Dispositional Implementation Solves the Superfluous Structure Problem

Colin Klein

Abstract. Consciousness supervenes on activity; computation supervenes on structure. Because of this, some argue, conscious states cannot supervene on computational ones. If true, this would present serious difficulties for computationalist analyses of consciousness (or, indeed, of any domain with properties that supervene on actual activity). I argue that the computationalist can avoid the Superfluous Structure Problem by moving to a dispositional theory of implementation. On a dispositional theory, the activity of computation depends entirely on changes in the intrinsic properties of implementing material. As extraneous structure is not required for computation, a system can implement a program running on some but not all possible inputs. Dispositional computationalism thus permits episodes of computational activity that correspond to potential episodes of conscious awareness. The Superfluous Structure Problem cannot be motivated against this account, and so computationalism may be preserved.

1. The Problem With Structure

I took the train to work today. The infrastructure of the Chicago Transit Authority ensures that I could have taken many different trains, and that the same train could have taken different routes. But I took only one train, and it travelled along only one track. There are some facts—travel time, say—about my *transportation experience* that supervene only on the facts about my train and its journey. As long as the same train went in the same way along the same track, I would have had exactly the same transportation experience; what went elsewhere in the system was quite irrelevant. Of course, in one sense those other facts were relevant—they likely caused some of the facts about my train. But holding fixed the facts about the actual activity of my train, we hold fixed the facts about my transportation experience; it is in that sense that the rest of the system did not matter. Transportation experiences are thus potentially *episodic*: it would be possible to have an identical trip even if the rest of the system were to vanish.

As with transportation experiences, perhaps too with conscious experiences. It is nice to have complicated brains, especially ones that can support a multitude of different experiences. However, you might think that as far as any *particular* train of awareness goes, what we experience depends only on what is actually going on in our brains. Extra structure is handy, but strictly irrelevant for episodes of awareness. If you think this, you think that conscious states can, at least potentially, be episodic as well.

Based on these observations, Tim Maudlin has argued for the striking conclusion that conscious states can neither supervene on computational states nor be explained by appeal to computation.[1] The essence of the argument is simple. To count as implementing some program π, a fairly large bit of structure has to be in place. Further, this structure is non-optional: its presence is required lest we be forced to admit that every sequence of events implements π. However, in ordinary cases where π runs on some input τ, much of that structure is not computationally active.[2] Hence, whether a system is computing $\pi(\tau)$ is not just a matter of actual activity; equivalently there can be no episodic computations of $\pi(\tau)$. So the nature of consciousness cannot be computational. Similarly so for transportation,[3] or for anything else that admits of single episodes that supervene on actual activity—consciousness is a snappy example, but the problem generalizes to many things about which the computationalist might want to theorize.

Call the thesis that conscious states supervene on computationalist states the *Computationalist Thesis.* Maudlin argues that the compu-

tationalist thesis cannot be held consistently with what we have just asserted. The argument runs thus:

1. Conscious states supervene on actual activity. (Activity Thesis)
2. Therefore, no two things can differ in conscious state without differing in actual activity. (from [1])
3. Conscious states supervene only on computational states. (Computationalist Thesis)
4. Two things that differ in computational status—that is, in whether or not they are computing anything at all—must differ in conscious state. (from [3])
5. Two things may differ in computational status without differing in actual activity. (Premise)
6. Therefore, two things can differ in conscious state without differing in actual activity (from [4] and [5]).

Since [2] and [6] are straightforward contradictions, something has gone wrong. It is tough to see where, though. If the computationalist really is committed to the two supervenience theses, then the only possible point of attack is premise [5]. But this seems hopeless—most authors agree that abandoning standard constraints on structure would result in everything implementing every Turing machine; computationalism would then collapse into panpsychism, which is hardly a happy result. Therefore, Maudlin concludes, we must abandon the weakest premise of the bunch—the computationalist thesis itself.

Call the general problem the *superfluous structure problem* (SSP). In order to count as an implementation, it looks like a system must have more structure than is strictly active during a particular computation. This puts computationalism in conflict with the activity thesis. SSP is wider than it may appear at first. If successful, SSP will block computationalism in any of domain where the activity thesis is plausible.[4] Indeed, I suspect that SSP is one motivation behind Searle's oft-misunderstood critiques of computationalism in his (Searle, 1990). Many authors assume that the problem for computationalism is the overgeneration worry he lays out in §IV. But there is a deeper worry that Searle sets out in both §IV and §VI. Searle writes that if you do not posit conscious homunculi then:

> . . . you are left only with a pattern of events to which someone from the outside could attach a computational interpretation. Now, the only sense in which the specification of the pattern by itself provides

> a causal explanation is that if you know that a certain pattern exists in a system you know that some cause or other is responsible for the pattern ((Searle, 1990) 32).

But, he goes on, being-mappable is not the right sort of thing to explain conscious behavior. One way of understanding this is as a worry that the ordinary requirements on computation require mappings to more structure than is active. This additional activity is necessary to certify a particular system as an implementation. Certification does not actually do any work, though. So if you think that work is what matters, you will think that computationalist accounts are misguided.

I think that a computationalist can nevertheless avoid SSP. In section two I will review reasons to think that the simple counterfactual account of implementation that motivates [5] is flawed. In section three, I argue that we can construct a *dispositional* theory of computation that overcomes the problems with using a simple counterfactual account. I use this account to develop an theory of episodic implementation in section four. On a dispositional account of episodic implementation, computational activity supervenes entirely upon actual activity. If the computationalist claims that conscious states supervene on the activity of episodic implementations, he can deny premise [5]—and thereby preserve computationalism. I conclude with an examination of the difference between being an implementation and computing a function, showing how the dispositional account avoids SSP by inverting the usual pattern of dependence between the two notions.

2. The Simple Counterfactual Account of Implementation

Taking the computationalist thesis to heart, let us suppose that that a system has a conscious experience E just in case it implements some program π running on some input τ. Determining whether a system S has the experience E at some time, therefore, reduces to the problem of determining whether S is an implementation of a TM running π on τ.

That problem has received extensive discussion in the literature. At a first pass, one might think that meeting the following three conditions would be enough:

I1 There exists a mapping from both the states of the tape and the states of the abstract specification of π on the one hand to dynamic states of S on the other.

I2 The transitions of the states and the tape when started on τ map to those prescribed by the machine table for π.

I3 Had the input to S been different, S would still have fulfilled I1 and I2.

[I1] and [I2] ensure that the material of S goes through the right sorts of motions as it computes. Mappings are cheap, though, and we can gerrymander mappings between any physical system and any Turing machine. Since gerrymandered mappings typically involve processes that are fleeting and transient, altering the 'input' to them would cause them to fail to correspond to our mapping. [I3] thus requires a system to support a mapping not just in the actual case but also for any counterfactual input as well.[5] Gerrymanders will not satisfy this, and so I3 rules them out.

These three conditions are the simple counterfactual account of implementation. On the simple counterfactual account, it is clear why premise [5] comes out true. There will be points where a Turing machine did move left, say, but *would have* moved right if the tape square it was reading had been different. [I3] requires the truth of counterfactuals like these. However, what makes those counterfactuals true might be completely inert machinery—if one engine drives the TM head left and another right, the right-moving engine will be completely silent during a move to the left. Its mere presence does not count as part of the computational activity of the system, but implementation requires its presence. Hence [5] above comes out true.

If the simple counterfactual account were the correct theory of implementation, then SSP would likely be unavoidable. However, there are at least three reasons to believe that the simple counterfactual account cannot be an adequate account of computation.

First, it certifies *epiphenomena* as proper computations: if I simulate a run of $\pi(\tau)$ on my (non-TM) home computer, the pixels on my display will satisfy conditions [I1]-[I3]. But clearly, the pixels of my display are not implementing anything—their activity is simply an epiphenomenon of the activity going on in the bowels of my CPU.[6] Second, counterfactuals can be *blocked*. For all I know, a madman may lurk behind me, waiting to smash my computer to bits the moment I try to save this file. If this is the case, then my computer no longer satisfies [I3] with respect to the word processing program: there is at least one chain of input to S for which the relevant counterfactuals about the program would not be true. But surely the mere presence of some external agent should not negate the computational status of my computer.[7] Third, counterfactuals can be made true in *illegitimate* ways. Suppose I map some actual set of transitions to the states of a turtle as it ambles along. I then set up a computer with a camera next to it in such a way that *if* some 'input' had been different, the turtle would have been prodded in

such a way as to produce a different (appropriate) ambling. We should still not say that the turtle computes (if anything, the computer next to it does). Nevertheless, [I3] is still satisfied; it is just satisfied in a jury-rigged, illegitimate manner.[8]

The simple counterfactual account must therefore be replaced with something more robust. There are two (nonexclusive) options for doing so. First, we could look for further counterfactual conditions that rule out problematic cases. Second, we could appeal to constraints on the implementing matter itself. For example: if we require that state transitions to be *caused* by the previous states, rather than just following them in the right sequence, then the problem of epiphenomena is avoided.[9]

I will defend a version of the second strategy. If we require the states of π to map to *dispositions* of the implementing system, then we have an adequate account of computation. This account will also form the basis for a solution to SSP.

3. Dispositions and Computation

Dispositions are (or supervene on) intrinsic properties of objects, and give them conditional causal powers. Typically, dispositions of an object are picked out by specifying the conditions under which the disposition would manifest and the effect that this manifestation would have. Matches have the disposition to *light* when *struck*, batteries in a flashlight have the disposition to *power the bulb* when the *switch is flipped*, and so on.

Dispositions are natural properties to appeal to when we analyze functional systems. Cummins emphasizes that we can analyze many complex systems into a series of interlocking dispositions, where the effect of one disposition includes the manifestation conditions of another disposition (Cummins, 1980). Similarly, implementing a Turing machine can be decomposed into sets of dispositions associated with each entry in the machine table. In the case of a TM, both the manifestation conditions and the effects will typically be complex. The manifestation conditions for a particular effect will include both something about the previous state of the system and the current state of the tape. The effect will itself include both changes to the tape, movement of the head along the tape, and the satisfaction of part of the manifestation conditions for a disposition associated with the next state.

I will make those conditions more formal in a moment. Note first though that *if* TM states could be successfully identified with sets of dispositions, then we could overcome many problems with the simple

counterfactual account. To begin with, dispositions make true counterfactuals under ordinary circumstances. To say that a match has the disposition to light when struck is to say that, under ordinary circumstances, a match *would* light if struck—this is true, of course, even if the match is never actually struck. Similarly, requiring a machine to have the right set of state-dispositions makes true input counterfactuals under normal circumstances: if the input had been different, a different set of manifestation conditions would have obtained, and so a different sequence of disposition-manifestations would have occurred. Hence, a dispositional account and a simple counterfactual account should converge in happy cases.

Dispositions are considerably more powerful than simple counterfactuals, however. The manifestation of a disposition can be blocked in non-ordinary circumstances. If a match is struck in anaerobic conditions it will not light, even though it still possesses the disposition to light when struck. Similarly, my computer may still have the correct set of state-dispositions even if, as a matter of fact, the manifestation of those dispositions would be blocked by external circumstances. Hence, something can implement $\pi(\tau)$ even if some input counterfactuals are blocked.

Something has a disposition in virtue of a first-order intrinsic property.[10] This first-order property is important. For one, it makes it the case that dispositions *cause* their typical effects when those effects occur. Hence, we may eliminate merely epiphenomenal implementations of π, because the states to which we map do not have the right sort of intrinsic causal powers. Further, we can distinguish cases where the disposition actually causes an effect from ones where the effect is reliably caused by some entirely extrinsic system. A mere stick of wood does not have the disposition to light when struck. It does not gain that disposition even if someone lurks waiting to set it ablaze should it happen to be struck. Similarly, we may distinguish cases where the computational counterfactuals are made true in the correct way from the cases where the appropriate computational counterfactuals are made true by merely extrinsic supporting machinery. Hence worries about illegitimate implementations need not arise.

Finally, dispositional theory looks tantalizingly close to something that could solve SSP. Although dispositions can be picked out in terms of their effects on other objects, neither the possession of that disposition nor the manifestation of it requires the presence of an external object. Consider the disposition of a battery to power a flashlight bulb. The disposition holds in virtue of some intrinsic chemical properties of the battery. The manifestation of that disposition consists in a certain chemical event. This chemical event should, under ordinary flashlight-

circumstances, be sufficient to light the bulb. However, the very same disposition can manifest—that is, the very same chemical event can occur—without the presence of a flashlight bulb. Similarly, we might think that computationally relevant dispositions can exist and manifest even if some of the other structure in terms of which they are picked out is not present.

Alas, SSP does not fall so easily. To see why, let us make the proposal on the table more precise. First, let us assume that a system S has features that map on to each of the common features of the Turing architecture—that is, it has a single discretely divided linearly ordered mark-preserving tape, and a head that can move in either direction along the tape and make marks upon it. Let us suppose that π is specified via a standard machine table—that is, a list of each possible combination of state and tape symbol and a corresponding instruction that specifies both what the system should do to the tape and head, and what state should follow. Call each of these scenarios an *entry* in the machine table. Each entry specifies some (possibly empty) set of changes to be made to the tape and the head position. It also specifies an *outgoing* set of entries that could follow it (normally called the next state). The entry that *will* be executed is drawn from the outgoing set but depends also on the state of the tape. The opposite of the outgoing set relation is the *incoming* set; the instructions for an entry will not execute unless it has been immediately preceded by some entry in the incoming set and the tape is as specified for the entry.

Given this, say that some system S implements π on τ just in case:

D1 For each entry in the machine table of π, there is a disposition d of S such that for the associated state x of the tape:

- **D1a** d will manifest just in case both x and any entry in the set of incoming transitions obtains.
- **D1b** The manifestation of d consists in changing the tape and moving the head as per the machine table.
- **D1c** When the outgoing transition is a non-halting one, then the manifestation of d is part of the manifestation conditions for each entry in the set of its outgoing transitions.
- **D1d** When the outgoing transition is a halting one, d does not form the partial implementation conditions corresponding to any other state, and ensures (if need be) that no further computationally relevant transitions will occur.

D2 When run on τ, dispositions of S manifest in the proper order as specified in the machine table for a run of π.

[D1] is the heart of the proposal: it says that for each entry in the machine table, S is disposed to transition as specified just in case the tape is as appropriate *and* is in the state that some previous state has partially enabled. When it manifests, its manifestation consists in causing the head position and tape to change in appropriate ways, and to act as partial manifestation conditions for the next state.

A note about 'states'. There is nothing in S corresponding directly to the present state of a TM; the interlocking sets of dispositions correspond to individual entries of the machine table. However, there is an important sense of state that is preserved. The entries for each state in a machine table all share a partial condition of manifestation—namely, that which is caused by any previous transition that is listed as causing that state. To say that S is in some state is to say that the manifestation conditions of each of a set of dispositions partially obtain. Which disposition actually manifests will depend on the state of the tape as well: thus the dispositions that belong to a state share a part manifestation condition, but differ in their total manifestation conditions according to the state of the tape. [D1d] ensures that the occurrence of the special transition to a halt state ensures that no new states will obtain and (if need be) that the parts of the system are put in a state where they can no longer do any anything. The final condition, [D2], just ensures that a run of $\pi(\tau)$ consists in the appropriate dispositions actually manifesting as prescribed.

This theory of implementation enjoys the benefits we expected from a theory of implementation: it ensures that the right counterfactuals are true under normal circumstances (and true for the right reasons), while allowing that the effects of some dispositions might be blocked under less than ideal circumstances. Alas, it *still* does not avoid SSP. For note that satisfying [D1] may well require the possession of dispositions that never manifest on a particular run. Our dispositional theory requires inert, unmanifested dispositions to be in place to take care of different input. Hence SSP is not yet solved. However, the dispositional theory does offer a way to revise the computationalist thesis that *does* solve SSP.

4. Episodic Implementation

The only stumbling block of a dispositional account is the requirement that *all* possible state-transitions of π correspond to some disposition of S. Suppose we had a theory of episodic implementation—that is, a theory that permits S to implement $\pi(\tau)$ without the possibility of implementing π on every input. Then we could avoid SSP by chang-

ing the computationalist thesis slightly. If conscious states supervene on episodes of implementation of $\pi(\tau)$ rather than full implementations, then (since episodically implementing $\pi(\tau)$ would *not* require the presence of superfluous structure) premise [5] would be false and SSP avoided.

For those used to the simple counterfactual account, episodic implementation might seem like madness. The whole point of excess structure was to block overgeneration; abandon that structure and it looks like we are back where we started from. Upon closer inspection, though, a dispositional account of episodic implementation does not overgenerate.

Formulating a dispositional account of episodic implementation is straightforward. Take the set of entries p in the machine table for π that specify transitions that occur on a run of $\pi(\tau)$. Say that S' episodically implements $\pi(\tau)$ if it satisfies [D1] and [D2] for p. That is, an episodic implementation of $\pi(\tau)$ just consists in manifesting exactly those dispositions that are needed to compute $\pi(\tau)$. Leave it entirely unspecified what *other* dispositions the system might have, so long as they do not conflict with the activity of S as it computes $\pi(\tau)$. An episodic implementation of $\pi(\tau)$ has the dispositions to compute $\pi(\tau)$, and those dispositions alone manifest during an episode of implementation.

Any example will help to make the thesis clear. Let π be specified by the machine table in Figure 1: Suppose some S fully implements π.

State	0	1
1	1:4	**R:2**
2	**L:3**	**R:1**
3	**Halt**	**0:3**
4	Halt	R:4

Figure 1. The Machine Table of π

When started on a tape with an even number of '1's this machine will leave the tape with one more; with an odd number, one less. Imagine the dispositions of S in virtue of which it counts as a full implementation of π to involve whatever material you please. If S is run on a tape containing an odd number of contiguous '1's—say on a tape with $\tau = 3$ '1's—only the dispositions corresponding to the bold entries above will manifest.

Strip down S to create a machine S'. S' has only the dispositions of S that correspond to the bolded entries in the machine table of π; further, they manifest in exactly the same way when S' is started on τ. Although we specified some of these dispositions in S by reference to their relations to other dispositions of S that S' lacks, that should not worry anyone. As we saw above, although we may pick out some dispositions in terms of potential causes and effects, we can make sense of those dispositions manifesting without causing those effects.

S' is an episodic implementation of $\pi(\tau)$. It has exactly the dispositions that an ordinary run of a full implementation of π on τ would manifest. I claim that this is sufficient for it to compute $\pi(\tau)$ in any relevant sense. Now for the obvious objection: does this over-generate? Are we forced to say that the world is chock full of S'-like things?

Certainly not. For one, we are now quite far from cheap mappings between facts about the world and transitions of π. Our mapping identifies stable properties of the world that manifest repeatedly. On a run on τ, our conditions require each of the states to manifest their disposition at least once, and one (corresponding to being in state one on a tape square with a '1') will have to manifest more than once. Episodic implementations of more complex programs will require the very same dispositions to manifest even more times.

The presence of these stable dispositions also allows us to distinguish episodic implementations from superficially similar non-computing systems that are problematic for simple counterfactual accounts. Mark Bishop imagines something similar to episodic implementation and concludes that the result would be a sort of 'wind-up toy' that simply unfolds in a way that maps onto appropriate state-transitions without properly computing (in his (Bishop, 2002), §7). The armature of Maudlin's stripped-down machine presents a similar challenge. For both sorts of system, it looks like we can construct trivial machines that are disposed to do the computationally appropriate thing at each point in response to the state of the system and the tape.

Closer examination reveals this to be a groundless worry. The dispositional theory does not claim that it is sufficient for an episode of implementation that a system have the appropriate dispositions at the computationally appropriate times. It requires that a system have standing dispositions to respond in the correct ways that *manifest* at the correct instant. True 'wind-up toys' presumably change a '1' to a '0' when in state 3 only incidentally. They would *not* do so at other points, even under normal conditions. So if while in state 3 at some later stage they were moved artificially along the tape back to a '1', they wold not be disposed to change that 1 to a 0—indeed, they would just continue in whatever sequence they had been wound to follow. Hence, they do not

have the required dispositions, and the dispositional account correctly rules them out.[11]

Requiring standing dispositions—again, stable, causal properties of a system—allows for a middle ground between profligate mapping and full implementation. This should be clear for those who like counterfactual accounts. The presence of standing dispositions makes true a subset of the counterfactuals true of a full implementation of π. In this case, our episodic implementation happens to have just those dispositions that manifest when π is started any tape with an odd sequence of contiguous '1's. Hence, S' manifests not just the dispositions that π manifests on τ, but also those that would manifest on a tape with five ones, seven ones, etc. So while our episodic implementation of $\pi(\tau)$ is more limited than a full implementation of π, it is bound by constraints that are not satisfied by patches of wall and buckets of water.

The fact that an episodic implementation still makes true (under ordinary circumstances) a range of input counterfactuals further shows that not everything counts as an episode of implementation of $\pi(\tau)$. For consider the following TM π', specified by the machine table in figure 2: Our system S' is also a perfectly ordinary *full* implementation of

State	0	1
1	Halt	R:2
2	L:3	R:1
3	Halt	0:3

Figure 2. The Machine Table of π'

π'; indeed, any episodic implementation of $\pi(\tau)$ will implement π' (or a small number of similar machines).[12] So there is no reason to think that an episodic implementation of $\pi(\tau)$ overgenerates unless there is reason to think that full implementations of π' overgenerate. But there is no reason at all to think that. Nor is there reason to think that there is a general problem in the area. Some simple episodic implementations may be extremely common, of course—but that is only because some simple full programs are extremely common as well, and widely implemented. That is not a general problem for the computationalist about consciousness; unless she is a panpsychist, she will assume that the episodes of consciousness supervene on less commonly implemented programs.

This leads to what I think is the most attractive feature of a dispositional theory of implementation. Call any proper subset p of machine-table entries of π a *subroutine* of π. Call a machine table that replaces the entries of π not in p with simple halt instructions a *restriction* of π. Now, if for any τ that activates only dispositions corresponding to members of p, the dispositional theory of computationalism allows us to assert the *equivalence of subroutines, episodic implementations, and restrictions* of π. According to a dispositional theory of computation, exactly the same actual activity occurs when a full implementation of π runs on τ, an episode of implementation of $\pi(\tau)$ occurs, or a restricted program π' is run on τ. The computationalist should thus assert that these are entirely equivalent, at least as far as conscious experience is concerned.

This equivalence is exactly what the computationalist should want if he thinks that conscious states can be episodic. What consciousness depends on is actual activity. As such, if executing a subroutine of π on τ is sufficient for having some experience E, then an episodic implementation of $\pi(\tau)$ should be sufficient to have E as well. And that is just the same as saying that a program that could implement *only* the E subroutine would still be conscious of E should it manifest.

We now have the tools in hand to defeat SSP. Revise the computationalist thesis as follows:

Computationalist Thesis (Episodic version): Conscious states supervene on episodes of implementations of some program.

Actual activity alone determines whether something is episodically implementing $\pi(\tau)$. Therefore, it follows that no two things can differ in episodic implementation status without also differing in actual activity. The revised version of premise [5] is false, and the argument for SSP unsound.

5. Conclusion

What generates SSP is a particular way of thinking about implementation and computation of an input. On standard theories of implementation, one first gives an account of what it takes to implement π. One then uses this to construct a definition of computing π for some τ. That just amounts to implementing π *tout court*, and then being started on τ. Thus the property of being a *computer of* π is logically prior to the activity of *computing* π *on* τ. Being a computer of π thus requires more structure than is invoked in the activity itself. Running into SSP is an unavoidable consequence of this strategy.

Dispositional computationalism allows us to reverse the order of analysis. We can first determine what dispositions the activity of computing π on τ would require, and define episodic implementation in terms of that activity. A full implementation can be built up out of a sufficient set of episodic implementations that are related in the proper way. In the case of π above, we could take the two overlapping subroutines—one invoked in odd cases and the other invoked in even cases—and define the conditions on episodic implementation of each.

To be a full implementation of π is just to implement both episodic implementations in a way that ensures that the overlapping entries of each correspond to the same dispositions. More generally, take any arbitrary set σ of subroutines of π such that for any τ, there is some subroutine in σ that includes all of the entries sufficient to compute $\pi(\tau)$. Then S will be a full implementation of π just in case there is a single set of dispositions that is sufficient to episodically implement every member of σ.

On this order of analysis, episodic implementation—and thus computational activity—comes first. Computing an input does not require superfluous structure; for any input there exists a set of actual disposition manifestations, the occurrence of which is sufficient for computing that program on that input. Of course, when we talk about full implementation, the dispositional account will say many familiar things: it will allow for the truth of the same counterfactuals about input, for example, and a full implementation in normal conditions will always satisfy simple counterfactual conditions.

The main advantage of moving to a dispositional account is the flexibility it gives in more complicated cases and the focus on activity rather than structure. It is the latter that keeps us from falling prey to SSP, since computing an input is always a matter of changes to local bits of an implementing systems. Computationalism about consciousness thus need not conflict with intuitions about the actual activity upon which conscious states should supervene.[13]

Notes

[1]The argument (as well as a defense of the activity thesis) is found in (Maudlin, 1989). Maudlin's paper has not received the attention that it deserves, in part because much of the paper is devoted to constructing an elaborate machine, Olympia, to support his claims. I will not discuss the details of Olympia in this paper; focus on the example can easily obscure the main point, and the main point is interesting even if Olympia fails. For aficionados of the original, I discuss Olympia in and around footnote 11. For additional discussion, see also (Barnes, 1991).

[2]You might worry about the correct way to cash out phrases like 'actual activity' or 'computationally active'. The notion is presumably meant to capture something

like: only intrinsic properties of positive events can form the supervenience base for conscious states. The non-occurrence of some property, or the instantiation of some merely extrinsic property, cannot be required for conscious states. Rather than spend time on these equally problematic notions, we may simply recast the claim as a constraint on theories about the supervenience base. Computationalists should hold that the instantiating stuff is different in *some* way or other when that stuff is in a computational state rather than merely potentially in that state. Read 'actual activity' as picking out that difference.

[3]Of course, there are independent reasons to think that computation cannot exhaust the nature of transportation. Being a transportation system also requires that you move some material around, for one! Thanks to an anonymous reviewer for urging me to clarify the point.

[4]Michael Antony has offered a parallel argument against functionalism as a whole (Antony, 1994). Much of what follows could also be applied *mutis mutandis* to Antony's argument, but that is a larger task for another time.

[5]For a careful and detailed defense of this solution, see (Copeland, 1996).

[6]The example first appears in (Dreyfus, 1972).

[7]Thanks to Karen Bennett for the example.

[8]This is inspired by the device in (Maudlin, 1989). Note that it differs from the emergentism case because there is a causal connection between the turtles and any future state—directly in the actual case, indirectly via the computer in the counterfactual case.

[9]Note that this is not necessarily in tension with a counterfactual account: if I think that causation itself admits of a purely counterfactual analysis, then this expanded account may be cashed out in entirely counterfactual terms. Similarly, the dispositional account will still be a counterfactual account if dispositions can be exhaustively analyzed in terms of counterfactuals. There are strong reasons to think that they cannot be so analyzed, in part for the same sorts of worries about blocking and illegitimate counterfactuals that cause trouble for the simple counterfactual account of implementation (Martin, 1994; Fara, 2005). I take no stand on that issue here, but I do emphasize that even sophisticated counterfactual accounts of dispositions (like the one sketched in (Lewis, 1997)) make appeal to intrinsic properties of the possessor of a disposition. Note that the presence of intrinsic properties upon which dispositions supervene is crucial to the account I give, and so any reconstruction in counterfactual terms will have to preserve that feature.

[10]I remain neutral between identifying the disposition with the first-order property or treating it as a second-order property that is realized in each case by some first-order property; for discussion of why this might matter, see (Kim, 1998). As for whether second-order properties are properly the causes of events, see the discussion in (Jackson and Pettit, 1990). For the present discussion, we only need the causal facts to supervene on the intrinsic properties of the first-order realizer of the disposition; we need not take a stand on whether it is identical to those properties.

[11] A further complication arises for the full version of Maudlin's Olympia. If the armature is moved or the input changed, the supplemental copies of Klara will kick in and ensure the computationally appropriate transitions. Being careful about what we count as the relevant dispositions, however, will show that either the full version of Olympia does not implement π or else is not a demonstration of SSP.

Take some putative disposition d corresponding to a entry of π. Either it is a disposition solely of the armature, or it is a disposition of a complex disjunctive object consisting of parts of the armature and parts of the multiple copies of Klara.

If the former, then the whole does not implement a TM, for the dispositional account requires a *single* disposition that corresponds to the relevant machine table entry. In that case, the multiple copies of Klara make true input counterfactuals illegitimately, and as we saw in section 3 the dispositional account rules out such cases. On the other hand, the single disposition could be a single disposition that holds of a gerrymandered swath of the armatures plus copies of Klara. But then, contrary to Maudlin's assumption, the copies of Klara *are* computationally active during an ordinary run of $\pi(\tau)$, because they are (part of) something that is manifesting a computationally relevant disposition! Hence they are not superfluous. Further, any attempt to block off versions of Klara from the armature constitutes a change to the intrinsic properties of some gerrymandered slice of Olympia. So SSP does not threaten: there is no change in the computational status of Olympia without a change in either dispositional structure or actual activity.

[12]For any unspecified entry in the machine table, there will be a finite number of computational transitions that S could make; should it fail to make any of those, it will have halted. For an n-state $(m-1)$-symbol machine, there will be $2mn+1$ ways to complete an unspecified machine table entry. While the number of completions may be large for some episodic implementations, it is still far from overgeneration. Further, any completion will share a subroutine, and the computationalist thesis arguably *should* treat them as equivalent.

[13]Thanks to Adam Elga for introducing me to Maudlin's work and for a number of productive discussions on the topic, and to Karen Bennett and Sinan Dogrammaci for helpful comments on an earlier version of this paper. Special thanks is due to an anonymous reviewer whose comments helped immensely in refining and focusing the current work.

References

Antony, M.: 1994, 'Against Functionalist Theories of Consciousness'. *Mind and Language* **9**, 105–123.

Barnes, E.: 1991, 'The Causal History of Computational Activity: Maudlin and Olympia'. *The Journal of Philosophy* **88**, 304–316.

Bishop, M.: 2002, 'Counterfactuals Cannot Count: A Rejoinder to David Chalmers'. *Consciousness and Cognition* **11**, 642–652.

Copeland, B. J.: 1996, 'What is Computing?'. *Synthese* **108**, 335–359.

Cummins, R.: 1980, 'Functional Analysis'. In: N. Block (ed.): *Readings in the Philosophy of Psychology*, Vol. 1. Harvard University Press, pp. 185–191.

Dreyfus, H.: 1972, *What Computers Can't Do*. MIT Press.

Fara, M.: 2005, 'Dispositions and Habituals'. *Nous* **39**, 43–82.

Jackson, F. and P. Pettit: 1990, 'Program Explanation: A General Perspective'. *Analysis* **50**(2), 107–117.

Kim, J.: 1998, *Mind in a Physical World*. MIT Press.

Lewis, D.: 1997, 'Finkish Dispositions'. *The Philosophical Quarterly* **47**, 143–158.

Martin, C.: 1994, 'Dispositions and Conditionals'. *The Philosophical Quarterly* **44**, 1–8.

Maudlin, T.: 1989, 'Computation and Consciousness'. *The Journal of Philosophy* **48**, 407–32.

Searle, J.: 1990, 'Is the Brain a Digital Computer?'. *Proceedings and Addresses of the American Philosophical Association* **64**(3), 21–37.