Implementation as Resemblance[1]

Abstract

This paper 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 non-mathematical 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 don't?

Q2. Among physical computing systems, what distinguishes those that compute the same thing from those that don't?

(1) concerns the difference between laptops and calculators on the one hand, and rocks and tables on the other. (2), 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).

Different answers to Q1 and Q2 are possible. I shall be concerned with *implementationist* theories, which hold that a physical system computes if it implements some computational system, or 'computation', for short.[2] Thus:

[1] Draft of March 5, 2020. Word count: 4947.

[2] What of alternatives to implementationism? On one reading, Piccinini's (2015) mechanistic account answers Q1 and Q2 by direct appeal to the notion of a computing mechanism. So construed, the implementation relation plays no part in the mechanistic account. This sort of approach is worth exploring, but is beyond the present scope.

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, however. They say little about what computations are, and what implementation amounts to. Different accounts of implementation emerge from different specifications of these details.

This paper 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. Sections 6 argues that the resemblance offers an interesting new perspective on some old problems in the philosophy of physical computation. Finally, Section 7 deals with an objection, and Section 8 concludes.

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

2. The Resemblance Account

I propose to begin at the beginning. In his landmark 1936 paper, Turing offers the following description 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. (Turing 1936, 231)

As we know, Turing arrived at this conception of *a*-machines 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. (For, if not, then we could distinguish between (tokens of) types which differ to an arbitrarily small degree. But our perceptual apparatus is not nearly as sophisticated as this. See Turing (1936, 249 - 252) and Sieg (2009) for discussion.)

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 and limitations 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, such as λ-definability or Herbrand-Godel general recursivity, were unsatisfactory 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. *A*-machines bear on the computational powers of human workers because, and to the extent that, the former resemble the latter in certain respects. The resemblance account generalizes and precisifies this insight. On the resemblance account, physical computation is a matter of resembling a computational architecture:

> **The Resemblance Account.** A physical system computes just in case, and to the extent that, it resembles a 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, really I am scouting a family of views. Different ways of filling in the sketch I give deliver different particular resemblance accounts. I will mention the major choice-points as they arise.

3. Computational Architectures

Turing's *a*-machines are an example of what I'll call a *computational architecture*. To a first approximation, computational architectures are ‘blueprints’ for physical computing devices. 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 Section 5). Turing’s description, for instance, constitutes a blueprint which specifies the features a physical system must have in order to ‘counts as’ an *a*-machine.

Accordingly, a physical system performs *a*-machine computations only if it has these features too. Anything lacking these features doesn't count as an *a*-machine, hence *a fortiori* doesn't 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 latter machine states be *determined* by earlier states. Other features are more abstract, and concern the patterns or regularities the machine or its components exhibit. Others still concern what states of the device represent. The symbols on the tape refer to natural numbers, for instance, and a machine as a whole may be taken to represent, in some sense, the function it computes. The upshot of all of this is that a physical must exhibit these sorts of features if it is to 'count as' 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 (in the present sense), we should look to work on computer architecture and engineering. These disciplines truck in 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 a highly specific set of features.[3] Some of these features are described explicitly in the microarchitecture description, for instance that the system have a datapath with a certain pipelining scheme, certain 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, that

[3] See Harris and Harris (2013).

it have a certain clock rate, and so on. As in the *a*-machine case, nothing counts as a MIPS microarchitecture unless it has these features.

Stepping back, it seems to me that three kinds of features are commonly cited in the descriptions offered by computer architects and engineers. *Physico-mechanical features* concern the physical and micro-physical structure of a system: its components and their relationships, interactions, and composition. Other features concern the patterns or regularities exhibited by various states and components, and I'll call these features *syntactic*. Finally, *semantic* or *representational* features concern what the states or processes of the device represent. However, I don't 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 new and different sorts of features.

Computational architectures can be more or less fine-grained. Some omit irrelevant details, as when they indicate that one must build a column that supports 500kg, but does not say whether it must be made of wood or stone. Others are more exacting, and demand that a five meter tall fluted marble column be put here with such-and-such capital ornaments. Similarly, in the computational case we might be told that the device is to have a read/write head, but not told what it is made of. Other times we are told that the device must be made of silicon, have 500MB of L3 cache, have four cores, run at 2.4GHz, and so forth.

However, the question of which specific physico-mechanical, syntactic, or semantic properties are required for computation is best left to computer scientists and engineers. It is a

highly non-trivial task designing a computing system, involving a tremendous amount of epistemic labour.[4] Consequently, I doubt that philosophers have much to contribute on this front.

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, and here I will offer one.

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. (What it is for an architecture to 'specify' a feature depends to some extent on one's background metaphysics; see Section 5 for more.)

Recent work on similarity can be used to make this precise.[5] On this theory, resemblance is always determined relative to a distinguished class of features *F*, called the feature set. If *C* is a computational architecture and *P* is a physical system, we'll let $F_C \subseteq F$ be the features specified by *C* and $F_P \subseteq F$ be the set of features of *P*. Then we can say that *P* resembles *C*, with respect to *F*, to degree *n*, just in case

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

[5] The account presented here follows Weisberg's 'weighted feature-matching account' (2012). For recent criticisms and elaborations, see Parker (2015) and Fang (2017). Some philosophers distinguish between resemblance and similarity, but here I use the terms interchangeably.

$$|F_C \cap F_P| - |F_C - F_P| - |F_P - F_C| = n \quad (1)$$

For convenience, I'll write *Res(F, C, P)* = *n*. Here $F_C \cap F_P$ are the features shared by *C* and *P*. $F_C - F_P$ are the features specified by *C* which *P* lacks. And $F_P - F_C$ are the features had by *C* not specified by A. This equation, in effect, measures the extent to which *P* 'fits' *C*'s specification.

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 *Res(F, C, P)* = *n* and $|F_C \cap F_P| = n$. Similarly, we can say that *C* and *P resemble each other simpliciter* just in case *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's 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 they don't perfectly resemble each other. On the other, too low a degree threatens to trivialize the notion of physical computation: there are plausibly some simple physico-mechanical properties shared by paradigmatically non-computing systems and any computational architecture. With this complication flagged, 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've glossed computational architectures as 'blueprints', and I've said that blueprints 'specify' features. But what does all this mean? Presumably we don't wish to add 'blueprints' as a new fundamental to our ontology, so we'd better find a way to cash them out 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 background metaphysics. Here I'll mention two, but I don't take these to exhaust the alternatives. In fact, I think the resemblance account can be developed in a way that accommodates a wide variety of views in metaphysics, and I take this to be a virtue of the account.

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 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 some scientific models are said to be abstract particulars.[6] On this account, computational architectures are taken to literally *have* certain physico-mechanical, syntactic, or semantic features. In this case, resemblance amounts to property sharing, so that *Res(F, C, P)* is a measure of the degree to which *P* has features also had by *C*. And other deflationary approaches are possible too. For instance, we might take computational architectures to be linguistic entities -- descriptions, say -- so that implementation boils down to some sort of semantic relation, such as accurate description. Degree of resemblance then amounts to how well a given computational architecture describes a given physical system.

At any rate, which way you go turns, to a large extent, on your ontological tastes. I take it to be a virtue of the resemblance account that it can be developed in a way that satisfies a wide variety of ontological palates.

---

[6] See, e.g., Giere (1988, Chapter 3). A related approach, although arguably more deflationary, is found in Godfrey-Smith (2009), who suggests that models are fictional entities. Copeland and Shagrir mention, but do not endorse, a view of computation in this vicinity too. As they explain, their 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" (2011, 234).

6. Triviality Arguments

So far I've motivated the resemblance account by noting that it matches the thought and talk of computer scientists. Computer scientists routinely describe computational architectures, such as Turing machines, in spatio-temporal terms. I take it to be a point in favour of the resemblance account that it reflects this practice. But the resemblance account is also attractive on theoretical grounds. In particular, the resemblance account offers a novel response to so-called triviality arguments. This section sketches that response.

It is instructive to recall how triviality worries emerge for standard 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 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*.[7]

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 → S, we have:

[7] Chalmers (1996) is representative of this approach. Other proponents include Millhouse (2017), Schweizer (2019), Sprevak (2010), and Scheutz (2001), among many others.

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, according to (SF). This is the core of the triviality worry.[8]

In light of triviality worries, few philosophers endorse structuralism in this unalloyed form.[9] Various additional constraints are added to (SF) 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); that only appropriately `natural' or 'simple' physical states feature in the mapping (Scheutz, 2001; Godfrey-Smith, 2009; Millhouse, 2017); that the physical states have representational properties (Shagrir, 2001, 2018; 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 those with the appropriate counterfactual,

[8] See Sprevak (2019) for more.
[9] Schweizer (2019) is a recent exception.

representational, etc. features get in. A 'top down' response, by contrast, enriches the account of the systems in the image of the mapping. To the extent that formalists take computations to be formal, mathematical objects, they *must* take bottom up approaches to triviality. The few 'top down' approaches they can take typically complicate the structure of computations while preserving their overall formal, mathematical character.

Concerning triviality, the resemblance account employs a 'top down' response. The account denies that computational architectures are exhausted by their formal, mathematical features. Instead, they may be replete with physico-mechanical, representational, etc. features. Since implementation is a matter of resembling a computational architecture in these respects, and since most physical systems *don't* have the right arrangement of features, most physical systems won't implement many, or even any, computational architectures. For instance, most physical systems don't have a read/write head, an indefinitely extensible tape, etc., so most systems won't implement *a*-machines. Similarly, most physical systems aren't composed of a silicon board, have a certain pipelining scheme, and so on, so most physical systems don't implement a MIPS microarchitecture.[10]

There is some reason to be dissatisfied with 'bottom up' approaches; here I'll mention just one. The worry is that structuralism cannot adequately explain *why* physical computation necessarily involves causal, semantic, etc. features.[11] From the structuralist's perspective, computation is fundamentally a mathematical phenomenon, captured by pure mathematical computing systems such as Turing machines (construed set-theoretically), DFAs, and the like.

[10] However, attentive readers will note that this response turns, to some extent, on the specific choice of features in question. There subtleties must await another occasion.

[11] Another is that structuralism fails to capture the implementation conditions of many computational models; see Rescorla (2014).

Methodologically, the strategy is to *start* with a prior mathematical notion of computation, and define a notion of physical computation in terms of it.[12] But given this outlook, the requirement that physical computation essentially involve causal, semantic, etc. 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 trivialization (as it appears to), then what could possibly explain why physical computation *must* be causal, semantic, or whatever?

This explanatory problem doesn't 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, etc. 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. From this perspective, *what it is* to be an *a*-machine is just to have these features. But since this characterization comes with causal-mechanical features 'built in', so to speak, it is straightforward to explain why a physical system must have these features in order to carry out *a*-machine computations. The reason is simply that *a*-machine computations *just are* a certain kind of causal-mechanical (etc.) process. From this perspective, a causal-mechanical requirement isn't an *ad hoc* maneuver designed to save the theory from triviality, but instead reflects a basic fact about the nature of *a*-machine computations, namely, that they are a kind of causal-mechanical process.

[12] See Chalmers (1994, 341-342) for an especially clear statement of this approach.

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, ch. 7). This section explains how a suitable stand-in for 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. Medium independence entails multiple realizability: if a property or process is medium independent, then it can be realized in different physical media. Note, however, that the converse fails. *Being a corkscrew* is multiply realizable, since many different corkscrew designs might do the trick, but not medium independent. *Being a corkscrew* is a matter of interacting with a specific physical medium, namely cork.

It seems undeniable that computations are medium independent, hence multiply realizable. As Ned Block once pointed out, for instance, an AND-gate might be realized either by transistors, or by mice, string, and cheese (Block 1995). Moreover, the literature on unconventional computation is replete apparent cases in which the same computation (e.g., a sorting task) is performed by wildly different physical systems. But it's not clear that the

resemblance account can capture this apparent datum. The trouble is that there appears to be no single computational architecture, in the above sense, common to the wide variety of computing systems hypothesized by computer scientists. There is no architecture, for instance, which both silicon and murine AND gates resemble.[13]

What should the resemblance theorist make of this? The solution, I think, becomes clear once we reflect on the role played by medium independence (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, e.g., two systems compute the same logical function despite physical dissimilarities. Now, ordinarily this role is played by the alleged medium independence (multiple realizability) of computations. But if the resemblance account can supply a way to abstract from physical details, it can furnish a way to describe physical systems at the desired level of abstraction. And this, I submit, is just what is needed for computational theorizing. The rest of this section explains how this might go.

To begin, while above resemblance is characterized 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 F_i ($1 \leq i \leq m$) to degree n just in case $\sum_{i=1}^{m} Res(F_i, P, C) = n$. Moreover, by adding a coefficient to equation (1) we can discount (or boost) the contribution of a particular class of features to the overall resemblance score.[14] Doing so gives

[13] 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.
[14] Cf. Weisberg (2012).

$$x(|F_A \cap F_P| - |F_A - F_P| - |F_P - F_A|) \quad (2)$$

I will write $Res(F, P, C) = n$ as shorthand. In general, then, if $x_i \in \mathbb{R}$ $(1 \leq i \leq m)$ are coefficients, the generalized resemblance score is given by $\sum_{i=1}^{m} Res(F_i, x_i, P, C) = n$.

By appropriately choosing coefficients we can define a notion of *pattern resemblance* between systems. For instance, if F_1, F_2, F_3 are classes of physico-mechanical, syntactic, and semantic features, respective, with corresponding coefficients x_1, x_2, x_3, then by setting $x_1 = x_3 = 0$ and $x_2 = 1$ we can say that P *pattern resembles* C to degree *n* just in case $\sum_{i=1}^{3} Res(F_i, x_i, P, C) = n$, that P *pattern resembles C* simpliciter just in case P pattern resembles C to a high enough degree, and so on.

How does all this help? Medium independence is naturally thought to concern what I've 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 resemblance account can furnish a level of description appropriate for this part of computational theorizing, without abandoning the insight that inclusion of specific physico-mechanical features is central to avoiding triviality.

8. Summary

The resemblance account holds that a physical system computes to the extent that it resembles a computational architecture. This view is grounded in Turing's influential approach to thinking about computational architectures, an approach which persists in computer science today. The view is also motivated on theoretical grounds: it offers a novel and natural response to triviality arguments about computational implementation. While I haven't 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, Vol 3*, Ed. Osherson et al. MIT Press, 377-425.

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

Chalmers, D. (1996). Does a Rock Implement Every Finite-State Automaton? *Synthese, 108(3)*, 309-333.

Copeland, B. J. (1996). What is Computation? *Synthese 108*, 335-359.

Copeland, B. J. & Shagrir, O. Do Accelerating Turing Machines Compute the Uncomputable? *Minds and Machines, 21(2)*, 221-239.

Fang, W. (2017). Holistic modeling: an objection to Weisberg's weighted feature-matching account. *Synthese 194*, 1743-1764.

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

Godfrey-Smith, P. (2007). Models and Fictions in Science. *Philosophical Studies 143(1)*, 101-116.

Godfrey-Smith, P. (2009). Triviality arguments against functionalism. *Philosophical Studies, 145(2)*, 273-295.

Harris, D. M., & Harris, S. L. (2013). *Digital Design and Computer Architecture* (2nd ed.). Morgan Kaufmann.

Millhouse, T. (2017). A Simplicity Criterion for Physical Computation. *The British Journal for the Philosophy of Science, online first*.

Parker, W. (2015). Getting (even more) serious about similarity. *Biology & Philosophy, 30*, 267-276.

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

Piccinini, G. (2012). Computationalism. In *The Oxford Handbook of Philosophy of Cognitive Science*, Ed. Samuels et al. Oxford University Press, 222-249.

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

Rescorla, M. (2014). A theory of computational implementation. *Synthese*, *191*(6), 1277–1307.

Scheutz, M. (2001). Computational versus Causal Complexity. *Minds and Machines, 11(4)*, 543-566.

Schweizer, P. (2019). Triviality Arguments Reconsidered. *Minds and Machines*, DOI: 10.1007/s11023-019-09501-x.

Shagrir, O. (2001). Content, Computation and Externalism. *Mind 110(438)*, 369-400.

Shagrir, O. (2018). In defense of the semantic view of computation. *Synthese*. DOI: 10.1007/s11229-018-01921-z.

Sieg, W. (2009). On Computability. In: *Handbook of the Philosophy of Science, the Philosophy of Mathematics*, Ed. Irvine, A. Elsevier.

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

Sprevak, M. (2019). Triviality arguments about computational implementation. In *Routledge Handbook of the Computational Mind*, ed. Colombo, M. Routledge, 175 - 191.

Turing, A, M. (1936). On Computable Numbers, with an Application to the *Entscheidungsproblem. Proceedings of the London Mathematical Society, 42(1)*, 230-265.

Weisberg, M. (2012). Getting Serious about Similarity. *Philosophy of Science*, *79*(5), 785–794.