Harbecke and Shagrir 2019; Fresco 2021).
- **Semantic anti-individualism** holds that a system's computational identity is always determined, in part, by its relational semantic properties in addition to its intrinsic non-semantic properties (Shagrir 2001, 2020; Horowitz 2007; Sprevak 2010).

1 Thanks to Soyeong An, Matteo Biachetti, Preston Lennon, Daniel Olson, Richard Samuels, Stewart Shapiro, Declan Smithies, Damon Stanley, and audiences in Bergamo and online at BSPS 2021 for comments and discussion. Special thanks to three anonymous referees, whose suggestions led to substantial improvements in the final paper. Any errors that remain are entirely my own.

2 So-called by Fresco and Milkowski (2021). Known also as 'the multiplicity of computations' (Piccinini 2008, 2015; Coelho Mollo 2018; Lee 2021), and 'the problem of simultaneous implementation' (Shagrir 2001, 2020; Dewhurst 2018b, 2018a).

Although these views are in some respects attractive, ultimately I find them unsatisfactory. Each either seriously distorts or otherwise fails to capture certain important aspects of computational practice. Consequently, this paper develops a solution that better comports with that practice. This solution leads to the following middle position with respect to computational identity:

The Middle Ground holds that a system's computational identity sometimes depends solely on its intrinsic features and sometimes also on its relational features, and that sometimes it depends on a system's semantic features and other times not (Rescorla 2013; Lee 2021).

Here is the plan. I begin in section 2 with a more careful formulation of the indeterminacy problem, and argue in section 3 that extant solutions are unsatisfactory. I present my alternative solution in section 4. Section 5 concludes.

2. The Indeterminacy of Computation

Historically, the indeterminacy of computation has been used to motivate semantic accounts of computational identity (Shagrir 2001, 2020). I will present the problem in more neutral terms. At the broadest level, the goal is to capture descriptive and explanatory practice in the computational sciences. The problem is that under *prima facie* plausible assumptions this practice is incoherent. At the core of the problem are the following two claims:

The Uniqueness Condition. Successful computational explanation requires a physical computing system to have a single, unique computational identity — fall under a unique computational type — at a given time.

Simultaneous Implementation. Physical computing systems simultaneously implement multiple distinct computations.

The trouble is that, under the following auxiliary assumptions, these claims are inconsistent:

The Identity-Implementation Link. The computational identity of a physical system is determined by the computation(s) it implements.

Successful Explanations. There are successful computational explanations.

In the remainder of this section I explain each of these claims and offer some overall clarificatory remarks about the character of the indeterminacy problem. I begin with the two auxiliary assumptions.

2.1 The Identity-Implementation Link

This is an empirical claim about how computer and cognitive science taxonomize physical computing systems. While physical systems can be taxonomized in any number of non-computational respects (e.g., in terms of their mass or shape), these sciences taxonomize systems *computationally* in terms of the computations they implement. Computations are defined in terms of some mathematically characterized computational formalism, such as a Turing machine, finite state automaton, or Java program.[3] In general, computations consist of:

- A non-empty set of internal states.
- Sets of input and output objects. These are the 'domains' over which computations are performed. Either set may be empty.
- A transition function defined over all of the above.

More specific computations are determined by a choice of internal states, input and output sets, and the action of the transition function. Two kinds of computation will play an important role in what follows:

1. **String-Theoretic Computations** compute over string-theoretic or 'linguistic' domains. For instance, on Turing's (1936) original definition, Turing machines compute functions over unary strings.
2. **Numerical Computations** perform computations defined over non-string-theoretic entities, such as natural numbers or truth values. Examples include register machines (Cutland 1980) which compute number-theoretic functions, and Boolean circuits (Savage

3 Also called 'syntactic structures' (Shagrir 2001) or 'computational models' (Rescorla 2013).

2008) which compute Boolean functions.[4]

It is standard to think of mathematical computations as abstract descriptions or models of physical systems, and we say that a physical system implements a computation when that computation 'accurately describes' the system.[5] This is usually cashed out in terms of a structural relation such as isomorphism, so that a physical system implements a computation only if, and perhaps also if, (a) there is a grouping of microphysical states, inputs, and outputs into state, input, and output types, and (b) an assignment of these to mathematical inputs, outputs, and internal states such that (c) under this assignment the physical system and computation are isomorphic (Ritchie and Piccinini 2019).

A simple example is a digital circuit with two inputs and a single output. The circuit outputs 5-10V iff both inputs are in the 5-10V range; otherwise, it outputs 0-5V (Table 1 in Appendix). What computation does this circuit implement? It depends on how the system's microphysical states are grouped into state types and then assigned mathematical values. One option assigns '1' to the 5-10V state type and '0' to the 0-5V state type. Under this mapping, the circuit implements logical AND (Table 2).

This example illustrates that implementing a computation fixes the computational identity of both a system and its states. If two token physical states map to distinct mathematical states, they are computationally type distinct; otherwise, they are type identical. Similarly, if two systems implement the same overall computation, they are computationally type identical; otherwise, they are computationally type distinct. Given this connection between implementation and computational identity, we can frame the views encountered in the introduction as views about implementation. For instance, semantic views hold that to implement a computation a physical system must have certain semantic properties, while non-semantic views allow that a

4 As we will see, string-theoretic and numerical computations have very different implementation conditions. But from a mathematical point of view, the differences between these two kinds of computation are less important. It is typical to start with string-theoretic computations and then define numerical computations by taking strings as representations of numbers (e.g., Davis 1982). However, it is also possible to develop computability theory directly in terms of numerical computations (e.g., Davis, Segal, and Weyuker 1994, ch. 2-4).

5 I will move back and forth between talk of physical systems 'satisfying computational descriptions' and 'implementing' or 'realizing' computations, taking these to be roughly equivalent.

system might implement a computation even if it lacks semantic properties altogether.[6]

2.2 Successful Explanations

It is routine in computer and cognitive science to explain the properties or behaviour of physical systems in terms of the computations they implement. This holds for both artificial computing systems, such as laptops or microprocessors, and natural computing systems, such as the brain. I take it that at least some of these explanations are successful. Computer and cognitive scientists routinely offer explanations of this sort in journal articles and at conferences and other academic venues. These explanations are accepted, at least sometimes, by other members of these disciplines. All else equal, we should take this practice at face value and believe that there are successful computational explanations.

2.3 The Uniqueness Condition

According to the Uniqueness Condition, successful computational explanation requires that a system fall under a single computational type. Given the Identity-Implementation Link, this amounts to the requirement that a system implement a single computation at a time. This claim is intuitively quite plausible, but it is also supported by two more systematic considerations.

The first is that it explains certain aspects of computational practice. As Oron Shagrir (2001, 377–9) notes, when computational scientists explain some phenomenon, they type identify the system responsible for that phenomenon uniquely in terms of the computation it implements. Edge detection in the visual system is explained by the fact that the visual system implements a *specific* computation on retinal signals, for instance. The most straightforward explanation of this practice is that computational scientists are being guided by something like The Uniqueness Condition. If the Uniqueness Condition were false, it would be puzzling why computational scientists type identify physical systems the way they do.

A second consideration concerns the contrastive character of many computational

6 Not everyone will agree with this way of framing the debate. For instance, Oron Shagrir (2001, 382) claims that computational identity involves semantic properties over and above implementation. But this is because Shagrir endorses a wholly non-semantic account of implementation. Rather than claiming that computational identity involves something in addition to implementation, it is more straightforward to see Shagrir as proposing a semantic account of implementation.

explanations (cf. Sprevak 2019, 186). This character is apparent even in very simple cases. For instance, the explanation why a logic gate outputs '0' (rather than '1') given '1' and '0' as input is that it computes AND (rather than OR). But it is highly plausible to think that contrastive explanation requires a contrast (Hitchcock 2013). The fact that a system implements computation C_1 (rather than) C_2 explains the fact that it exhibits effect E_1 (rather than E_2) only if the system doesn't implement C_2. That is, if the logic gate implements AND at the exact same time it implements OR, it is hard to see how we can cite the fact that it implements AND *rather than* OR to explain why it does what it does. It thus seems that to satisfy the contrastive character of computational explanation, a system must implement a single computation at a time, which is just what the Uniqueness Condition says.

2.4 Simultaneous Implementation

The three claims encountered so far are consistent. But trouble emerges once we notice that physical systems implement multiple distinct computations simultaneously. Shagrir (2001, 2020) illustrates this with the example of a tristable circuit, which is sensitive to three different voltage levels: 0-2.5V, 2.5-5V, and 5-10V. The circuit has two input signals and one output signal, and it outputs 5-10V iff both input signals are 5-10V, 0-2.5V iff both input signals are 0-2.5V, and otherwise, it outputs 2.5-10V (Table 3). What computations does this circuit implement? Different possibilities arise from different groupings of states into state types or assignments of abstract values to these types:

1. One option assigns distinct mathematical values to each stable state. For instance, we might assign the labels '0', '$\frac{1}{2}$', and '1' to 0-2.5V, 2.5-5V, and 5-10V, respectively. In this case, the gate computes an 'averaging' operation over its inputs (Table 4). Because this option exploits all the stable states of the gate, it is sometimes known as a 'maximal task' (Piccinini 2015, 41).
2. Another option groups states in the 2.5-10V range into one state type, states in the 0-2.5V range into another, and assigns '1' to the former and '0' to the latter. Under this assignment, the gate implements logical OR (Table 5). Because it ignores the distinction

between 2.5-5V and 5-10V, the gate performs a non-maximal task under this assignment.

3. A third option groups 5-10V into one state type and assigns it '1', and it groups 0-5V into another and assigns it '0'. In this case, the gate computes logical AND (Table 6).

These options all rely on different groupings of physical states into state types. Others hold fixed the groupings and vary the mathematical values assigned to them. For instance:

4. Keep the grouping from (2), but flip the assignment of '0' and '1'. In this case, the gate also computes logical AND, albeit in a different way than in (3) (Table 7).[7]

Of course, these observations are not restricted to Shagrir's tristable gate. Similar considerations apply to nearly any simple computing system. And because more complex computing systems are typically composed of simpler components like this one, if these atomic components simultaneously implement many computations, it is reasonable to expect the same for more complex systems as well.[8]

Simultaneous implementation is attractive. In general, it is useful if a single physical component can perform multiple different computational tasks, for then the same component can be repurposed as the need arises. Simultaneous implementation arguably makes this possible, insofar as it ensures that a single physical component can subserve multiple distinct computations. A real-world illustration of this comes from contemporary microchip manufacture (Hennessy and Patterson 2003, ch. 1). The manufacturing process involves casting silicon wafers into segments known as 'die'. Multiple die are bonded together to form a chip. The casting process is imperfect, and not every cast die is functional. To keep manufacturing costs down, it is thus useful if a single die component can be repurposed to perform different computations, depending on failures in the casting process. Simultaneous implementation may thus play an

7 There is some dispute about whether this is a genuine alternative to the computation in (2) (Piccinini 2020b, 153). There is also dispute about whether there is one kind of indeterminacy at play here, or two (Papayannopoulos, Fresco, and Shagrir, forthcoming). My own view is that there is just one; see sections 3.2 and 4.4.

8 Note that simultaneous implementation falls short of pancomputationalism, the claim that every physical system simultaneously implements every computation. See (Shagrir 2001) for discussion.

important role in computer engineering, and perhaps elsewhere as well.[9]

2.5 A clarification

What sort of indeterminacy is at issue in the indeterminacy of computation? On its face, it appears to be a metaphysical problem about the nature of computational identity. If a system's computational identity is determined by the computation it implements, and if that system simultaneously implements multiple computations, then that system's computational identity is indeterminate between each of the computations it implements. The problem is not simply that we lack evidence favouring one computation over another. Rather, the problem is that there is no clear reason to think that any one computation in particular determines a system's 'true' computational identity.

Ultimately, I think that this initial appearance is deceiving. I will later argue that physical computation is not metaphysically indeterminate. What indeterminacy there is, is epistemic. Before that, however, I will take the problem at face value and will consider solutions that address it on these terms.

3. Extant Responses

A natural response to the indeterminacy problem, as I've formulated it, is to reject one of the foregoing claims. Which one? We can immediately set aside solutions that reject either of the auxiliary assumptions (the Identity-Implementation Link and Successful Explanations). All parties to the current debate accept that computational systems and states are taxonomized with respect to the computations they implement. Similarly, few philosophers nowadays doubt that there are successful computational explanations. Although solutions that reject either of these principles are not entirely off limits, they incur substantial costs and should be pursued only as a last resort.

This leaves Simultaneous Implementation and the Uniqueness Condition. This section surveys the main strategies for rejecting each of these claims, and argues that they are

9 Fresco, Copeland, and Wolf (2021) make a similar point in the context of natural computing systems. They suggest that it is evolutionarily beneficial if a given neural component can subserve multiple distinct neural processes.

unsatisfactory. My overall complaint is that these solutions either distort or otherwise fail to capture certain important aspects of computational practice. However, I should emphasize from the start that I will not attempt to show that these strategies *cannot* capture these aspects once and for all. My aims are more modest. I take the considerations surveyed below to shift the burden onto those who wish to reject either of these principles: they must show that their accounts have the resources to accommodate the problem cases I will identify, or they must provide grounds for denying that they must accommodate them in the first place.

3.1 The anti-individualist response: reject the Uniqueness Condition

A first line of response accepts that a physical system might simultaneously implement different computations, but denies that this undermines computational explanation. Here is the rough idea. We describe systems computationally in order to explain a specific phenomenon. A computational explanation is successful if it reveals how a system comes to exhibit that phenomenon. This is compatible with the system simultaneously satisfying other computational descriptions, appropriate for other explanatory targets. For this reason, computational explanation does not require that a system implement a single computation *full stop*. Rather, it requires only that a system implement a single computation responsible for the phenomena we wish to explain.

This response has been pursued primarily by anti-individualists about computation, who cash it out in terms of three more specific claims: (1) that the phenomenon relevant for assessing a system's computational identity in a given context is the task it performs in that context; (2) that a system always performs a single task in a given context; and (3) that the task a system performs in a given context is determined by its relationship to its broader environment. The primary point of disagreement between anti-individualists concerns the nature of this relationship: is it best understood semantically or non-semantically? This can be framed as a disagreement about the nature of computational tasks: are computational tasks always identified in terms of (wide) semantic properties, or not? Semantic anti-individualists say yes, while non-semantic anti-individualists say no.

Semantic views start from the observation that computational tasks are often described in terms of mathematical or environmental entities. Standard examples are computing addition over numbers or computing zero-crossings of environmental luminance levels. On the face of it, performing these tasks requires a system to represent the entities in question — numbers and environmental properties, respectively.[10] Accordingly, a system performs these computations only if it has appropriate semantic properties. Non-semantic views, by contrast, hold that the relevant relational properties are not semantic properties, but wide functional properties. Gualtiero Piccinini is the foremost defender of this view. On his version, these wide functional properties comprise a system's immediate mechanistic context. This includes things like: "the relation between the forces exerted on input devices (such as keyboards) and the signals relayed by input devices to the computing components, and on the other hand, the relation between the computing components' outputs and the signals released by the output devices)" (Piccinini 2008, 43, 221).[11]

Next I will consider two objections to anti-individualist views. The first challenges both semantic and non-semantic views, while the second challenges Piccinini's non-semantic view in particular.

Objection 1: some tasks depend only on intrinsic properties

Anti-individualists claim that the task performed by a system is always determined by the system's relationship to its broader environment. But is this true? Consider the task of sorting a list of binary strings. It is straightforward to describe a Turing machine computation *M* for this task. A physical system implements *M* if it transforms lists of physical strings in the right way. Yet, as I will suggest, implementing this computation plausibly involves neither semantic nor wide functional properties. If this is right, then this computation is a *prima facie* counterexample to the claim that all computational tasks — and thereby all correct computational descriptions — depend on a system's relational properties.

To begin, observe that a system can perform this task even if it lacks semantic properties altogether. Piccinini (2008) plausibly argues that implementing string-theoretic computations

10 This has been challenged, but I will grant this assumption here. See (Egan 1995) and (Peacocke 1999) for discussion.

11 Fresco (2021) defends a 'long-arm' anti-individualist view, in contrast to Piccinini's 'short-arm' view. Although I focus on Piccinini's version here, the objections raised below apply equally to both.

involves only functional, non-semantic properties of a system. This suggests that the semantic view fails to capture *M*'s implementation conditions. One obvious counter-maneuver the semanticist might make is to argue that these conditions involve semantic properties only tacitly, making them easily overlooked. Perhaps when a state realizes a certain string-theoretic type it represents that type. Or perhaps in order to compute a sorting function it must represent one string as coming before or after another in the ordering. Whether such responses succeed remains to be seen. For my own part, I am skeptical that they can be executed without seriously distorting computational practice.[12]

What about the wide functional account? Whether a system implements *M* sometimes depends solely on its intrinsic functional features. The most straightforward case is a contemporary digital computer. For such devices, there is a reasonably sharp distinction between core computing components (e.g. the CPU and memory) and I/O devices (e.g. the keyboard and display). For the wide functionalist, the latter components comprise the device's immediate mechanistic context. Whether the device implements *M* is thus partly determined by the state of those components.

But suppose we hold fixed the internal dynamics of the CPU and memory, while varying the state of the input and output devices. I take this to be nomologically possible. For instance, it could be the result of a manufacturing error which causes input and output signals to become momentarily scrambled, but which leaves the internal dynamics of the CPU intact. In this scenario, I claim that we would still judge that the CPU performs the sorting task. Because the internal dynamics of the device remain unchanged, the internal physical states would appear to go through the same patterns of activity as in a typical, unscrambled case. It is thus plausible to think that the device implements *M* in this case as well, even though it bears a warped relationship to its broader mechanistic context.

It might be objected that in this scenario the device *doesn't* perform the sorting task, but performs some other task or even no task at all. I have two responses to this objection. First, this response sits awkwardly with certain aspects of computational practice. I/O glitches are an unfortunate fact of any computer scientist's life. In such cases, upon inspecting one's code and

12 See (Rescorla 2014) for other examples in this vein.

determining that it compiles properly and runs correctly, it is natural to say, not that the device fails to sort, but rather that it correctly sorts yet fails to communicate this computation to the user.

Second, even if we grant that the device performs a different task, at best this objection shows that a system's computational identity is not always determined by the task it performs. The scrambled device goes through *some* sequence of internal states accurately described by *M*. This sequence exhibits certain properties that may be of interest, for instance having to do with running time or memory usage. These properties can legitimately be explained by appeal to *M*. Thus, for the purposes of explaining *these* specific phenomena, it is plausible to think that *M* constitutes the computational identity of the device.

To be clear: I do not deny that whether a system performs this computation depends *causally* on its input (and perhaps output) transducers. It plausibly does, at least in ordinary cases. Nor do I deny that *some* string-theoretic computations might depend constitutively on a system's wide functional properties. I grant that they might, and I will consider an example in due course. Nor do I even deny that sometimes the phenomenon relevant for determining computational identity is the task a system performs. Rather, the claims I reject are (1) that whether a device implements *M* always depends constitutively on its wide functional properties, and (2) that a system's computational identity is always determined by the task it performs in a given context.

Objection 2: some tasks are characterized semantically

Next I consider a semantically characterized computation that poses a problem for Piccinini's wide functionalism in particular. Other philosophers have emphasized the importance of semantically characterized computations for descriptive practice in computer and cognitive science (e.g. Shagrir 2001; Rescorla 2017; Lee 2021). And Rescorla (2013) argues that in computer science physical systems are sometimes computationally distinguished in terms of their semantic properties. Here I want to throw another consideration into the mix by considering a case in which two systems ought to be computationally type *identified* in virtue of their semantic properties. As I shall argue, certain explanatory benefits accrue from describing systems this way, yet it is unclear how to account for these benefits in non-semantic terms.

Consider the task of sorting a list of numbers (rather than strings). It is straightforward enough to describe an abstract register machine computation *R* for this task. Abstract register machine computations differ from string-theoretic computations in that they take operations on numbers, rather than strings, as computationally primitive. *R* is thus a numerical computation, in the sense introduced in section 2. Moreover, it is plausible to think that implementing *R* requires a system to have certain semantic properties, such as representing particular numbers (Rescorla 2013).

Descriptions couched in terms of *R* allow us to abstract away from details which are irrelevant for certain explanatory purposes. For example, consider two microprocessors that realize *R*, one of which uses binary notation and another of which uses ternary notation.[13] Describing these systems as performing computations over lists of numbers (rather than lists of strings) allows us to capture important similarities between their properties and behaviour, the most obvious of which is that under a semantic description the two devices can be seen to perform the same sorting task over numbers. Insofar as the numerical sorting task performed by these systems is relevant for determining their computational identity, under the semantic task description sanctioned by *R* the devices are computationally identical.

This case poses a dilemma for the wide functionalist. Horn one: they can accept that the devices are computationally identical, but try to account for this fact in non-semantic terms. This is easier said than done. Because the systems use different notations, their internal functional dynamics differ. Moreover, the broader environments of which they are parts must presumably be sensitive to their specific notational schemes too. Yet if computational tasks depend on their wide functional properties, then it would appear that these systems perform different tasks: one performs a task on binary strings, while the other performs a task on ternary strings.

To get around this, the wide functionalist might go even wider. For instance, they might appeal to the ways that users respond to or interact with the devices. This move is suggested by some of Piccinini's remarks (2015, 44 fn. 12). Here the idea would be to appeal to the fact that

13 There are a few interesting instances of the ternary computers in the history of computer science. Perhaps the most notable are the Setun machines, developed in the 1950s and 60s. For discussion see (Brusentsov and Ramil Alvarez 2011) and references therein.

users interact with the devices in similar ways to ground the fact that these devices are computationally equivalent. But what are these 'similar ways'? This must presumably be spelled out in a way that *doesn't* rely on the way that users *interpret* these devices or otherwise imbue them with content, on pain of collapsing into the semantic view. This is a non-trivial task, made more difficult by the fact that semantic properties very plausibly supervene on such broad environmental relations between a system and its users (Rescorla 2013; Shagrir 2020). It remains to be seen whether the wide functionalist can accomplish this task.

Horn two: the wide functionalist might instead hold that the binary and ternary devices are computationally distinct. This seems to be Piccinini's preferred strategy. He points out that any semantic task description depends on a more basic non-semantic task description (Piccinini 2015, 33–36). We cannot describe the devices as sorting lists of *numbers* unless we can antecedently describe them as sorting lists of (non-semantically characterized) *strings*. But it is this more basic, non-semantic description that is most relevant for questions about computational identity. This is because (a) the relevant description is the one that is required for computational explanation, and (b) on Piccinini's view computational explanation is a species of mechanistic explanation (Piccinini 2015, ch. 5). In particular, the non-semantic description is what will ultimately feature in a full-blown mechanistic explanation of a system. This includes, among other things, a specification of "the notation being used, the algorithm followed, [and] the architecture that executes the algorithm" (Piccinini 2015, 39).

On this picture, "computing systems and their states have non-semantic identity conditions" (Piccinini 2015, 49). It would thus seem that the proper attitude towards the binary and ternary realizations of R is that they are computationally distinct, owing to their different notation. Of course, this response does not deny that the binary and ternary devices and their states can be described in semantic terms for certain purposes. As Piccinini points out, "once computational states are individuated non-semantically, semantic interpretations may (or may not) be assigned to them" (Piccinini 2015, 49).[14]

What should we make of this response? Piccinini certainly has a point. There may be computation without representation, but there is no computation without notation, as it were. But

14 Thanks to an anonymous referee for urging me to clarify this point.

this doesn't automatically show that the descriptions relevant for computational identity are always non-semantic. That claim follows only under the assumption that computational explanation (properly so called) is full-blown mechanistic explanation. But as I will suggest next, some computational explanations succeed only by *ignoring* certain mechanistic details such as notation. And for such explanations it is natural to computationally type identify physical systems and their states in semantic terms.

The explanations I have in mind come from algorithmic analysis, the branch of computer science that investigates the resource requirements of various algorithms. Computer scientists make a few simplifying assumptions so that the results of such analyses apply as widely as possible. One is to focus on asymptotic, as opposed to exact, running time. Another is to describe algorithms in a way that is largely machine-independent, abstracting away from details such as notation or architecture (Cormen et al. 2001, 21–23). This is usually achieved by describing algorithms as operating on numbers, rather than strings. This allows computer scientists to assume that primitive computational steps (e.g. basic logical or arithmetical operations) operate in constant time. Overall, these assumptions ensure that a given analysis can be used to explain the performance of a wide variety of physical systems, at least in many cases, regardless of their mechanistic details.

To illustrate, suppose that *R* employs Insertion Sort, a well-known sorting algorithm. Given a list of numbers of length *n*, Insertion Sort sorts in $\Theta(n^2)$ primitive computational steps in the worst case.[15] Recall that primitive steps here are characterized numerically, rather than string-theoretically. This is important, because the binary and ternary devices presumably differ in the exact amount of time required to carry out a primitive numerical operation, owing to their different notations. Nevertheless, such details are irrelevant for explaining facts about the asymptotic running time of the devices. Thus, we can appeal to *R* to explain why both devices sort on average in a certain amount of time: the reason is that they both implement a specific $\Theta(n^2)$ sorting computation over *numbers*.

The crucial point is that this explanation would not be improved by adding in mechanistic details about notation. Indeed, accounting for those details would obscure the fact that the two

15 (Cormen et al. 2001, 26). Crudely, the running time of an algorithm is $\Theta(f)$ for some function f if for sufficiently large inputs the running time is bounded above and below by f, up to some constant factor.

devices have similar asymptotic running times. This similarity emerges only under a notation-indifferent description, such as that sanctioned by *R*. Indeed, this is an instance of a familiar general point: for certain explanatory purposes, less (detail) is more (cf. Potochnik 2017, ch. 2). Thus, relative to the goal of explaining the similar asymptotic running time of these devices, we ought to describe them semantically, as sorting lists of numbers. And under this descriptive idiom, the devices are computationally identical.

Of course, this is consistent with the claim that non-semantic, mechanistic descriptions are important for *other* explanatory purposes. If we wish to explain their exact running times, down to the number of clock cycles used, mechanistic details are undoubtedly relevant. But there is little reason to think that this goal occupies an explanatorily privileged position. Relative to one explanatory aim, we ought to describe the binary and ternary devices as computationally identical; relative to another, computationally distinct. There would seem to be little pressure from within computational practice for regarding either description as capturing their 'true' computational identity.

3.2 The individualist response: reject Simultaneous Implementation

Recently, some mechanists have attempted to improve on Piccinini's response. According to Dewhurst (2018b) and Coelho Mollo (2018) we should reject Simultaneous Implementation, not the Uniqueness Condition. These philosophers advance individualistic versions of the mechanistic view, on which the uniquely correct computational description of a system is determined wholly by its local, intrinsic properties. They disagree, however, about which intrinsic properties matter.

Dewhurst argues that the computational identity of a system is determined by its intrinsic physical properties. On this approach, the computational identity of Shagrir's tristable gate is given, in effect, by Table 3. Only devices that transform the same voltage levels, in the same way, are computationally identical with the gate, and any system that differs in either respect is computationally distinct. However, this proposal faces a serious challenge. The problem is that it entails that systems which differ even a little in respect of their intrinsic physical features are computationally distinct. While Dewhurst is upfront that this is a cost (2018b, 110–1), it is far from clear that the price is worth paying (Fresco and Milkowski 2021; Shagrir 2020).

Coelho Mollo (2018) offers a more promising approach that retains the spirit of Dewhurst's proposal. Coelho Mollo suggests that a system's computational identity is fixed not by its intrinsic physical properties, but rather by its intrinsic functional organization. On this approach, particular voltage levels don't impact a system's computational identity, but its overall functional organization does. In particular, any device that doesn't exhibit the same overall functional pattern is computationally distinct from Shagrir's gate.[16]

Coelho Mollo's view nicely captures the implementation conditions for certain computations, but I do not believe that it is plausible as a general account of computational identity. It struggles with two kinds of case in particular.

Objection 1: some computations depend on wide functional properties

The first sort of case concerns wide functional properties. On Coelho Mollo's view, any two systems that differ in respect of their overall functional profile are computationally distinct (Coelho Mollo 2018, 3494 fn. 20). The trouble is that functionally distinct systems are sometimes computationally identical.

Consider a system that implements the Turing machine *M* described above for sorting lists of binary strings. Suppose that this device is composed of bistable circuits (e.g. like Table 1), but that one of the AND gates in this device malfunctions and the only available replacement is a tristable circuit (e.g. like Table 3). Even though the tristable circuit registers the difference between 0-2.5V and 2.5-5V, the other circuits in the device do not, and so the tristable circuit can be safely swapped in. Because of this, the system will exhibit the same overall pattern of string manipulation as before the swap, and so will continue to implement the string-theoretic computation *M*. Insofar as both gates make the same overall contribution to the string-theoretic sorting task performed by the system, it is thus plausible — contra Coelho Mollo's claim — to think that in this context the tristable AND gate is computationally identical to the bistable AND gate (cf. Schiller 2018).

16 This view resembles more traditional causal views of computation (e.g. Chalmers 1996a), but goes beyond them by endorsing the empirical claim that functionally characterized computations are realized by mechanisms.

It is not immediately clear that this worry is insurmountable, however. Elsewhere, Coelho Mollo points out that a functional decomposition of a system always takes place in light of a particular target capacity or teleofunction (2018, 3495). Functional differences that are irrelevant to these capacities may thus be ignored for the purposes of computational identification. Given this, one might say that insofar as the tristable circuit is located in a larger mechanism sensitive only to the difference between 0-5V and 5-10V, the difference between 0-2.5V and 2.5-5V is functionally irrelevant. Thus, in this context, the functional organization of the tristable gate is identical to that of the bistable gate after all, in which case it would follow that the two are computationally identical.

This seems to me a reasonable analysis of the situation. But Coelho Mollo rejects this manoeuvre, insisting that the tristable and bistable gates are never computationally identical: "devices that differ in the number of their stable states do not count as computationally equivalent … regardless of whether those functional differences are exploited … by the overall computational system in specific computations" (Coelho Mollo 2018, 3496, fn. 22). However, it is unclear why we should insist in this case the bistable and tristable gates are computationally distinct. Indeed, this seems to be a case in which wide functional considerations *do* impact a circuit's computational identity (cf. Piccinini 2015, 43–44). Even if the bistable and tristable circuits differ in respect of their overall computational potential, in this specific instance they certainly appear to be computationally identical.

Objection 2: some tasks are characterized semantically (redux)

The second sort of case involves semantically laden computations, like the register machine *R* for sorting numbers. Coelho Mollo offers an interesting new response to this sort of case: he denies that a theory of computational identity must capture such computations in the first place. This is because such computations concern 'logical' or 'mathematical' identity, which, strictly speaking, do not concern computational identity properly so-called (Dewhurst 2018b, 110; cf. Coelho Mollo 2018, 3495). It is thus no strike against a non-semantic view that it fails to capture *R*, for it simply falls outside the scope of the theory.

Coelho Mollo illustrates this response with the help of a binary digital circuit. Consider

Table 8, which represents the functional profile of the circuit in Table 1. In Table 8, 'EC1' and 'EC2' are arbitrary labels for digit types. As Coelho Mollo points out, the description in table 8 underdetermines the logical function computed by the circuit. If the EC1 represents logical '1' and EC2 represents logical '0', it computes AND. Under the reverse assignment, it computes OR. And nothing about the intrinsic functional features of the circuit tells us which it computes. According to Coelho Mollo, this is as it should be:

> Computational individuation … leaves logical individuation indeterminate. This is a welcome result, since … logical individuation is at least one step above computational individuation … what [Oron Shagrir and Mark Sprevak] point out is correct: the mechanistic view does not have the tools to distinguish between dual logic gates. However, such a feat is not something we should be asking of a theory of computational individuation, for computational individuation takes place below the level of logical functions. (Coelho Mollo 2018, 3495)

I have two concerns about this manoeuvre. First, it turns crucially on the distinction between 'computational' versus 'logical' individuation. But what motivates this distinction? True, the computational sciences register a distinction between string-theoretic and numerical computations, as noted in Section 2. But computational scientists treat both of these as furnishing different but equally legitimate notions of *computational* individuation.

Moreover, there are general theoretical reasons for treating both as genuine kinds of computational individuation. Describing systems computationally allows us to explain them computationally. Paradigmatically, computational explanation reveals how some phenomenon is produced by a routine step-by-step process.[17] If an explanation of some phenomenon fits this pattern, it thereby constitutes a computational explanation of that phenomenon. This is so whether the phenomenon of interest is characterized in narrow functional, wide functional, or semantic terms.

To illustrate, consider the task of sorting uninterpreted strings versus the task of sorting interpreted strings. We've seen that an explanation of the former task may be couched wholly in

17 At least to a very rough first approximation. In saying this I do not claim that *all* computational explanations identify routine, step-by-step processes. I merely claim that it is sufficient for a given explanation to constitute a computational explanation that it take this form.

narrow functional terms, while an explanation of the latter must cite specific semantic properties. Despite this difference, both tasks can be explained by identifying routine, step-by-step sorting processes. There is thus arguably a single, unified kind of explanation at work in both cases. Insofar as computational explanation depends upon computational individuation, it seems that Coelho Mollo's 'logical' individuation is in the end just another kind of computational individuation.

The second concern is that this move doesn't solve the indeterminacy problem — at best, it relocates it. Even if we grant that computational identity properly so-called does not concern the logical or mathematical function computed by a physical system, the problem reemerges in the guise of 'logical' indeterminacy. For we are still owed an account of what makes it the case that a physical system computes one logical function rather than another. Our goal, recall, is to account for taxonomic practice in the computational sciences. Computational identifications that appeal to the logical operations or mathematical functions computed by physical systems are a central part of this practice. Insofar as Coelho Mollo's response fails to address indeterminacy of logical function, it thus loses sight of the main issue. If we wish to disarm indeterminacy worries about computational practice as it stands, we must address questions of 'logical' indeterminacy as well.[18]

3.3 Taking Stock

Where does this leave us? I've argued that extant responses struggle to capture the identity/implementation conditions of at least some computations:

- Anti-individualistic accounts struggle to capture string-theoretic computations with individualistic implementation conditions, such as *M*.
- Non-semantic accounts (both individualistic and anti-individualistic) struggle to capture certain numerical computations with partly semantic implementation conditions, such as *R*.
- Non-semantic individualistic accounts struggle to capture string-theoretic computations when those computations depend on wide functional properties. This typically happens

18 But see (Papayannopoulos, Fresco, and Shagrir, forthcoming) for an alternative take on the situation.

when a device computes a non-maximal task given its broader context.

The moral is that an adequate response should have the resources to capture each of these different kinds of computation. I develop such a response next.

4. A New Solution

So far, I have framed the indeterminacy problem as the problem of reconciling four *prima facie* plausible but jointly inconsistent principles. However, I will argue next that the inconsistency between these principles is illusory. Under a proper understanding of computational implementation, no contradiction arises.

My solution has two components. The first appeals to the relativity of implementation: the view that a physical system implements a computation only relative to a specific pairing of abstract mathematical values with physical components. These pairings are called 'labelling schemes' (Copeland 1996; Chalmers 1996a). Although a system may implement different computations relative to different labelling schemes, relative to a fixed scheme, it will typically implement a single computation. The second part of my response argues that the Uniqueness Condition will be satisfied if computational explanations are relativized to specific labelling schemes. And this will be so even if the system simultaneously implements other computations, relative to other schemes. I will take each part in turn.

4.1 The relativity of implementation

Labelling schemes capture how mathematical computations are applied to physical systems. This involves two more specific tasks: (1) grouping microphysical states, inputs, and outputs, into state, input, and output types, and (2) assigning an abstract 'label', such as a syntactic type, truth value, or natural number, to these types. Both of these tasks can be described formally as functions, which I will call the 'grouping' and 'assignment' functions, respectively. A labelling scheme is then the composition of a grouping and labelling function.

The grouping function maps microphysical states to state types, so that microphysical states mapping to identical state types are thereby grouped together. How states are grouped depends on their properties. For instance, some schemes considered in connection with Shagrir's tristable circuit group microphysical states according to their electromagnetic properties, while others might group according to functional or semantic properties. The assignment function takes these state types and maps them to the states of some computation. This captures the assignment of abstract labels to those types.

Indeterminacy phenomena can be generated both at the level of the grouping function (by varying the grouping of microphysical states into state types) and at the level of the assignment function (by varying the labelling of state types). Both possibilities are illustrated by Shagrir's gate. For the former, notice that we can group microphysical states into two state types (e.g., 0-5V vs 5-10V) or three (e.g., 0-2.5V vs 2.5-5V vs 5-10V). For the latter, notice that even a fixed grouping underdetermines which logical values we ought to assign to these state types. A two-state grouping can be assigned either AND or OR, for instance. Because an adequate solution to the indeterminacy of computation ought to address indeterminacy both at the level of grouping and at the level of assignments, taking labelling schemes as the composition of a grouping and an assignment function thus allows us to address both aspects of indeterminacy at once.

Not everyone will agree with this approach. Some philosophers hold that the assignment function is less important than the grouping function. They claim that how we choose to label state types is arbitrary, a matter of free choice (Coelho Mollo 2018, 3494; Piccinini 2020b, 153). According to these philosophers, to solve the indeterminacy problem, we need only focus on the grouping function. This is more plausible in some cases than others. Perhaps in the case of a single circuit considered in isolation (e.g. Table 1), it does not matter which state type is labelled '1' or '0'. But the choice of label is arguably more important when the circuit is embedded in a more complex system. This is because poorly chosen labels can *misdescribe* the computation being performed. Suppose the gate is located in a broader system in such a way that it has determinate semantic content. And suppose the 5-10V state represents logical 1 while 0-5V represents logical 0. In this case, if we describe the gate using a scheme that assigns 0 to 5-10V and 1 to 0-5V, we mischaracterize the computation performed by this device. For, while the device actually computes AND in this scenario, we describe it as computing OR. Thus to capture the full range of computational phenomena, we arguably need to account for both groupings and assignments.

When a system implements a computation, it does so under a specific labelling scheme. Under a different labelling scheme, it may implement a different computation. Given this, we should not say that physical systems implement computations *simpliciter*. Rather, we should say that physical systems implement computations *relative* to specific labelling schemes. In this respect, describing physical systems computationally closely resembles the more general practice of describing systems mathematically. Applying some branch of mathematics to a physical system — such as arithmetic, real analysis, or, in our case, computability theory — requires that we specify how that branch is to be applied. Finite cardinals apply relative to sortal concepts. Reals apply relative to choices of unit magnitude. And computations apply relative to labelling schemes.[19]

Importantly, relative to a specific labelling scheme, a system typically implements a single, determinate computation. While it is in principle possible to devise schemes that fail to single out a particular computation, none of the schemes considered in connection with the indeterminacy of computation are indeterminate in *this* sense. Return to Shagrir's gate. Each of the different computations implemented by the gate depends on a different grouping of states into state types, or on a different labelling of state types. Relative to a fixed scheme, however, the gate implements a single, unique computation. That is, *relative to a specific labelling scheme* the computational identity of a physical system is as determinate as one could reasonably want.[20]

I can now state the first part of my solution. I propose that we accept Simultaneous Implementation, in the sense that a system may simultaneously implement different computations relative to different labelling schemes. The next task is to argue that accepting Simultaneous Implementation in this sense does not undermine computational explanation.[21]

19 Blackmon (2013) also emphasizes the relativity of implementation. For connections between the applicability of mathematics and computational implementation see (Matthews and Dresner 2017) and (Schweizer 2019a).

20 At least, up to any indeterminacy in the properties cited by the specific scheme. Although this is itself a serious issue, I will not pursue it here.

21 To be clear, I do not claim that a system's computational identity is determined by the *total* set of computations it implements (as in Milkowski 2013). Rather, I claim that computational identity is determined only relative to a fixed labelling scheme. Relative to a fixed scheme, a system's computational identity is determined by the computation implemented under that scheme.

4.2 Relativity and computational explanation

To begin, recall that computational description is useful largely because it allows us to reason about a system while abstracting away from irrelevant physical details. Describing Shagrir's circuit as an AND gate, for instance, allows us to ignore physical idiosyncrasies that make no difference to its overall pattern of behavior. We can explain that it outputs '0' because it was given '1' and '0' as input without worrying about whether the device tokens '1' because it has a voltage level of 6V or because it has a voltage level of 7V, because these physical states are indistinguishable from the perspective of a labelling scheme relative to which the gate computes AND.

This much is uncontroversial. The important point is that these benefits are not undermined by the fact that the gate simultaneously implements OR under a *different* labelling scheme. It is still true that it will output '0' when given '1' and '0' relative to the AND scheme, even if it is simultaneously true that, relative to the OR scheme, the same pattern of physical activity would be described as outputting '1' given '1' and '0'. In general, there is no incompatibility between the claim that a system implements one computation relative to one labelling scheme while simultaneously implementing a distinct computation relative to another labelling scheme.[22] For this reason, we can explain a device's behavior by citing the fact that it implements AND (say), as long as we bear in mind the labelling scheme that sanctions this particular computational description.

Furthermore, there can be genuine computational contrasts relative to specific labelling schemes. This allows us to capture the contrastive character of many computational explanations, noted in section 2. For it is false that Shagrir's gate implements OR relative to the scheme under which it implements AND. We can thus explain that the gate outputs '0' *rather than* '1' when given '1' and '0' as input because it computes AND *rather than* OR. This is because, relative to the AND-scheme, the device really does output '0', not '1', when given '1' and '0' as input.

It should now be apparent that the inconsistency identified in section 2 does not arise on a

22 Although I will not argue for it here, this appears to be a general feature of applications of mathematics. To take Frege's (1884) example, a single expanse of matter may simultaneously constitute one deck and fifty-two cards. There is no inconsistency in describing the matter simultaneously as *one* and as *fifty-two*, because these descriptions apply relative to the concepts DECK and CARD, respectively.

relativistic conception of implementation. According to the Uniqueness Condition, computational explanation requires a system to fall under a unique computational type. But relative to an explanatorily salient labelling scheme, a system typically will implement a unique computation and thereby fall under a single computational type. Relative to a specific scheme, we can thus respect the Uniqueness Condition while recognizing that, relative to other schemes, a system might implement other computations.

Moreover, in light of this solution, there is arguably little need to identify any one way of regarding a system computationally as capturing its 'true' computational identity in some absolute sense. Rather, we should ask which way of regarding it computationally best serves our purposes, given our explanatory goals and interests. It may be that, for certain theoretical purposes, we should regard a system one way (e.g., as computing AND), whereas for other purposes we should regard it another way (e.g., as computing OR). Relative to appropriate labelling schemes, each of these may be a legitimate — indeed, true — computational description of that system. But neither description captures the system's computational identity in some further global sense, divorced from our specific explanatory goals and interests.

Finally, I should emphasize that computability theory unifies these different ways of regarding systems computationally.[23] That theory provides a formal, domain-neutral characterization of computational processes. Applications of this theory to particular physical systems reveal how a phenomenon is produced by such a process. As with mathematical theories generally, computability theory can be applied in different ways, relative to different labelling schemes. Nevertheless, insofar as a single, unified mathematical theory is applied in structurally similar ways, it is plausible to think that such applications furnish a common kind of description and explanation. Thus, my solution is consistent with the claim that the various different ways of identifying and explaining systems are nonetheless ways of identifying and explaining systems *computationally*.

4.3 Indeterminacy vs underdetermination

One might worry that this solution merely relocates the indeterminacy problem. Even if

23 In this respect, computability theory and related branches of theoretical computer science play a familiar unifying role encountered elsewhere in science (Kitcher 1981).

computational identity is determinate relative to a labelling scheme, we are still owed an account of what makes a labelling scheme explanatorily relevant in a particular case. Absent an answer to this question, are we not left with a new indeterminacy problem, now pitched at the level of labelling schemes?[24]

In response to this worry, I would point out that this problem is really quite different from the original indeterminacy worry. To see this, recall the proposals from section 3. Those proposals attempted to solve the problem by tying computational identity to specific kinds of properties. The assumption guiding this approach is that the only way to satisfy the Uniqueness Condition is to identify a fixed class of properties which determine a system's computational identity in *every* theoretical context. But in my view, this assumption is unwarranted. Relative to a well-defined labelling scheme, *whatever* properties it cites, a system's computational identity is fully determinate, in which case the Uniqueness Condition will be satisfied relative to that scheme. This is consistent with different schemes being appropriate in different theoretical contexts, for different descriptive or explanatory aims.

This way of approaching the indeterminacy problem reveals that what we *thought* was a metaphysical problem about computational identity turns out to be an epistemic problem about explanatory relevance. It is, moreover, a quite familiar issue. For the question of what makes a given computational description of a system explanatorily relevant is just an instance of the more general question of what makes a given *scientific* or *mathematical* description explanatorily relevant. This is of course a difficult question, among the hardest in the philosophy of science. But there is little reason to think that we cannot make headway on it. For this reason, trading the metaphysical problem for an epistemic one leaves us at least no worse off than we were before.

What makes a particular labelling scheme explanatorily relevant will tend to be highly context-sensitive, depending on the properties and behavior of the system under consideration as well as our explanatory goals and interests (Schweizer and Jablonski 2013; Lee 2021). How these factors interact is a complicated affair, and I am somewhat skeptical that there is an illuminating general story to be told here. Nevertheless, this skepticism flagged, a few points are worth mentioning.

24 Thanks to an anonymous referee for pressing me on this point.

First, because labelling schemes map physical components to abstract values, which schemes apply to a system will depend on the system's properties. Not just anything goes. A system that lacks semantic properties will, quite obviously, not be explained in terms of a labelling scheme that groups microphysical states according to their semantic features. Which schemes are available thus varies on a case by case basis, and will plausibly depend on intrinsic features of a system as well as its relation to the broader environment. In general, identifying schemes applicable to a system or class of systems will be partly an empirical matter.

Second, although systems will typically fall under a variety of different labelling schemes at once, the range of which must be determined empirically, specific explanatory tasks typically require that we focus on a specific scheme or kind of scheme. This point is usefully illustrated by the cases considered in section 3:

1. Labelling schemes that group physical states according to their semantic features are appropriate for explaining how a system performs a semantically characterized task. The most obvious such tasks include computing functions over non-string theoretic entities, such as numbers or truth-values. Semantic schemes are also appropriate for capturing coarse-grained similarities between systems that disappear when we consider their functional or mechanistic properties alone. Facts about asymptotic running time are one example, although there may be others.
2. Labelling schemes that group physical states according to their non-semantic features are appropriate for explaining non-semantically characterized tasks, such as computing string-theoretic functions. They are also appropriate for explaining highly specific facts about exact running time or memory usage.
3. Schemes that cite wide functional properties are typically required when a system's broader context is insensitive to certain functional differences registered by the system itself. The example we saw concerned bistable and tristable gates, both of which may perform the same binary digital computation when situated in a larger context which is sensitive to two digits only.

Of course, I am happy to grant that in some cases we may not have enough evidence to choose between different computational descriptions, sanctioned by different labelling schemes. While this situation is less likely to occur for artificial computing systems, it is perhaps the norm for complex natural computing systems such as the brain. Determining which scheme is explanatorily most appropriate for neural or cognitive phenomena is a non-trivial problem. That's putting it lightly. But this is hardly a problem unique to computation. Just about *any* scientific description of the world — computational or otherwise — is underdetermined by the available evidence. Thus, I would emphasize again that there is no special problem for computation here, only an instance of a quite familiar general phenomenon.

4.4 Comparisons and elaborations

I will round off the discussion by situating my view with respect to a few others in the literature.

Extant Responses

My view resembles the responses in section 3 in certain respects, but departs from them at crucial junctures as well. Each of those views is partially correct, because physical systems *can* be computationally individuated with respect to their narrow functional, wide functional, or semantic properties. However, I do not hold that a system's computational identity is always determined by its narrow functional, wide functional, or wide semantic properties, respectively. Rather, I hold that each of these can determine computational identity in different contexts, for different explanatory ends.

Of these views, mine is perhaps closest to computational anti-individualism. For I agree that computational explanation does not require a system to implement a single computation at a time. However, because these views do not endorse a relativistic conception of implementation, they are forced to reject the Uniqueness Condition. This is a cost, because the condition is *prima facie* attractive. My view, by contrast, is able to endorse the Uniqueness Condition. Point for me.

The computational individualist pays a similar price, because they are forced to reject Simultaneous Implementation. Once again, insofar as this principle is *prima facie* plausible, it is a mark in favour of my account that I can endorse it. Moreover, we saw that the computational individualist is forced to distinguish 'computational' from 'logical' individuation. This too is a cost, insofar as this distinction has little grounding in either the descriptive or explanatory aspects of computational practice. No such distinction is forced on my account. Given the unifying role of computability theory on my account, I am able to capture the sense in which narrow functional, wide functional, and semantic descriptions/explanations may all, in certain cases, constitute computational descriptions/explanations.

Computational perspectivalism

My account foregrounds the role of explanatory contexts in determining an explanatorily relevant labelling scheme. In this respect it resembles certain perspectivalist views in the philosophy of science. Over the next few paragraphs, I will argue that my view is perspectival in only a weak epistemic sense.[25]

To begin, we can draw a rough and ready distinction between *ontic* and *epistemic* varieties of perspectivalism. Ontic perspectivalism about some subject matter holds that there are no non-perspectival facts about that subject matter. Epistemic perspectivalism, by contrast, holds that our knowledge of some subject matter is crucially mediated by factors such as our explanatory goals and interests, scientific models and theories, or instrumentation, which constitute our epistemic perspective on the world.

To make matters concrete, I will contrast my view with two recent proposals, due to Schweizer (2019b) and Dewhurst (2018a). Schweizer holds that physical computation "is founded upon an observer-dependent act of ascription" so that "there is no deep or metaphysically grounded fact about whether or not a physical system 'really' implements a given computational formalism" over and above facts about how agents choose to interpret that system (Schweizer 2019a, 31). Dewhurst offers a more attenuated view, couched in terms of the mechanistic account of computation. On Dewhurst's view, "what it means for a mechanism to perform the function of computing is to possess the right kind of physical structure to be interpreted as performing this function from an explanatory perspective" (Dewhurst 2018a, 581). Differences aside, these views hold that a system computes if an agent, taking up a particular

25 Scientific perspectivalism is itself a large, challenging topic, and I cannot hope to do justice to all the subtleties involved here. For more, see (Chakravartty 2010), (Massimi 2018), and (Massimi and McCoy 2019), among many others. For detailed discussion of computational perspectivalism in particular see (Coelho Mollo 2019).

explanatory perspective, in some sense interprets that system as computing. Thus, these views arguably fall into the ontic perspectivalist camp with respect to physical computation.[26]

In contradistinction to these views, I do not hold that computation depends upon 'observer-dependent acts of ascription' or on a system being interpreted in a particular way. Rather, I hold that it depends upon a labelling scheme. Labelling schemes are functions from physical states to abstract labels — a kind of mathematical object. As with mathematical objects more broadly, labelling schemes in this sense arguably exist whether anyone ascribes them to anything.[27] Furthermore, the fact that a system implements a computation relative to a labelling scheme does not on its own entail that computational facts are perspective-dependent. Indeed, given an explanatorily adequate labelling scheme it follows by inference to the best explanation that the computational features ascribed under that scheme are real, perspective-independent properties of the system in question. This illustrates that my view is certainly compatible with (although not necessarily committed to) a realist, perspective-independent attitude towards physical computation.

Thus although I agree that computational ascriptions must be made from *some* explanatory perspective, this alone does not mean that computational facts are perspectival. The fact that we must adopt some explanatory perspective or other is an inevitable aspect of scientific inquiry — we must always confront the world with a particular set of theories, models, instruments, etc. Computational inquiry is no different. Our computational perspective on a physical system is codified by a specific labelling scheme (or class of labelling schemes) relative to which we ascribe computational features to that system. But this is an epistemic point, not an ontological one. And there is no straightforward inference from this epistemic point to the stronger, ontological claim that computation *per se* is perspectival (Chakravartty 2010). Thus my view is committed to no more than a weak form of epistemic perspectivalism, a form it arguably shares with scientific theories quite generally.

26 At times, Dewhurst's remarks suggest a more epistemic reading; see (Coelho Mollo 2019) for an interpretation along these lines.

27 There is, of course, longstanding philosophical disagreement about the nature and status of mathematical objects. Rather than rehearse that debate here, I would instead emphasize that my view is compatible with a wide range of background views in the philosophy of mathematics. While some such views might lead to ontic perspectivalism, others arguably do not.

Computational Pluralism

One final point of contact worth mentioning concerns computational pluralism. Pluralism about a subject matter X holds, roughly, that there are multiple distinct yet equally legitimate accounts of, theories about, or kinds of X. Insofar as I allow that different properties impact computation in different cases, I endorse a version of computational pluralism. On this version, there are multiple distinct yet equally legitimate ways of individuating systems computationally. In contrast with other pluralist views in the literature, however, I hold that pluralism about computational identity is a natural consequence of the application of computability theory to physical systems.

To see this, consider an alternative pluralist view proposed by Lee (2021). Like me, Lee holds that computational systems and states can be individuated with respect to different properties in different cases. However, Lee's pluralism is grounded in the thought that computation "does not have an essential nature that supplies a single, privileged individuation" (Lee 2021, 234). By contrast, my approach does not rely on the idea of essential natures. Rather, it relies only on the observation that when applying a branch of mathematics to a physical system — computability theory, in our case — we do so relative to a specific pairing of physical and mathematical components. This leads naturally to computational pluralism, because different pairings may be more or less useful for different theoretical purposes in different circumstances.

5. Conclusion: The Middle Ground

Returning to the issue raised at the top of the paper, we have seen that the response offered here accommodates the fact that different properties impact computational identity in different cases. It thus sits naturally with the following intermediate view:

> **The Middle Ground.** A system's computational identity sometimes depends solely on its intrinsic features and sometimes also on its relational features, and that sometimes it depends on a system's semantic features and other times not.

Ultimately, which properties impact a system's computational identity is an empirical issue, to be determined by careful examination of the computational sciences. Work on computational identity would thus benefit from a more systematic survey of these sciences which goes beyond discussion of, e.g., isolated logic gates.[28] Such a survey would (a) taxonomize the explanatory uses of computation in a variety of representative cases, in order to (b) identify the specific kinds of labelling schemes — the different ways of identifying systems computationally — that support these explanatory aims. Although such a survey must for now remain on the agenda for future investigation, the view developed here ensures that worries about computational indeterminacy present no obstacle to this work.

28 For promising initial moves in this direction, see (Fresco, Copeland, and Wolf, 2021) and (Fresco 2021).

Appendix

Input 1	Input 2	Output
5-10V	5-10V	5-10V
5-10V	0-5V	0-5V
0-5V	5-10V	0-5V
0-5V	0-5V	0-5V

Table 1: A bistable circuit

Input 1	Input 2	Output
1	1	1
1	0	0
0	1	0
0	0	0

Table 2: Bistable AND gate

Input 1	Input 2	Output
5-10V	5-10V	5-10V
5-10V	2.5-5V	2.5-5V
5-10V	0-2.5	2.5-5V
2.5-5V	5-10V	2.5-5V
2.5-5V	2.5-5V	2.5-5V
2.5-5V	0-2.5	2.5-5V
0-2.5V	5-10V	2.5-5V
0-2.5V	2.5-5V	2.5-5V
0-2.5V	0-2.5	0-2.5V

Table 3: A tristable circuit

Input 1	Input 2	Output
1	1	1
1	$\frac{1}{2}$	$\frac{1}{2}$
1	0	$\frac{1}{2}$
$\frac{1}{2}$	1	$\frac{1}{2}$
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$\frac{1}{2}$	0	$\frac{1}{2}$
0	1	$\frac{1}{2}$
0	$\frac{1}{2}$	$\frac{1}{2}$
0	0	0

Table 4: Tristable ‘averaging’ computation.

Input 1	Input 2	Output
1	1	1
1	1	1
1	0	1
1	1	1
1	1	1
1	0	1
0	1	1
0	1	1
0	0	0

Table 5: Tristable OR gate.

Input 1	Input 2	Output
1	1	1
1	0	0
1	0	0
0	1	0
0	0	0
0	0	0
0	1	0
0	0	0
0	0	0

Table 6: Tristable AND gate.

Input 1	Input 2	Output
0	0	0
0	0	0
0	1	0
0	0	0
0	0	0
0	1	0
1	0	0
1	0	0
1	1	1

Table 7: Tristable AND gate, version 2.

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

Table 8: The functional profile of Table 1.

References

Blackmon, James. 2013. “Searle’s Wall.” *Erkenntnis* 78 (1): 109–17.

Brusentsov, Nikolay Petrovich, and José Ramil Alvarez. 2011. “Ternary Computers: The Setun and the Setun 70.” In *Perspectives on Soviet and Russian Computing*, edited by John Impagliazzo and Eduard Proydakov, 357:74–80. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-22816-2_10.

Chakravartty, Anjan. 2010. “Perspectivism, Inconsistent Models, and Contrastive Explanation.” *Studies in History and Philosophy of Science Part A* 41 (4): 405–12. https://doi.org/10.1016/j.shpsa.2010.10.007.

Chalmers, David J. 1996a. “Does a Rock Implement Every Finite-State Automaton?” *Synthese* 108 (3): 309–33.

———. 1996b. *The Conscious Mind*. Oxford University Press.

Coelho Mollo, Dimitri. 2018. “Functional Individuation, Mechanistic Implementation: The Proper Way of Seeing the Mechanistic View of Concrete Computation.” *Synthese* 195 (8): 3477–97.

———. 2019. “Against Computational Perspectivalism.” *The British Journal for the Philosophy of Science*, August, axz036.

Copeland, B. Jack. 1996. “What Is Computation?” *Synthese* 108 (3): 335–59.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms*. 2nd ed. McGraw-Hill Higher Education.

Cutland, Nigel. 1980. *Computability, an Introduction to Recursive Function Theory*. Cambridge: Cambridge University Press.

Davis, Martin. 1982. *Computability & Unsolvability*. New York: Dover.

Davis, Martin, Ron Segal, and Elaine Weyuker. 1994. *Computability, Complexity and Languages*. 2nd ed. San Diego: Academic Press.

Dewhurst, Joe. 2018a. “Computing Mechanisms Without Proper Functions.” *Minds and*

Machines 28 (3): 569–88.

———. 2018b. “Individuation Without Representation.” *The British Journal for the Philosophy of Science* 69 (1): 103–16.

Egan, Frances. 1995. “Computation and Content.” *The Philosophical Review* 104 (2): 181. https://doi.org/10.2307/2185977.

Frege, Gottlob. 1884. *The Foundations of Arithmetic*. Evanston, Illinois: Northwestern University Press.

Fresco, Nir. 2021. “Long-Arm Functional Individuation of Computation.” *Synthese*. https://doi.org/10.1007/s11229-021-03407-x

Fresco, Nir, B. Jack Copeland, and Marty J. Wolf. 2021. “The Indeterminacy of Computation.” *Synthese*. https://doi.org/10.1007/s11229-021-03352-9

Fresco, Nir, and Marcin Milkowski. 2021. “Mechanistic Computational Individuation Without Biting the Bullet.” *British Journal for the Philosophy of Science* 72 (2): 431–38.

Harbecke, Jens, and Oron Shagrir. 2019. “The Role of the Environment in Computational Explanations.” *European Journal for Philosophy of Science* 9 (3): 37. https://doi.org/10.1007/s13194-019-0263-7.

Hennessy, John L, and David A Patterson. 2003. *Computer Architecture: A Quantitative Approach*. San Francisco, CA: Morgan Kaufmann Publishers.

Hitchcock, Christopher. 2013. “Contrastive Explanation.” In *Contrastivism in Philosophy*, edited by Martijn Blaauw, 11–35. Routledge Studies in Contemporary Philosophy 39. New York: Routledge/Taylor & Francis Group.

Horowitz, Amir. 2007. “Computation, External Factors, and Cognitive Explanations.” *Philosophical Psychology* 20 (1): 65–80. https://doi.org/10.1080/09515080601085856.

Kitcher, Philip. 1981. “Explanatory Unification.” *Philosophy of Science* 48 (4): 507–31.

Lee, Jonny. 2021. “Mechanisms, Wide Functions, and Content: Towards a Computational Pluralism.” *British Journal for the Philosophy of Science* 72 (1): 221–44.

Massimi, Michela. 2018. “Four Kinds of Perspectival Truth.” *Philosophy and Phenomenological*

Research 96 (2): 342–59.

Massimi, Michela, and Casey D. McCoy, eds. 2019. *Understanding Perspectivism: Scientific and Methodological Prospects*. Routledge Studies in the Philosophy of Science 20. New York: Taylor & Francis.

Matthews, Robert J., and Eli Dresner. 2017. "Measurement and Computational Skepticism." *Nous* 51 (4): 832–54.

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

Papayannopoulos, Philippos, Nir Fresco, and Oron Shagrir. Forthcoming."On Two Different Kinds of Computational Indeterminacy." *The Monist*.

Peacocke, Christopher. 1999. "Computation as Involving Content: A Response to Egan." *Mind and Language* 14 (2): 195–202.

Piccinini, Gualtiero. 2008. "Computation Without Representation." *Philosophical Studies* 137 (2): 205–41.

———. 2015. *Physical Computation: A Mechanistic Account*. Oxford, UK: Oxford University Press.

———. 2020a. "Computation and Information Processing." In *Neurocognitive Mechanisms*, 128–55. Oxford University Press. https://doi.org/10.1093/oso/9780198866282.003.0007.

———. 2020b. *Neurocognitive Mechanisms: Explaining Biological Cognition*. 1st ed. Oxford University Press. https://doi.org/10.1093/oso/9780198866282.001.0001.

Potochnik, Angela. 2017. *Idealization and the Aims of Science*. Chicago: The University of Chicago Press.

Rescorla, Michael. 2013. "Against Structuralist Theories of Computational Implementation." *The British Journal for the Philosophy of Science* 64 (4): 681–707.

———. 2014. "A Theory of Computational Implementation." *Synthese* 191 (6): 1277–1307. https://doi.org/10.1007/s11229-013-0324-y.

———. 2017. "From Ockham to Turing – and Back Again." In *Philosophical Explorations of the Legacy of Alan Turing: Turing 100*, edited by Juliet Floyd and Alisa Bokulich, 279–

304. Cham: Springer International Publishing.

Ritchie, J. Brendan, and Gualtiero Piccinini. 2019. "Computational Implementation." In *Routledge Handbook of the Computational Mind*, edited by Mark Sprevak and Matteo Colombo, 192–204. London: Routledge.

Savage, John E. 2008. *Models of Computation: Exploring the Power of Computing*. http://cs.brown.edu/people/jsavage/book/.

Schiller, Henry. 2018. "The Swapping Constraint." *Minds and Machines* 28 (3): 605–22.

Schweizer, Paul. 2019a. "Computation in Physical Systems: A Normative Mapping Account." In *On the Cognitive, Ethical, and Scientific Dimensions of Artificial Intelligence: Themes from IACAP 2016*, edited by Matteo Vincenzo d'Alfonso, International Association for Computing and Philosophy, Annual Meeting, and Don Berkich, 27–47.

———. 2019b. "Triviality Arguments Reconsidered." *Minds and Machines* 29 (2): 287–308.

Schweizer, Paul, and Piotr Jablonski. 2013. "Abstract Procedures and the Physical World." *Proceedings of the AISB'13 Symposium on Computing and Philosophy*, 66–73.

Shagrir, Oron. 2001. "Content, Computation and Externalism." *Mind* 110 (438): 369–400.

———. 2020. "In Defense of the Semantic View of Computation." *Synthese* 197: 4083–4108.

Sprevak, Mark. 2010. "Computation, Individuation, and the Received View on Representation." *Studies in History and Philosophy of Science Part A* 41 (3): 260–70.

———. 2019. "Triviality Arguments About Computational Implementation." In *Routledge Handbook of the Computational Mind*, edited by Mark Sprevak and Matteo Colombo, 175–91. London: Routledge.

Turing, Alan. 1936. "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society* 42 (1): 230–65.