Home, Pause, or Break: A Critique of Chalmers on Implementation

Brandon N. Towl

Washington University in St. Louis
towl@sbcglobal.net

David Chalmers's theory of implementation proposes an implementation/computations relation that supports the theses of *computational sufficiency* and *computational explanation*. These theses face some well known challenges, and so one should expect that a full theory of implementation to meet these challenges. In this critique, I show that Chalmers's theory is silent on some of the important metaphysical details need for such a full account. I then turn to sketching out some of the possible moves and countermoves to be made in providing such details. The intent here is not to refute Chalmers's theory, but rather to show where much of the hard philosophical works still needs to be done.

Key words: *computation, implementation, isomorphism, computational sufficiency, computational explanation, metaphysic of computation*

1. Why a Theory of Implementation?

With his usual insight and clarity, Chalmers fills a lacuna in the literature on cognition and computation with an analysis of *implementation*. Such an analysis certainly seems needed, as it could potentially settle some debates about which systems are computational (and which are not), which systems are multiply realized (and which are not), and which research programs would benefit from a computational perspective (and which ones would not).

Chalmers's aim in this paper is, wisely, much narrower than these topics. Chalmers provides his analysis of the notion of implementation with the

©2011 Institute for Cognitive Science, Seoul National University

hopes that such an analysis can support two theses: the thesis of *computational sufficiency* (that the right kind of computational structure suffices for the possession of a mind) and the thesis of *computational explanation* (that computation provides a general framework for the explanation of cognitive processes, and of behavior).

Vindication of these two theses would be a coup for both AI research programs and for cognitive science in general. Critics of AI have taken great pains to discredit these theses, as Chalmers notes. Even today, many of their criticisms are taken seriously by both friends and foes of AI and computational cognitive science. These critics have a legitimate complaint because the application of computational explanations is at once too permissive and too limited. The application of computational explanations is too permissive because just about *any* system can be described computationally—not just computers and mathematicians, but also thermostats, digestive systems, economies, and rooms with philosophers locked in them. At the same time, describing systems computationally is too limited in that there are a number of features of our mental lives that seem to outstrip the mere manipulation of symbols according to algorithms—features like semantic content, qualitative feel, skilled expertise, and perhaps many others. (Note, though, that Chalmers' general framework does not assume the algorithmic manipulation of symbols—see section 4.) Permissiveness plus computational sufficiency leads to too many things having minds, while limitedness makes computational explanation inadequate as a general theory of the mind.

Chalmers' insight is to embrace the permissive nature of computational description while putting some important qualifications on it. The first qualification is that although any system implements some computation or other, only a limited number of systems are going to have the kind of complexity to support the computations that we would consider cognition. Thus, even though computational description by itself is permissive, only computational structures *of the right kind* should be considered as possessing a mind (or better yet, as possessing cognition or intelligence).

The second qualification is that computational explanations capture the structure of a system involving only its *organizationally invariant* properties. And since mentality seems to involve only organizationally invariant properties, computational explanations are a fine general framework for their explanation. By contrast, digestion involves properties that are not organizationally invariant, and so something would naturally be left out by a merely computational explanation of digestion.

The third qualification is that the thesis we are ultimately interested in through this discussion is the thesis that *cognition* is computation, not that *all mentality* is computation. Some mental states (or processes) are certainly cognitive: thinking, deciding, knowing, remembering, etc. Many are not, or at least not obviously so. For example, there are emotional states (fear, envy, anger, enjoyment), conative states (acting, trying, wishing, wanting), and "bare" sensations (throbs, itches, tingles).[1] Computational structure need only be sufficient for *cognition*, and computational explanation is appropriate for this more limited domain.

If Chalmers' analysis of implementation is correct, it will have some pay off. For example, we will have a general formula for when something is implementing a computation, and perhaps a guide as to which computation is being implemented. We will be able to see the role of computational explanations and to justify them. And we will be able to respond to a variety of objections raised by critics of AI research, such as Searle's Chinese Room argument and the argument concerning a wall interpreted as a WordStar processor (see Searle, 1980 and 1992, respectively). In short, if Chalmers is right, all of these problems are revealed as pseudo-problems and all of us philosophers (or, at least, the metaphysicians) can head home and leave the nitty-gritty details to the AI researchers and cognitive scientists to work out (in peace, without our meddling).

It would be useful at this point to pause and consider what a theory of implementation needs to do in order to achieve these feats. In barest outlines, it should 1) say what, precisely, the relata of the implementation relation are, 2) analyze the relation between these relata in a satisfying way, and 3) square (1) and (2) with the way we commonly talk about computational systems (and how we distinguish them from non-computational systems).

[1] Some of these might be cognitive states as well, or could be taken as cognitive states for the purposes of some explanatory framework or other. Which states are cognitive, and which are not, will not matter for the critique I provide in this article; the main point is only that we need not treat *all* mentality as cognitive.

While Chalmers does much towards (2), and says some important things for (3), it becomes apparent once one starts investigating (1) that more needs to be said about it. The following considerations, then, are offered as friendly suggestions about the elements of Chalmers' theory that need to be explained, expanded, or possibly re-examined.

Before getting to those considerations, I want to clarify my commitments. I take it that giving a theory of implementation is doing metaphysics (though metaphysics applied to a particular domain). Some philosophers, and many non-philosophers, are generally skeptical about metaphysics. I do not have much to say to allay that concern, but the skeptics can rest easy knowing that I am *not* suggesting that artificial intelligence and cognitive science must halt until these issues are resolved. (Indeed, if scientists ever stopped their work in order to wait for the philosophers to get their act together, science would never get done.)

Nor am I suggesting that we break with Chalmers and try to develop a different notion of implementation. Chalmers might well be on the right path. However, insofar as a theory of implementation is something for metaphysicians to develop, we would expect such a theory to answer certain questions and resolve standing confusions within this domain. I cannot claim that the questions or standing confusions are mine; indeed, they have been around for a while. In the last twenty years or so, a lot of good work has been published on these questions and confusions. What I am suggesting is that it is not always clear how to take Chalmers' theory and use it to gain traction on some of these questions and confusions. So there are still some metaphysical issues to work through in order to give a theory of implementation. So the philosophers (particularly the metaphysicians) cannot go home yet; some of the hard work has yet to be done, and we are just at the start of a much larger project.

I suggest, then, that the metaphysician's response to Chalmers should be neither to go home nor to break with him, but simply to pause, take stock, and think about the directions these investigations should take. To give an example of such pausing, and in order to lay out some of the issues I have been hinting at, I am going to introduce a dilemma that theories of implementation face when trying to get specific about what computations are. There are two obvious choices: computations are either abstract formalisms or concrete processes. Each choice has its questions and problems. Thus, a theory of implementation must go beyond committing to one view or the other; it has to go some ways to dispel some of these problems.

2. A Dilemma: Formalism vs. Process

There is a dilemma that theories of implementation may face depending on what one thinks the relata of the implementation relation are. On the surface, it is obvious what the relata of this relation are: an implementation relation relates a computation and an implementation of that computation. But theories of implementation will differ depending on how they conceive of implementations and computations.

In particular, I want to focus on what a *computation* is. There seem to be two mutually exclusive ways to conceive of computations. One way is to view computations as a kind of formalism only; on this view, computations are abstract relations between abstract objects. Another way is to view computations as actual (types of) processes; on this view, computations are concrete processes carried out by concrete objects, though perhaps described at a level of detail that abstracts away from many of the physical properties of the system.

Any theory of implementation must adopt one or the other of these two views—or, if it does not, it must provide a third alternative. My first goal, then, is to present (only very briefly) some issues that arise when one takes up one or the other of these views. My second goal is to try and situate Chalmers' paper within this debate. The reason why these steps are important is that failing to say something about these views means failing to be clear about what sort of thing a computation is, which in turn prevents one from giving a fully articulated theory of implementation.

So what about the first view, that computation is just an abstract formalism? What I have in mind here is a function or mapping between abstract objects. These abstract objects might be numbers, or symbols, or something else. Multiplication, for example, would be a mapping from ordered pairs of numbers to their products: $<2, 3> \rightarrow 6$. (Copeland (1996) can be seen as articulating these "mapping" relations in terms of labeling and specifications of algorithms.)

When Chalmers writes about an isomorphism between the "formal structure" of a computation and the causal structure of an implementing system, the preceding is a natural way to interpret him. This is also a natural way to understand the word "abstract," when it is claimed that computation is an "abstract specification of causal organization."

This way of reading the word "computation" also makes sense of those box-and-arrow diagrams that cognitive scientists and neuropsychologists were fond of twenty years ago (and that some are still fond of today). Such diagrams are literally "maps" of the cognitive processes in question: representations of a process that leave out some details while retaining important organizational information.

Indeed, there might be multiple such representations. A box-and-arrow diagram can provide one way of displaying the relevant mapping. A large chart is another way. A set of instructions in Java or C++ are yet others. One of the advantages of Chalmers' framework is that none of this is specified ahead of time—the framework can capture implementation regardless of how the implementing system is ultimately represented.

Taking 'computation' to refer to an abstract formalism seems to get the implementation relation wrong, however. Recall that Chalmers says that a physical system implements a computation when the causal structure of the physical system mirrors the formal structure of the computation. And by "mirrors," Chalmers means that there is the appropriate sort of isomorphism. This is obviously the right move to make; mirror cannot mean "duplicate," for example, because a formal structure is something atemporal and non-physical (in just the way a number is atemporal and non-physical), and a causal process, by contrast, takes time to unfold and features physical objects and their properties. Thus the causal process and the formal structure are not duplicates—they do not even belong to the same ontological category. The best we can get is an isomorphism (see section 2.2).

Implementation is to be had, then, when there is an isomorphism between the formal structure of the computation and the causal structure of the system implementing that computation. A system that implements multiplication, for example, has a causal structure isomorphic to the formal structure of multiplication—so, just as multiplication maps <2, 3> to 6, the structure implementing multiplication will take two inputs (which can be plausibly mapped to 2 and 3) and produces an output (plausibly mapping to 6).

On this reading, the computation is not, by itself, a process, although it can describe a process. A computation, seen this way, is no more a process than a road map is the process of getting from point A to point B. Or, to further extend the analogy from above, a multiplication chart or a box-and-arrow diagram (or any number of things) can describe what output to expect from a system instantiating multiplication, given two inputs, but neither the chart nor the diagram would be the process of multiplying.[2] (Also notice that, on this view, cognition would not be computation; it would simply be computationally describable.)

Are computations just descriptions, though? There seem to be some considerations against this view. First, it seems to go against most definitions of "computation," definitions that categorize computation as an *active* sort of process (Boyle, 1994; Copeland, 1996; Piccinini & Scarantino, 2011). Second, computations have a certain logical structure, which means that there is a normative component to them. There is such a thing as performing a computation correctly or incorrectly (Piccinini, 2007). It is hard to see how this could be so if computations were just descriptions.

These two considerations might just be quibbles; it could be that we have the wrong definition of computation, after all, and that with the right definition, we will be able to get the appropriate sort of normativity as well. There is a third, more pressing consideration: if computations are descriptions, we are still left with the question of how a concrete system can satisfy an abstract computational description. The answer to this question is, of course, what Chalmers is trying to give us: implementation is to be understood as a mapping between formal computation and causal structure, formal computation is a kind of description, and how the causal structure of a system satisfies a (computational) description is to be explained by reference to isomorphism.

[2] This is not to imply that computations can be any sort of description. Chalmers is clear on this. Computations would have to be, at the very least, descriptions of the *causal structure* of the system, or part of the system, in question. Still, computations are descriptions on this view, and any system with a causal structure can thus be described computationally.

But this answer seems incomplete. We can know (or discover) *that* two systems are isomorphic in some way, without knowing at all how this is possible. To see this, it would be helpful to contrast a system where we *do* know how the isomorphism is possible with one where things are not so clear. Consider, then, the difference between a road map, on the one hand, and a computational description of a non-artificial cognitive system, like a brain, on the other. With road maps, the isomorphisms between the ink blobs on the paper map and the actual physical locations of the roads are plentiful and unmysterious. The paper map just is the way that it is because *we* made it that way, and we made it that way in order to understand the physical relations of actual streets and building. We also know exactly which properties matter when using the map (the distance between two ink lines is relevant to navigation, whereas the fact that it was printed on 20# offset paper does not). And we understand how the paper map *is* a map in the abstract sense. For street maps, then, the "hard work" is just making sure that the map is accurate (i.e., using the isomorphisms to make sure that the map is accurate).

By contrast, we are not in a similar situation with non-artificial cognitive systems. For example, we are not yet at the stage where we understand how neural structure stands to thought with propositional content (though not for lack of trying). For another example, we do not know how the purely-syntactic processes of the brain manage to mirror the semantic relations that hold between propositions about objects in the world. Even the debate about whether cognition is computation itself shows that there is still work to be done. No one argues about whether making one's way around a strange city using an atlas counts as following a map.

Since we are interested in cognition (and not so much interested in road maps), it seems that we need to say a little bit more about implementation than just the fact that it is a kind of isomorphism between a formal structure and a causal process.[3] To summarize, then, there are some issues left unresolved on the view that computation is an abstract formalism: computations

[3] Indeed, there is already some interesting work in this area. For a small sample see Goel (1991), Kentridge (1995), Klein (2008), Scheutz (1999), as well as Chalmers' own work in that area.

are processes, whereas formal descriptions are not; computations are normative, whereas formal descriptions are not; and isomorphism is not enough to explain how physical systems can implement formal descriptions—at least, it is not enough for the kinds of systems we are interested in here.

Perhaps these issues can be resolved. But the fact that they have been raised has led some philosophers in recent years to turn to alternative theories of implementation. It might be useful to look at the other alternative then—that computations are concrete processes—to see if, by adopting this view, Chalmers can avoid getting entangled in the aforementioned debates.

So, supposing that a computation is not just a formal description of some sort, could it be a concrete, active *process*?

This option does not seem farfetched if we shift our ways of talking about the relevant concepts. On this view, it would be more apt to speak of an implementing system as *computing* or *performing a computation*, rather than just implementing a computation. For example, while it would be technically correct to say that "this system is an implementation of multiplication," it would be more straightforward and understandable to say "this system is *performing* multiplication," or simply "this system is multiplying."

Reading computation as a process avoids some of the pitfalls mentioned above. Computations turn out to be processes, processes that implementing systems carry out. Thus, the causal structure of the system mirrors the formal structure of the computation in the sense that the computational description specifies an actual proper subset of the causal processes happening in the system. More directly: the computation *just is* a proper subset of the causal processes happening in the system (referred to in such a way that many of the fine-grained details of the system's states are left out). The formal structure of the computation is isomorphic with the causal structure of the system because the formalism describes the very process of computing that the system is doing (or can potentially do).

This second view makes use of Chalmers's term "causal organization" in the phrase "an abstract specification of causal organization." When I drive my car, there are a lot of things going on: I am making small adjustments with the steering wheel, I am applying pressure to the gas pedal, the wheels are turning, the engine is heating up, etc. But I can abstract from many of the details of any particular episode of driving and simply summarize the activity as "driving" (or, a better analogy might be to the activity "driving from point A to point B"). In the same way, we can summarize the diverse events happening in a calculator (or an abacus, or a brain) as "multiplying." The activities picked out in this way will be invariant with regards to whether or not they preserve the formal mapping in question.

While this reading avoids the pitfalls above, it runs into a problem of its own: it is rather permissive and hence threatens to make the notion of computation rather unhelpful.[4] Chalmers' view is permissive in the sense that 1) every system implements some computation or other and 2) any given system can implement more than one computation. Chalmers does clearly deny that *any* system can implement *any* given computation—such a claim is ridiculously strong, and thus should be denied. But Chalmers's view still leaves many possibilities open. Indeed, as long as an implementing system has a causal structure that preserves the mapping given by the formalism, it is said to implement the relevant computation.

What is the problem with this kind of permissiveness? After all, is it not the case that (1) and (2) are true? The answer is "no," if computation is a kind of causal activity. We might believe that every system implements some computation or other, in the sense that every system can be computationally modeled or described. But would we want to say that every system computes? Hardly. But if computation is the process of computing according to a formalism, and implementing a computation is having the right kind of causal process going on, then anything that has the right kind of causal process going on is computing according to a formalism.

To see how permissive this is, consider a pool table in the middle of a game. The first player lines up her cue with the cue ball and the seven ball and makes a shot. The cue ball hits the seven ball at an angle, and consequently both balls shoot off in different directions. Anyone familiar with basic physics can predict exactly which directions those are, and the speeds at which the balls will travel.

[4] This is not to imply that the abstract notion is not also permissive; it might be. (Many thanks to any anonymous reviewer for pointing this out.) But the permissiveness is more of a pressing problem for the computation-as-process view than it is for the computation-as-abstract-formalism view, and so I raise it here.

This event can be fairly accurately (and adequately) described using some vector addition, perhaps along with a few physical laws involving friction and inelastic collisions. But is the pool table actually doing vector addition or carrying out any sort of computation? (A different example that makes the same point is Fodor's question of whether the solar system computes its own laws of motion (Fodor, 1975, p. 74.)

I would be very hesitant to say so. The events on the pool table can be *described* with vector math, clearly. Perhaps it is the case that we could use such a pool table to do vector math—that is, we could *interpret* the motion of the cue ball as representing one vector, the motion of the seven ball as another, and, if everything is favorable (an almost-perfectly straight table, little-to-no friction, input forces that are consistent and free from human error, etc.). Then *we* would be performing a computation, using the pool table—just as we might do with a calculator or abacus.

If this second view of computation entails that all systems are literally performing computations by dint of having a causal structure, it is not clear what advantages there are to talking about computation to begin with (and, hence, it is unclear why we would need to bother with a theory like Chalmers's, on this view). Better to drop the talk about computation and simply talk about causal structure. Then again, if there are only some systems that perform computations, then there must be something else to computation than *just* being an abstract specification of causal structure. Whatever that extra ingredient is, it is likely to be a key part of a theory of implementation.

Note that Chalmers cannot simply respond that the added ingredient is complexity. I grant that only computations of a certain complex sort are candidates for mentality (cognition). But the issue here is not about which computational/causal structure can be considered cognitive. The question is which causal systems are to be considered computational in the first place.

Nor can Chalmers turn to organizationally invariant properties as the missing ingredient. The pool table, after all, has organizationally invariant properties: the ratio of the distance from pocket to pocket, the number of balls considered "in play," the relative locations and velocities of the balls, etc. (Indeed, anything that a computer version of "virtual" pool would have to encode and keep track of would be a candidate for an organizationally invariant property of a pool table.) And I would guess that those organizationally invariant properties are there whether we use the pool table to calculate vectors or not. But the table is only computing (here, adding vectors) if we are actively using the table's causal structure to do so—and even that is being charitable.

And so there is something of a dilemma here.[5] If a computation is just a formal structure, it seems to make the implementation relation come out wrong. If a computation is just a subset of the causal processes going on in a system, then the theory seems unhelpful (either everything computes—which is false—or only some things compute—in which case, it would be helpful to know which things, and why).

Here is another way of bringing this dilemma into focus. Is a computation a description or is it just the case that a description of a computation is also a description of the implementation? If you think of a computation as the formal structure itself, a computation describes its implementation. If you think of a computation as a process, then it just happens to be that any description of the computation will be a description of the implementation as well. What I hope I have shown is that both views face some challenges.[6]

3. Are Computations like Propositions?

This sort of dilemma is certainly not new; indeed, it is a major thread in debates about computation and computational theories of mind and the reason why some people want a theory of implementation in the first place (Goel, 1991; Searle, 1990; Sterelny, 1990; Copeland, 1996). A theory of implementation, then, should say something to answer these questions. Here is one simple way to do so: suggest another alternative conception of

[5] Putting things in terms of this kind of dilemma is, perhaps, overstating the case a bit—I am sacrificing precision for clarity here.

[6] As one reviewer has noted, some of these issues arise generally for the view that scientific theories are sets of models, and are not particular to Chalmers' theory of implementation (and thus, computation on Chalmers' theory is not any worse off than models in any other area of science). This could well be the case. This does not show that the philosophical problems are solved, however; it merely indicates that the problems generalize.

what a computations is. After all, one might worry that I have set up a false dichotomy here between computations as abstract formalisms, and computations as concrete processes. Maybe our choices are not so stark. Maybe there is an option falling between “mere formalism” and “subset of causal processes.” However, for this reply to work, one needs to put another viable candidate on the table. So here is one idea worth considering.

Consider, for a moment, propositions. Propositions are supposed to be the “meanings” associated with actual sentences (utterances or inscriptions). They abstract away from properties of individual languages, like spelling and vocabulary. They also represent states of affairs in the world. They have truth values, and which truth value a proposition has depends on whether or not the state of affairs represented by the proposition obtains in the real world.

Propositions are thus a unique sort of thing. On the one hand, they are not sentences (utterance, inscriptions, etc.). They may be said to be the forms of, or types of, sentences, or even the meaning behind sentences, but they are not sentences themselves. On the other hand, propositions are not things in the world either. They represent states of affairs that obtain in the world, but they themselves are not objects in the world (though sentences that mean a particular proposition can be). Propositions are not part of, or subsets of, the things they represent.

Perhaps computations have a status similar to propositions. Computations are not specific formal descriptions, just as propositions are not specific sentences. Nor are computations subsets of the causal processes going on in a system; they describe those processes. Similarly, propositions are not parts of or subsets of the things in the world, though they can describe those things.

The analogy here looks promising. But this is my main point: to say that something is *promising* is to say that it holds out hope for the future. By itself, this analogy does not give us any answers. What propositions are, and how they relate to things in the world (like objects and sentences) is still debated. There is no obvious consensus. If propositions are supposed to provide a useful analogy for thinking about computations, the ontological status of propositions should be less contested than the status of computations—but this is not so.

The need for more work is not a problem by itself. Perhaps it is worth working out what propositions are, and then using propositions as an analogy for understanding computations. (Or perhaps if we succeed in understanding computations first, we could use that understanding to solve some puzzles about propositions by analogy.) But, again, my aim was not to break with Chalmers' theory of implementation, showing where it is wrong or defective. My aim in this article is merely to identify the hard work that still needs to be done.

4. Hard Work

And hard work it is. In responding to Chalmers, I have purposively avoided retracing these debates in the literature, settling instead for bare outlines of some of the questions raised.[7] For what it is worth, his paper does move the debate along in ways that are useful. At the same time, I think Chalmers is still a ways away from securing computational sufficiency and computational explanation, which were the main goals of his essay to begin with. The philosophers can pause to take stock, but we cannot go home just yet.

References

Boyle, C. F. 1994. Computation as an intrinsic property. *Minds and Machines*, 4:4, 451-67.

Copeland, B. J. 1996. What is computation? *Synthese*, 108:3, 335-59.

Fodor, J. A. 1975. *The Language of Thought*. Cambridge, MA: Harvard University Press.

Goel, V. 1991. Notationality and the information processing mind. *Minds and Machines*, 1:2, 129-166.

Kentridge, R. W. 1995. Symbols, neurons, soap-bubbles and the neural computation underlying cognition. *Minds and Machines*, 4:4, 439-449.

Klein, C. 2008. Dispositional Implementation Solves the Superfluous Structure Problem. *Synthese*, 165:1, 141-153.

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

Piccinini, G. 2010. Computation in physical systems. In E. Zalta (Ed.), *Stanford Encyclopedia of Philosophy*.

[7] For an outline of this literature, see Piccinini (2010).

Piccinini, G., & Scarantino, A. 2011. Information processing, computation, and cognition. *Journal of Biological Physics*, 37:1, 1-38.

Scheutz, M. 1999. When physical systems realize functions. *Minds and Machines*, 9:2, 161-196.

Searle, J. R. 1980. Minds, brains and programs. *Behavioral and Brain Sciences*, 3, 417-57.

Searle, J. R. 1990. Is the brain a digital computer? *Proceedings and Addresses of the American Philosophical Association*, 64:3, 21-37.

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

Sterelny, K. 1990. *The Representational Theory of Mind*. Oxford: Blackwell.