mark.sprevak@ed.ac.uk

Final version due to appear in M. Sprevak & M. Colombo (Eds.)
Routledge Handbook of the Computational Mind, London: Routledge

Triviality arguments about computational implementation

Mark Sprevak
University of Edinburgh

6 June 2018

This chapter examines four triviality arguments: arguments from Ian Hinckfuss, John Searle, Hilary Putnam, and David Chalmers. The arguments are of varying strength, scope, and plausibility. Despite the differences, I argue that they succeed in ruling out a classical 'mapping' theory of computational implementation. This theory takes isomorphism between a formal computational model and a physical system to be a sufficient condition for implementation. More sophisticated theories depart from a classical mapping account but defeat triviality arguments only at cost. Focusing our attention on performance with respect to the triviality arguments allows costs associated with a theory of implementation to be measured. I conclude by tentatively proposing a theory that aims to minimize cost: pluralism about computational implementation. Triviality arguments should be welcomed. They provide powerful and informative constraints on viable theories of computational implementation.

1 Introduction

Triviality arguments seek to show that computational implementation in physical systems is trivial in some worrisome way. A triviality argument might show that a theory of implementation attributes computations to too many physical systems, or that it attributes too many computations to those physical systems that compute, or both. Triviality arguments threaten to make trouble both for computational functionalism (the metaphysical claim that implementing a certain computation is sufficient for having a certain mental state or process) and for computational

explanation in science (the scientific practice of explaining mental and behavioral phenomena in terms of physical computations).

In this chapter, I examine four triviality arguments. These are arguments of Ian Hinckfuss, John Searle, Hilary Putnam, and David Chalmers. Their arguments may appear to give us cause for dismay. However, seen in a more positive light, they help us formulate an improved theory of computational implementation and choose between competing alternatives. If a theory of computational implementation blocks, or otherwise avoids consequences of, a triviality argument, that is a desirable property of the theory. Depending on how bad those consequences are, it may even be a necessary condition for a theory of computational implementation to be adequate. Triviality arguments mark out red lines that a theory of implementation should not cross.

In Section 2, I describe the aim of a theory of computational implementation. In Section 3, I discuss the structure and target of a triviality argument. In Section 4, I give four triviality arguments about implementation. In Section 5, I explore how far these triviality arguments reach – how trivial is 'trivial'? In Section 6, I reply to the objection that we should simply accept the conclusions of the triviality arguments. In Section 7, I describe some popular lines of response to the triviality arguments. In the Conclusion, I argue that we should learn to love the triviality arguments: they shine light on what would otherwise be murky territory for theory builders. I propose a theory of implementation that aims to minimize the cost of responding to triviality arguments: pluralism about computational implementation.

2 Computational implementation

Roughly speaking, a theory of implementation aims to describe the conditions under which a physical system does and does not implement a computation.[1] More precisely, a theory of implementation aims to tell us, for a physical system, X, and abstract computation, Y, the conditions under which 'X implements Y' is true or false. X may be any physical system – an electronic PC, a brain, or the entire universe. Y may be any abstract formal computation – a Turing machine, a cellular automaton, an artificial neural network, or a C++ program.[2] A theory of implementation tells us which conditions the physical system needs to satisfy for it

[1] The discussion in this chapter is phrased in terms of physical implementation. If there are non-physical states – qualia, ectoplasm, etc. – a theory of implementation may aim to cover their computational implementation conditions too.

[2] I do not consider here how we should characterize the class of abstract computations. For the purpose of this chapter, I assume that we know (at least roughly) which abstract formal structures count as computations.

to implement the computation. Such a theory gives us the *truth conditions* of claims about computational implementation. This serves not only as a semantic theory but also to explicate the concept (or concepts) of computational implementation as they appear in the computational sciences. A theory of implementation says what we mean by our talk of computational implementation and explains how it reduces to (hopefully straightforward) conditions regarding abstract computations and physical systems.

A theory of implementation is sometimes also described as a theory of the computational implementation *relation*. The relation is envisioned as a metaphysical bridge between the abstract realm of mathematical entities (Turing machines, finite state automata, …) and the concrete realm of physical systems (electronic PCs, brains, …). The relation either obtains or does not obtain between specific abstract entities and specific physical systems. If it obtains, then the physical system implements the computation; if not, then not. Conceived this way, a theory of implementation has an explicitly metaphysical task: to describe a special metaphysical relation within our ontology. Although fine as an informal gloss, describing the goal of a theory of computational implementation in these terms is a strategic error. It hypostatizes the implementation relation as part of the description of the problem and it lumbers us with the task of how such a relation fits into our wider ontology. These are not problems we need. The truth of claims about physical systems implementing computations does not require the existence of a special metaphysical relation between mathematical entities and physical objects. Still less does it require the existence of mathematical entities to stand in such a relation. Rendering claims about computational implementation true or false no more requires the existence of a special implementation relation than rendering claims that use the expression 'the average man' true require the existence of an average man. A special metaphysical relation of computational implementation may exist, but it is not a commitment that a theory of computational implementation should make at the outset.

Instead, a theory of implementation should aim to answer two questions about the truth conditions of *talk* of computational implementation:

COMP
: Under which conditions is it true/false that a physical system implements an abstract computation?

IDENT
: Under which conditions is it true/false that a physical system implements one abstract computation rather than another?

The first question concerns the computational status of a physical system. The

second concerns its computational identity.[3] Neither has an easy or uncontroversial answer.

In seeking to answer COMP and IDENT, a theory of implementation should try to provide a theory that is extensionally adequate. The computational sciences already (explicitly and implicitly) classify physical systems into those that compute and those that do not. They further classify members of the class of computing systems by their computational identity. Many complex judgments are made here, but a few simple ones can be briefly stated: electronic PCs compute and their plastic cases do not; my electronic PC is running Microsoft Word and not Grand Theft Auto; not every electronic PC in the world is currently running the Tor browser. These judgments would be regarded as 'obviously correct' by a scientist or engineer with a working grasp of computational implementation. Such judgments, made with confidence and receiving widespread agreement in the relevant scientific arenas, are important data points for a theory of implementation to capture.

A good theory of implementation need not capture every single data point or find every one of its claims vindicated in existing practice. We might expect some divergence between what the right theory of implementation says and current judgments made in science. This violation of extensional adequacy, however, may be small or large. At one end, it may require only minor adjustment or qualification of existing scientific practice about relatively unimportant cases. For example, a theory of implementation might claim that my PC does not, strictly speaking, compute the addition function because my PC can only store a finite number of digits. Or it might make a judgment about which current practice is largely silent; for example, that a straight wire computes the identity function. At the other extreme, a theory of implementation might say that current scientific practice is massively in error. For example, it might say that all (or nearly all) physical systems implement all (or nearly all) computations. Here, there is no question of resolving the disagreement by minor tweaks to the scientific practice. We are instead put in a bind: reject the scientific practice or reject the theory of implementation. The triviality arguments aim to demonstrate a violation of extensional adequacy of the latter kind.

It is worth noting that extensional adequacy is only one desideratum of a good theory of implementation. Other desiderata include that the theory of implementation be *explanatory*: it should explain computational implementation in terms of

[3] Attribution of multiple computational identities to a physical system is commonplace in the sciences. Sometimes the computations are related, for example, when a physical system satisfies several related formal computational descriptions (e.g. gradient descent, backpropagation, AdaGrad, and some specific machine-learning Python program). Sometimes the computations are not related, for example, when the physical system has a sufficient number of physical properties to support the implementation of multiple unrelated computational models. Anderson (2014) argues that the brain is a computing system of the latter type.

notions that are better understood than computational implementation. The theory of implementation should be *non-circular*: it should explain computational implementation in terms of notions that are not themselves explained by computational implementation. The theory should be *naturalistic*: it should not make the truth of implementation claims depend on human beliefs, interests, or values.[4] In Section 7, we will see that theories of implementation that avoid the triviality arguments often do so at the cost of giving up one or more of these desiderata.

3 Triviality arguments

Triviality arguments attack the extensional adequacy of a theory of implementation. The target theory of computational implementation in this chapter is a 'mapping' account of computational implementation. A mapping account is a theory of implementation that practitioners in the computational sciences tend to produce when questioned. It is also the starting point for almost every more sophisticated theory of implementation. A mapping account of implementation says that a sufficient condition for computational implementation is isomorphism[5] between the physical states and transitions of a physical system and the abstract states and transitions of a computation:[6]

(M) A physical system X implements a formal computation Y if there is a mapping f that maps physical states of X to abstract states of the formal computation Y, such that: for every step-wise evolution $S \rightarrow S'$ of the formalism Y, the following conditional holds: if X is in physical state s where $f(s) = S$ then X will enter physical state s' such that $f(s') = S'$

In Section 7, we will see that some alternative accounts of implementation treat M as a necessary but not a sufficient condition for computational implementation. Part of the motivation for this comes from the recognition that M is vulnerable to triviality arguments.

M is simple, clear, explanatory, non-circular, and naturalistic. M also explains why computations are multiple realizable. Different physical systems (silicon chips, vacuum tubes, brass cogs and wheels, neurons) can implement the same computation because, despite their physical differences, their various physical activities can be isomorphic to the same abstract structure.

[4] See Sprevak (2012) for more on these desiderata. See Piccinini (2015) for discussion of additional desiderata.

[5] The mapping relation is sometimes called a 'homomorphism'. The correct term depends on how one groups physical states into physical state types, which as we will see, is controversial.

[6] This condition is adapted from Chalmers (2012).

4 The arguments

It is worth noting two points before proceeding.

First, the conclusion of a triviality argument does not flow from M alone. Additional assumptions, including empirical assumptions about the typical behavior of physical systems, are required. These assumptions may be true but it is worth bearing in mind that they are unlikely to be necessary truths. In other possible worlds, implementation under M is not trivial.

Second, an uncontroversial claim about implementation is sometimes confused with the triviality claims. This claim is that any physical system could, under the right circumstances, implement a given computation. A rock could, if we were to make tally marks on it, compute the addition function. A wall could, if we were to attach electromagnets which control its physical states the right way, run Microsoft Word. That almost any physical system *could* implement an arbitrary computation is not at issue in the triviality arguments. What is at issue is whether they *do* implement that computation. One might expect there to be a difference between these two conditions: merely having the potential to implement (if certain further conditions are met) and actually implementing. (Compare: all hydrogen atoms have the potential to be fuel inside a star, but not all are). The triviality arguments aim to show that this seemingly reasonable expectation is false. (Almost) all physical systems don't merely have the *potential* to implement any given computation, they actually *are* implementing it. The threat of the triviality arguments is not a threat about universal *realizability* in some modally qualified sense, but a threat about universal *realization*.

4.1 Hinckfuss' pail

William Lycan describes a thought experiment originally suggested by Ian Hinckfuss:

> Suppose a transparent plastic pail of spring water is sitting in the sun. At the micro level, a vast seething complexity of things are going on: convection currents, frantic breeding of bacteria and other minuscule life forms, and so on. These things in turn require even more frantic activity at the molecular level to sustain them. Now is all this activity not complex enough that, simply by chance, it might realize a human program for a brief period (given suitable correlations between certain micro-events and the requisite input-, output-, and state-symbols of the program)? And if so, must the functionalist not conclude that the

> water in the pail briefly constitutes the body of a conscious being, and has thoughts and feelings and so on? Indeed, virtually any physical object under any conditions has enough activity going on within it at the molecular level that, if Hinckfuss is right about the pail of water, the functionalist quickly slips into a panpsychism that does seem obviously absurd … (Lycan, 1981, p. 39)

There is no mention of trivial implementation here, but there is clear violation of extensional adequacy. Physical systems that we do not normally think of as implementing computations (pails of water) are implementing computations and, perhaps more worryingly, their computational identity is shared with that of our brains and bodies. Hinckfuss is assuming some form of computational functionalism. He argues that the pail implements the same computation as the human brain and body, and for this reason it has the same mental properties as us.

Hinckfuss' thought experiment may be unnerving, but it is not obvious that it is fatal to M.

First, it is not obvious that the conclusion is unacceptable. Even if we follow the chain of reasoning to the end, panpsychism is arguably not an untenable position to be left holding (Chalmers, 1996b; Goff, 2017; Strawson, 2006). There are good reasons, however, to jump off the chain of reasoning before we reach the conclusion. Computational functionalists typically do not claim that every aspect of mental life supervenes on the computation that a physical system performs. Usually, they claim this only for a subset of mental life, perhaps a small subset: non-conscious aspects of mental life (Block, 1978; Chalmers, 1996b) or aspects of mental life that exclude cognitive processes like central cognition (Fodor, 2000). If one were to remove large chunks of mental life from the thought experiment, it may no longer seem absurd or even objectionable that a pail of water could, over a brief time interval, share the same computations as our brain and body.

Second, it is not clear how the thought experiment is supposed to generalize to a triviality result. What is imagined is the physical *possibility* of an unusual implementation. We need a reason to think that such an implementation is actual and the norm in order to have a triviality result. There are plenty of worrisome physical possibilities: spontaneous unmixing of scrambled eggs, movement of all molecules of air in the room into one corner, quantum tunneling of my legs through the floor. Science and engineering generally assume that these events will not happen. It is safe to do so because, even though the events are possible, they are extraordinarily unlikely. Perhaps a pail of water implementing the same computation as a human brain and body is like that. If so, it is hard to see why it would provide a reason to think that computational implementation is trivial.

4.2 Searle's wall

John Searle describes a different thought experiment:

> [I]t is hard to see how to avoid the following results: 1. For any object there is some description of that object such that under that description the object is a digital computer. 2. For any program and for any sufficiently complex object, there is some description of the object under which it is implementing the program. Thus for example the wall behind my back is right now implementing the Wordstar program, because there is some pattern of molecule movements that is isomorphic with the formal structure of Wordstar. But if the wall is implementing Wordstar, then if it is a big enough wall it is implementing any program, including any program implemented in the brain … (Searle, 1992, pp. 208–209)[7]

This suggests a different argument that helps us bridge the gap between the mere possibility of an unusual implementation and a triviality result. Consider, according to M, why Searle's electronic PC implements WordStar. Inside his PC are many microscopic physical changes: changes in electrical, thermal, vibrational, and gravitational state of the physical parts of his PC. His PC implements WordStar because among these physical changes is one set of changes – the set of electrical changes – that has a structure that is isomorphic to the formal structure of WordStar. Searle claims that the same is true of his wall. Inside the wall are many microscopic physical changes: there are atoms and molecules undergoing electrical, thermal, vibrational, and gravitational changes. Searle suggests there are *so many* patterns of physical activity inside his wall that there is certain to be at least one pattern with a structure isomorphic to the formal structure of WordStar. Therefore, just like Searle's PC, his wall implements WordStar. The same reasoning applies to other computations and to other physical systems provided they are 'sufficiently complex'.

Again, there are problems with this argument that should leave one unpersuaded.

First, the restriction to physical systems that are 'sufficiently complex' is underspecified. Searle's idea seems roughly that the system should have enough patterns of physical activity for it to implement any computation. But if 'sufficiently complex' just means having enough patterns, that is only a restatement of what needs to be shown: that the physical system should be subject to trivial implementation under M. We need an independent characterization of the class of physical systems affected by triviality. We need some idea how large this class is, which systems it contains,

[7] Although Searle's argument is phrased in terms of programs, I will understand it here as covering any abstract computation; for discussion, see Section 5.

and whether its members are relevant to scientific practice. If one characterizes 'sufficiently complex' in terms of a system's physical size – as Searle seems to hint – small enough physical systems relevant to scientific practice may squeak through as immune to triviality.

Second, and more worryingly, why should we believe, even of the largest physical systems, that there are always a sufficient number of patterns of physical activity to make implementation trivial? One might agree with Searle that a wall contains many patterns of activity. But one might not accept that it contains *every* pattern of physical activity – or that it contains any single specific pattern, such as one isomorphic to WordStar. How can we be *sure*, as Searle says, that the wall contains this particular pattern? Searle does not say much about this, but it is crucial to establish his conclusion. He needs to show, not just that there are *many* patterns of physical activity, but that we should be *certain* there will be a pattern isomorphic to any abstract computation we like.

4.3 Putnam's rock

In the Appendix of *Representation and Reality* (1988), Hilary Putnam presents an argument that addresses all the preceding concerns. While Searle and Hinckfuss aim to show that a trivializing mapping exists but do not actually show us the mapping, Putnam provides a method for finding the relevant mapping. Given a physical system, X, and formal computation, Y, Putnam demonstrates how to map X's states to Y's states so that X implements Y according to M. Putnam also provides an informative characterization of the class of physical systems vulnerable to trivial implementation.

Putnam restricts his triviality argument to inputless finite state automata (FSAs).[8] In Section 5, we will see how to extend his argument to other types of computer. Putnam says that the class of physical systems that are vulnerable to his triviality argument is the class of physical systems that are 'open'. An 'open' physical system is one that is not isolated from, and therefore is in causal interaction with, its environment. Nearly all physical systems in which we are interested are open in this sense.[9] To illustrate his argument Putnam chooses a simple inputless FSA that transits between two abstract states, $A \rightarrow B \rightarrow A \rightarrow B$. He argues as follows.

Pick any open physical system (say, a rock) and any time interval, t_0 to t_n. Consider the 'phase space' of the rock over this time interval. The phase space is a representation of every one of the rock's physical parameters, including the physical parameters

[8] See Hopcroft and Ullman (1979); Sudkamp (1998) for a description of FSAs.

[9] One possible exception is the entire universe (Copeland, Shagrir and Sprevak, 2018).

of the rock's constituent atoms, molecules, and other microscopic parts.[10] Over time, the rock will trace a path through its phase space as its physical parameters change. The rock's physical parameters will change due to endogenous physical causes (its atoms changing state, vibrations, atomic decay, etc.), and because of external causal influences (gravitational, electromagnetic, vibrational influences, etc.). Putnam argues that some external influences are certain to play the role of 'clocks' for the rock: due to these influences the rock will not return to precisely the same set of values of its physical parameters in the time interval. Putnam calls this the 'Principle of Noncyclical Behavior'. Putnam argues that this principle is likely to be true of any open physical system.[11]

Consider the rock's phase-space trajectory from t_0 to t_n. By the Principle of Noncyclical Behavior, this path will not cross itself at any point. Putnam assumes that the path through phase space is also continuous in time: the path passes through a different point in phase space at each moment in time. Putnam calls this the 'Principle of Continuity'. Provided these two principles hold, each point in the rock's trajectory falls within a unique region of its phase space. Putnam divides the rock's phase space into four regions through which the rock's state travels during the time interval, and he labels these r_1, r_2, r_3, r_4. These regions describe the rock's state during four, equally spaced time intervals between t_0 and t_n. Regions in phase space are groups of possible physical states. A region in phase space therefore defines a possible physical state *type* for the system. Consequently, we can describe the rock's physical state during the time interval in the following way: in the first time interval, the rock is in the physical state type defined by r_1; in the second, it is in the physical state type defined by r_2; in the third, in the physical state type defined by r_3; and in the fourth, in the physical state type defined by r_4.

Using regions of phase space to characterize a system's physical state type is a powerful tool. It reveals that the rock undergoes multiple changes in its physical state type during the time interval. One set of such changes is this: $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_4$. But, Putnam observes, it is not the only set of changes. The rock also undergoes the following changes: $r_1 \vee r_3 \rightarrow r_2 \vee r_4 \rightarrow r_1 \vee r_3 \rightarrow r_2 \vee r_4$. In other words, as well as traveling through four neighboring regions of its phase space (r_1, r_2, r_3, r_4), the rock oscillates between two disjoined regions of its phase space ($r_1 \vee r_3$ and $r_2 \vee r_4$). It is worth stressing there is nothing inherently objectionable about identifying a physical state type of a system with a disjunction of regions of that system's phase

[10] Putnam only considers the classical properties of physical systems. However, it is not clear how quantum mechanical properties would constrain implementation under M.

[11] Poincaré's recurrence theorem says that physical systems, if they satisfy certain conditions, return to a total physical state arbitrarily close to that of their initial conditions after a sufficiently long time. However, this theorem only applies to closed systems and the Poincaré recurrence period for an open system is likely to be extremely long – longer than the lifetime of the universe.

space. Many physical state types used for legitimate implementation are defined this way: for example, net thermal energy and net electric charge. The physical states that implement the formal states of electronic PCs are highly disjunctive in this sense: they are diverse configurations of electrical signals that could occur in multiple electronic components scattered throughout the machine. Putnam maps $r_1 \vee r_3$ to computational state A and $r_2 \vee r_4$ to computational state B. We now have an isomorphism between the physical states and transitions of the rock and the formal states and transitions of the FSA. According to M, the rock implements the FSA. The same reasoning applies to other physical systems and other inputless FSAs. Every open physical system implements every inputless FSA under M.

There are three common objections to Putnam's argument.

First, Putnam's argument assumes that any disjunction of regions in phase space defines a legitimate physical state type for implementing a computational state. We have already seen there is nothing inherently objectionable about mapping a disjunction of phase-space regions to a single computational state. However, it is less clear whether mapping from an *arbitrary* disjunction of phase-space regions is permissible. Critics of Putnam suspect that some disjunctions of phase space are 'legitimate' candidates to implement a computational state, others are not. And this suspicion seems reasonable as far as it goes. The problem is that it is hard to say *which* disjunctions are illegitimate and *why*. As we will see in Section 7, answering this question turns out to be the central point on which theories that aim to replace M disagree. Which further conditions – semantic, teleological, causal, natural kind, pragmatic, etc. – should a group of physical states satisfy to count as a 'legitimate' implementer of a computational state? Without an answer to this question, the current objection to arbitrary disjunctions lacks bite.

Second, many computations have inputs (and outputs), but Putnam's triviality argument only applies to inputless FSAs. Putnam's response is that although *physically specified* inputs and outputs partially constrain implementation, the computational states that lie between input and output are still open to his triviality argument. An appeal to inputs and outputs would, at best, collapse computational functionalism into a form of behaviorism – not good. A separate reply, developed by Godfrey-Smith (2009) and Sprevak (2012), is that the inputs and outputs of computations are rarely constrained in terms of having a specific physical nature. Inputs and outputs are typically described as abstract states within the abstract computational formalism (e.g. as numerals, characters, strings, or activation values). These abstract states could, in principle, be implemented by any physical state type (electrical signals, vibrations in the air, turns of a brass cog). This removes from us the luxury of having physically constrained inputs and outputs and returns us to finding mappings between abstract states and physical states. As abstracts states, inputs and

outputs seem just as vulnerable to Putnam's triviality argument as internal states.[12]

Third, a computer should not be destroyed by arbitrarily small physical changes. But there is no guarantee that the mappings described by Putnam would hold under (even slightly) different physical conditions, e.g. if one photon more had hit the rock before t_0. The mapping given by Putnam also only covers FSA states that actually occur during the computation. It does not cover abstract states or transitions that could have occurred but did not. Computers, especially complicated ones, often have states and transitions that are not instantiated on a given run but could have been under other conditions.[13] On the basis of these two points, one might propose two counterfactual conditions that a genuine implementation of a computation should satisfy. First, the physical transitions identified by the mapping should be 'reliable': they should not fail under arbitrarily small changes. Second, the mapping should be 'exhaustive': it should map every abstract state and transition of the computer to a physical state and transition. Putnam's argument only tells us about what *actually* happens to the physical system, so it cannot guarantee that either of these counterfactual conditions are satisfied.

4.4 Chalmers' clock and dial

The counterfactual objection to Putnam's triviality argument was once believed to be fatal (Block, 1995; Chalmers, 1995; Chrisley, 1995; Maudlin, 1989). Chalmers (1996a) showed that it is not. A more sophisticated version of Putnam's construction can work against a counterfactually strengthened version of M.

Chalmers defines a 'clock' as a component of the physical system that reliably transits through a sequence of physical states over the time interval.[14] He defines a 'dial' as a physical component of the system with an arbitrary number of physical states such that if it is put into one of those states it stays in that state during the time interval. Chalmers' counterfactually strengthened triviality result is that every physical system with a clock and a dial implements every inputless FSA.

This argument involves a similar construction to Putnam's, but over possible, as well as actual, trajectories in phase space. In one respect the construction is simpler, since the only states that need to be considered are the physical system's clock and dial; the other physical states can be safely ignored. Chalmers' strategy is to identify a mapping between each formal FSA state and a disjunction of physical states $[i, j]$

[12]For more on this point, see discussion of the 'transducer layer' in Godfrey-Smith (2009) and 'strong' and 'weak' input–output constraints in Sprevak (2012).

[13]See Maudlin (1989) for an illustration of this point with Turing machines.

[14]Chalmers' 'clock' is different from the clocks in Putnam's argument. Chalmers' clocks are inside the system and they change their physical state in a counterfactually robust way.

of the implementing system, where i corresponds to a numbered clock state, and j to a numbered dial state, and show that the relevant physical states stand in the right counterfactual relations to each other.

Suppose the system starts in physical state $[1, j]$. It will reliably transit to $[2, j]$, $[3, j]$, and so on, as the clock progresses. Suppose that the system starts its actual run in dial state 1. The start state of the FSA can then be mapped to $[1,1]$, and the subsequent abstract states of the FSA to $[2,1]$, $[3,1]$, and so on. At the end of this mapping process, if some FSA states have not come up, then choose one of those formal states as the new start state of the FSA and map $[1, 2]$ to it. Then pair physical states $[2,2], [3,2]$, and so on with the abstract states that follow in the evolution of the FSA. Continue until all the un-manifested states of the FSA have been considered. Now, for each abstract state of the FSA, we have a non-empty set of associated physical states $\{[i_1, j_1], [i_2, j_2], \ldots, [i_n, j_n]\}$. Map the disjunction of these states to each FSA state. The resulting mapping between physical and formal states satisfies the counterfactually strengthened version of M.

Chalmers argues that almost all open physical systems have a clock and a dial. If for any reason a physical system does not have a clock or a dial, they can be added by attaching a watch to the physical system. If trivial implementation can be achieved simply by adding a watch to a rock, something has clearly gone wrong with the account of implementation.

5 The reach of the arguments

None of the preceding arguments show that every physical system implements every computation. In what sense do they show that implementation is 'trivial'? Broadly, we can measure their approximation towards unbounded triviality along three dimensions: time, physical systems, and abstract computations.

First, time. One reason why Hinckfuss' argument seems weak is that trivial implementation is only claimed for a brief time period and even then conditional on some lucky accident. Searle's argument tries to pump our intuitions to raise the chance of trivial implementation happening more often and over a longer time interval, but it gives no proof. The arguments of Putnam and Chalmers are unrestricted in their time interval. They also provide a high degree of certainty (conditional on various empirical assumptions) that trivial implementation occurs in that time interval. The time interval (t_0 to t_n) could, in principle, be as long or short as one likes: one second, one year, or 10^9 years (provided the physical system is still around).

Second, physical systems. Hinckfuss and Searle hint that only macroscopic systems

(like walls or pails of water) are vulnerable to trivial implementation. On Putnam's account, a physical system is vulnerable to trivial implementation provided it satisfies the Principles of Noncyclical Behavior and Continuity. On Chalmers' account, a physical system is vulnerable to trivial implementation provided it has a clock and a dial. There is no reason why the conditions described by Putnam and Chalmers should not be satisfied by microscopic as well as macroscopic systems. A few atoms, or even a single atom, could vary its state in such a way as to satisfy the conditions. Thus, Putnam's and Chalmers' arguments appear to threaten not just macroscopic systems but also systems that are small or simple in Hinckfuss' and Searle's terms.

Third, abstract computations. Which abstract computations are trivially implemented? Familiar abstract computations include FSAs, Turing machines, register machines, pushdown automata, cellular automata, random access machines, and artificial neural networks. These, however, are just a tiny sample from the vast population of possible computational architectures. Which of these abstract computers are subject to trivial implementation? We can put an upper bound on the triviality arguments. At least some computers are impossible to implement in a physical system. Plausible examples of such include infinitely accelerating computers (for which each step takes place faster than the previous step), computers that use infinite time or storage, and computers that manipulate real-valued quantities with unlimited precision.[15] These formalisms are normally regarded as *notional* computers and studied for their abstract properties alone (for example, proving which sentences in the arithmetic hierarchy they decide). They cannot be, and they are not intended to be, physically implemented. The triviality arguments thus have a limit. Not every abstract computation is implemented because not every abstract computation is implementable.

Searle claims that all 'programs' are trivially implemented. However, it is hard to be sure what he means by this. In computer science, a 'program' refers to a piece of data that plays a certain role within a certain kind of computer, a programmable computer. Programmable computers form a relatively small subclass of the population of all abstract computers (most Turing machines are not programmable). Physical systems that implements programs must *ipso facto* implement a programmable computer. Searle thinks that his triviality result applies to a wider range of computers than just the programmable ones. Therefore, he does not seem to have this standard definition of 'program' in mind. It appears he has a more generic meaning, roughly synonymous with *algorithm* or *computational method* in mind. But then his argument is unrestricted with regard to formal computations, and we have seen that this is not right.

Chalmers and Putnam restrict their triviality claims to only one type of computer:

[15] Blum et al. (1998); Copeland (2002); Piccinini (2011).

inputless FSAs. We have seen how to extend their claims to FSAs with inputs and outputs. What about other abstract computers? Below, I argue that their triviality claims generalize, beyond FSAs, to all abstract computers with finite storage. More to the point, their claims generalize to every *physically implementable computation*.

The argument for this generalization relies on three premises:

1. There is an isomorphism between the physical activity of any open physical system, A, (with a clock and dial) and any FSA, B.
2. If there is an isomorphism between A and B, and an isomorphism between B and C, then there is an isomorphism between A and C.
3. There is an isomorphism between any computer with finite storage, C, and an FSA, B.

The first premise is the conclusion of the triviality arguments. The second premise states the formal property of transitivity of the isomorphism relation. The third premise requires some justification, which is given below. The argument runs as follows. Pick any open physical system, A, and any abstract computer with finite storage, C. By premise 3, there is an isomorphism between C and some FSA, B. By premise 1, there is an isomorphism between A and FSA B. By premise 2, there is an isomorphism between A and C. Hence, A implements C.

Justifying premise 3 is not hard. The state of any abstract computer with finite storage, C, can be redescribed by a single monolithic 'state' variable, X. The state variable, X, enumerates every possible way in which that abstract computer could be. For example, each value of X (x_1, x_2, …, x_n) may denote a possible tape and head state combination for a Turing machine, a possible grid pattern for cells for a cellular automaton, and a possible setting of activation levels and weights for an artificial neural network. Since C has finite storage, its monolithic state variable X can only take a finite number of possible values – otherwise, one could exploit it to gain a computer with infinite storage. The next value of C's giant state variable is determined by its current value.[16] For example, the next value (tape and head state combination) of a Turing machine is determined by its current value (tape

[16]I focus here only on deterministic computers. It is worth considering how the argument might generalize further to non-deterministic computers. This would require proving a version of the Putnam/Chalmers triviality result for non-deterministic FSAs. One way to do this would be to augment their assumptions to require that the physical system contain some random physical element. One could then partition the random element's physical state into physical types with appropriate probabilities for the FSA's states and map abstract states of the non-deterministic FSA to appropriate disjunctions of triples of states of the clock, dial, and random element. What would remain is to prove a non-deterministic analogue of premise 3. This would require showing that there is an isomorphism between any non-deterministic computer with finite storage, C, and some non-deterministic FSA, B. The argument could follow the line of reasoning given in the main text for premise 3.

and head state combination). The behavior of C can then be fully described by a directed graph (perhaps a large one) of C's state variable and possible transitions between its values (and any inputs or outputs where relevant). This directed graph specifies an FSA, B. Conversely, FSA B, under the scheme for labeling of ways C could be with values of a giant state variable, fully describes C. This establishes an isomorphism between B and C, which is what premise 3 requires.

The class of computers with finite storage is a large one. There are good reasons for thinking it is the class of computers that are physically implementable.[17] Any physical implementation of a computer only has access to finite physical resources to do its work. These resources may be plentiful (e.g. all the energy and time in the system's forward light cone), but they are finite. This means that any physical implementation of a computer will eventually fail on a large enough input. For example, any PC one builds will eventually fail to recognize $\{a^n b^n \mid n \geq 0\}$ for large enough n. A physical system only has finite resources and so it only has finite storage. Consequently, a physical system can only implement an abstract computer with finite storage. The upper limit on storage is likely to be large – large enough that it is often ignored. We often *idealize* our electronic PCs by treating them as if they had unlimited storage. But strictly speaking, only implementation of abstract computers with finite storage is possible.

The preceding points indicate that the triviality arguments have a very far reach indeed. There is no space between the computers that *can* be implemented and those that are implemented *trivially*. The triviality arguments are effectively unrestricted in this respect. If a computation is physically implementable, it is trivially implemented.[18]

6 What is so bad about trivial implementation?

One might wonder whether we should just accept that computational implementation is trivial. After all, M has many other virtues: it is simple, clear, explanatory, and makes the truth of claims about computational implementation objective. Maybe these are worth the cost of accepting that implementation is trivial.

M remains problematic for at least three reasons:

1. M's violation of extensional adequacy needs explanation.

[17] See Rabin and Scott (1959).

[18] Chalmers (2012) effectively blocks this consequence for his preferred architecture of combinatorial state automata by modifying M to require that computational states be implemented by 'independent' components of a physical system. See Section 7.

2. Combined with computational functionalism, M entails panpsychism.
3. M drains computational explanation in the sciences of their power.

All of these considerations have some force, but I will argue that the third consideration is the most powerful.

The premise of the current response is that violating extensional adequacy is a price worth paying to keep M. Given this, it is not immediately obvious how pointing (again) to violation of extensional adequacy has any further persuasive force. Violating extensional adequacy may be bad, and provide good grounds to reject M, but the defender of the current response already factors this cost in. She is willing to pay it to keep M. The question is whether there are further problems that she needs to address as a consequence. One such problem is to explain *why* M diverges so much from the judgments of experts in the computational sciences and what we should do about this. A hard-nosed revisionist might suggest that we have discovered that the experts in the computational sciences were wrong and they should revise their claims about computational implementation accordingly (e.g. start saying that PC cases do run Grand Theft Auto). A less revisionary response would be to explain away the disagreement with M by appeal to pragmatic factors. For example, one might say that computational implementation is, strictly speaking, trivial but this is not displayed in scientific practice – and need not be so displayed – because scientists attend to, and talk about, only the implementations that interest them. Other implementations are objectively there, but they are not of interest and so they are not discussed. Talk of 'the' computations that a physical system implements should be interpreted as a pragmatic shorthand to direct our attention to the implementations that interest us and away from implementations that do not.[19] Whichever response one pursues, it is clear that work remains for a defender of M to do here.

The second consideration raises problems for M but only conditional on computational functionalism being true. Computational functionalism claims that if a physical system implements certain computations then it has certain mental states. According to the triviality arguments, nearly every physical system implements nearly every computation. Consequently, nearly every physical system has nearly every mental state. Computational functionalism was popular in the 1970s and 1980s (Block, 1978; Fodor, 1987; Putnam, 1975). But it is not clear how widely the view is held today. Cognitive science appeals to physical computations to explain behavior

[19] A parallel could be drawn with how Lewis (1970) and Mackie (1974) treat causes and background conditions. According to Lewis and Mackie, there is no objective distinction between a cause and a background condition. Lewis and Mackie diffuse disagreement between this and our everyday talk (which distinguishes between causes and background conditions) by appeal to the pragmatics of causal discourse. Talk of causes and background conditions is merely a pragmatic device to direct our attention to those causes that interest us.

and mental processes. However, contemporary cognitive science is usually silent on the metaphysical question of whether that computation is sufficient for mental life. Consequently, it is unclear how much weight, if any, to accord consideration 2 as a source of problems for M.

The third consideration should cause more concern. Cognitive science explains, predicts, and describes human behavior and mental processes in terms of computations. Decision making, categorization, inference, and belief revision are explained by the brain implementing distinctive computations. Cognitive science explains particular aspects of that behavior or mental processing (behavioral or psychological 'effects') by appeal to the brain implementing specific computations. Specific effects occur because the brain implements one computation rather than another. This explanatory methodology is threatened by the triviality results. If implementation is trivial, then no distinctive computations are implemented by the brain. The brain, like almost every other physical system, implements almost every computation. Explaining psychological effects by appeal to distinctive computations cannot work because there are no distinctive physical computations. Granted that computational explanation is important in cognitive science, accepting the triviality results while continuing to pursue cognitive science looks hard.

7 How to respond to triviality arguments

There is widespread agreement that unmodified M makes computational implementation trivial. There is also widespread agreement that trivial implementation is incompatible with the explanatory goals of cognitive science. Unfortunately, there is no agreement about what to do about it. Competing proposals try to modify M in different ways in order to block the triviality arguments. These proposals tend to fall into four broad categories.

Physical/causal structure proposals. These proposals dive deeper into the physical and causal nature of physical systems to find differences there that matter for implementation. Chalmers (1996a) claims that physical states that implement distinct abstract states should be 'independent' components of the physical system. What 'independent' means here is not entirely clear and Chalmers suggests that it means the physical states occupy different spatial regions. Unfortunately, as he acknowledges, this condition is too strong; it rules out legitimate implementations (Sprevak, 2012). As an alternative, Godfrey-Smith (2009) suggests that physical states that implement the same abstract state should be 'physically similar' to each other and 'physically different' from those that implement different abstract states. Again, the meaning of 'physical similarity' is unclear but in any case the condition looks too strong. Two

physical states within the same computation that implement different abstract states are often not just physically similar, they are physically identical (this happens in virtualized machines and in an ordinary PC when memory is reorganized during a memory remap (Sprevak, forthcoming)).

Semantic proposals. Some physical computers manipulate representations. For example, an electronic PC manipulates electrical signals that represent numbers when it computes an answer to an addition problem. Semantic accounts of implementation suggest that only those physical systems that manipulate representations are candidates to implement computations. As Fodor said, '[t]here is no computation without representation' (1981, p. 180). The computational identity of a physical system is determined by which representations it manipulates and how it manipulates them (Sprevak, 2010; Shagrir, 2010). Walls and rocks do not implement computations because they do not manipulate representations, or if they do, they do not manipulate them in the right way. Like the physical/causal proposals, semantic proposals face the worry that they are too strong. Not every physical computation manipulates representational content. And not every computation that manipulates representations has its computational identity fixed by how it manipulates representational content. Some computers, like parsers, appear to be 'purely syntactic' and do not manipulate representations at all (Piccinini, 2008).

Teleological proposals. Physical parts of a computer often have teleological functions related to the computation. For example, a memory unit inside a PC has the function of storing physical symbols. Teleological accounts of implementation suggest that only physical systems with parts with the right teleological functions are candidates to implement computations. The computational identity of a physical system is determined by the teleological functions of its parts and how those parts are related (Bontly, 1998; Piccinini, 2015). Walls and rocks do not implement computations because their parts do not have the right teleological functions for computing. Like the physical/causal and semantic proposals, teleological proposals face the worry that they are too strong. Teleological functions are essentially relational properties of a physical system: they depend on the physical system satisfying conditions about how users treat it, the intentions of designers, its evolutionary history, or something similar. Yet an intrinsic physical duplicate of a computer – one that lacks these relations – still seems capable of computing. Having the right teleological function (in more than a purely nominal sense) is not necessary to implement a computation. A further wrinkle is that naturalistic accounts of teleological function suggest that natural teleological functions are sparsely instantiated and have a large degree of indeterminacy in their identity (Burge, 2010; Shea, 2013). It is unlikely there is a rich and determinate enough set of natural teleological functions to ground the computational claims of cognitive science.

Anti-realist proposals. Physical computers are often physical systems that are useful to us, or salient to us, as computers in light of our human interests, values, and cognitive and perceptual machinery. An electronic PC is a convenient and useful means for us to implement a word processing program, in a way that a rock is not. Anti-realist accounts suggest that it is because certain physical systems offer a useful means for us to implement a computation that they implement that computation. The computational identity of a physical system is determined how the physical properties of the system fit with our human-centric interests and perceptual and cognitive abilities. Computers have physical activity that we can observe, predict, or manipulate in appropriate ways. Walls and rocks do not implement computations, notwithstanding their isomorphism to abstract computations, because we cannot conveniently exploit those isomorphisms to do computational work. The relevant isomorphisms are 'there' but essentially worthless to us because we do not know how to control the relevant physical states that map to inputs or outputs or how to predict or control the relevant intermediate physical states. 'Usefulness' here is not a simple matter of subjective preference. One person wanting something to implement a computation shouldn't make it so. An anti-realist account of implementation would likely appeal to what is generally useful, explanatory, perspicuous, informative, or practicable in science.

Anti-realism is ultimately the route down which Searle and Putnam wish to push us. Their goal is to show us, not that computational implementation is trivial, but that computational implementation *unconstrained by human interests and values* is trivial. Unlike physical/causal proposals, semantic proposals, and teleological proposals, anti-realism gets the extension of computational implementation more or less correct. It fits closely with the judgments associated with COMP and IDENT in existing computational practice. However, it has a cost. Anti-realism gives up on the *naturalistic* desideratum of a theory of computational implementation. Anti-realism guarantees that computational implementation is not a natural or objective matter of fact. Indeed, to the extent that implementation is non-trivial, computational implementation is 100 percent a function of our (perhaps a broad 'our' referencing a scientific community) interests, values, and perceptual and cognitive abilities. Scientific explanations that appeal to computational implementation would have to acknowledge this. They explain only by invoking a mind-dependent, observer-relative notion. Given that computational explanation in cognitive science is valued as a way to understand the mind in objective, non-mind-dependent terms, this should cause worry.

8 Conclusion

Triviality arguments teach us that unmodified M is unacceptable. We have seen little agreement about how to respond to them. Different theorists plump for different options, with different costs, in order to block the triviality arguments. It is unfortunate that there is so little agreement here. But on the positive side, it is good that triviality arguments provide significant, hard-to-meet constraints on theorizing. Without constraints that push back against us, we would be theorizing about implementation in the dark. The triviality results are not an embarrassment. They provide a valuable source of information that can guide us towards improved theories of computational implementation.

What would be a fair price to pay to avoid the triviality arguments? Each option canvassed in Section 7 incurs a cost. Typical costs were that the theory either gets the extension of computational implementation wrong or it gives up on the naturalistic desideratum of a theory of implementation. Before closing, I wish to suggest that we can minimize costs by adopting a form of pluralism about computational implementation. Each of the constraints described in Section 7 has some truth in it: it describes how implementation is constrained in *some* circumstances. The mistake the accounts make is that they say implementation is constrained in the same way in *every* circumstance. Scientific practice allows for more diversity than that: implementation may be sometimes constrained one way, sometimes in another. The relevant constraints in different circumstances could invoke physical relations, semantic content, teleological functions, or human-centric interests and values. The problem we saw in Section 7 was that appealing to a single constraint does not do the work in all circumstances (or at least not without considerable cost). A pluralist account of computational implementation says that the factor that constraints implementation varies between contexts. Whatever constraint is present (without cost!) in that context – be it physical, semantic, teleological, or pragmatic – does the work of constraining implementation in that context. Absent such constraints, implementation is of course trivial. But in any context where implementation is not trivial, at least one such constraint kicks in. If more than one constraint is available, then conflicting claims about implementation may arise (e.g. the system both implements a computation and does not implement the computation relative to different standards). This may result in groups of researchers talking past each other, or more commonly, acknowledgment that there is a legitimate sense in which the relevant physical system does and does not perform the computation. A pluralist approach offers a way to provide answers to COMP and IDENT that reflect the nuances of scientific practice and respond to the triviality arguments while avoiding the cost of a one-size-fits-all approach.

Acknowledgements

I would like to thank Matteo Colombo for helpful comments on a previous version of this chapter.

Bibliography

Anderson, M. L. (2014). *After Phrenology: Neural Reuse and the Interactive Brain*. Cambridge, MA: MIT Press.

Block, N. (1978). "Troubles with Functionalism". In: *Perception and Cognition: Issues in the Foundations of Psychology, Minnesota Studies in the Philosophy of Science*. Ed. by C. W. Savage. Vol. 9. Minneapolis: University of Minnesota Press.

— (1995). "The mind as the software of the brain". In: *An Invitation to Cognitive Science, Vol. 3, Thinking*. Ed. by E. E. Smith and D. N. Osherson. Cambridge, MA: MIT Press, pp. 377–425.

Blum, L., F. Cucker, M. Shub and S. Smale (1998). *Complexity and Real Computation*. New York, NY: Springer.

Bontly, T. (1998). "Individualism and the nature of syntactic states". In: *The British Journal for the Philosophy of Science* 49, pp. 557–574.

Burge, T. (2010). *Origins of Objectivity*. Oxford: Oxford University Press.

Chalmers, D. J. (1995). "On implementing a computation". In: *Minds and Machines* 4, pp. 391–402.

— (1996a). "Does a rock implement every finite-state automaton". In: *Synthese* 108, pp. 309–333.

— (1996b). *The Conscious Mind*. Oxford: Oxford University Press.

— (2012). "A computational foundation for the study of cognition". In: *Journal of Cognitive Science* 12, pp. 323–357.

Chrisley, R. L. (1995). "Why everything doesn't realize every computation". In: *Minds and Machines* 4, pp. 310–333.

Copeland, B. J. (2002). "Accelerating Turing machines". In: *Minds and Machines* 12, pp. 281–301.

Copeland, B. J., O. Shagrir and M. Sprevak (2018). "Zuse's Thesis, Gandy's Thesis, and Penrose's Thesis". In: *Computational Perspectives on Physics, Physical Perspectives*

on Computation. Ed. by M. Cuffano and S. Fletcher. Cambridge: Cambridge University Press.

Fodor, J. A. (1981). "The mind–body problem". In: *Scientific American* 244, pp. 114–125.

— (1987). *Psychosemantics*. Cambridge, MA: MIT Press.

— (2000). *The Mind Doesn't Work That Way*. Cambridge, MA: MIT Press.

Godfrey-Smith, P. (2009). "Triviality arguments against functionalism". In: *Philosophical Studies* 145, pp. 273–295.

Goff, P. (2017). *Consciousness and Fundamental Reality*. Oxford: Oxford University Press.

Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. 2nd ed. Reading, MA: Addison-Wesley.

Lewis, D. K. (1970). "Causation". In: *The Journal of Philosophy* 70, pp. 556–567.

Lycan, W. G. (1981). "Form, Function, and Feel". In: *The Journal of Philosophy* 78, pp. 24–50.

Mackie, J. L. (1974). *The Cement of the Universe*. Oxford: Oxford University Press.

Maudlin, T. (1989). "Computation and consciousness". In: *The Journal of Philosophy* 86, pp. 407–432.

Piccinini, G. (2008). "Computation without representation". In: *Philosophical Studies* 137, pp. 205–241.

— (2011). "The physical Church–Turing Thesis: Modest or Bold?" In: *The British Journal for the Philosophy of Science* 62, pp. 733–769.

— (2015). *The Nature of Computation*. Oxford: Oxford University Press.

Putnam, H. (1975). "The mental life of some machines". In: *Mind, Language and Reality, Philosophical Papers, volume 2*. Cambridge: Cambridge University Press, pp. 408–428.

— (1988). *Representation and Reality*. Cambridge, MA: MIT Press.

Rabin, M. O. and D. Scott (1959). "Finite automata and their decision problems". In: *IBM Journal of Research and Development* 3, pp. 114–125.

Searle, J. R. (1992). *The Rediscovery of the Mind*. Cambridge, MA: MIT Press.

Shagrir, O. (2010). "Brains as analog-model computers". In: *Studies in History and Philosophy of Science* 41, pp. 271–279.

Shea, N. (2013). "Naturalising representational content". In: *Philosophy Compass* 8, pp. 496–509.

Sprevak, M. (forthcoming). "Review of Susan Schneider, *The Language of Thought: A New Philosophical Direction*". In: *Mind*.

— (2010). "Computation, individuation, and the received view on representation". In: *Studies in History and Philosophy of Science* 41, pp. 260–270.

— (2012). "Three challenges to Chalmers on computational implementation". In: *Journal of Cognitive Science* 13, pp. 107–143.

Strawson, P. F. (2006). "Realistic monism: Why physicalism entails panpsychism". In: *Journal of Consciousness Studies* 13, pp. 3–31.

Sudkamp, T. A (1998). *Languages and Machines*. 2nd ed. Reading, MA: Addison-Wesley.