Implementation as Resemblance

André Curtis-Trudel*†

This article advertises a new account of computational implementation. According to the resemblance account, implementation is a matter of resembling a computational architecture. The resemblance account departs from previous theories by denying that computational architectures are exhausted by their formal, mathematical features. Instead, they are taken to be permeated with causality, spatiotemporality, and other nonmathematical features. I argue that this approach comports well with computer scientific practice and offers a novel response to so-called triviality arguments.

1. Theories of Implementation. Theories of physical computation address two questions:

> Q1. What distinguishes physical systems that compute from those that do not?
>
> Q2. Among physical systems that compute, what distinguishes those that compute the same thing from those that do not?

Q1 concerns the difference between laptops and calculators, on the one hand, and rocks and tables, on the other. Q2, by contrast, concerns the distinction between one laptop computing dot products and another computing Fourier transforms. An adequate account of physical computation should answer both (Sprevak 2019).

*To contact the author, please write to: Department of Philosophy, 337E University Hall, Ohio State University, Columbus, OH 43210; e-mail: curtistrudel.1@osu.edu.

†For helpful comments and discussion I would like to thank Samuel Fletcher, Chris Pincock, Richard Samuels, and Stewart Shapiro. I would also like to thank audiences at the Ohio State Graduate Student Workshop, the 2019 Society for the Metaphysics of Science conference, and the American Philosophical Association (APA) 2020 Central Division meeting for comments and feedback. Special thanks go to Michael Rescorla, whose APA commentary led to substantial improvements in the article.

Philosophy of Science, 88 (December 2021) pp. 1021–1032. 0031-8248/2021/8805-0023$10.00
Copyright 2021 by the Philosophy of Science Association. All rights reserved.

https://doi.org/10.1086/714872 Published online by Cambridge University Press

Different answers to these questions are possible. I will be concerned with *implementationist* theories, which hold that a physical system computes if it implements some computational system, or 'computation', for short. Thus:

> A1. A physical system computes just in case it implements some computation.
>
> A2. What a physical system computes is determined by the computation it implements.

These answers are schematic, in that they say little about what computations are and what implementation amounts to. Different accounts of implementation emerge from different specifications of these details.

This article introduces a new account implementation, which I call the *resemblance account*. I sketch the account in sections 2–4. Section 5 deals with some background metaphysical issues. Section 6 argues that the resemblance offers an interesting new perspective on an old problem in the philosophy of physical computation. Finally, section 7 deals with an objection, and section 8 concludes.

A caveat before proceeding: my aim is advertisement. I wish to show that the resemblance account offers a novel approach to physical computation, worthy of further investigation. Regrettably, however, this means some issues will not receive the treatment they deserve. These issues must wait for another occasion, and I will flag them as they arise.

2. The Resemblance Account. I propose to begin at the beginning. In his landmark paper, Turing offers the following characterization of an *a*-machine: "The machine is supplied with a 'tape' (the analogue of paper) running through it, and divided into sections (called 'squares') each capable of bearing a 'symbol'. . . . The configuration [of the machine] determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (i.e. bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left" (1936, 231). As we know, Turing arrived at this characterization by carefully considering the activity of human workers proceeding effectively. The restriction that *a*-machines may only 'observe' one symbol at a time, for instance, is justified on the grounds that human workers can only distinguish between finitely many different primitive symbol types.[1]

1. For, if not, then we could distinguish between (tokens of) types that differ to an arbitrarily small degree. Unfortunately, our perceptual apparatus is not nearly as sophisticated as this. See Turing (1936, 249–52) and Sieg (2009).

https://doi.org/10.1086/714872 Published online by Cambridge University Press

The importance of Turing's insight is not hard to appreciate. By linking the characterization of an *a*-machine directly to the activities of actual human workers, Turing's analysis sheds light on the computational capacities of humans working effectively. Very roughly, the computational power of *a*-machines bears on the computational power of effective human workers because the former resemble the latter in certain important respects: both have certain 'perceptual' limitations, both follow only finitely many instructions one at a time, and so on. Indeed, alternative analyses of effective calculability, such as λ-definability or Herbrand-Gödel general recursivity, were found to be unsatisfactory largely because they fail to adequately illuminate the basic activities of a human working effectively.

I mention all of this because it seems to me that Turing's analysis contains the essentials of the resemblance account. At the core of Turing's analysis is a specific kind of computation, or, as I will call it, a *computational architecture*. The *a*-machine architecture bears on the computational powers of human workers because, and to the extent that, the former resemble the latter in certain respects. The resemblance account codifies this insight:

> **The Resemblance Account.** A physical system implements a computational architecture just in case, and to the extent that, it resembles that computational architecture.

In the following two sections I flesh out the notion of a computational architecture and explain the notion of resemblance at play. But it should be noted that while I talk of *the* resemblance account, I am in fact scouting a family of views. Different ways of filling in the sketch I give deliver different particular resemblance accounts. I mention the major choice points as they arise.

3. Computational Architectures. To a first approximation, computational architectures are 'blueprints' for physical computing devices. These blueprints specify which features a system must have in order to count as a computing device of a particular sort (I return to the question of what blueprints are, and what specification amounts to, in sec. 5). Turing's description, for instance, constitutes a blueprint that specifies the features a physical system must have to be an *a*-machine. Accordingly, a physical system performs *a*-machine computations only if it has these features too. Anything lacking these features does not count as an *a*-machine and a fortiori does not perform *a*-machine computations either.

But what are these features? Some concern the physical or mechanical features of the device. For instance, Turing's characterization requires that *a*-machines *scan* the tape, *write* symbols, and *shift* left or right; that they have a read/write head and a tape divided into squares; and that later machine

https://doi.org/10.1086/714872 Published online by Cambridge University Press

states be determined by earlier states. Other features are more abstract and concern the patterns or regularities the machine or its components exhibit. For instance, two machines might compute the same string-theoretic function for increasing the length of a string of strokes by one but in different ways. One machine might append a stroke to the rightmost end of an input string, while another might append a stroke to the leftmost end of an input string. Others features, still, concern what states of the device represent. A string of strokes may refer to a natural number, and a machine as a whole may be taken to represent the function it computes. For instance, the two machines just mentioned may both represent the successor function. The upshot is that a physical device must exhibit these sorts of features if it is to count as or implement an *a*-machine or if it is to perform *a*-machine computations.

However, while Turing's characterization is illustrative, it is not representative of contemporary computer design. For a state-of-the-art understanding of computer architectures, we should look to work in computer engineering (Hennessy and Patterson 2003). This discipline produces highly specific descriptions of computational architectures. For instance, for a physical system to count as a MIPS (microprocessor without interlocked pipelined stages) microarchitecture, it must exhibit certain features. Some of these features are specified explicitly in the microarchitecture description, for instance, that the system have a data path with a certain pipelining scheme, components for sign extension operations, and so on. Others are left tacit, such as the requirement that the system be cast in a silicon wafer or that it have a certain clock rate. As in the *a*-machine case, nothing counts as a MIPS microarchitecture unless it has these features.

More broadly, it seems to me that three kinds of features are commonly cited in the descriptions offered by computer architects and engineers. *Physical-mechanical* features concern the physical and microphysical structure of a system: what its component are made of and their relationships, interactions, and composition. Other features concern the abstract patterns or regularities exhibited by various states and components, and for want of a name I will call these features *syntactic*. Finally, *semantic* or *representational* features concern what the states or processes of the device represent. However, I do not take this list to exhaust the features that may be specified by a computational architecture. Indeed, it would be a mistake to try to specify such a list once and for all. Instead, we should regard this list as open to addition or amendment, as computer engineers and architects devise new sorts of computing systems with different sorts of features.

The blueprint metaphor brings out another important feature of computational architectures. Blueprints in general can be more or less fine-grained. Some omit irrelevant details, as when they indicate that one must build a column that supports 500 kg but do not say whether it must be made of wood or stone. Others are more exacting and demand that a 5 meter tall fluted marble

https://doi.org/10.1086/714872 Published online by Cambridge University Press

column be put here with such-and-such capital ornaments. Similarly for computational architectures. We might be told that the device is to have a read/write head but not be told what it is made of. At other times we are told that the device must be made of silicon, have 500 MB of L3 cache, have four cores, run at 2.4 GHz, and so forth.

Obviously, the question of which specific physical-mechanical, syntactic, or semantic properties are required for computation is best left to computer scientists and engineers. It is a highly nontrivial task designing a computing system, involving a tremendous amount of epistemic labor.[2] Consequently, I doubt that philosophers have much to contribute on this front. Rather, I take it that a philosophical theory of implementation should be flexible enough to accommodate developments in computer science. As we will see, the resemblance account is flexible in just this way.

4. Resemblance. The next task is to say what it is for a physical system to 'resemble' a computational architecture. This marks a choice point in the theory. Some philosophers might be content to rest with an intuitive or commonsensical notion of resemblance. This is fine as far as it goes, but there are benefits to working with a more precise account. To a very rough first approximation, the idea pursued here is that a physical system resembles a computational architecture to the extent that it (*a*) has features 'specified' by the architecture and (*b*) lacks features not 'specified' by the architecture. A physical system resembles an *a*-machine, for instance, to the extent that it has a read/write head, a control unit, tape, and so on.

I will use Weisberg's (2012) *weighted feature-matching account* of similarity to make this idea precise.[3] Simplifying somewhat, this account holds that similarity, or resemblance, is always determined relative to a distinguished finite class of features F, called the feature set. If C is a computational architecture and P is a physical system, we will let $F_C \subseteq F$ be the features specified by C and $F_P \subseteq F$ be a designated set of features of P. The degree of fit between P and C is then measured by the following equation:

$$|F_C \cap F_P| - |F_C - F_P| - |F_P - F_C|. \tag{1}$$

For convenience, I will write this as Res(F, C, P). Here $F_C \cap F_P$ are the features shared by C and P, $F_C - F_P$ are the features specified by C that P

2. To get a sense of the complexity involved, note that a modest contemporary microprocessor houses approximately 4.5 billion transistors and executes upward of 140 instructions in parallel.

3. Some philosophers distinguish between resemblance and similarity, but I will use the terms interchangeably.

https://doi.org/10.1086/714872 Published online by Cambridge University Press

lacks, and $F_P - F_C$ are the features had by P not specified by C. Set cardinality is $|\cdot|$. We say that P resembles C, with respect to F, to degree n just in case $\text{Res}(F, C, P) = n$.

This account treats resemblance as a graded notion. I think this is the most basic notion of resemblance, and we can use it to define other notions as the need arises. For instance, we can say that P perfectly resembles C just in case $\text{Res}(F, C, P) = |F_C \cap F_P|$. Similarly, we can say that C and P resemble each other simpliciter just in case $\text{Res}(F, C, P) \geq m$, for some predetermined 'cutoff' degree of resemblance m. For present purposes I think it is enough to work with the basic notion, but I am open to the possibility that a more refined notion is appropriate for thinking about implementation.

Given this account of resemblance, implementation is in the first instance a matter of degree, so that a physical system implements a given computational architecture to a greater or lesser degree. Some philosophers might be uncomfortable with this result, preferring an absolute notion of implementation instead. But as I just mentioned, given a graded notion of implementation we can use it to define an absolute notion if we want. Moreover, there is some independent reason for working with a graded notion. Computer scientists often talk about different physical systems being better or worse implementations of a given architecture. This practice is naturally understood as relying on a graded notion of implementation, and it is not clear how to capture this talk, without distortion, with an absolute notion.

One further issue deserves comment. Exactly what degree of resemblance is required for implementation? On the one hand, perfect resemblance seems too exacting: it seems useful to allow that a physical system may implement an architecture even when the two do not perfectly resemble each other. On the other, too low a degree threatens to trivialize the notion of physical computation: there are plausibly some simple physical-mechanical properties shared by paradigmatically noncomputing systems and any computational architecture. There is a genuine question here, but for now I propose to duck it. For the time being, I will simply say that implementation requires a 'sufficiently high' degree of resemblance, noting that this is just a placeholder for what will undoubtedly be a complicated theory of just what a 'sufficiently high' degree amounts to.

5. Interlude: Matters of Metaphysics. So far I have glossed computational architectures as 'blueprints', and I have said that blueprints 'specify' features. But what does all this mean? Presumably we do not wish to add blueprints as a new fundamental to our ontology, so I had better find a way to characterize them in more familiar terms. Perhaps unsurprisingly, this marks another choice point for the theory. There are different ways of cashing out the notion of a blueprint, according to different tastes in metaphysics. Here I will mention a few, but I do not take these to exhaust the alternatives.

https://doi.org/10.1086/714872 Published online by Cambridge University Press

One option takes computational architectures to be highly specific universals. In this case, implementation boils down to instantiation, and a computational architecture 'specifies' a class of features in something like the way that a conjunctive universal 'specifies' its conjuncts, for instance, by having them as parts. So construed, Turing's description of *a*-machines is a description of a universal, the instantiation of which is a matter of having a read/write head, a tape, and so forth. A physical system will instantiate a computational architecture more or less, to the degree that it instantiates the conjuncts that compose the architecture in question. In this case, Res(*F*, *C*, *P*) is a measure of the degree of instantiation of a given universal.

Philosophers who balk at universals will prefer a more deflationary approach. Here too there are many options available. One treats computational architectures as abstract particulars, perhaps in something like the way that scientific models are said to be abstract particulars (Giere 1988). Copeland and Shagrir mention, although do not endorse, a view in this vicinity. As they explain, such a view "recognizes an ontological level lying between the realization (or physical-device) level and the level of pure-mathematical ontology. . . . At this level are to be found notional or idealized machines that are rich with spatio-temporality and causality" (Copeland and Shagrir 2011, 234). On this account, computational architectures are taken to literally have certain physical-mechanical, syntactic, or semantic features. Accordingly, Res(*F*, *C*, *P*) measures the degree to which *P* has features also had by *C*. And other deflationary approaches are possible, too. Rescorla, for instance, suggests that computational architectures are abstract descriptions of physical systems (2014, 1278). On an account such as this one, implementation turns out to be a semantic relation, and degree of resemblance then amounts to how well a given computational architecture describes a given physical system.[4]

Undoubtedly there are trade-offs to be navigated between these alternatives. Which way to go turns both on quite general questions about the nature of abstract objects and scientific models, as well as more local concerns about the conception of computational architectures most appropriate for theorizing about implementation. Work remains to be done investigating these trade-offs, and I take it to be a virtue of the resemblance account that it provides a neutral framework in which these questions can be addressed.

6. Triviality Arguments. So far I have motivated the resemblance account by noting that it matches computer scientific practice. Computer scientists routinely describe computational architectures, such as Turing machines,

4. Thus, on this way of looking at things, Rescorla's descriptivist account turns out to be a species of the resemblance account. Thanks to Michael Rescorla for urging me to clarify this point.

https://doi.org/10.1086/714872 Published online by Cambridge University Press

in spatiotemporal terms. I take it to be a point in favor of the resemblance account that it reflects this practice. But the account is also attractive on theoretical grounds. In particular, the resemblance account offers a novel response to so-called triviality arguments.

To better situate this response, it is instructive to recall how triviality worries emerge for more traditional theories of implementation. The most popular theory holds that implementation is a relation between a physical system, on the one hand, and an abstract, mathematical computation, on the other. In the simplest case a computation is taken to be a finite automaton, composed of a finite set of states plus a transition function. More complicated computations may include inputs, outputs, or states with internal combinatorial structure. These details aside, however, the characteristic feature of the received view is that computations are exhaustively characterized by their formal, mathematical structure. Following Rescorla (2014), I will call this *structuralism* about implementation.[5]

Structuralism holds that a physical system implements a computation if its state transitions 'mirror' the state transitions of some formal computation. 'Mirroring' is typically taken to be a structure-preserving map, or isomorphism, between physical and formal states. Thus, according to a simple structuralist view, if $C = (S, T)$ is a computation with states $S = \{S_1, S_2, \ldots, S_n\}$, and transition function $T : S \rightarrow S$, we have:

> **Simple Structuralism.** Physical system P implements computation C just in case there exists a mapping f from states of P to S such that if P is in a state P_i for which $f(P_i) = S_k$, and $T(S_k) = S_m$, then P goes into state P_j for which $f(P_j) = S_m$.

Notoriously, however, mappings are cheap and computations are abundant. For nearly every physical system P and every computation C, there is a structure-preserving map between P and C, in which case nearly every physical system implements every computation. This is the core of the triviality worry (Sprevak 2019).

In light of triviality worries, few structuralists endorse such a simple view. Various additional constraints are added in an effort to avoid triviality. Some common requirements are that implementing systems satisfy counterfactual conditionals (Block 1995; Copeland 1996), that they exhibit appropriate causal structure (Chalmers 1996; Scheutz 2001), that distinct physical states be mapped to distinct formal states (Chalmers 1996; Godfrey-Smith 2009),

5. Chalmers (1996) is representative of this approach. For other structuralist views, see Copeland (1996), Scheutz (2001), Godfrey-Smith (2009), Sprevak (2010), and Millhouse (2019), among many others.

https://doi.org/10.1086/714872 Published online by Cambridge University Press

that only appropriately 'natural' or 'simple' physical states feature in the mapping (Scheutz 2001; Godfrey-Smith 2009; Millhouse 2019), that the physical states have representational properties (Sprevak 2010), or that the physical states be states of functional mechanisms (Piccinini 2012, 2015). And others are surely possible.

All of these tactics are a kind of 'bottom up' response to triviality. They attempt to cut down the class of implemented computations by constraining which physical systems figure in the preimage of the implementation mapping. Only systems with the appropriate counterfactual, representational, and so on, features get in. The resemblance account, by contrast, employs a 'top down' response to triviality. For the account denies that computational architectures are exhausted by their formal, mathematical character. Instead, they may be replete with physical-mechanical, representational, and so on, features. Since implementation is a matter of resembling a computational architecture in these respects, and since most physical systems do not have the right arrangement of features, most physical systems will not implement many, or even any, computational architectures. For instance, most physical systems do not have a read/write head, an indefinitely extensible tape, an so on, so most systems will not implement an *a*-machine.

There is some reason to be dissatisfied with 'bottom up' approaches. One worry is that structuralism cannot adequately explain why physical computation must involve causal, semantic, and so on, features. From the structuralist's perspective, computation is fundamentally a mathematical phenomenon, captured by pure mathematical computing systems such as set-theoretic characterizations of Turing machines, deterministic finite automata, and the like. Methodologically, the strategy is to start with a prior mathematical notion of computation and define a notion of physical computation in terms of it. But given this methodology, the requirement that physical computation essentially involve causal, semantic, and so on, features looks less like a discovery about the nature of physical computation and more like an ad hoc maneuver designed to save the theory from triviality. If computation is fundamentally a mathematical phenomenon, as the structuralist holds, and if that account of computation leads to triviality, as it appears to, then what could possibly explain why physical computation must be causal, semantic, or whatever?

This explanatory problem plausibly does not arise for the resemblance account. Because the resemblance theorist denies that computation is essentially mathematical, there is no additional task of explaining why physical computation must involve causal-mechanical, representational, and so on, features. Recall Turing's characterization of *a*-machines. On that characterization, it is constitutive of *a*-machines that they have a read/write head, that later states be determined by earlier states, and so on. But since *a*-machines come with these causal-mechanical features 'built in', so to speak, it is straightforward to explain why a physical system must have these features in order

https://doi.org/10.1086/714872 Published online by Cambridge University Press

to carry out *a*-machine computations. The reason is that *a*-machine computations are a certain kind of causal-mechanical (etc.) process. From this perspective, a causal-mechanical requirement is not an ad hoc maneuver designed to save the theory from triviality but instead reflects a basic fact about the nature of *a*-machine computations.

7. Medium Independence. The resemblance account responds to triviality by enriching the character of implemented computations. But this maneuver may seem to cut against the idea, widely endorsed, that computations are medium independent (Piccinini 2015, 122–23). And this, in turn, may seem to be a compelling reason to abandon the account. In response to this worry, this section explains how a suitable analogue of medium independence can be developed within the resemblance framework.

To a first approximation, a property or process is medium independent if it can be realized in different physical media. *Cooking lentils* is not medium independent, because it can be realized in only quite specific physical media; *powering a drivetrain* is, because it can be accomplished by otherwise quite different physical systems (internal combustion engines, electric motors, etc.). Medium independence is closely related to multiple realizability. A property or process is multiply realizable, roughly, if it can be realized by different kinds of physical systems. And it seems undeniable that computations are multiply realizable and perhaps even medium independent. As Block (1995) points out, an AND gate might be realized by transistors. But it might also be realized by a clever arrangement of mice, string, and cheese. Moreover, the literature on unconventional computation is replete with cases in which the same computation (e.g., a sorting task) is performed by wildly different physical systems.

The trouble for the resemblance account is that there appears to be no single computational architecture common to the wide variety of computing systems hypothesized by computer scientists. There is no architecture, for instance, that both silicon and murine AND gates resemble. But absent such an architecture, it seems the resemblance account cannot capture what these systems have in common.[6]

What should the resemblance theorist make of this? The outline of a solution becomes clear once we reflect on the role played by ideas of medium independence and multiple realizability in computational theorizing. In general, we require a way to describe physical systems that abstracts away from (some of) their physical details. So abstracted, we can consider whether, for

6. There is, I suppose, a disjunctive architecture composed of both silicon and murine components. But such a device is a metaphysician's contrivance, not a genuine deliverance of computer science.

https://doi.org/10.1086/714872 Published online by Cambridge University Press

example, two systems compute the same logical function, despite their physical dissimilarities. Thus, to accommodate medium independence, it is enough if the resemblance account can furnish a level of description that abstracts away from certain physical features and allow us to capture what, for instance, the different AND gates have in common. The rest of this section explains how this might go.

To begin, we need to import some more machinery from the weighted-feature matching account. First, while resemblance is characterized above in terms of a single class of features, we can also define a notion of resemblance that discriminates between different classes of features. If $F_1, F_2, \ldots, F_m$ are classes of features, then we can say that P resembles C with respect to the $F_i (1 \leq i \leq m)$, to degree n just in case $\Sigma_{i=1}^{m} \mathrm{Res}(F_i, P, C) = n$. Second, by adding coefficients we can discount (or boost) the contribution of a particular class of features F_i to the overall resemblance score:

$$x_i(|F_{i,C} \cap F_{i,P}| - |F_{i,C} - F_{i,P}| - |F_{i,P} - F_{i,C}|). \tag{2}$$

I will write $\mathrm{Res}(F_i, x_i, P, C)$ as shorthand for this. Thus, if $x_i \in \mathbb{R}$, $(1 \leq i \leq m)$ are coefficients, the generalized resemblance score is given by $\Sigma_{i=1}^{m} \mathrm{Res}(F_i, x_i, P, C)$.

The strategy is to use this generalized notion of resemblance to define a new notion of resemblance between systems that accommodates intuitions about medium independence. If F_1, F_2, F_3 are classes of physical-mechanical, syntactic, and semantic features, respectively, with corresponding coefficients x_1, x_2, x_3, then by setting $x_1 = x_3 = 0$ and setting $x_2 = 1$ we can say that P pattern resembles C to degree n just in case $\Sigma_{1}^{3} \mathrm{Res}(F_i, x_i, P, C) = n$. (As above, we may define other notions as needed. For instance, we can say that P pattern resembles C simpliciter just in case P pattern resembles C to a high enough degree.)

How does all this help? Medium independence is naturally thought to concern what I have called 'syntactic' features. We say that silicon and mouse-and-string systems compute AND, when they do, because at a certain abstract level of description they exhibit the same patterns, regardless of their physical substrate. The resemblance account can accommodate this fact by noting that different AND gates pattern resemble each other. Thus, the account furnishes a level of description appropriate for capturing what the two systems have in common, while at the same time recognizing that in order to avoid triviality, there must be more to implementation than mere pattern resemblance.

8. Conclusion. The resemblance account holds that a physical system implements a given computational architecture just in case, and to the extent that, it resembles that computational architecture. This view is grounded in

https://doi.org/10.1086/714872 Published online by Cambridge University Press

Turing's influential approach to thinking about computational architectures, an approach that persists in computer science today. While I have not here attempted an exhaustive assessment of the resemblance account, the considerations surveyed here are promising. For this reason I take the resemblance account to be worthy of further investigation.

REFERENCES

Block, N. 1995. "The Mind as the Software of the Brain." In *An Invitation to Cognitive Science*, 2nd ed., ed. D. N. Osherson, et al., 3:377–425. Cambridge, MA: MIT Press.

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

Copeland, B. J. 1996. "What Is Computation?" *Synthese* 108 (3): 335–59.

Copeland, B. J., and O. Shagrir. 2011. "Do Accelerating Turing Machines Compute the Uncomputable?" *Minds and Machines* 21 (2): 221–39.

Giere, R. N. 1988. *Explaining Science: A Cognitive Approach*. Chicago: University of Chicago Press.

Godfrey-Smith, P. 2009. "Triviality Arguments against Functionalism." *Philosophical Studies* 145 (2): 273–95.

Hennessy, J. L., and D. A. Patterson. 2003. *Computer Architecture: A Quantitative Approach*. San Francisco: Kaufmann.

Millhouse, T. 2019. "A Simplicity Criterion for Physical Computation." *British Journal for the Philosophy of Science* 70 (1): 153–78.

Piccinini, G. 2012. "Computationalism." In *The Oxford Handbook of Philosophy of Cognitive Science*, ed. R. Samuels, E. Margolis, and S. P. Stitch, 222–49. Oxford: Oxford University Press.

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

Rescorla, M. 2014. "A Theory of Computational Implementation." *Synthese* 191 (6): 1277–307.

Scheutz, M. 2001. "Computational versus Causal Complexity." *Minds and Machines* 11 (4): 543–66.

Sieg, W. 2009. "On Computability." In *Philosophy of Mathematics*, ed. A. Irvine, 535–630. Burlington, MA: North-Holland.

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

———. 2019. "Triviality Arguments about Computational Implementation." In *Routledge Handbook of the Computational Mind*, ed. M. Sprevak and M. Colombo, 175–91. London: Routledge.

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

Weisberg, M. 2012. "Getting Serious about Similarity." *Philosophy of Science* 79 (5): 785–94.

https://doi.org/10.1086/714872 Published online by Cambridge University Press