Forthcoming in M. Colombo & M. Sprevak (Eds.) *Routledge Handbook of the Computational Mind*, London: Routledge

mark.sprevak@ed.ac.uk

Triviality arguments about computational implementation

Mark Sprevak
University of Edinburgh

24 March 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 minimise 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 behavioural 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 (if sound) might appear to be cause for dismay. But seen in a more positive light, their arguments help us. They help us to formulate an improved theory of computational implementation and choose between competing alternatives. If a theory of computational implementation blocks, or otherwise avoids bad consequences of, a triviality argument, that is a desirable property of the theory. Depending on how bad the consequences, it may even be a necessary condition for an adequate theory of computational implementation. Triviality arguments mark out red lines that a theory of implementation should not cross.

In Section 2, I describe the form a theory of computational implementation should take. In Section 3, I discuss the structure and target of a triviality argument. In Section 4, I give four triviality arguments. In Section 5, I explore how far these triviality arguments range – how trivial is 'trivial' here? In Section 6, I reply to the objection that we should simply accept the conclusion of the triviality arguments. In Section 7, I describe some common responses to the triviality arguments. In the Conclusion, I argue that we should learn to love the triviality arguments: they shine a light on what would otherwise be dark and murky territory for theory builders. I tentatively propose a theory of implementation that aims to minimise costs associated with 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, it 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 model that counts as computational – a Turing machine, a cellular automaton, an artificial neural network, or a C++ program.[2] A theory of implementation tells

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

[2] I do not worry here about how we should characterise the class of abstract computational models. For the purpose of this chapter, I assume that we know (or at least roughly know) which abstract formal systems count as computational. The examples given in the text are meant to be paradigm cases.

us which conditions the physical system needs to satisfy in order for it to implement the specific 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 science. The theory of implementation says what we mean by implementation talk and how it reduces to (hopefully straightforward) conditions regarding formal systems and physical systems.

A theory of implementation is sometimes also advertised as a theory of the computational implementation *relation*. This relation is envisioned as a metaphysical bridge between abstract mathematical computations (Turing machines, finite state automata, …) and concrete physical systems (electronic PCs, brains, …). An implementation relation either obtains or not between specific abstract mathematical computations and specific physical processes. If it obtains, then the relevant entity implements the computation; if not, then not. Conceived in these terms, the job of a theory of implementation is a metaphysical task: to give an account of the implementation relation. However, describing a theory of implementation in these terms, although fine as an informal gloss, is a strategic error as a starting point for a theory of implementation. It hypostatises the implementation relation from the start and lumbers us with needless problems about how such a relation fits into a wider ontology. The truth of the claim that a physical system 'implements a computation' is not committed to the existence of a relation between mathematical abstracta and physical objects. (Still less is it committed to the existence of the mathematical abstracta that stand in such relations.) It is only committed to the truth (and falsity) of computational claims about physical systems. Rendering these claims true or false no more requires the existence of a special metaphysical relation than rendering claims that use the expression 'the average man' true require the existence of an average man. A metaphysical computational implementation relation may exist, but it is not a commitment a theory of computational implementation should make at outset.

At a minimum, a theory of implementation should aim to answer these two questions:

COMP
: Under which conditions is it true/false that a physical system implements some abstract computational model?

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

The first question concerns the computational versus non-computational nature

of the physical system. The second concerns its computational identity.[3] Neither question has an easy or uncontroversial answer.

In seeking to answer COMP and IDENT, a theory of implementation should aim to provide a theory that is *extensionally adequate*. Scientific and engineering practice – in computer science, electrical engineering, neuroscience, and physics – already (explicitly and implicitly) classifies physical systems into those that compute and those that do not. It further classifies members of the former class by their computational identity. Many complex judgements 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 running the Tor browser at the moment. These claims are not meant to be controversial. They would be regarded as 'obviously correct' by a scientist or engineer with a working grasp of computational implementation. Such judgements about implementation, made with confidence and receiving widespread agreement in relevant scientific arenas, are important data points for a theory of implementation to capture.

A good theory of implementation need not succeed in capturing every single one of these data points or find all of its consequences confirmed by existing practice. We might reasonably expect *some* divergence between what a theory of implementation says and current judgements in science. But the violation of extensional adequacy may be small or large. At one end, it may only require some minor adjustment, fleshing out, or qualification of existing scientific practice. For example, a theory of implementation might make a claim that my PC does not, strictly speaking, compute the addition function because the PC can only store a finite number of digits – some numbers will be too large for it to add. Or it might make a claim about which current practice is largely silent – for example, that a straight wire computes the identity function. At the other end, a theory of implementation may say that current scientific practice is massively and irretrievably in error. For example, it might claim that all (or nearly all) physical systems implement all (or nearly all) computations. Here there is no question of the disagreement being resolved by a tolerable tweak to scientific practice. The triviality arguments aim to show a violation of extensional

[3] A physical system may implement more than one computation and so have more than one computational identity. Attribution of multiple computations to a physical system is common in science. Sometimes these computations are related, for example, when a physical system satisfies multiple related abstract formal models (e.g. implementing gradient descent, back propagation, AdaGrad, and some specific machine-learning Python program). Sometimes the formal models are not related in any interesting way: the physical system may be 'multi-functional' in the sense that diverse computational models are instantiated in the same physical machinery and active in different contexts. Anderson (2014) argues that the brain is a multi-functional computing system of this type.

adequacy of the latter kind. Triviality arguments aim to put us in a bind: either we entirely reject the scientific practice or we reject the theory of implementation.

Extensional adequacy is only one desideratum on a theory of implementation. Other desiderata include that the theory of implementation be *explanatory*: it should explain computational implementation in terms of conditions that are better understood than computational implementation. The theory of implementation should be *non-circular*: it should explain computational implementation in terms of conditions that are not themselves explained by computational implementation. The theory should be *naturalistic*: it should not make the truth of implementation claims depend on the beliefs, interests, or values of cognitive agents.[4] In Section 7, we will see that when theories of implementation avoid the triviality arguments they 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. We will see that these violations of extensional adequacy are not just worrisome for their own sake (i.e. bad for extensional adequacy), they also create other problems for using computational explanation in scientific practice.

The target of the triviality arguments in this chapter is a 'mapping' account of computational implementation. A mapping account is the one that practitioners in science tend to produce when questioned. It is also the starting point for almost every more sophisticated theory of computational implementation. A mapping account of implementation says that a sufficient condition for computational implementation is the obtaining of an *isomorphism*[5] between the *physical states and transitions* of a physical system and the *abstract formal 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 $f(s') = S'$

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

[5] The mapping relation is sometimes called an 'isomorphism' or 'homomorphism'. For our purposes the difference does not matter. Which term is correct depends on how one individuates physical states, and we will see this is controversial.

[6] This condition is adapted from Chalmers (2012). I have avoided talk of inputs and outputs because, as we will see in Section 4, they can be treated as part of the formal 'state'.

The formal state, *S*, is not meant to refer to just one element of the abstract computational model. Instead, it refers to the specification of the totality of the formal elements that define the computational model at any given step. For example, the formal state, *S*, for a Turing machine would include both the head state and the tape state at any given step; these two entirely specifying a Turing machine's 'state' at any step in its evolution.

In Section 7 we will see that more sophisticated accounts of implementation treat M as necessary but insufficient for computational implementation. Part of the motivation comes from the recognition that M is vulnerable to triviality arguments.

M has many virtues: it is simple, clear, explanatory, non-circular, and naturalistic. M captures the multiple realisability (or medium independence) of computation. Different physical systems (silicon chips, vacuum tubes, brass cogs and wheels, neurons) can implement the same formal computation under M. These physical systems can share a formal structure even if the physical states within that structure differ in their physical nature. The triviality arguments aim to show that this virtue conceals a vice. M licences more than multiple realisability: it licences universal *realisation*. The accounts of Section 7 try to tame this aspect of M without removing support for multiple realisability.

4 The arguments

Two points to note before proceeding.

First, the conclusions of the triviality arguments do not flow from M alone. Additional assumptions, including empirical assumptions about the behaviour of typical physical systems over time, are required. These assumptions may be plausible but it is worth bearing in mind that they are not necessary truths. In other possible worlds, implementation according to M may not be trivial.

Second, a relatively uncontroversial claim is sometimes confused with the triviality claims. This claim is that any physical system could, in the right circumstances, implement any computation. A rock could compute addition if, say, we were to make tally marks on it. A wall could run Microsoft Word if, say, we were to attach powerful electromagnets that constrain its physical states the right way. That a physical system *could* be used to implement a computation is not at issue in the triviality arguments. What is at issue is whether a physical systems *does* implement the computation. One might expect there to be a difference between these two conditions: between merely having the potential to implement (if certain further conditions are met) and actually implementing. (Compare: all hydrogen atoms could be the fuel of

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 compute, they actually *are* computing (and implementing nearly every computation). The threat of the triviality arguments is not a threat about universal *realisability* in some modally qualified sense, but universal *realisation*.

4.1 Hinckfuss' pail

William Lycan describes a scenario 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 in this description, but there is violation of extensional adequacy. Physical systems that we do not normally think of as functional realisers of a computation (the pail of water) are realisers and, perhaps more worryingly, their functional identity is shared with that of our brains and bodies. The kind of functionalism at issue is clearly computational functionalism. Hinckfuss' observation is meant to support the idea that the pail implements the same computation as the human brain and body, and so has the same mental properties.

Hinckfuss' thought experiment may be unnerving, but it is not obvious that it raises a serious problem for M.

First, it is not clear what is so terrible about the conclusion. Panpsychism is arguably not an absurd position (Chalmers, 1996b; Goff, 2017; Strawson, 2006). More to the point however, computational functionalism need not, and rarely does, claim that all aspects of mental life are realised by the computation a system performs. Typically,

this claim is only made for some of the non-conscious aspects of mental life (Block, 1978; Chalmers, 1996b; Fodor, 2000). If one were to remove consciousness from the thought experiment – and perhaps non-conscious mental processes like central cognition (Fodor, 2000) – it may no longer may seem absurd or even surprising that a pail of water could, over a brief time interval, instantiate the same computations as our brain and body. Arguably there is still a challenge here to extensional adequacy, but the challenge is significantly blunted.

Second, is not clear how the thought experiment is supposed to generalise. Showing that this unusual implementation is *possible* is not enough. We would need a reason to think that such unusual implementation is common or the norm to have a triviality result. Perhaps the case described is so rare or unlikely that we should not worry about it in actual scientific practice. There are plenty of strange events that are possible: spontaneous unmixing of scrambled eggs, movement of all molecules of air in the room into one corner, quantum tunnelling of your legs through the floor. Science and engineering practice depends on these events not happening. We can safely assume this because these events are so extraordinarily unlikely. Perhaps a pail of water implementing the same computation as a human brain is like this.

4.2 Searle's wall

John Searle has several arguments against the computational theory of mind. One is this:

> … it 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]

The quoted thought experiment suggests an argument for going from the mere possibility of implementation to a universal triviality result. Consider, according to M, why an electronic PC implements WordStar. Inside the PC are many microscopic

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

physical changes: changes in electrical, thermal, vibrational, and gravitational state of physical parts inside the PC. The PC implements WordStar because among these is one set of changes – the set of electrical changes – that has a structure that is isomorphic to the formal structure of the WordStar computation. Searle claims that the same is true of his wall. Inside the wall are many microscopic physical changes. There are many atoms and molecules undergoing electrical, thermal, vibrational, and gravitational changes. Searle suggests there are *so many* patterns of microscopic physical activity that there is certain to be a pattern with a structure isomorphic to the WordStar computation. Therefore just like the PC, the wall implements WordStar. The same applies to other formal computations and to other physical systems that are 'sufficiently complex'.

There are some problems with this argument that should leave one unpersuaded that it shows implementation under M is unacceptably trivial.

First, the restriction to physical systems that are 'sufficiently complex' is underspecified. Roughly, the idea is that the system should have enough patterns of activity for it to implement any computation. But we might reasonably demand a more informative characterisation of the class of physical systems than this. If 'sufficiently complex' just means having enough patterns, that is no better than saying that the physical systems that are vulnerable are those that exhibit trivial implementation under M. We would like to know how large the vulnerable class is, which physical objects it contains, and whether its members are relevant to scientific practice. Even if one were to define the class in terms of its members' physical size ('sufficiently large'), small or simple physical systems relevant to scientific practice may still be immune.

Second, and more worryingly, why should one believe even of the large physical systems that there are enough patterns of physical activity to render implementation trivial? One might agree with Searle that a large system like a wall contains *many* patterns of physical activity. But one might not accept his claim that a wall contains *every* pattern of physical activity – or even that it contains some specific pattern, such as the one isomorphic to WordStar. How can we be *sure* that the wall contains such a pattern? The burden of proof seems to rest on Searle. He needs to show, not just that there are many patterns of physical activity, but that a pattern isomorphic to any formal computation is *certain* (or highly likely) to occur.

4.3 Putnam's rock

Hilary Putnam in the Appendix of his book *Representation and Reality* presents an argument for trivial implementation that addresses both these concerns (1988).

While Searle and Hinckfuss aim to show that a trivialising mapping exists without showing us the mapping, Putnam provides a method for finding the relevant mapping. Given an arbitrary 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 and non-circular characterisation of the class of physical systems that are vulnerable to trivial implementation. This class turns out to be larger than Hinckfuss' or Searle's arguments suggest.

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 computational formalisms. Putnam also restricts his triviality argument to physical systems that are 'open'. An 'open' physical system is one that is not isolated from, and so in causal interaction with, its environment. Nearly all physical systems in which we are interested are open in this sense.[9] To illustrate the argument Putnam chooses a simple inputless FSA that transits between two formal states: $A \rightarrow B \rightarrow A \rightarrow B$. Putnam's argument is as follows.

Pick an arbitrary open physical system (say, a rock) and an arbitrary 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 possible value of each one of the rock's physical parameters, including those 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 the totality of its physical parameters change. The rock's physical parameters will change due to endogenous physical factors (its atoms changing state, vibrations, atomic decay, etc.), and because of external causal influences (gravitational, electromagnetic, vibrational influences, etc.). Putnam argues that some of these external influences function as 'clocks' for the rock: due to these influences the rock will not return to precisely the same set of values of all its physical parameters in the time interval. Putnam calls this the 'Principle of Noncyclical Behavior'. It is an empirical assumption, but Putnam argues that it is likely to be true of almost any physical system in which we are interested.[11]

Consider the rock's phase-space trajectory from t_0 to t_n. By the Principle of Noncyclical Behavior, we know this path will not cross itself at any point. Putnam assumes that the path is also continuous: it passes through a unique point in phase space at

[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).

[10] Putnam only considers the classical physical states in his argument. However, it not clear how including quantum states would help 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. This theorem only applies to closed systems. The Poincaré recurrence period for an open system is likely to be extremely long – longer than the expected lifetime of the universe.

each moment in time. Putnam calls this latter the 'Principle of Continuity'. Provided these two principles hold, each point in the rock's phase-space trajectory falls into a unique region. Putnam subdivides the rock's phase space into four distinct regions through which it travels, labelled r_1, r_2, r_3, r_4. These regions bound the rock's state during four time intervals between t_0 and t_n. A physical system's position in its phase space completely characterises its physical state. Regions in phase space correspond to groups of physical states – a region in phase space defines a possible 'type' of physical state for the system. We can describe the rock's physical state type 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 of its phase space, 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 define physical state types is a powerful theoretical tool. It reveals multiple changes in physical state of the rock over the time interval. One set of such physical changes is: $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow r_4$. But, Putnam observes, this is not the only one. The rock also exhibits this set of 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 travelling through four neighbouring distinct regions of its phase space (r_1, r_2, r_3, r_4), the rock also moves between two disjoined distinct regions of its phase space ($r_1 \vee r_3$ and $r_2 \vee r_4$). In principle, there is nothing wrong with identifying a physical state type with a disjunctively described, non-contiguous region of that system's phase space. Many legitimate physical state types are defined this way: for example, the net thermal energy and net electric charge of the rock. It is also how the physical states that implement computational states for many electronic PCs are defined: as diverse configurations of electrical signals that could occur in multiple electronic components scattered throughout the machine. Putnam suggests that we map the physical state type characterised by $r_1 \vee r_3$ to computational state A, and the physical state type characterised by $r_2 \vee r_4$ to computational state B. This supplies the required isomorphism between the physical transitions of the rock and the formal transitions of our FSA. The rock's physical state transitions between $r_1 \vee r_3$ and $r_2 \vee r_4$ mirror the FSA's formal transitions: $A \rightarrow B \rightarrow A \rightarrow B$. According to M, the rock implements FSA. The same technique shows that any other open physical system implements any other inputless FSA. We have Putnam's triviality result: 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 an *arbitrary* disjunction of regions in phase space always defines a legitimate physical state type for implementing a computational state. We have seen that there is nothing wrong with identifying an implementing physical state type with a disjunction of phase-space regions. However,

it is unclear whether taking an arbitrary disjunction of phase-space regions is permissible. Many suspect that it is not (Scheutz, 2012). Certain disjunctions pick out legitimate physical states for implementation, others do not. This is reasonable as a response to Putnam as far as it goes. The problem is that it is hard to say *which* disjunctions of phase space are legitimate and *why*. Indeed, this is the central point on which theories that aim to replace M differ. Which additional conditions should a region of phase space satisfy – semantic, teleological, causal, natural kind, and/or an other – to be legitimate for implementing a computational state? Objecting to arbitrary disjunctions of phase-space regions is plausible as the start of a response to Putnam, it lacks bite unless we can say which disjunctions are forbidden and why.

Second, Putnam's triviality argument only considers inputless FSAs. As many computations have inputs (and outputs), one may question whether Putnam's argument is relevant to scientific practice. Putnam considers this. His reply is that although physically specified inputs and outputs partially constrain implementation, intermediate computational states between inputs and outputs would still be open to his triviality argument. Every open physical system with specific physical inputs and outputs would implement every FSA with those inputs and outputs. This collapses computational functionalism into a form of behaviourism – which is not good. A separate consideration, developed by Godfrey-Smith (2009) and Sprevak (2012), is that the inputs and outputs of many computations are not specified in physical terms at all. The inputs and outputs of formal computations are often specified as abstract elements within the computational model itself (e.g. as abstract numerals, characters, strings, or activation values). These inputs and outputs could be implemented, in principle, by any physical state (an electrical signal, a vibration in the air, a turn of a brass wheel). This takes us away from the realm of physically clamped inputs and outputs and back to finding a structure-preserving mapping between physical states and abstract formal elements – and why can't a disjunctive region of phase space implement an abstract formal input state?[12]

Third, Putnam's argument provides a mapping for how the physical state of the rock *actually* changes over the time interval. But there is no guarantee that the rock would behave the same way had the physical conditions been (even slightly) different – e.g. if one photon more had hit the surface of the rock before t_0. The mapping described by Putnam probably would fail under other physical conditions. One might think that a genuine computer should be more robust than this. A separate consideration is that Putnam's mapping only covers the formal FSA states that occur

[12]For more on this, see discussion of the 'strong' and 'weak' input-output constraint in Sprevak (2012) and of the 'transducer layer' in Godfrey-Smith (2009). Maudlin (1989), p. 411 describes how computers differ from other functionally defined systems in that a computer's inputs and outputs are not defined by the physical types it manipulates (unlike, for example, valve lifters or mouse traps).

during the computation. It does not cover formal states or transitions that could have occurred but did not. Computers, especially complicated ones, often have aspects of their formalism that are not employed on any particular run but could have been used under other conditions.[13] Shouldn't those formal elements and transitions be implemented at least as physical possibilities? Based on these two observations, one might propose to strengthen M in two ways. First, the physical state transitions picked out by M should be, in some sense, *reliable*: they should not fail under suitably small changes to the physical system. Second, M's mapping should be *exhaustive*: it should map every formal element of the computational model to a physical state with the right (counterfactual-supporting) physical state transitions. Putnam's argument only focuses on what actually happens in the time interval, not on what could or might happen, so it cannot show that these counterfactual conditions are satisfied. Hence, Putnam's argument fails to show that implementation is trivial.

4.4 Chalmers' clock and dial

The counterfactual objection might look like a fatal objection to Putnam's argument and a counterfactually-strengthened version of M might look safe from triviality (Block, 1995; Chalmers, 1995; Chrisley, 1995; Maudlin, 1989). This turns out to be false. David Chalmers gives a version of Putnam's triviality argument that works against a counterfactually-strengthened version of M (1996a).

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.

The 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 $[i, j]$ 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. Here is the argument.

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

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

Suppose the system starts in physical state $[1, j]$, then 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 formal states in the evolution 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 formal states that follow in the evolution of the FSA. Continue until all of the un-manifested states of the FSA have been covered. Now, for each formal state of the FSA, we will have a non-empty set of associated physical states $\{[i_1, j_1], [i_2, j_2], \ldots, [i_n, j_n]\}$. Assign 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 physical systems we are likely to meet have a clock and a dial. If for any reason a physical system does not have a clock or a dial, those components can be added by attaching a wristwatch to the physical system. If trivial implementation can be achieved simply by adding a wristwatch, clearly something has gone wrong with the theory of implementation.

5 The scope of the problems

None of the preceding arguments claim to show that every physical system implements every computation. What then is the force of the triviality worry? Broadly speaking, we can measure it along three dimensions: *time*, *physical systems*, and *formal computations*.

First, time: When – under which conditions – does trivial implementation occur? One reason why Hinckfuss' argument seems weak is that trivial implementation is only claimed for a brief time period and conditional on some lucky accident. Searle's argument tries to pumps intuitions to raise the chance of trivial implementation happening 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 empirical assumptions) that trivial implementation will occur in the time interval. A specific mapping may fail outside its chosen time window. But we are free to choose a different time interval. The time interval in Putnam's and Chalmers' arguments could in principle be any length: 1 second, 1 year, or 10^{99} years (provided the physical system is around).

Second, physical systems: Which physical systems are subject to trivial implementation? Hinckfuss and Searle suggest that only large macroscopic systems (like walls or pails of water) are likely to be vulnerable. On Putnam's account, a physical system

is subject to trivial implementation provided it satisfies the Principles of Noncyclical Behavior and Continuity. On Chalmers' account, trivial implementation occurs provided the physical system has a clock and a dial. There is no reason why these conditions described by Putnam and Chalmers cannot be satisfied by microscopic as well as macroscopic physical systems. A small number of atoms, even a single atom, could vary its physical state in such a way as to satisfy these conditions over many time intervals. Putnam's and Chalmers' arguments threaten not just 'complex' physical systems but also those that are 'simple' in Hinckfuss' and Searle's terms.

Third, formal computations: Which formal computational models are trivially implemented? Familiar formal computational models include FSAs, Turing machines, register machines, pushdown automata, cellular automata, random access machines, and artificial neural networks. But there are many others – the space of formal models of computation is vast. Which are subject to trivial implementation? We can put an upper bound on the range of the triviality arguments here. That is because at least some computational models are known to be impossible to implement in a physical system. Possible examples include infinitely accelerating computers (for which each step takes place faster than the previous step), computers that use infinite time or storage to reach an answer, or computers that manipulate real-valued quantities with unlimited precision.[15] These formal computations are typically considered as *notional* formal models and studied for their formal properties (for example, proving which sentences in the arithmetic hierarchy they decide), without an eye to physical implementation. Not every formal computational model *can* be implemented, so we can be sure that not every computational model *is* implemented.

Searle claims that all 'programs' are trivially implemented. It is hard to to be sure what he means by this. A program is 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 class of all models of computation (most Turing machines are not programmable Turing machines). If a physical system implements a program, that physical system should implement a programmable model, load the program, and execute it. Searle does not seem to have this in mind: he thinks that his triviality result applies to other computational models than just (rather sophisticated) programmable ones. Alternatively, 'program' might refer to a concrete physical entity that is fed to a physical computer (e.g. a sequence of electrical stored pulses in RAM, or marks on a punch card). But here there is no question of implementing the program as the program is already implemented. Searle seems to have a third, quasi-generic, meaning of 'program' in mind, roughly synonymous with 'algorithm' or 'formal method'. But then his argument is unrestricted with regard to formal computational models, and we have seen that this is not right.

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

Chalmers and Putnam more clearly restrict their triviality argument to only one kind of formal computational model: inputless FSAs. We have seen how one might extend their argument to FSAs with (formally specified) inputs and outputs. What about other models of computation?[16] In the remainder of this section, I argue that their result generalises beyond FSAs to all formal architectures with finite storage. More to the point, a triviality argument for M, once established for FSAs, generalises to every physically implementable formal architecture.

This argument 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 computational model with finite storage, C, and an FSA, B.

The first premise is the conclusion of their triviality arguments. The second premise expresses a formal property of the isomorphism relation: isomorphism is transitive. We will see how to justify the third premise in a moment. The argument runs as follows. Pick any open physical system, A, and any computational model with finite storage, C. By (3), we know there is an isomorphism between C and some FSA, B. By (1), we know there is an isomorphism between physical system A and FSA B. By (2), we know there is an isomorphism between A and C. Hence, according to M:

4. A implements C

Justifying (3) is not hard. Any formal architecture with finite storage, C, can be specified at each step with a single 'state' variable, X. Each value of X (x_1, x_2, …, x_n) would, for example, correspond to a possible tape and head state combination for a Turing machine, a possible grid pattern for cells in a cellular automaton, or a possible setting of activations levels, connections, and weights for an artificial neural network. Since C has finite storage, our monadic state variable, X, can only take a finite number of possible values – otherwise, one could exploit it to gain a formal model with infinite storage. The next formal state of C is determined by

[16] Chalmers (2012) claims that his triviality argument can be avoided if we switch from FSAs to another formal computational model he calls 'combinatorial state automata' (CSA). However, his strategy for doing so depends not so much on the formal nature of CSAs but on modifying M to require that each element of a CSA be implemented by an 'independent physical component' (see Section 7).

its current state.[17] For example, the next tape and head state of a Turing machine is determined by its current tape and head state. Formal model C can be fully described by a directed graph (perhaps very large) of monadic states $(x_1, x_2, \ldots, x_n)$ with transition relations between them. This graph uniquely specifies an FSA, B. Conversely, B's transition table uniquely specifies C under the labelling scheme we have adopted. The labelling scheme provides an isomorphism between B and C. This is the isomorphism we need between B and C.

(4) suggests that the triviality arguments have a very wide reach. This is because any implementable formal computational model seems to be restricted to finite storage. There is no deductive proof of this, but there are good reasons to believe it.[18] Reasons stem from the assumption that any physical implementation of a computer will only have access to finite resources to do its work. These resources may be large (e.g. all the mass and energy in its forward light cone), but they will be finite. That means that if a physical implementation of a computer, since it is restricted to finite physical resources, will eventually run out of storage. For example, any PC one builds will eventually fail to recognise $\{a^n b^n \mid n \geq 0\}$ for large enough n. The limits on storage of physical machines are usually large enough that they are not remarked on. We may *idealise* an electronic PC as an implementation of a formal machine that has infinite storage, like a Turing machine. But strictly speaking, only implementation of computational models with finite storage is physically possible.

(4) shows that there is no difference between those formal computational models that *can* be implemented and those that are implemented *trivially*. Therefore, the triviality argument concerning M is effectively unrestricted. For all formal computations that it is possible to physically implement, implementation is trivial.

6 What is so bad about trivial implementation?

One might wonder whether we could live with the triviality results. After all, M has other virtues: it is simple, clear, explanatory, and makes the truth of claims about

[17] I focus here only on deterministic computations. It is worth considering how the argument might be generalised to non-deterministic models. This would require proving a version of the Chalmers triviality result for non-deterministic FSAs. One way to do this would be to modify his argument to require that the physical system have a clock, dial, and some random physical element. One could then partition the random element's physical state into physical types with appropriate probabilities for the FSA and map formal states of the non-deterministic FSA to appropriate disjunctions of triples of states of the clock, dial, and random element. What remains is to prove an analogue of (3): that any non-deterministic formal architecture with finite storage is isomorphic to some non-deterministic FSA. This final step is straightforward and could follow the same line of argument as in the main text.

[18] For relevant discussion, see Rabin and Scott (1959).

computational implementation objective. Maybe this is worth the cost of accepting that implementation is trivial. The triviality results matter for three reasons:

1. They massively violate existing judgements about implementation in science
2. Combined with computational functionalism they entail panpsychism
3. They drain computational explanations of their explanatory power

Although all considerations have some force, I will argue that the third is crucial, and why trivial implementation is unacceptable.

The first consideration (violating extensional adequacy), while carrying some weight, is not decisive in this context. After all, we have already conceded that extensional adequacy is violated but accept this to gain other benefits. However, we would still be left with a puzzle about why our theory of implementation diverges so much from scientific practice. A hard-line response would be to say that expert scientists are massively in error and they should revise their claims about computational implementation based on the triviality arguments accordingly (e.g. acknowledge that PC cases do run Grand Theft Auto). A more nuanced response would be to explain the observed divergence in terms of pragmatic factors. Under this view, implementation is, strictly speaking, trivial but this is not displayed in scientific practice – and need to be so displayed – because scientists attend to, and talk about, only a few select implementations that interest them. Other implementations may be objectively there, but since they are not of interest they are not discussed. Talk of specific computations that a physical system implements is just pragmatic shorthand to direct our attention to the implementations that interest us most.[19]

The second consideration (entailing panpsychism) is also rarely decisive. Computational functionalism claims that a sufficient condition for a physical system to instantiate a particular mental state or mental process is for that physical system to implement a particular computation. Computational functionalism was popular in the 1970s and 1980s (Block, 1978; Fodor, 1987; Putnam, 1975). But it is not clear how much contemporary cognitive science is committed to it. Cognitive science appeals to physical computations to explain behaviour and mental processes. But scientific practice is normally silent about the metaphysical issue of whether computation is sufficient for mental life. Therefore, it is unclear how much weight to give to any inference to panpsychism (or to instantiation of other aspects of mental life) as such an inference is conditional on a questionable metaphysical assumption.

[19] A parallel here is with how Lewis (1970) and Mackie (1974) treat causes and background conditions. The distinction between a cause and a background condition has no basis in objective fact according to Lewis and Mackie. Lewis and Mackie explain the apparent divergence between science (and everyday talk) and their results by appeal to the pragmatics of causal discourse. Contrasting causes and background conditions is a pragmatic device to direct our attention to those causes that interest us.

The third consideration should cause more concern. Cognitive science explains, predicts, and models human behaviour and mental processes in terms of computations. Decision making, categorisation, inference, and belief revision are explained by the brain and body implementing distinctive computational models. The scientific methodology is to explain particular aspects of behaviour or psychological processing (behavioural or psychological 'effects') by appeal to the implementation of particular computational models. The explanation of why one effect rather than another is produced is because one computational model rather than another is implemented. Psychological phenomena are explained by distinctive physical computations. This methodology appears threatened by the triviality results. If implementation is trivial, then no distinctive computations are implemented by the brain and body. The brain and body, like almost every other physical system, implement every finite computation across every time period. Explaining by appeal to *distinctive* physical computations cannot succeed because there are no distinctive physical computations. Granted this methodology is central to contemporary cognitive science, unless one wants to abandon this science, accepting the triviality results should be off the menu.

7 How to respond to triviality arguments

There is widespread agreement that unmodified M makes computational implementation trivial and that this result is unacceptable to cognitive science. Unfortunately, there is little agreement about what to do about it. Various proposals try to modify M to block the triviality arguments. A natural strategy is to search for additional constraints on physical states that can map to a formal computational state. Proposals to modify M in this way fall into several categories. (The proposals below only sketch options – the points raised are not intended to be conclusive objections but only to indicate open challenges).

First, physical/causal structure accounts. These accounts dive deeper into the physical and causal structure of physical systems to find extra differences that should matter for implementation. Chalmers (1996a) proposes that physical states that implement distinct formal states should be 'independent' components of the physical system. The precise content of this condition is unclear and Chalmers suggests that it be understood as requiring that the physical states occupy different spatial regions. The problem, as Chalmers says, is that this rules out certain legitimate implementations. Godfrey-Smith (2009) proposes that the members of a group of physical states that implement a formal state should be 'physically similar' to each other and 'different' from those that implement other formal states. What counts as 'similarity' here is somewhat unclear. But no matter how similarity is understood,

this condition appears to be too strict. Physical implementers of different formal states may not just be similar but physically identical during a computation (this happens inside a PC – the very same transistor junction that implements a formal state A at time t_i may implement a different formal state B at another time t_j in that computation). The challenge that a physical/causal constraint faces is that it is hard to find a general physical or causal condition that does not make implementation overly strict – that does not rule out legitimate implementations.

Second, semantic accounts. These are based on the idea that computation involves manipulation of representations. This appears to be true for at least some computations: a PC that adds numbers manipulates representations of numbers. What is unclear is whether manipulating representations is necessary for computational implementation. Also requiring attention is to say which semantic properties are necessary (referential properties, procedural semantic properties, normative aspects of semantic content), and where those semantic properties should be instantiated (in the input, output, or intermediate states of the implementation) (Shagrir, 2012; Sprevak, 2010). A semantic constraint faces two main challenges. First, it looks like not all computations have semantic content. Some implementations, like parsers, appear to be 'purely syntactic' and do not operate on representations (Piccinini, 2008).[20] Second, semantic properties classically depend on agents with minds who adopt interpretations, have beliefs and other intentional attitudes (Grice, 1957; Lewis, 1969). Natural semantic properties – semantic properties that do not require minded agents – are notoriously hard to find. But unless natural semantic properties can be found, a semantic account of implementation collapses into a variant of the anti-realist response (see below).

Third, teleological accounts. These are based on the idea that computation involves the exercise of distinctive teleological functions. Which teleological functions these are, and where they are instantiated in physical systems, differs between accounts (Bontly, 1998; Piccinini, 2015). Teleological accounts accounts face a similar worry to the second facing semantic accounts. Like semantic properties, teleological functions look classically like mind-dependent properties: a PC has its teleological functions based on the intentions of its users and/or designers. Advocates of a teleological constraint claim that some teleological functions can also arise in a naturalistic way.[21] But like with the semantic account, naturalising teleology has proved hard. We need success to be sure here if the teleological account is to be the foundation of a theory of implementation. A related worry, and perhaps a more serious one, is that even if a naturalistic account of teleological function succeeds,

[20] For a response to this worry see Sprevak (2010), Section 3.3.

[21] Millikan (1984) argues that a physical system's evolutionary history may ground its current teleological function. For a different, non-etiological account of natural teleological function, see Piccinini (2015).

natural teleological functions are likely to be sparsely instantiated and have a large degree of indeterminacy (Burge, 2010; Shea, 2013). It is unclear whether there is a rich and detailed enough set of natural teleological functions to ground every computational claim of cognitive science.

Fourth, anti-realist accounts. These accounts constrain computational implementations to those that are somehow useful, pertinent, or salient to us given our interests, values, and cognitive make-up. Trivialising mappings such as those of Putnam and Searle are obviously neither useful nor salient to us. The anti-realist suggests that it is because of this that these cases do not count as legitimate implementations. Science should not change its values to accommodate trivial computations. Rather, those 'computations' do not count as legitimate precisely because their mappings are not scientifically valuable to describe. Anti-realists differ in the precise nature of their constraint. Clearly, the constraint shouldn't just be a matter of subjective preference: one person wanting something to implement a computation shouldn't make it so. A workable anti-realist account would likely appeal to what is generally useful, explanatory, perspicuous, informative, or practical in science. The anti-realist response is where Searle and Putnam wish to push us. Their message is not that implementation is trivial but that implementation unconstrained by human interests and values is trivial.

The main cost of the anti-realist response is that it gives up on a key desideratum for a theory of computational implementation: that the theory be naturalistic. According to the anti-realist, computational implementation is not natural or objective. To the extent that implementation is non-trivial, it is entirely a function of our (perhaps a broad 'our' referencing a scientific community) interests and values. Scientific methodologies that appeal to computational implementation to explain have to acknowledge that they cannot do so without invoking an essentially mind-dependent, observer-relative notion. Given that computational explanation is valued as a way to understand the mind in objective, non-mind-dependent terms, this is a worrisome cost.

8 Conclusion

Triviality arguments teach us that unmodified M is unacceptable. This was not obvious and we owe Hinckfuss, Searle, Putnam, Chalmers, Godfrey-Smith, and others our gratitude for uncovering it. We have seen that the price of avoiding triviality arguments is likely to be high with respect to the desiderata for a theory of implementation. This is reflected by different theorists plumping for different (costly) options in order to deal with the triviality arguments. This sounds like bad

news, but in another respect it is good news. It means that triviality arguments provide a meaningful, hard to meet, constraint on our theorising. Without constraints like this, which push back against us, we would be theorising in the dark. The triviality results are not an embarrassment to current theories. They provide valuable information that can guide us towards viable theories of computational implementation.

What is a fair cost to pay to avoid the triviality arguments? We have seen each of the options of Section 7 incurs a cost because they claim that implementation is constrained in the same kind of way in every circumstance. Before closing, I wish to suggest that we can minimise costs by adopting a form of pluralism about computational implementation. Each of the conditions described in Section 7 has a grain of truth in it: it describes how implementation is constrained in some circumstances. Scientific practice reflects this diversity: implementation is sometimes constrained one way, sometimes in another. The relevant constraints on implementation may invoke specific physical relations, semantic content, teleological functions, or human-centric interests and values. The constraint may be naturalistic in some contexts and relative to our interests in others. The problem we saw in Section 7 was that appealing to one constraint does not seem to work in all circumstances (or at least not without considerable cost). A pluralist account of computational implementation says that the factor that constraints implementation may vary between contexts. Whatever constraint is recognised to exist (without cost!) in that context – be it restriction regarding physical relations, semantic content, teleological function, or subjective interests – does work of constraining implementation in that context. Absent such constraints, implementation is trivial. But in any circumstance within scientific practice where implementation is not trivial, at least one such constraint kicks in. If more than one constraint is available – for example, if two competing anti-realist constraints are present – then two conflicting claims about implementation may arise (e.g. the system both performs and does not perform a computation relative to those different standards). This may result in scientific groups talking past each other, or more commonly in acknowledgement that there is a legitimate sense in which the physical system does perform a computation and a sense in which it does not. A pluralist approach offers a way to provide answers to COMP and IDENT that reflect the nuances of scientific practice while avoiding the costs of a simple monist 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. (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.

Grice, P. (1957). "Meaning". In: *Philosophical Review* 66, pp. 377–388.

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

Lewis, D. K. (1969). *Convention*. Cambridge, MA: Harvard University Press.

— (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.

Millikan, R. G. (1984). *Language, Thought and Other Biological Categories*. Cambridge, MA: MIT Press.

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.

Scheutz, M. (2012). "What it is not to implement a computation: A critical analysis of Chalmers' notion of implementation". In: *Journal of Cognitive Science* 13, pp. 75–106.

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

Shagrir, O. (2012). "Structural representations and the brain". In: *The British Journal for the Philosophy of Science* 63, pp. 519–545.

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

Sprevak, M. (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.