Computation, Consciousness, and "Computation and Consciousness"*

Colin Klein
Macquarie University
Sydney NSW, 2109
Australia
colin.klein@mq.edu.au

*For *Routledge Handbook of the Computational Mind*, ed Mark Sprevak and Matteo Columbo. Draft 1D, compiled October 8, 2016.

1 Three Preliminaries

1.1 Ms W

The *Symphonie Philosophique* reserves by ancient custom a chair for the viola de gamba. The current virtuosa, Ms W, is universally acclaimed for her work in other ensembles. Yet the symphony dropped early music from its repertoire many seasons ago. In her many appearances, Ms W. has never played a note.

Subscribers differ on Ms W's aesthetic contributions. One group insists that her presence clearly enhances the aesthetic value of performances. As Miles Davis taught us, music is as much about the notes you don't play as the ones you do. Her presence gives the symphony far more *possibilities* against which the actual music can be juxtaposed, and thereby a richness and splendor that other ensembles cannot match. (The most breathless insist that if you couldn't play viola de gamba parts, you really hardly *count* as an orchestra at all.) Skeptics are more blunt. She does not add to the music. How could she? She does not play.

The core of this dispute is over the constitutive role of counterfactual properties. One side thinks that aesthetic value (whatever it is) depends both on what actually happens and on what *could have* happened. Aesthetic value is, on this picture, a *counterfactually loaded* property. Many goods are like this (Pettit, 2015). The other side thinks that aesthetic value is not counterfactually loaded, and that only the actual matters to the aesthetic. Many properties are also like this. Hence there's room for debate.

The question of whether a property is a counterfactually loaded one is behind many philosophical disputes. This piece will talk about an under-appreciated place where the conflict arises, and how it causes headaches for the computationalist about conscious experience.

1.2 The Action Max

When I was young, several unscrupulous companies offered a low-cost alternative to video game consoles. The Action Max came with a small light-gun

setup and a VCR tape. I watched the commercials in breathless wonder: how could an ordinary VCR play video games? My parents were less interested in the answer, so I had to wait until I was an adult to learn what disappointment I had avoided. The Action Max played a tape of a thrilling space battle that you could shoot at. But it was just an ordinary VCR tape: nothing you did would make a difference to the outcome. The same ships would explode, the same dangers would be narrowly avoided, the same triumphal success awaited at the end. Calling this a *video game* seems like a stretch. That is not a claim about what happens on the screen, or even how well the actual action matches up with the players intentions. Even if the movie exactly matched what a player *actually* did, there is no room for departure. What makes something a game is options and contingency, and the Action Max lacked those. *Being a video game*, in short, is a counterfactually loaded property.

Video games are a kind of computation, and the point holds true of computation more generally. Consider a humble Turing machine (TM). Call him Simon. Simon implements a reasonably complex machine table that lets him reduce his input into its prime factors. Put Simon on some input τ and he'll click away merrily. Now suppose our unscrupulous Turing machine dealer offers a discount on a simpler model, Theodore. Theodore cuts costs: he does not have a machine table, and instead just has a tape recording that contains each transition that Simon would make on τ. Each change to the tape, each change of internal state indicator, each movement will be faithfully reproduced. Theodore will also factor τ, and just as fast a Simon does! Hardly a bargain, you protest. Given any input other than τ, Theodore will click along just the same. But computation must be counterfactually sensitive. Simon, and Simon alone, counts as implementing σ because he could do the right thing on *any* input. Theodore is not just worse than Simon; his lack of counterfactual sensitivity means he doesn't even count as computing in the first place.

Computation thus seems like a counterfactually loaded property. This is important for more than just the fight against shady Turing machine dealers. *Computationalism* about consciousness is the thesis that phenomenal properties supervene on computational ones—that is, that being the right kind of computer is enough to be conscious. Computationalism is exciting because it is a broad tent: brains compute, silicon does too, and so your laptop (it seems) could be just as conscious as you. A class of skeptical arguments pushes back by saying that computationalism is far *too* permissive, implausibly so. The molecular meanderings of buckets of water and stretches of brick

can be mapped to the actual activity of a TM too (Searle, 1990; Putnam, 1991). Surely bricks aren't conscious, though!

Nor do they compute, goes the standard response. At a minimum, computing some function π requires that a system's activity *would have been* isomorphic to some abstract machine's for any of the inputs over which π is defined. Bricks don't manage that. Attention to counterfactuals is thus necessary to avoid explosion. Which is all again by way of saying that computation is a counterfactually loaded notion. Two things can differ in computational status—in whether or what they are computing—without differing in what they're actually doing.

1.3 Anaesthetics

What about consciousness? Start with things that remove it. The mechanism of action of inhalational anaethetics remains something of a scientific puzzle, especially given the ability of chemically inert gasses like xenon to (reversibly) remove consciousness at high concentrations. Most theories agree that interactions with the lipids in the neural membrane are crucial, as they alter the ability of the neuron to fire given appropriate stimulation (Frost, 2015). Here there's an important nit to pick. The action can't *just* just via changing the membrane properties, however. For anaesthetics affect all neurons indiscriminately (or let's suppose). What makes a difference is that neurons that would have fired are silenced. Effects on neurons that *wouldn't* have fired anyway (one might think) don't make a difference to consciousness. To knock you out, in other words, I only need to make the firing neurons quiet—the already quiet ones aren't making a difference to what you experience.

Indeed, that seems to be a quite general intuition not just about the loss of consciousness, but about the contents of consciousness altogether. A given episode of phenomenal experience supervenes (most think) on brain processes. That is to say, what you're aware of over an interval supervenes on the *actual activity* of the brain over that interval. Intuitively, neurons that are completely silent and inert over the interval can't make any contribution to what you're aware of. This is to say very little about what 'activity' amounts to: the important thing is just that phenomenal awareness depends on what a system does, not on what it would do or might have done in other circumstances.

This is not a universally held thesis, but denying it forces you to say really odd things. Consider Tononi's influential Information Integration Theory of consciousness (IIT). (Tononi, 2004) is committed to the importance of a 'qualia space' (Tononi, 2004) or a 'cause-effect space' (Tononi and Koch, 2015) that traces out all of the possible informational states a system might be in. A conscious experience is "a different point in the multidimensional qualia space", and "in every case, it is the activity state of all elements of the complex that defines a given conscious state, and both active and inactive elements count." (Tononi, 2004, 9). (He would, one imagines, be a supporter of Ms W's work.)Yet this is a very *weird* commitment. As Fekete and Edleman point in a nice discussion

> if silent units indeed contribute to a subjects phenomenal experience, cooling some of them reversibly (which would inactivate them without causing them to fire) would alter the experience. Having offered this prediction, Tononi stops short of explicitly addressing the crux of the problem: how can presently inactive silent or silenced units contribute to present experience? ... It seems to us that making experience depend on a potentiality without explaining how it actually comes to pass falls short of completing the explanatory move. (Fekete and Edelman, 2011, 813)

That does seem to be the common intuition. Consciousness is an active, constructive process, and inactive units cannot add anything.

Of course, spelling out 'actual activity' brings more complications than one might expect.[1] Let's put them aside. I assume that there is some intuitive distinction between the active and the inert, and that it would be surprising

[1] In an awake, conscious adult the whole brain is constantly active, and this high tonic level of activity appears to be a prerequisite for conscious experience (Raichle and Snyder, 2007). While action potentials are discrete and distinct there is near-constant churn at the synapses, and balanced excitatory and inhibitory input can be neurocomputationally important even if it doesn't result in firing (Logothetis, 2008). Firing *delays* are also neurocomputationally important due to spike-timing dependent plasticity and timing- and phase-dependent processing (Izhikevich, 2006). Further afield, insofar as the actual activity thesis requires on a notion of *simultaneous* activity it is incompatible with special relativity. Whatever the supervenience base for conscious experience, it arguably ought to be something that is invariant across inertial reference frames (Power, 2010). The latter may seem especially *recherché*, but the easiest ways to avoid it—moving to causality or ordering rather than strict simultaneity—would require a careful rethink of the setup.

if the inert actually made a contribution to consciousness (however that is spelled out).

2 Maudlin's Argument

Simple stories have led to a troubling conflict. Computation is counterfactually loaded: it depends in part on what could happen. Consciousness isn't: it depends only on what does happen. That mismatch seems like it ought to trouble the computationalist.

Tim Maudlin's (1989) "Computation and Consciousness" argues just that. Maudlin's argument remains relatively obscure, in part because the bulk of his paper is devoted to constructing an elaborate example that is easily misinterpreted. That is a shame, as the core argument is simple and powerful. The preliminaries above are there to bring out what I take to be the core of Maudlin's argument. I present it here in schematic form:

1. If computationalism is true, then there is some Turing machine π which, when run on input τ, is sufficient to have phenomenal experience.
2. Two things can differ in whether they implement π because of differences in wholly inert machinery.
3. Two things cannot differ in whether they are conscious because of differences in wholly inert machinery.

   [Contradiction]

$\therefore$ Computationalism is false.

Premise 1 is meant to be a basic commitment of the computationalist, while premise 3 depends on the intuition sketched above. Premise 2 seems plausible insofar as computation is counterfactually loaded. But an ordinary TM like Simon may have no wholly inert machinery while processing τ—everything is active at some point during the run.

Maudlin's central innovation is a thought experiment designed to show how to make a TM where the counterfactuals are supported by wholly inert machinery. (Interested readers should consult his paper for details of the machines Olympia and Klara: I'll give a streamlined version here.) Return to Simon and Theodore. Let us suppose that Simon is configured to implement to

compute π on τ. 'No problem!' says the computationalist. 'Turing machines can be conscious!' Now Theodore apes what Simon does simply by following a pre-recorded sequence. Following a pre-recorded sequence seems like the sort of thing that isn't enough to be conscious. (In Maudlin's example, the corresponding machine simply sprays water from a hose.) 'No problem!' says the computationalist. 'That thing is too simple to be conscious. But it doesn't compute, either, and our thesis is about computation!'

Now the kicker. Consider a deluxe machine, Alvin. Alvin is a copy of Theodore with an extra sensor and a powered-down copy of Simon strapped on the back. On a run on τ, Alvin simply reads off of Theodore's tape as usual. If the input deviates from τ, however, the extra sensor switches off the Theodore-bits, powers up the Simon-bits, sets them in the appropriate way, and starts it running.[2]

Alvin is now counterfactually sensitive: he has all of the range of Simon. Hence, Maudlin claims, he computes π. But Alvin shows that premise 2 is correct: the difference between computing and not computing is in the inert bits of his machinery. Note that we could change Alvin to a non-computing thing by sticking a passive bit of insulation in the right spot and thereby prevent the Simon-copy from from powering up. On a run on τ, Alvin's actual activity doesn't differ from the non-computing, non-conscious Theodore's—again, the added bits are all completely inert. If Theodore's simplistic activity isn't enough to be conscious, then Alvin's isn't either.

What to do? Giving up on premise 3 would be too strange. As Fekete and Edleman noted, it would be weird to think that we could be changed to zombies solely by (say) anesthetizing some nerves that were't going to fire anyway. Giving up premise 2 won't work: counterfactuals are necessary to prevent an explosion in the things that count as computing. But that leaves only computationalism to abandon. Hence consciousness cannot depend on computation. The modal mismatch between their supervenience bases assures this.

A surprising conclusion, not in the least because it depends on what appear to be relatively lightweight assumptions. To be clear, I do not think Maudlin's argument succeeds. But I think that it has the feature of all beautiful thought

[2]Part of the cleverness of Maudlin's example is to show how this switch function can be made using stuff that itself remains wholly inert on a run on the special input τ. I leave that as an exercise for the reader.

experiments in philosophy: pushing back helps us clarify our assumptions. In the case of computationalism, it forces us to look at what seemed like a very simple and general thesis and constrain it in interesting ways.

Before moving on, it is worth distinguishing Maudlin's argument from a number of other arguments in the literature. Though it has interesting relationships to several other anti-computationalist arguments, it is *sui generis*. I think the reception of the argument has been blunted in part because many readers confuse it with one of these other, related arguments.

Maudlin's work bears an important relationship to various 'exploding implementation' arguments, which claim that any sufficiently complex object can implement any finite-state automaton (Searle, 1990; Putnam, 1991; Chalmers, 1996b; Bishop, 2009a,b). Maudlin's argument does not require exploding implementation to be true, however. Indeed, he presupposes (at least for the sake of argument) that appropriate counterfactual restrictions are sufficient to avoid explosions. One might instead reframe his argument as a dilemma for the computationalist: either one accepts explosion, or else one accepts a modal mismatch between computation and consciousness.

Olympia and Alvin may seem like instances of a 'funny implementation' strategy. Funny implementation arguments (such as Block (1978)'s Chinese Nation) rely on intuitions that a non-standard implementation of a program couldn't possible be conscious. Though Olympia has occasionally been taken this way (see e.g. Fekete and Edelman (2011)), I agree with Maudlin that the force is different. The argument is not merely that machines like Alvin look *strange*. Rather, the argument is that Theodore isn't conscious by the computationalist's own lights because he does not compute, and Alvin does not differ from Theodore in ways that ought to matter for consciousness. The problem is thus not with the oddness of the machine, but its relationship to other machines that aren't conscious by anyone's lights.

The argument also bears an interesting relationship to some of Searle's (1980; 1990) critiques of computationalism. (This link is explored further by Bishop (2009a,b)) Many of Searle's critiques can be read as expressing a concern over the fact that various conditions have to be in place for something to *count as* a computer. Yet it doesn't make sense to ask whether something 'counts as' conscious—either you are or you aren't. Searle's concerns tend to focus on the need for something with non-derivative intentionality to interpret something as a computer (which is in turn wrapped up with questions about the status of the symbols used by the computer, which is Searle's ultimate concern).

Maudlin's argument is related, but more sparse. There is no issue here about the semantic interpretability of the symbols that Alvin uses. Even if all it takes to 'count as' a computer is counterfactual input sensitivity, that alone is enough to cause a mismatch between the demands of the phenomenal and the computational.

3 Consciousness, Processes, and Programs

There are many possible responses to Maudlin. Rather than catalog them individually[3], I want to come at things from a slightly different angle.

Begin with an odd feature of of Maudlin's argument. Computationalism, in my hand-waving introduction of the thesis, was a claim about *computation* broadly construed. Maudlin's argument, by contrast, reads computationalism more narrowly as the claim that some *Turing machine* can be conscious. Maudlin is explicit about the latter: his 'necessity condition' claims that computationalism "asserts that any conscious entity must be describable as a nontrivial Turing machine running a nontrivial program." (1989, 420). No defense is given of this. Presumably the idea is that TMs are universal, and that universal machines can compute whatever function you please. Hence TMs (despite their notoriously unwieldy nature) are as good a candidate for conscious experience as anything else.

Yet this embodies a substantial assumption. As Sprevak notes when discussing Searle, while a universal TM can compute any function that any other architecture can compute,

> ... it is not true that a universal computer can run any program. The programs that a computer (universal or otherwise) can run depend on that machines architecture. Certain architectures can run some programs and not others. Programs are at the algorithmic level, and that level is tied to the implementation on particular machines. (2007, 759)

I think this is absolutely right. Computationalism could be formulated in terms of input-output functions. It needn't be, and it shouldn't be.

[3] A task mostly done, albeit in a Maudlin-sympathetic way, by (Bartlett, 2012).

Let's flesh that out. Begin with some terminology. An *architecture* is a set of primitive operations and basic resources available for building computations (Pylyshyn, 1984, 93). The Turing machine architecture has as primitives (i) changing (or preserving) a single square of the tape (ii) moving the head along the tape one square left or right, and (iii) changing state to one of a finite number of other states.A computational architecture specifies its primitives at a relatively high level of abstraction: we ought not care what a TM tape is made of, or how long it takes the head to move to a new position. A *computational process*, as I use the term, is a temporally extended series of primitive operations. A *program* is a set of instructions for building a machine that, when combined with appropriate context, generates a computational process. The relationships between contexts and outputs defines the mathematical function that a machine computes.

Many different machines can compute the same mathematical function: this is the upshot of TM universality. But different architectures can do so in very different ways, because they have different primitives available from which to build computational processes. That is why, as Sprevak notes, we need to make clear whether computationalism is a thesis about mathematical functions (in which case we don't have to care about the architecture of our machine) or whether it is better expressed in terms of processes, architectures, or programs (in which case we do).

Consciousness can't depend on what function a machine computes. Sprevak (2007) gives several good arguments. Here is another. Take some computable[4] function f; make it one that a TM takes many steps to perform. Any computable function can be a primitive in some architecture: that is, one can posit a computational architecture that performs f as a single operation. Much of early chip design consisted in identifying useful functions that could be made into primitives: most computer architectures today have, for example, a primitive that does bit shifting, which takes multiple steps on a TM. The Intel 8087 introduced primitive support for functions that approximated trigonometric functions, for example. From the point of view of that architecture, computing sines and cosines is a single operation, and so occurs in a single computational step.[5] Of course, the implementation of that primitive might be itself be complex. That is irrelevant, just as the number

[4]Or non-computable function, for that matter. The point can be made with computable functions, though I have argued elsewhere that Olympia is herself actually an oracle machine (Klein, 2015).

[5]More or less. As you might expect, approximating transcendental functions in floating point is not for the faint of heart; see (Ferguson et al., 2015).

of gears or wires or gallons that implements a single primitive in a TM is computationally irrelevant.

Return to consciousness. Take any mathematical function f_c, the computation of which is supposed to be enough for an extended bout of phenomenal experience. We can posit an architecture has f_c as a computational primitive. But surely, executing a single primitive in a single computational step is not enough for conscious experience. From the computational point of view, there's no structure at all there. Hence there's no way to see how (for example) different elements of that conscious episode might change, because there's no structure in the computational supervenience base that could be changed.

Yet phenomenal experiences clearly have a combinatorial structure, and computationalists ought to say that this is mirrored by the combinatorial structure of the computations upon which it supervenes. Indeed, the combinatorial structure of computation is one of the things that makes computationalism so great in the first place.

The upshot is clear: phenomenal consciousness isn't a matter of what function you compute It must depend on something more *computationally* fine-grained. That is not an abandonment of computationalism, though. It is a move from an implausible formulation to a more sensitive one.

The link between computational complexity and complexity of phenomenal experience also allows us to say something about a traditional objection to computationalism, usually attributed to Ned Block.[6] For any function you please, so long as it has a finite domain you can imagine a simple lookup table that computes the same function. The typical response is either to deny that sufficiently large lookup tables are physically possible (Dennett, 1998), or else to deny that lookup tables count as computations. The former feels unprincipled, and the latter would be surprising: lookup tables are widely used in computer programming, and play critical roles in lower-level programming (Duntemann, 2011, Ch 10).

Instead, we might point out that lookup tables are suspicious precisely because lookup requires a batch of information to be present as a computational

[6]From his (Block, 1981). Though the lookup table arises as an objection to the Turing test—the link to phenomenal consciousness comes via a conflation with the the Nation of China thought experiment from (Block, 1978).

primitive, and because lookup is usually envisioned as either a primitive operation or composed out of a small and repetitive handful of primitives. That mismatch between the demands of computation and of consciousness is what drives our intuition that that simple lookups can't be sufficient for consciousness.

Input-output function is thus the wrong grain of analysis for consciousness. That's not a huge surprise: it's arguably the wrong grain of analysis for cognitive science as well (Pylyshyn, 1984). Instead, we should look to architectures and the processes that belong to them. Note that this doesn't (yet) require constraining consciousness to a particular architecture. Architectures themselves can be taxonomized. 'Turing machine' picks out a very general architecture which itself has many species depending on the number of symbols and states available. Different species of TM have different complexity profiles as well—you can do things more quickly with numerous symbols than you can with just two, for example.

Higher-level taxonomy is also possible: there are important differences between architectures that process input serially from ones that have primitives that operate over the entire input at once.[7] Similarly, architectures differ in the primitive data structures available to them, the kinds and access parameters of memory available, and so on. Such concerns are far from ad hoc. Architecture is important because it places constraints on the time and space it would take to compute a particular mathematical function. These differences in computational complexity are the bread and butter of computer science (Aaronson, 2015). They may well matter for consciousness.

A final caveat before returning to Maudlin. I have talked about architectures and processes rather than *programs*. The relationship between a program and the process it gives rise to can be a complicated affair. I follow in thinking that programs "would not be of much use unless they could be used to engender (not just to describe) computations. Programs, that is, are treated as much as *prescriptions* as *descriptions*" (1996, 36).[8] In *some* cases, pro-

[7]Indeed, sufficiently exotic architectures can have surprising advantages over traditional architectures—Dewdney (1984) gives a delightful example of the speedups available in list sorting with by analog computers that utilize dry spaghetti and a sufficiently large forklift.

[8]Of course, not all computations involve running programs, and some computational processes (like those implemented in brains) weren't built by someone following a program. This has caused confusion in the past, but it's simply another reason not to put too much weight on programs as the unit of analysis. Instead, when a cognitive neuroscientist (say) gives something that looks like pseudocode but for the brain, we can say that it

grams simply specify a series of primitive operations to be performed. But that is typically only the case for very, very low-level languages. Higher-level languages are usually intended to be architecture-independent. They must be *compiled* before they specify a process in a particular architecture. Compilation is thus a complex relationship of *translation*.[9] In less familiar programming styles, a program may not even specify the temporal order of operations, leaving that to be determined by context (Klein, 2012). Because of cases like these, I think it is better to consider programs as a series of *directions* for building a computational process. Those directions must themselves be interpreted in order to actually build a process.[10]

If we don't distinguish programs and the processes they can give rise to after compilation, we can also generate additional, and unnecessary, versions of Maudlin's argument. Bishop (2009b), for example, gives several purported counterexamples to computation that rely on the fact that optimizing compilers might eliminate certain code paths. Similarly, we might note that compilers can speed execution by unrolling loops or by baking control logic into a simple lookup table. But that means that a good compiler might take a program that *appears* to specify a complex process and run it in an architecture where the complexity is packed into a single primitive.

Fair enough. That only shows that the computationalist ought to be wary of—or at least very, very careful about—formulating their thesis in terms of programs. It is the processes that computers run, and the computational processes that are build out of them, that are the appropriate grounds for a supervenience claim.

4 Alvin's True Nature

If you buy all that, then it should now be pretty clear what the computationalist ought to say about machines like Olympia and Alvin. Both were

specifies instructions to build a model (given some implicit semantics) that is isomorphic to the brain in the relevant respects. In general, respecting the distinction between descriptions of models and descriptions of the world is the key to solving many puzzles about implementation (Klein, 2013).

[9] Thanks to Peter Clutton for emphasizing this point to me.

[10] 'Interpretation' here is meant to be metaphysically lightweight: humans can interpret simple programs to build a machine, but the power of computers is that they can also interpret programs given in a suitable format.

claimed to be TMs on the basis of their actual and counterfactual similarity to what a TM would do. But there are other architectures that *emulate* TMs. Olympia and Alvin belong to one of those, I claim.

Consider an architecture $\mathcal{A}$, which has all of the primitives available to TMs plus an additional lookup instruction: on one state i, they transform the tape according to a certain pre-determined sequence j. Otherwise, they transform the tape according to j up to the point where it diverges from i, and then switch over to running a machine table p. One might have j match exactly what what a TM running p on i would do. That is what Olympia and Alvin do. Yet nothing constrains machines in $\mathcal{A}$ to do so. Some machines in $\mathcal{A}$ might change the tape in a way that no TM actually could—for example, the same state and input might lead to a different output at a later computational step, which is something that a TM cannot do. This suggests that $\mathcal{A}$ machines are different in kind from TMs, even when they happen to act in similar ways.

On Maudlin's view of computation, there is no space between 'acting like a Turing machine' and 'being a Turing machine.' The latter is just defined in terms of the former. But there's good reason to reject this notion, precisely because it can't distinguish between implementation and emulation. By contrast, the view of implementation I have advocated is one on which an architecture defines a set of *mechanisms* (Piccinini, 2007, 2015). These mechanisms have common spatiotemporal parts, each of which interact causally. By those lights, Alvin and Olympia don't have the right set of parts interacting in the right way to implement a TM.

Here's another way to put the point. Upgrading Alvin to a machine that computed an entirely different function on i would be a trivial matter from the point of implementation: one would need to change only the sequence j. On the other hand, changing a TM to compute a different function can require arbitrarily many changes to the machine table.

From a programming perspective, recall, the architecture provides the functional primitives from which more complex processes must be built. This means that the basic operations defined by the computational architecture are also the *basic loci of intervention* upon a computational system. An architecture that provides both multiplication and addition as primitive operations can be directly intervened upon in a different way than one that provides only addition as basic and derives multiplication.

Considerations about computational architectures are explanatorily critical insofar as we take an interventionist stance on explanation. This is another important departure from accounts of computation that focus solely on isomorphism between machine tables and implementing machinery (Chalmers, 2011). Explanatory claims are claims about causal influence. Claims about causal influence require a well-defined sense in which one could manipulate the system in question from one state to another (Woodward, 2003, 115ff). To say that something belongs to a computational architecture, then, is to define the basic computational interventions that could be made upon it. That is what a mechanist approach to computation gives us.

Indeed, it's worth nothing that a great number of responses to Maudlin and related arguments rely precisely on thinking about the causal structure of the implementing mechanism. Barnes (1991) focuses on the causal chain that leads from input to output. Klein (2008) argues that we ought to look to the dispositions that make true computational counterfactuals, and that these in turn must have their categorical basis in well-defined parts of the system. Bartlett (2012) complains that Klein doesn't provide a story about what counts as a well-defined part. Whatever the story, however, it seems like it ought to link into a more general mechanistic story—perhaps by appealing (for example) to the mutual manipulability of part and whole (Craver, 2007).

Yet it is clear that Alvin and Olympia do have different sets of parts to that of an ordinary TM. Since Alvin and Olympia don't have the right causal structure, they don't count as TMs. (This doesn't mean that TMs could actually be conscious, as per the argument in 3, but it's important to note that these guys aren't even in the running.) Further, the causal structure they *do* have puts them in a class of architectures that aren't a plausible candidate for consciousness. Those are claims about the causal structure of the machines. But then causal structure—how the spatiotemporal parts of the machines themselves are actually arranged and interact—is critical for determining whether a computing machine is conscious.

The move to mechanism thus has what might be counterintuitive consequence. Suppose I emulate a Turing machine on my computer: that is, suppose I run a program that specifis a TM machine table and a tape, and displays what the transitions would be on some virtual tape. Does my computer thereby have a TM architecture? No.[11] Its spatiotemporal parts are

[11]Assuming my computer is not actually a TM in the first place, or that its components are not assembled out of TMs, or so on. The argument that follows is not meant to show

just as they were when they left the factory. The primitive operations of the TM are implemented in some unknown but presumably sprawling and complicated way by the underlying architecture that my laptop actually has.

At best, then, say that my computer has a TM as a *virtual* architecture, running on top of the actual one. Because of the complex relationship between virtual architectures and actual ones, it may be difficult to disentangle which is which. Disentangle we must. For if the path we've been led down is the right one, then we computationalists must say something that is suspiciously close to what Searle (1980) says—there is no reason to think that a simulation of an architecture is sufficient for consciousness. Virtual machines, even the virtual machine that would arise from a simulation of my brain in all computationally relevant detail, are not the right kinds of thing to support conscious experience. The fact that a virtual machine has similar 'organizational invariants' (Chalmers, 1996a) to an actual architecture does not show that it is the same as a process that actually has that architecture.

Thankfully, we needn't inherit Searle's pessimism. For the thrust of all of this has been to say that if (as Maudlin claims) computation depends on actual activity, then we really ought to look at the actual activity of actual computing processes. If I think that a certain kind of TM is enough to be conscious, then something actually has to get down on its knees and move around the tape in order to be conscious Once we do, I suspect, we may well find that the constraints on architectures are actually pretty minimal.[12] Still, I think Maudlin's argument, when seen in this light, is not a failure. Even if it does not take down computationalism. But it forces important constraints on the computationalist thesis. Which things satisfy these constraints, and why, is a nontrivial empirical question.[13]

---

that a computer only ever belongs to one architecture, but rather that it can belong to more than one architecture only under very specific conditions involving the coincidence of spatiotemporal parts.

[12]Though not necessarily. As I understand them, Fekete and Edelman (2011) argue that only computational architectures that can be formulated in terms of trajectories through state-space (rather than successions of points in state-space) are candidates for the grounds of conscious experience. While I like their overall picture, I worry about the details: the move to trajectories seems to require a phenomenally smooth, continuous transition between any two whole phenomenal states. That would, among other things, rule out *any* architecture that has transitions between discrete states, including TMs (see also Spivey (2007) for a similar argument). That seems to me to be both restrictive and implausible: if I am shown a sequence of disconnected images, it seems like my phenomenal state *does* make non-continuous transitions.

[13]Thanks to Peter Clutton, Stephen Gadsby, Annelli Janssen, and Antonios Kaldas for

References

Aaronson, S. (2015). Why philosophers should care about computational complexity. In Copeland, B. J., Posy, C., and Shagrir, O., editors, *Computability: Gödel, Turing, Church, and Beyond*. MIT Press, Cambridge.

Barnes, E. (1991). The causal history of computational activity: Maudlin and Olympia. *The Journal of Philosophy*, 88(6):304–316.

Bartlett, G. (2012). Computational theories of conscious experience: Between a rock and a hard place. *Erkenntnis*, 76(2):195–209.

Bishop, J. M. (2009a). A cognitive computation fallacy? Cognition, computations and panpsychism. *Cognitive Computation*, 1(3):221–233.

Bishop, M. (2009b). Why computers cant feel pain. *Minds and Machines*, 19(4):507–516.

Block, N. (1978). Troubles with functionalism. In Savage, C. W., editor, *Perception and Cognition, Issues in the Foundations of Psychology, Minnesota Studies in the Philosophy of Science*, volume 9, pages 261–325. University of Minnesota Press, Minneapolis.

Block, N. (1981). Psychologism and behaviorism. *The Philosophical Review*, 90(1):5–43.

Cantwell Smith, B. (1996). *On the origin of objects*. The MIT Press, Cambridge.

Chalmers, D. (1996a). *The Conscious Mind: In Search of a Fundamental Theory*. Oxford University Press, New York.

Chalmers, D. J. (1996b). Does a rock implement every finite-state automaton? *Synthese*, 108(3):309–333.

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

Craver, C. (2007). *Explaining the brain*. Oxford University Press, New York.

Dennett, D. (1998). Can machines think? In *Brainchildren: Essays on designing minds*, chapter 1, pages 3–30. MIT Press.

thoughtful comments on an earlier draft, and to Sean Power for helpful discussion. Work on this paper was supported by Australian Research Council grant FT140100422.

Dewdney, A. (1984). On the spaghetti computer and other analog gadgets for problem solving. *Scientific American*, 250(6):19–26.

Duntemann, J. (2011). *Assembly language step-by-step: Programming with Linux*. John Wiley & Sons, New York.

Fekete, T. and Edelman, S. (2011). Towards a computational theory of experience. *Consciousness and cognition*, 20(3):807–827.

Ferguson, W., Cornea, M., Anderson, C., and Schneider, E. (2015). The difference between x87 instructions FSIN, FCOS, FSINCOS, and FPTAN and mathematical functions sin, cos, sincos, and tan. Technical report, Intel Corporation. https://software.intel.com/en-us/articles/the-difference-between-x87-instructions-and-mathematical-functions.

Frost, E. A. (2015). A review of mechanisms of inhalational anesthetic agents. In Kaye, A. D., Kaye, A. M., and Urman, R. D., editors, *Essentials of Pharmacology for Anesthesia, Pain Medicine, and Critical Care*, pages 49–60. Springer.

Izhikevich, E. M. (2006). Polychronization: computation with spikes. *Neural computation*, 18(2):245–282.

Klein, C. (2008). Dispositional implementation solves the superfluous structure problem. *Synthese*, 165(1):141–153.

Klein, C. (2012). Two paradigms for individuating implementations. *Journal of Cognitive Science*, 13(2):167–179.

Klein, C. (2013). Multiple realizability and the semantic view of theories. *Philosophical Studies*, 163(3):683–695.

Klein, C. (2015). Olympia and other O-machines. *Philosophia*, 43(4):925–931.

Logothetis, N. K. (2008). What we can do and what we cannot do with fMRI. *Nature*, 453:869–878.

Maudlin, T. (1989). Computation and consciousness. *The Journal of Philosophy*, 86(8):407–432.

Pettit, P. (2015). *The Robust Demands of the Good: Ethics with Attachment, Virtue, and Respect*. Oxford University Press, USA.

Piccinini, G. (2007). Computing mechanisms. *Philosophy of Science*, 74(4):501–526.

Piccinini, G. (2015). *Physical computation: A mechanistic account*. Oxford University Press, Oxford.

Power, S. E. (2010). Complex experience, relativity and abandoning simultaneity. *Journal of Consciousness Studies*, 17(3-1):231–256.

Putnam, H. (1991). *Representation and reality*. The MIT Press, Cambridge.

Pylyshyn, Z. W. (1984). *Computation and cognition*. Cambridge University Press, Cambridge.

Raichle, M. E. and Snyder, A. Z. (2007). A default mode of brain function: A brief history of an evolving idea. *Neuroimage*, 37(4):1083–1090.

Searle, J. (1990). Is the brain a digital computer? In *Proceedings and Addresses of the American Philosophical Association*, pages 21–37. JSTOR.

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(03):417–424.

Spivey, M. (2007). *The continuity of mind*. Oxford University Press, New York.

Sprevak, M. (2007). Chinese rooms and program portability. *The British Journal for the Philosophy of Science*, 58(4):755–776.

Tononi, G. (2004). An information integration theory of consciousness. *BMC neuroscience*, 5(42):1–22.

Tononi, G. and Koch, C. (2015). Consciousness: Here, there and everywhere? *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 370(1668):20140167.

Woodward, J. (2003). *Making Things Happen*. Oxford University Press, New York.