The Swapping Constraint[*]

Henry Ian Schiller

Abstract

Triviality arguments against the computational theory of mind claim that computational implementation is trivial and thus does not serve as an adequate metaphysical basis for mental states. It is common to take computational implementation to consist in a mapping from physical states to abstract computational states. In this paper, I propose a novel constraint on the kinds of physical states that can implement computational states, which helps to specify what it is for two physical states to non-trivially implement the same computational state.

1 Introduction

The computational theory of mind (CTM) holds that mental states are realized by a physical system just in case that system implements a particular computation. Computationalists have faced serious difficulty when it comes to saying exactly what it is for a physical system to implement a particular computation. One option is to give a *structuralist* account of implementation. This is, broadly, to say that a physical system implements an abstract computation if the physical system's causal structure is a mirror of the computation's formal structure (Chalmers 1996). This gives us a criterion for mapping states in a physical system to states in a computation: if a physical state and a computational state have the same input-output dispositions, then the physical state implements that computational state.[1]

CTM proponents and opponents alike have made much of the fact that no two physical states are identical down to the last microphysical detail. Thus, for any given physical system, the mapping that occurs from physical states to computational states will actually be a mapping from disjunctions of physical states — or *coarse-grained* physical states — to computational states. The individual physical states which make up a coarse-grained physical state will all have the same input / output dispositions

[*] Thanks to Mark Sprevak for guidance and support in the early stages of this project, and for crucial feedback on an earlier draft. Thanks to Cory Juhl, and to several anonymous reviewers for crucial feedback on more recent drafts. Thanks are also due to Andy Clark, Jonny Lee, Becky Millar, Alex Rendon and audiences at the University of Edinburgh and the 2015 Northwestern Philosophy Conference,for helpful questions and discussion.

[1] The mapping function between physical states and computations must be a mapping from physical states to states in the computation; as we will see, physical states are fine-grained enough that multiple physical states will map to one computational state.

Issues relating to the difference in fineness-of-grain between physical and computational systems has lead to a number of so-called *triviality arguments* against CTM. The general thrust of these arguments is that complex enough open systems (like rocks, tables, and walls) can be shown to trivially map to any given abstract computer.[2] It is trivial that these open systems have 'behavioral patterns' with respect to inputs and outputs like heat or light, and as long as they can be described as behaving in a certain way, they can be mapped to a computation associated with that behavioral profile (Putnam 1987, Chalmers 1996, Godfrey-Smith 2009).[3]

As Godfrey-Smith notes, in order to avoid trivial computational implementations there needs to be some "constraint on the sets of total physical states that are disjoined and mapped to each formal state" (Godfrey-Smith 2009: 289). In this paper, I argue for a constraint on structuralist theories of implementation which would guarantee an appropriate mapping from physical states to computational states. The proposal, briefly, is that a physical state counts as instantiating a computational state only if it can be swapped with another physical state that allegedly serves as an instantiation of the same state and not change the overall behavioral dispositions of the system.

This paper will proceed as follows. First, I will present a sophisticated triviality argument against computationalism, due to Godfrey-Smith (2009).[4] Though the argument is sophisticated, I will — with an aim at clarity — focus on its application to a fairly simple model for computationalism, involving the implementation of *finite state automata*. The structure of Godfrey-Smith's argument generalizes to more contemporary and sophisticated models of computational implementation (such as those involving *combinatorial state automata*), and I hold that the constraint provided here does as well. Then, I will give an account of my proposed constraint, and explain how it rules out trivial computational systems. I will discuss some potential upshots of my proposal for discussion of aging and neural reuse.

[2] The sufficient level of complexity is just that the system is in a unique physical state at any given point, where uniqueness applies to the intrinsic properties of the system (Putnam 1987). I follow Godfrey-Smith (2009) in assuming an account of intrinsic properties along the lines of Langton & Lewis (1998).

[3] A non-structuralist response to triviality worries — which rejects some of these assumptions about what it takes to implement a computation — can be found in Rescorla (2014). This paper will focus on structuralist responses to issues of triviality.

[4] I will focus on Godfrey-Smith's argument because it is immune to objections which have been raised to earlier triviality arguments, and because there is not yet a satisfying response to it in the literature. However, I take what is said here to be in general a first step at outlining a constraint on computational triviality.

2 Triviality Arguments Against Computationalism

There are different ways of representing computations. One kind of abstract computer is a *finite state automaton* (FSA). An FSA is a rule-related set of abstractly-specified inputs, outputs, and states. While a physical computational system could take all sorts of inputs (photons, perceptual information, keyboard inputs) and produce all sorts of outputs (twitches in a muscle, sound, a display on a screen), the FSA distinguishes inputs and outputs only at a formal level (Input 1, Input 2, etc.). Two different physical systems, with different physical inputs and outputs, can thus be said to implement the same computation only if their physical inputs and outputs can be appropriately mapped to the same abstractly specified inputs and outputs of an abstract computation.

Let us give a specific FSA as an example. Our FSA will have two inputs (designated 'I_1' and 'I_2'), three outputs ('O_1', 'O_2', and 'O_3'), and three states ('S_1', 'S_2', and 'S_3'). We will also give the following (exhaustive) list of the FSA's dispositional properties, which specifies the six possible state and input combinations, as well as the state and output combinations that each of them leads to.

$(S_1, I_1) \rightsquigarrow (S_2, O_1)$
$(S_1, I_2) \rightsquigarrow (S_3, O_1)$
$(S_2, I_1) \rightsquigarrow (S_3, O_1)$
$(S_2, I_2) \rightsquigarrow (S_1, O_2)$
$(S_3, I_1) \rightsquigarrow (S_1, O_2)$
$(S_3, I_2) \rightsquigarrow (S_1, O_3)$

The above list specifies the state transition the system makes and output it produces if it is in one of three states S_{1-3} and receives one of two inputs I_{1-2}. Following Godfrey-Smith (2009), we can call this a 'Coke Machine' FSA (because its behavior matches that of a very simple vending machine). The 'Coke machine' is such that it takes 5c and 10c coins as inputs, and emits one of three outputs: a 'null' output (when it hasn't received enough money), a Coke, or a Coke plus 5c in change. A coke costs 15c, so the machine has three internal states. A state where it has no money in it, a state where it has 5c, and a state where it has 10c. Given that the machine is in the 'no money' state, if it recevies 5c it transitions into the 5c state, and produces a 'null' output. Given that the machine is in the 5c state, if it receives 10c it transitions into the 'no money' state and produces a coke (and so on). So in order to make a computer that a coke machine maps to, we need three abstract internal states (S_1, S_2, S_3), two abstract inputs (I_1, I_2), and three abstract outputs (O_1, O_2, O_3). Further, these states need to have the right sorts of transitions; for example, if we define the mappings from physical inputs to formal inputs as follows,

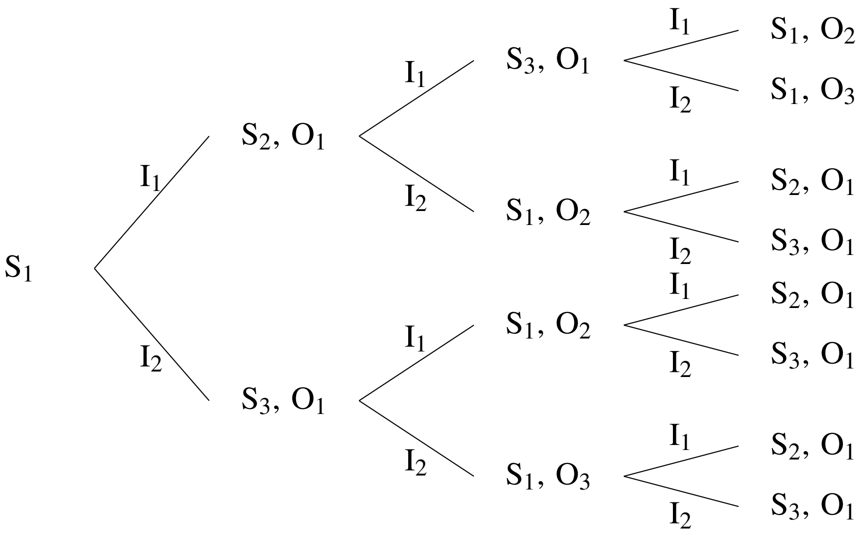

Figure 1 'Coke Machine' FSA (Godfrey-Smith 2009: 277)

Inputs	Outputs	States
I_1 = 5c	O_1 = null	S_1 = empty
I_2 = 10c	O_2 = coke	S_2 = 5c
	O_3 = coke + 5c	S_3 = 10c

then it needs to be the case that if the system is in S_1, and receives I_1, it transitions to S_2 and produces the null output (O_1).

We can graphically represent the Coke Machine FSA as a *contingency tree* (Figure 1), which maps all the different possible state transitions of the FSA.[5] This will be useful for clearly representing some of the moves made later in the paper, where contingency trees for computations are compared with those for physical systems.

Now that we have a computation (the Coke Machine FSA) and a way of giving it graphical representation (the contingency tree in Figure 1), let us turn our attention to how a physical system might map to this computation (and thus implement this computation). I follow Chalmers (1996) and Godfrey-Smith (2009) in adopting a 'simple mapping' criterion for implementation/realization. This specifies the way in which the physical inputs (designated with 'I^*_n'), physical outputs (designated with 'O^*_n') and physical states (designated with 'P_n') associated with some physical

[5] The contingency tree in Figure 1 is taken from Godfrey-Smith 2009. It has been pointed out to me by **redacted** that this is a nonstandard description of FSA, as it does not specify a non-arbitrary initial state, or terminating states.

system map to the abstract inputs, outputs, and states of the FSA. Below I quote Godfrey-Smith's characterization of the simple mapping criterion verbatim:[6]

> **Simple Mapping Criterion:** "A physical system realizes a given FSA during a time interval iff there is mapping M from states of the physical system onto states of the FSA, and from inputs and outputs of the physical system onto inputs and outputs of the FSA, such that: for every state-transition (S, I) → (S', O) of the FSA, if the physical system were to be in state P and receive input I* such that M(P) = S and M(I*) = I during this time interval, then it would transition to state P' and would emit output O* such that M(P') = S' and M(O*) = O" (Godfrey-Smith 2009: 275-6).

Imagining something like the Coke Machine posited by Godfrey-Smith as the implementor of the computation, certain mappings from physical states in the machine to formal states in the FSA will be intuitive. As noted by Putnam (1987) and others, it is very unlikely that any physical system will be in *exactly* the same physical state at any two given times, down to the last microphysical detail. Every time our switchboard is set to *left* it is in a slightly different physical state P. Thus, *left* is actually an arbitrary way of specifying the type of state the physical system is in. Saying that it's the right level of description because it lets us map the physical system to the Coke Machine FSA doesn't give us a satisfying explanation of why this is the right level of description.

Thus,unlike the contingency tree for the FSA, where each node on the tree corresponded to one of three possible state types, in the contingency tree for the physical system each of the 15 possible physical states depicted in the tree corresponds to a unique physical state type P_n.[7]

$$S_1 = Q_1 = P_1 \vee P_5 \vee P_6 \vee P_7 \vee P_8 \vee P_9$$

[6] Chalmers (2012) says something similar: "A physical system implements a given computation when there exists a grouping of physical states of the system into state-types and a one-to-one mapping from formal states of the computation to physical state-types, such that computational states related by an abstract state-transition relation are mapped onto physical state-types related by a corresponding causal state-transition relation" (Chalmers 2012: 229). It should be noted that Chalmers reverses the domain and range of the mapping: whereas I (and Godfrey-Smith) state it as a mapping from physical states (P) to formal states (S), Chalmers states it as a mapping from formal states (S) to physical states (P). My reason for stating it as a P to S function is that if physical states are fine-grained, then there will be many physical states mapping to formal states (a many-to-one mapping). So, doing it the other way give you a one-to-many mapping, which would not be a function (thanks to NAME REDACTED for pointing this out to me). One could specify the mapping this way if one had a reasonably good characterization of the physical state types being mapped to; but, of course, this is exactly what is at issue.

[7] See Putnam 1987 for the origins of this kind of disjunctive description.

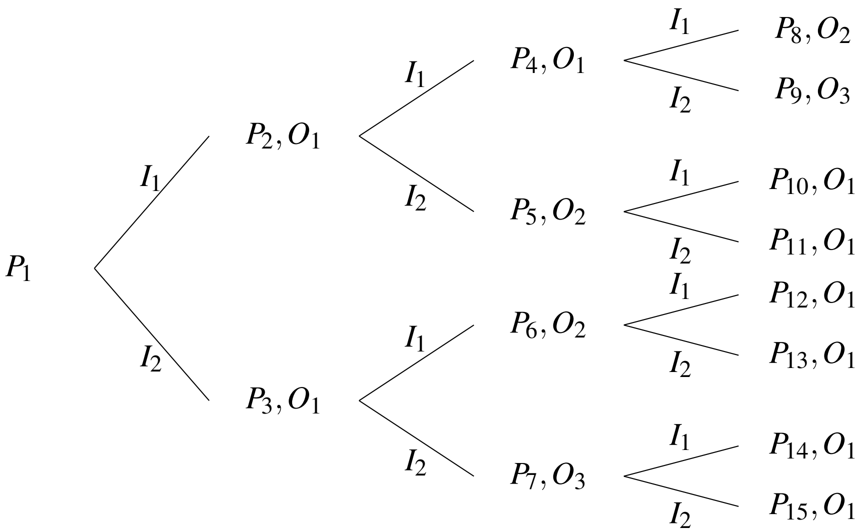

Figure 2 Contingency tree for a physical implementer of the 'Coke Machine' FSA.

$$S_2 = Q_2 = P_2 \vee P_{10} \vee P_{12} \vee P_{14}$$
$$S_3 = Q_3 = P_3 \vee P_4 \vee P_{11} \vee P_{13} \vee P_{15}$$

Disjunctions of physical states P are mapped to a coarse-grained physical state Q before being mapped to the corresponding computational state.

$$(Q_1, I_1) \rightarrow (Q_2, O_1)$$
$$(Q_1, I_2) \rightarrow (Q_3, O_1)$$
$$(Q_2, I_1) \rightarrow (Q_3, O_1)$$
$$(Q_2, I_1) \rightarrow (Q_1, O_2)$$
$$(Q_3, I_1) \rightarrow (Q_1, O_2)$$
$$(Q_3, I_1) \rightarrow (Q_1, O_3)$$

We can give the same kind of graphical representation we used for the FSA to depict the dispositions of a physical system through time. Figure 2 is a contingency tree for a physical system (like the switchboard, or a vending machine) that instantiates the Coke Machine FSA.

2.1 Triviality Argument

The mapping from the physical system contingency tree depicted in Figure 2 to the FSA depicted in Figure 1 relies a great deal on the behavioral properties of both the physical system and the FSA. A state P of a physical system maps to a state S of an FSA if there is the correct kind of fit between the behavioral profile of the physical system and the dispositional profile of the FSA:

> If a state P of a physical system is such that given input I_1 the system transitions to state P' and produces output O_1 and given input I_2 the system transitions to state P" and produces output O_2, and if states P' and P" are such that *etc. etc.*... then P maps to S.

This is how we determine the appropriate mapping from physical states to computational states.

But this means that any system whose states exhibit the same behavioral patterns with respect to arbitrarily defined inputs and outputs (the same input-output dispositions) as those of the states in the Coke Machine FSA will properly map to the FSA. If the states of a physical system are mapped to states in a computation on the basis of their input/output dispositions alone, then computationalism reduces to a sort of behaviorism at the level of states.

What gives this argument even more force against the computationalist position is, as Godfrey-Smith notes, that "any sufficiently complex physical system can be made into a behavioral system duplicate of an intelligent agent, via a change to the 'transducer layer' of that system" (Godfrey-Smith 2009: p. 284). The transducer layer is the part of a system which acts as an interface between system and environment (for example: the retina, in humans). Godfrey-Smith gives the following example using a bucket of water:

> The transducer layer that has to be given to the bucket of water to make it into a coke machine includes an input device and an output device. At the input end, we need the device to accept 5c and 10c coins. This is no problem; they can be dropped into the bucket. We do have to assume a stock of very physically similar 5c and 10c coins, and a uniform method of dropping. Each coin sends the bucket of water into a new unique physical state, and also generates a unique output. Here, the outputs are the effects of ripples in the water on air molecules at the surface. At each moment, the effects of the water surface on these air molecules are unique products of the prior state of the water and the particular impact of a coin.
>
> (Godfrey-Smith 2009: 287)

Though the outputs will be unique, we can build a *transducer device* that disjunctively maps them to the formal outputs O_1, O_2, and O_3. The contingency tree representing the dispositions of the Coke Machine implementer through time (Figure 2) is thus *formally identical* to the contingency tree for a trivial physical system with the right sort of behavioral profile.

We get the trivial implementation because the criteria for realization – the mapping criteria – actually has to be fairly weak. For example, it allows for disjunctions of outputs to be mapped to outputs in the formal model. We have already discussed why it is that the mapping requires us to map disjunctions of physical states to internal states in the formal model. One might object that a stronger mapping criteria would be able to rule out these trivial implementations. But a stronger mapping criteria is not something that we can just non-arbitrarily insist upon. [8]

3 The Swapping Constraint

One way of dealing with this kind of triviality argument is to identify further constraints (besides whatever is already supplied by the simple mapping criteria) on what it takes for a physical system to implement a computer.[9] Chalmers (1996), for example, notes that the state transitions of a physical system must be sufficiently reliable, and that the internal organization of the states needs to be taken into account. I am specifically interested in supplying a constraint on what it takes for two physical states to instantiate the same computational state, in a particular system. Such a constraint will allow us to identify the correct level of granularity for computational implementations.

What we would like is to be able to identify an appropriate level of physical description for individuating computations; to be able to say that only physical states that are sufficiently similar in X way can count as instantiating the same computational state. But what notion of sufficient similarity will place adequate constraints on the kinds of things that we can say are implementing computers?

It is obviously too much to ask that these states be microphysical duplicates because, as discussed above, there will always be microphysical differences between physical states in a system (so *nothing* would count as a physical computer). And we cannot just say that these states have to map to the same formal state in the FSA; they do, but this is insufficient to block against the triviality arguments discussed above. We need some non-arbitrary way of determining a level of description at a

[8] Godfrey-Smith makes a note of this issue as well. “The criteria for realization discussed above look weak because of the existential quantifiers; all that is required that a system have some physical states that map onto a given structure, or contain some states that are related in such a way that they occupy a given set of roles. But this weakness is often something that functionalism seeks, because of the message of multiple realizability, and the alleged ‘autonomy’ of high-level descriptions of complex systems” (Godfrey-Smith 2009: 289)

[9] This is more or less what Godfrey-Smith (2009) suggests in response to the triviality problem he raises.

coarser-grain than microphysical structure, but fine-grained enough to block triviality arguments.

Specifically, this needs to be a (non-arbitrary) way of determining a level of physical description such that for things we want to describe as computers, every physical disjunct in a coarse-grained physical state Qi is identical, but for trivial implementers (rocks, walls, buckets) such a non-arbitrary mapping from physical state disjuncts to coarse-grained physical states is not possible. I propose a constraint that tells us not only when it is appropriate to say that a physical state instantiates some FSA state, but also when it is appropriate to say that two physical states instantiate the *same* FSA state. This will give us a way of setting the correct level of description for a given computational implementation.

I propose the following simple constraint, which I call the *swapping constraint*, on what it is for physical states in a system to count as instantiating the same computational state. We can think of this proposal as a *necessary condition* on two physical states counting as both instantiating the same computational state in a given physical system.

> **Swapping Constraint:** What it is for two states in a given physical system to count as instantiating the same computational state type is for it to be the case that *a*) both states map to that computational state type via Chalmers' simple mapping criterion, and *b*) if those states were physically swapped in the system, this swapping would not alter the internal organization of the contingency tree that represents the physical system.

So, take some coarse-grained physical state Q, such that physical states P_n and P_m are disjuncts of Q, (and Q is the coarse-grained physical state that instantiates some computational state S). In order for the physical system to meet the swapping constraint, it has to be the case that P_n and P_m could switch positions in the contingency tree, and this would not alter the internal organization of the contingency tree for the computation being implemented by that system (it could be represented in the physical system contingency tree as having no effect on any subsequent physical states). In other words, we identify a Putnam-style disjunction of physical states, and then apply the constraint that these states need to be swappable in the physical system without compromising the system's computational integrity.

It might be objected that it looks as though meeting condition (a) of the swapping constraint entails meeting condition (b). That is, if two states, P6 and P8, exhibit the same behavior (i.e., produce the same output, O2, given the same inputs) then they will continue to exhibit this behavior if their positions are swapped.[10] The issue,

[10] Thanks to an anonymous reviewer for pressing me to say more about this issue.

however, is that these states exhibit the same behavior only insofar as there is a (arbitrary) mapping from the *distinct* physical outputs they produce to the same formal output O2. Because swapping will have an inevitable effect on the physical outputs they produce, and on the subsequent state transitions they make, this will require us to ‘redo’ the mapping.[11] This I will expand on this in detail in what follows. So, arbitrary trivial systems that meet condition (a) will not exhibit the swapping feature, because the transducer layer needs to be adjusted after the swap takes place in such a system in order to preserve the claim that it is running the same computation as before (I will go into greater detail below).

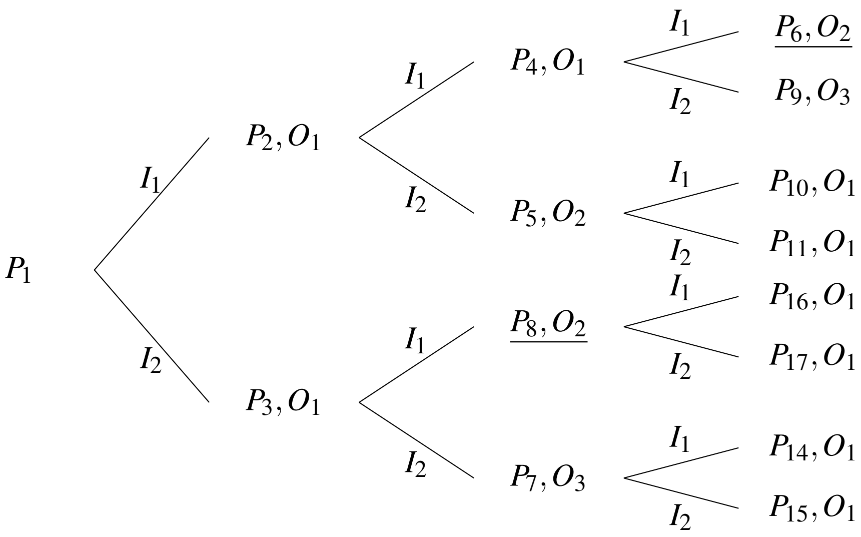

Figure 3 Contingency tree for a physical implementer of the ‘Coke Machine’ FSA with ‘swapped’ states. If the behavioral dispositions of the system would remain unchanged after the swap, then the physical states exhibit the right properties to be counted as implementing the same computational state.

This brings up an important point about how we are to interpret this constraint. The Swapping Constraint has two possible interpretations, one weak and one strong. A weak interpretation would allow for adjustments to be made to the transducer layer of a system in order to accommodate physical differences in swapped components. This is an interpretation of the constraint that I would like — for the time being — to rule out; without adequate restrictions on the sort of adjustments we can make, the

[11] As Godfrey-Smith notes, key to showing that some trivial system B implements some computation S is “that all B’s physical outputs, as well as inner states, are unique” (Godfrey-Smith 2009: 287).

Swapping Constraint will fail to distinguish trivial systems from nontrivial ones.[12] Thus, the Swapping Constraint should be interpreted as a strong constraint, which says that two physical states only count as implementing the same computational state in a physical system if a swapping can occur without adjustments to the system's transducer layer.[13]

Why think that the ability of two states to be swapped is the right thing to measure when trying to identify whether those two states instantiate the same computational state? Even if it should so happen that only true computational systems have this property, we need some reason for thinking why this is sufficiently explanatory.[14]

One answer we might give is that the swapping constraint has to do with the functional properties of the states in question. If a physical state P to be able to subjunctively fulfill a certain functional role (the role of another state) in virtue of its *actually* fulfilling a certain functional role, then this is a functional property of P. If a physical state P_1 is such that were it to have occurred in the physical contingency tree where P_{1000} in fact occurred it would have fulfilled the same functional role (produced the same output), then this is a functional property of P_1.[15] This is to say that the swapping constraint is based on the right kind of thing for a constraint on computational systems: functional properties. What proponents of CTM are after is an explanation of the mind in terms of computational properties; i.e., properties that can be specified at the level of computational, rather than physical, description. Functional properties are the 'right kind of thing' in this sense.

Further, the view has some intuitive plausibility with respect to how we think about the components of physical computers. Imagine a standard computer with discrete functional components. One such component will be a logic gate, which (in a physical computer) is a physical device implementing a Boolean function. If that physical component were to be swapped with a similar enough (but not microphysically identical) physical component, the computer could still run. Swapping it with a sufficiently dissimilar physical component will alter the subsequent state-transitions in the contingency tree representing the system (or it may even crash the

[12] It is also worth noting that this interpretation requires us to give a different account of transduction than one that is tacitly assumed. Transduction is typically thought of as an operation on the inputs / outputs of a system as a whole, but if we want to adjust the transducer layer so as to accommodate physical differences in swapped components, then we need to think of transduction from/to inputs and outputs at the level of states, rather than at the level of the system as a whole (thanks to an anonymous reviewer for pointing this out.)

[13] Thanks to an anonymous reviewer for pushing me to address this in detail.

[14] We might also think that only true computational systems have the property of being claimed by us to be true computational systems. But this would not make such a property a compelling one to use in developing a constraint.

[15] We might wish to put this in terms of a microphysical duplicate of the state P_1 being instantiated at the node of the contingency tree where P_{1000} is.

computer).[16] Components that are similar enough to preserve the contingency tree when swapped are, intuitively, the same kind of physical component (as far as the system's behavior is concerned).

In Figure 2 — which could be used to represent the Coke Machine FSA being instantiated by either a trivial or nontrivial physical system — states P_5 and P_8 are both mapped to the same coarse-grained physical state Q_1, which maps to the computational state S_1. If we were to swap them in the system, then in a nontrivial system we should expect the contingency tree to remain unchanged, with respect to its overall functional structure. The reason for this is that what is being swapped is two states that have the same relevant internal structure, which moves the system in to a new state and produces particular output.

Why should a trivial system fail to meet this expectation? Because the mapping from a trivial system to an FSA is based on a trivial reading, there is no reason to think that a trivial computation system could satisfy the requirements of the swapping constraint without some aspect of the input/output readings being altered. I will say a bit more about why I find this claim so compelling.[17]

I will demonstrate how the mapping constraint rules out trivial systems. I will do this by providing a completely arbitrary physical system, instantiating Godfrey-Smith's 'Coke Machine' FSA, and show how it meets condition (a) but not (b) of the Swapping Constraint. Most of this will be borrowed from Godfrey-Smith's own discussion of the trivial implementer.

Take an ordinary bucket of sea water. We will show how it implements the Coke Machine FSA (but fails to meet the swapping constraint). Let our arbitrary inputs be dropping a 5c coin (I1) and dropping a 10c coin (I2) in the bucket. Remember that the Coke machine has three states, and starts in the 'empty' state (S1). The bucket of water starts without any coins in it, so this is S1. However, we cannot simply define S1 as the state at which the bucket has no coins in it (or S2 as the state at which the bucket has a 5c coin in it, or S3 as the state at which the bucket has a 10c coin in it). S1 will also have to be a state at which the bucket has a 10c coin and a 5c coin in it.[18]

The bucket produces ripples when the coin is dropped in. The exact nature of the ripples produced is going to be different enough between ripples that they can

[16] That physical component could be swapped with a similar enough physical component in (a) a standard computer of the same kind, or (b) the same physical system at a different time.

[17] Thanks to an anonymous reviewer for pushing me to go into greater detail on this point.

[18] This is not crucial, but it is worth noting to highlight the difference between the bucket of water and a wall. In some sense the wall is 'more' trivial (which is why I use the bucket as the example) because the different states are not differentiable in the terms used here. We can add complexities to the case, such as drop levers, etc. in order to give a more direct mapping here. However, it should be noted that we run the risk of turning the bucket of water into a nontrivial computer.

be mapped to different formal outputs. It is worth noting that the ripples produced when a 5c coin is dropped in will probably be, overall, more similar than the ripples produced when a 10c coin is dropped in, even though *either* could count as, for example, O1 or O2 (only the latter could count as O3 – which is the output that includes change). Given these features of the system, how do we get the requisite mapping from physical states to computational states?

> All that then has to be done is collect the O_i^P that should map onto each of the three desired outputs of the coke machine. Those are O_1 (null output), O_2 (emit coke), and O_3 (emit coke and 5c change). So physical outputs O_1^P, O_2^P, and O_3^P (and some others), should map to O_1 physical outputs O_4^P, O_7^P (etc.) should map to O_2, and physical outputs O_6^P and O_*^P should map to O_3. All the designer has to do to generate coke machine behavior over the interval is build a transducer device that does nothing when it detects O_1^P (etc.), emits a coke when it detects O_4^P (etc.), and emits a coke and change in response to O_6^P (etc.) (Godfrey-Smith 2009: 287).

Let us consider (part of) a contingency tree for our trivial physical system. The physical inputs are 5c and 10c coins being dropped in the bucket, and the outputs are the different resulting ripple effectd. The bucket would have slightly different physical properties — and thus produce slightly different outputs — depending on very small changes in the way the coins are dropped, and on very small differences in the internal organization of the bucket at a given time (See Figure 4).[19] Receiving one input rather than another (I_1 rather than I_2) at a particular time changes how the bucket will behave at later times, with respect to different inputs. That is to say, if the system receives I_1 at some time t_1, versus receiving I_2 at t_1, then the way the system responds (the output it produces) to any given input I_n at some later time t_n will differ. Consider the tree above — if it is in physical state P_1 and it gets input I_1, then it will have a different disposition to respond to I_3 than it would have if it had gotten input I_2 while in physical state P_1.

Now imagine that physical states P_3 and P_4 (both of which map to S_3) in Figure 4 are swapped. In the initial contingency tree, when P_3 received a 5c coin, it moved to a new state and produced physical output O_4^P and when P_4 received a 5c coin, it moved to a new state and produced physical output O_7^P, and these were disjunctively mapped to O_2.

You throw a quarter into the bucket and it produces a certain kind of ripple. Later, in the same ‘state’, you throw another quarter in the bucket, and it produces

[19] This means that it is not entirely appropriate, in the contingency tree that follows, to represent only two physical inputs, rather than four (or more).

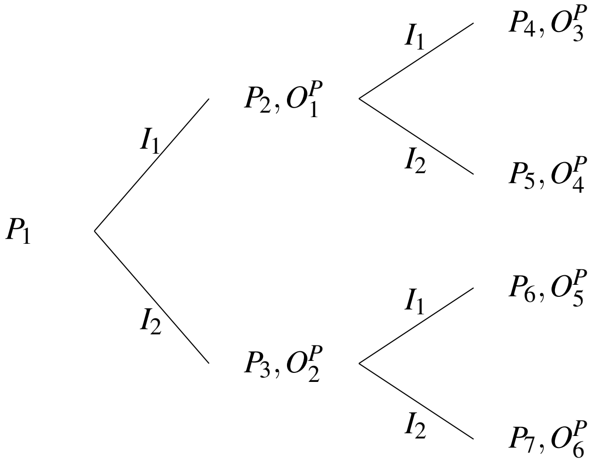

Figure 4 (Part of) the ‘coke machine’ FSA as run by a trivial system; note that the outputs are all physically distinct, and thus require a disjunctive mapping to a formal output.

a different ripple, though maybe it is similar in some ways to the first ripple. That the behaviors of these states are similar when you throw a quarter in is determined entirely by the fact that the quarter has been thrown in. But how do these states differ from physical states of the bucket mapped to different computational states? Well, the behavior of those states when a quarter is thrown in produces a similar ripple effect (because of the quarter) but we map it to a different disjunction of outputs. So: some quarter-outputs get mapped to O1, and some quarter-outputs get mapped to O2. The system is ‘trivial’ because the mapping here is determined by our need to map things in a certain way in order to get the kind of computational description we want.

I will briefly give this example again with a wall. Searle (1990) famously wondered whether the wall of his office was running the Wordstar program. In order to map a wall to an FSA (say, for Wordstar, or Godfrey-Smith’s coke machine) we develop a transducer device that maps the the wall’s physical outputs to FSA’s formal outputs.[20] Swapping states in a trivial physical system results in a physical contingency tree with different internal states and *completely different outputs*. It would require an entirely new transducer device with an entirely new mapping

[20] “All the designer has to do to generate coke machine behavior over the interval is build a transducer device that does nothing when it detects O_1^P (etc.), emits a coke when it detects O_4^P (etc.), and emits a coke and change in response to O_6^P (etc.) . . . It is as if a designer had enormous knowledge of the physical dispositions of the bucket of water, and very fine-grained ways of building input-output devices” (Godfrey-Smith 2009: 287).

system (corresponding to the new outputs) to map the outputs of the wall to the formal outputs of the FSA.[21]

Given the swapped states, the inputs and outputs will differ, and a new mapping will need to be created based on the new behavior of the system. Instead of one physical state, which reacts to an input in a particular way, we get another physical state, which will react to that input in a way that alters the subsequent behavior of the system. So while the wall has the behavioral properties of the FSA, those properties are not preserved through swapping. The constraint is based on the plausible notion that if a state performs a function in a system, it should be able to perform that function at any point in the system. If what it is for two states to instantiate the same computational state is for them to exhibit swapping, then physical states of trivial systems will fail to instantiate computational states, because the mapping from these states to FSA states is based on dispositional properties that the states do not exhibit after the swap.

3.1 Swapping Between Systems

The Swapping Constraint is a constraint on what it takes for two physical states to implement the same computational state *in a given physical system*. In one physical computer, the correct level of granularity might need to be fairly fine-grained (say, at the atomic level) in order for the states to meet the swapping condition. But in another system, physical similarity at a different level of granularity may be required. Taking the Swapping Constraint as a system-level constraint allows us to accommodate the fact that there may be no one correct level of granularity for describing the physical implementation of a computer. In other words, the level of physical description at which one physical computer's states might be grouped as equivalent will not be the same as another's. That a computer can be run by a macrophysical and microphysical implementer, the latter requiring a more fine-grained level of description than the former, is an aspect of multiple realizability.

Note that the (strongly interpreted) swapping constraint applies not only within the states of a single physical system, but also between sufficiently similar physical systems.[22] There is, however, a question of what happens when you treat the Swapping Constraint as a constraint on computation in general not just on states

[21] No corresponding worry arises for the nontrivial system; we might think of the initial mapping of the nontrivial system as fixing the granularity with which we describe its behavior. With respect to that level of grain, swapping does not change the behavior of the system. With respect to the level of grain used to describe the behavior of a trivial system, it does.

[22] Of course, what counts as sufficient similarity is, in some sense, going to be determined here by the swapping constraint itself. Perhaps the strongly interpreted swapping constraint offers a way of distinguishing physical computers of the same 'type' in one sense, helping to distinguish multiple realizability between tokes from multiple realizability between types. This relates to classic work on

in a particular (kind of) system. It should be clear how the Swapping Constraint explains how distinct physical states in a particular physical system might (or might not) implement the same computational states. The states in a system S will count as implementing the same state only if they can be swapped, and we should expect that within one system all functionally equivalent states will exhibit this property (I respond to some potential worries for this claim in the next section).

Further, sufficiently physically similar systems will exhibit the swapping constraint with each other. Physical computers that are built the same way, out of the same physical substrate will be able to swap components. But as many have noted, we would often like to treat systems made from wildly different physical substrate as performing the same computation. Sprevak (2010) offers the example of a computer built out of silicon, and a sophisticated computational device built out of tin cans and strings; these might be described as having computationally equivalent states (components like logic gates) but it would not be possible to swap states between the two systems without changing the transducer device.[23]

In other words, the swapping constraint may be able to fix the correct level of granularity for discussing computational implementation in one physical system, but cannot account for equivalencies between different kinds of implementation. If we want to treat the Swapping Constraint as a a constraint on computational implementation in general, then this will require that we say apparently equivalent components in different enough kinds of physical systems are not actually equivalent. This is a difficult view to accept without extensive argumentation, and so in this paper I simply advocate the weaker claim that the Swapping Constraint places a constraint on what it takes for two states in a given system to implement the same computational state.[24]

3.2 CSA Models

Other proposals have been made for placing constraints on the kinds of physical systems that might implement a computation. Perhaps most notably, Chalmers (1996, 2011, 2012) advocates a *combinatorial state automata* (CSA) model, and has rejected the FSA model as outdated. The foregoing discussion has focused on FSA

the type-identity theory (Lewis 1966), as well as some more recent work on multiple realizability (Shapiro 2000).

[23] Thanks to an anonymous reviewer for bringing this example to my attention, and for pressing me to say more about this issue in general.

[24] However, something like stronger view — that different material substrate realize different computations — has advocates in a related debate regarding mechanistic accounts of computation. Kaplan (2017), for example, responds to recent arguments against mechanistic accounts, made on the basis of similar charges (Chirimuuta 2014).Kaplan’s argument, briefly, is that we cannot expect the scope of mechanistic explanations to completely account for considerations involving multiple realizability.

models to make the exposition of the problem and solution as simple as possible. This section will briefly discuss how the proposed constraint might be applied to a CSA model.

The reason for focusing on the simpler FSA models is in part expositional: the triviality argument against FSA's is much easier to summarize. Godfrey-Smith (2009) gives a triviality argument against CSA models that is almost identical in structure to the one outlined in the foregoing for FSA models. There is no in-principle reason to think that my proposal for constraints on the implementation of FSAs could not also be used to constrain the implementation of CSAs.[25]

The swapping constraint might also help us to avoid some of the less desirable features of a CSA model. For example, Chalmers proposes a 'spatial independence' constraint on computational systems. He notes that this constraint "rules out the relevant trivializations, but at cost of ruling out some reasonable implementations as well" (Chalmers 2012: 232). Chalmers also notes the following about his proposal:

> . . . I entertained the possibility of adding further constraints, including further constraints on the physical state-types (e.g. a naturalness constraint) or on the physical state-transitions (e.g. a uniformity constraint), although I left the matter open.
>
> (Chalmers 2012: 232)

The Swapping Constraint may be viewed as supplying specifications for a 'naturalness constraint'. The idea of a nautralness constraint is that we constrain the types of physical states that get mapped to functional states. The swapping constraint fixes a level of physical similarity that is, I take it, appropriate for distinguishing physical kinds in this way. In order for states to be swappable, they will have to be (relatively) physically similar; i.e., of the same physical state type.

Chalmers suggests that we have some reliability constraint on the physical processes that get mapped to abstract states. In a sense, the proposal made here is a way of guaranteeing reliability without positing anything over and above a constraint on what the physical states must be like: the physical states being mapped to a computational state must be physically similar enough to one another to give rise to the same sorts of behaviors.

Working in a CSA model does address many of the issues faced by FSA frameworks; however, such a framework still faces many of the same challenges when it comes to specifying the correct level of physical description at which to individuate computational states. Thus, whether we think of physical computers as implementing

[25] Sprevak (2012) argues somewhat compellingly that the components of a CSA can actually be construed as FSA which are particularly permissive about inputs and outputs. See Section 4 of Chalmers 2012 for some discussion of this proposal.

FSA or CSA, we (a) want an explanation of how to group the (physical) components of those physical systems, such that we get a reliable mapping, and (b) can use the swapping constraint to provide an explanation of this kind.

4 Changes to Computational Systems Over Time

In this section I will briefly discuss how the swapping constraint deals with change to the physical systems that implement computations. Specifically, I address an important way in which change occurs over time in a human brain (the supposed implementor in a computational theory of mind): aging.

Internal states in a computation do not themselves change or age, but they are instantiated by physical parts (cells, synapses, tissue) that change and degrade over time. Thus, a physical state at one point in a computation may not be able to be swapped for a physical state at another point in the computation if that second state includes physical parts that have changed significantly. This is the case even if these states are playing the same general role in that computation. There might be a 'mismatch' of parts. Imagine a puzzle which slowly becomes waterlogged over time. If this occurs to all the pieces of the puzzle (say, while the puzzle is put together) the pieces may still fit. But if it only occurs to some, then the puzzle will likely be rendered useless.

It is worth noting, however, that alterations in brain structure due to aging are "intimately tied to alterations in cognitive function" Glisky (2007). That is to say, the process of aging might alter the computations that underwrite mental functions and mental states, resulting in new computations underwriting what seem like identical mental states. Some behavior or function being performed in a younger version of a system might, as a result of aging, have a correlate behavior or function in an older version of the system that is underwritten by a different computational process.[26] It would not make sense to have to swap physical states that are part of different computations altogether. If mental states are underwritten by computations, then mental states will have to be considered in coarse-grained functional terms, even within a single physical system. If a cognitive system is in mental state M1 at time period T1, then this mental state will be underwritten by some computation C1, which is describable by some FSA at T1. At time period T1+n, an older version of the same cognitive system might be in mental state M1 again, but because of the ways in which aging alters cognitive function, at this time period the mental state could be thought of as underwritten by some other computation C1+n. Thus, M1 can be thought of as a coarse-grained state to which behaviorally similar instances of mental states can be mapped.

[26] See Glisky & Kong 2008 for some experimental evidence in support of this claim.

If the brain is a physical computer, then it is such that it can accommodate (at the computational level) changes, over time, to the various components that are involved in implementing computations. Though these changes are accommodated, they also have an effect on cognition. There is obviously some sense in which the cognitive functions of an older brain might be thought of as the continuation of cognitive functions performed by that brain when it was younger. However, the (perhaps gradual) changes might be such that the physical system underwriting the computation being implemented has to be thought of as a new physical computer (at least quite narrowly speaking), capable of implementing different computations.[27] If aging does indeed lead to this kind of cognitive change, this can be accommodated by the swapping constraint.

A similar issue might be raised for cases of neural reuse and neural plasticity. Neural reuse is a phenomenon whereby neural circuits that have been established for one purpose are recycled and redeployed for other purposes (Anderson 2010). Neural plasticity, which can occur after a stroke or other traumatic neural event, is a reorganization of neural architecture such that functional neural circuits pick up some of the functionality of circuits that have been damaged by the event. It seems that the same computational states can be mapped to from radically different neural architectures within the same physical system. It is difficult to see how two physical computational states that occur within the same system, but over different sets of neural circuitry could be swapped without altering the internal organization of the physical contingency tree for the computation.

In other words, the constraint threatens to be too strict. It threatens to say, of cases of neural reuse, that two physical states (which cognitive neuroscientists would be happy to treat as instantiating the same computational state) must be instantiating different states because they cannot be swapped. Much worse, it might threaten to render these systems trivial.

Cases of neural reuse and neural plasticity seem to be instances where a computation is being run on a physical system and the physical states that are disjunctively mapped to the same computational state have a radically different structural organization. However, I think all that is happening in cases of neural reuse is that one computation (pre-neural event) has discontinued implementation and a new computation (post-neural event) is being implemented by the new physical system.

The computation being implemented pre-neural event may be the same computation as the one being implemented after the neural event. These may each be an instance of the same computation. That is to say, before and after the stroke or neural reorganization, the computation being run by the brain will be describable using the same FSA. However, they should be thought of as distinct instantiations of

[27] See Olson 2002, Shoemaker 2004 for some discussion of this issue in connection with theories of personal identity.

the computation, on different — albeit related — computational systems. As such, it should not be necessary that the physical states in one instantiation are able to 'swap' with their FSA counterparts in the other instantiation of the computation.

5 Conclusion

I have offered a necessary condition on what it is for two physical states to implement the same computational state as part of a nontrivial computational system. This characterization breaks from earlier responses to triviality arguments by focusing on the properties of particular physical states, rather than the properties of the system as a whole Chalmers (1996). Though the proposal of a 'swapping constraint' is quite specific, I hope the lesson here is a general one: we ought to think more about computational systems in terms of the functional similarity between the physical states they undergo through time.

References

Anderson, Michael. 2010. Neural reuse: a fundamental organizational principle of the brain. *Behavioral and Brain Sciences* 33(4).

Chalmers, David. 1996. Does a rock implement every finite-state automaton? *Synthese* 108(3).

Chalmers, David. 2011. A computational foundation for the study of cognition. *Journal of Cognitive Science* 12(4). 323–357.

Chalmers, David. 2012. The varieties of computation: A reply. *Journal of Cognitive Science* 13. 211–248.

Chirimuuta, Mazviita. 2014. Minimal models and canonical neural computations: The distinctness of computational explanation in neuroscience. *Synthese* 191(2). 127–153.

Glisky, Elizabeth. 2007. Changes in cognitive function in human aging. In David Riddle (ed.), *Brain aging: Models, methods, and mechanisms*, Boca Ratonn, FL: CRC Press.

Glisky, Elizabeth & Lauren Kong. 2008. Do young and older adults rely on different processes in source memory tasks? A neuropsychological study. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 34(4). 809–822.

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

Kaplan, David Michael. 2017. Neural computation, multiple realizability, and the prospects for mechanistic explanation. In David Michael Kaplan (ed.), *Explanation and integration in mind and brain science*, Oxford: Oxford University Press.

Langton, Rae & David Lewis. 1998. Defining 'intrinsic'. *Philosophy and Phenomenological Research* 58(333-345).

Lewis, David. 1966. An argument for the identity theory. *The Journal of Philosophy* 63(1).

Olson, Eric T. 2002. What does functionalism tell us about personal identity? *Noûs* 36(4). 682–698. http://www.jstor.org/stable/3506231.

Putnam, Hilary. 1987. *Representation and reality*. Cambridge, MA: MIT Press.

Rescorla, Michael. 2014. A theory of computational implementation. *Synthese* 191(6).

Searle, John. 1990. Is the brain a digital computer? *Proceedings and Addresses of the American Philosophical Association* 64(3). 21–37.

Shapiro, Lawrence A. 2000. Multiple realizations. *The Journal of Philosophy* 97(12). 635–654.

Shoemaker, Sydney. 2004. Functionalism and personal identity: A reply. *Noûs* 38(3). 525–533. http://www.jstor.org/stable/3506251.

Sprevak, Mark. 2010. Computation, individuation, and the received view on representation. *Studies in History and Philosophy of Science* 41. 260–270.

Sprevak, Mark. 2012. Three challenges to Chalmers on computational implementation. *Journal of Cognitive Science* .