A Computational Foundation for the Study of Cognition

David J. Chalmers
Department of Philosophy
Washington University
St. Louis, Missouri 63130
dave@twinearth.wustl.edu

May 13, 1994

Short abstract

Computation is central to the foundations of modern cognitive science, but its role is controversial. Questions about computation abound: What is it for a physical system to implement a computation? Is computation sufficient for thought? What is the role of computation in a theory of cognition? What is the relation between different sorts of computational theory, such as connectionism and symbolic computation? This article develops a systematic framework that addresses all of these questions. A careful analysis of computation and its relation to cognition suggests that the ambitions of artificial intelligence and the centrality of computation in cognitive science are justified.

Long abstract

Computation is central to the foundations of modern cognitive science, but its role is controversial. Questions about computation abound: What is it for a physical system to implement a computation? Is computation sufficient for thought? What is the role of computation in a theory of cognition? What is the relation between different sorts of computational theory, such as connectionism and symbolic computation? In this paper I develop a systematic framework that addresses all of these questions.

Justifying the role of computation requires analysis of *implementation*, the nexus between abstract computations and concrete physical systems. I give such an analysis, based on the idea that a system implements a computation if the causal structure of the system mirrors the formal structure of the computation. This account can be used to justify the central commitments of artificial intelligence and computational cognitive science: the thesis of computational sufficiency, which holds that the right kind of computational structure suffices for the possession of a mind, and the thesis of computational explanation, which holds that computation provides a general framework for the explanation of cognitive processes. The theses are consequences of the facts that (a) computation can specify general patterns of causal organization, and (b) mentality is an *organizational invariant*, rooted in such patterns. Along the way I answer various challenges to the computationalist position, such as those put forward by Searle. I close by advocating a kind of minimal computationalism, compatible with a very wide variety of empirical approaches to the mind. This allows computation to serve as a true foundation for cognitive science.

Keywords: computation; cognition; implementation; explanation; connectionism; computationalism; representation; artificial intelligence.

1 Introduction

Perhaps no concept is more central to the foundations of modern cognitive science than that of computation. The ambitions of artificial intelligence rest on a computational framework, and in other areas of cognitive science, models of cognitive processes are most frequently cast in computational terms. The foundational role of computation can be expressed in two basic theses. First, underlying the belief in the possibility of artificial intelligence there is a thesis of *computational sufficiency*, stating that the right kind of computational structure suffices for the possession of a mind, and for the possession of a wide variety of mental properties. Second, facilitating the progress of cognitive science more generally there is a thesis of *computational explanation*, stating that computation provides a general framework for the explanation of cognitive processes and of behavior.

These theses are widely held within cognitive science, but they are quite controversial. Some have questioned the thesis of computational sufficiency, arguing that certain human abilities could never be duplicated computationally (Dreyfus 1974; Penrose 1989), or that even if a computation could duplicate human abilities, instantiating the relevant computation would not suffice for the possession of a mind (Searle 1980). Others have questioned the thesis of computational explanation, arguing that computation provides an inappropriate framework for the explanation of cognitive processes (Edelman 1989; Gibson 1979), or even that computational descriptions of a system are vacuous (Searle 1990, 1991).

Advocates of computational cognitive science have done their best to repel these negative critiques, but the positive justification for the foundational theses remains murky at best. Why should *computation*, rather than some other technical notion, play this foundational role? And why should there be the intimate link between computation and cognition that the theses suppose? In this paper, I will develop a framework that can answer these questions and justify the two foundational theses.

In order for the foundation to be stable, the notion of computation itself has to be clarified. The mathematical theory of computation in the abstract is well-understood, but cognitive science and artificial intelligence ultimately deal with physical systems. A bridge between these systems and the abstract theory of computation is required. Specifically, we need a theory of *implementation*: the relation that holds between an abstract computational object (a "computation" for short) and a physical system, such that we can say that in some sense the system "realizes" the computation, and that the computation "describes" the system. We cannot justify the foundational role of computation without first answering the question: *What are the conditions under which a physical system implements a given computation?* Searle (1990) has argued that there is no objective answer to this question, and that any given system can be seen to implement any computation if interpreted appropriately. He argues, for instance, that his wall can be seen to implement the Wordstar program. I will argue that there is no reason for such pessimism, and that objective conditions can be straightforwardly spelled out.

Once a theory of implementation has been provided, we can use it to answer the second key question: *What is the relationship between computation and cognition?* The answer to this question lies in the fact that the properties of a physical cognitive system that are relevant to its implementing certain computations, as given in the answer to the first question, are precisely those properties in virtue of which (a) the system possesses mental properties and (b) the system's cognitive processes can be explained.

The computational framework developed to answer the first question can therefore be used to justify the theses of computational sufficiency and computational explanation. In addition, I will use this framework to answer various challenges to the centrality of computation, and to clarify some difficult questions about computation and its role in cognitive science. In this way, we can see that the foundations of artificial intelligence and computational cognitive science are solid.

2 A theory of implementation

The short answer to question (1) is straightforward. It goes as follows:

> A physical system implements a given computation when the causal structure of the physical system mirrors the formal structure of the computation.

In a little more detail, this comes to:

> A physical system implements a given computation when there exists a grouping of physical states of the system into state-types and a one-to-one mapping from formal states of the computation to physical state-types, such that formal states related by an abstract state-transition relation are mapped onto physical state-types related by a corresponding causal state-transition relation.

This is still a little vague. To spell it out fully, we must specify the class of computations in question. Computations are generally specified relative to some formalism, and there is a wide variety of formalisms: these include Turing machines, Pascal programs, cellular automata, and neural networks, among others. The story about implementation is similar for each of these; only the details differ. All of these can be subsumed under the class of *combinatorial-state* *automata* (CSAs), which I will outline shortly, but for the purposes of illustration I will first deal with the special case of *simple finite-state automata* (FSAs).

An FSA is specified by giving a set of input states $I_1, \ldots, I_k$, a set of internal states $S_1, \ldots, S_m$, and a set of output states $O_1, \ldots, O_n$, along with a set of state-transition relations of the form $(S, I) \rightarrow (S', O')$, for each pair (S, I) of internal states and input states, where S' and O' are an internal state and an output state respectively. S and I can be thought of as the "old" internal state and the input at a given time; S' is the "new" internal state, and O' is the output produced at that time. (There are some variations in the ways this can be spelled out—e.g. one need not include outputs at each time step, and it is common to designate some internal state as a "final" state—but these variations are unimportant for our purposes.) The conditions for the *implementation* of an FSA are the following:

> A physical system P implements an FSA M if there is a mapping f that maps internal states of P to internal states of M, inputs to P to input states of M, and outputs of P to output states of M, such that: for every state-transition relation $(S, I) \rightarrow (S', O')$ of M, the following conditional holds: if P is in internal state s and receiving input i where $f(s) = S$ and $f(i) = I$, this reliably causes it to enter internal state s' and produce output o' such that $f(s') = S'$ and $f(o') = O'$.[1]

This definition uses maximally specific physical states s rather than the grouped state-types referred to above. The state-types can be recovered, however: each corresponds to a set $\{s | f(s) = S_i\}$, for each $S_i \in M$. From here we can see that the definitions are equivalent. The causal relations between physical state-types will precisely mirror the abstract relations between formal states.

There is a lot of room to play with the details of this definition. For instance, it is generally useful to put restrictions on the way that inputs and outputs to the system map onto inputs and outputs of the FSA. We also need not map *all* possible internal states of P, if some are not reachable from certain initial states. These matters are unimportant here, however. What is important is the overall form of the definition: in particular, the way it ensures that the formal state-transitional structure of the computation mirrors the causal state-transitional structure of the physical system. This is what all definitions of implementation, in any computational formalism, will have in common.

2.1 Combinatorial-state automata

Simple finite-state automata are unsatisfactory for many purposes, due to the monadic nature of their states. The states in most computational formalisms have a combinatorial structure: a cell pattern in a cellular automaton, a combination of tape-state and head-state in a Turing machine, variables and registers in a Pascal program, and so on. All this can be accommodated within the framework of combinatorial-state automata (CSAs), which differ from FSAs only in that an internal state is specified not by a monadic label S, but by a vector $[S^1, S^2, S^3, \ldots]$.

The elements of this vector can be thought of as the components of the overall state, such as the cells in a cellular automaton or the tape-squares in a Turing machine. There are a finite number of possible values S^i_j for each element S^i, where S^i_j is the jth possible value for the ith element. These values can be thought of as "substates". Inputs and outputs can have a similar sort of complex structure: an input vector is $[I^1, \ldots, I^k]$, and so on. State-transition rules are determined by specifying, for each element of the state-vector, a function by which its new state depends on the old overall state-vector and input-vector, and the same for each element of the output-vector.

Input and output vectors are always finite, but the internal state vectors can be either finite or infinite. The finite case is simpler, and is all that is required for any practical purposes. Even if we are dealing with Turing machines, a Turing machine with a tape limited to 10^{200} squares will certainly be all that is required for simulation or emulation within cognitive science and AI. The infinite case can be spelled out in an analogous fashion, however. The main complication is that restrictions have to be placed on the vectors and dependency rules, so that these do not encode an infinite amount of information. This is not too difficult, but I will not go into details here.

The conditions under which a physical system implements a CSA are analogous to those for an FSA. The main difference is that internal states of the system need to be specified as vectors, where each element of the vector corresponds to an independent element of the physical system. A natural requirement for such a "vectorization" is that each element correspond to a distinct physical region within the system, although there may be other alternatives. The same goes for the complex structure of inputs and outputs. The system implements a given CSA if there exists such a vectorization of states of the system, and a mapping from elements of those vectors onto corresponding elements of the vectors of the CSA, such that the state-transition relations are isomorphic in the obvious way. The details can be filled in straightforwardly, as follows:

> A physical system P implements a CSA M if there is a vectorization of internal states of P into components $[s^1, s^2, \ldots]$, and a mapping f from the substates s^j into corresponding substates S^j of M, along with similar vectorizations and mappings for inputs and outputs, such that for every state-transition rule $([I^1, \ldots, I^k], [S^1, S^2, \ldots]) \rightarrow ([S'^1, S'^2, \ldots], [O^1, \ldots, O^l])$ of M: if P is in internal state $[s^1, s^2, \ldots]$ and receiving input $[i^1, \ldots, i^n]$ which map to formal state and input $[S^1, S^2, \ldots]$ and $[I^1, \ldots, I^k]$ respectively, this reliably causes it to enter an internal state and produce an output that map to $[S'^1, S'^2, \ldots]$ and $[O^1, \ldots, O^l]$ respectively.

Once again, further constraints might be added to this definition for various purposes, and there is much that can be said to flesh out the definition's various parts; a detailed discussion of these technicalities must await another forum (see Chalmers (forthcoming b) for a start). This definition is not the last word in a theory of implementation, but it captures the theory's basic form.

One might think that CSAs are not much of an advance on FSAs. Finite CSAs, at least, are no more computationally powerful than FSAs; there is a natural correspondence that associates every finite CSA with an FSA with the same input/output behavior. Of course infinite CSAs (such as Turing machines) are more powerful, but even leaving that reason aside, there are a number of reasons why CSAs are a more suitable formalism for our purposes than FSAs.

First, the *implementation* conditions on a CSA are much more constrained than those of the corresponding FSA. An implementation of a CSA is required to consist in a complex causal interaction among a number of separate parts; a CSA description can therefore capture the causal organization of a system to a much finer grain. Second, the structure in CSA states can be of great *explanatory* utility. A description of a physical system as a CSA will often be much more illuminating than a description as the corresponding FSA.[2] Third, CSAs reflect in a much more direct way the formal organization of such familiar computational objects as Turing machines, cellular automata, and the like. Finally, the CSA framework allows a unified account of the implementation conditions for both finite and infinite machines.

This definition can straightforwardly be applied to yield implementation conditions for more specific computational formalisms. To develop an account of the implementation-conditions for a Turing machine, say, we need only redescribe the Turing machine as a CSA. The overall state of a Turing machine can be seen as a giant vector, consisting of (a) the internal state of the head, and (b) the state of each square of the tape, where this state in turn is an ordered pair of a symbol and a flag indicating whether the square is occupied by the head (of course only one square can be so occupied; this will be ensured by restrictions on initial state and on state-transition rules). The state-transition rules between vectors can be derived naturally from the quintuples specifying the behavior of the machine-head. As usually understood, Turing machines only take inputs at a single time-step (the start), and do not produce any output separate from the contents of the tape. These restrictions can be overridden in natural ways, for example by adding separate input and output tapes, but even with inputs and outputs limited in this way there is a natural description as a CSA. Given this translation from the Turing machine formalism to the CSA formalism, we can say that a given Turing machine is implemented whenever the corresponding CSA is implemented.

A similar story holds for computations in other formalisms. Some formalisms, such as cellular automata, are even more straightforward. Others, such as Pascal programs, are more complex, but the overall principles are the same. In each case there is some room for maneuver, and perhaps some arbitrary decisions to make (does writing a symbol and moving the head count as two state-transitions or one?) but little rests on the decisions we make. We can also give accounts of implementation for nondeterministic and probabilistic automata, by making simple changes in the definition of a CSA and the corresponding account of implementation. The theory of implementation for combinatorial-state automata provides a basis for the theory of implementation in general.

2.2 Questions answered

The above account may look complex, but the essential idea is very simple: the relation between an implemented computation and an implementing system is one of isomorphism between the formal structure of the former and the causal structure of the latter. In this way, we can see that as far as the theory of implementation is concerned, a computation is simply an *abstract specification of causal organization*. This is important for later purposes. In the meantime, we can now answer various questions and objections.

Does every system implement some computation? Yes. For example, every physical system will implement the simple FSA with a single internal state; most physical systems will implement the 2-state cyclic FSA, and so on. This is no problem, and certainly does not render the account vacuous. That would only be the case if every system implemented *every* computation, and that is not the case.

Does every system implement any given computation? No. The conditions for implementing a given complex computation—say, a CSA whose state-vectors have 1000 elements, with 10 possibilities for each element and complex state-transition relations—will generally be sufficiently rigorous that extremely few physical systems will meet them. What is required is not just a mapping from states of the system onto states of the CSA, as Searle (1990) effectively suggests. The added requirement that the mapped states must satisfy reliable state-transition rules is what does all the work. In this case, there will effectively be at least 10^{1000} constraints on state-transitions (one for each possible state-vector, and more if there are multiple possible inputs). Each constraint will specify one out of at least 10^{1000} possible consequents (one for each possible resultant state-vector, and more if there are outputs). The chance that an arbitrary set of states will satisfy these constraints is something less than one in $(10^{1000})^{10^{1000}}$ (actually significantly less, because of the requirement that transitions be reliable). There is no reason to suppose that the causal structure of an arbitrary system (such as Searle's wall) will satisfy these constraints. It is true that while we lack knowledge of the fundamental constituents of matter, it is impossible to *prove* that arbitrary objects do not implement every computation (perhaps every proton has an infinitely rich internal structure), but anybody who denies this conclusion will need to come up with a remarkably strong argument.

Can a given system implement more than one computation? Yes. Any system implementing some complex computation will simultaneously be implementing many simpler computations—not just 1-state and 2-state FSAs, but computations of some complexity. This is no flaw in the current account; it is precisely what we should expect. The system on my desk is currently implementing all kinds of computations, from EMACS to a clock program, and various sub-computations of these. In general, there is no canonical mapping from a physical object to "the" computation it is performing. We might say that within every physical system, there are numerous computational systems. To this very limited extent, the notion of implementation is "interest-relative". Once again, however, there is no threat of vacuity. The question of whether a given system implements a given computation is still entirely objective. What counts is that a given system does not implement *every* computation, or to put the point differently, that most given computations are only implemented by a very limited class of physical systems. This is what is required for a substantial foundation for AI and cognitive science, and it is what the account I have given provides.

If even digestion is a computation, isn't this vacuous? This objection expresses the feeling that if every process, including such things as digestion and oxidation, implements some computation, then there seems to be nothing special about cognition any more, as computation is so pervasive. This objection rests on a misunderstanding. It is true that any given *instance* of digestion will implement some computation, as any physical system does, but the system's implementing this computation is in general irrelevant to its being an instance of digestion. To see this, we can note that the same computation could have been implemented by various other physical systems (such as my SPARC) without it's being an instance of digestion. Therefore the fact that the system implements the computation is not responsible for the existence of digestion in the system.

With cognition, by contrast, the claim is that it is *in virtue* of implementing some computation that a system is cognitive. That is, there is a certain class of computations such that *any* system implementing that computation is cognitive. We might go further and argue that every cognitive system implements some computation such that any implementation of the computation would also be cognitive, and would share numerous specific mental properties with the original system. These claims are controversial, of course, and I will be arguing for them in the next section. But note that it is precisely this relation between computation and cognition that gives bite to the computational analysis of cognition. If this relation or something like it did not hold, the computational status of cognition would be analogous to that of digestion.

What about Putnam's argument? Putnam (1988) has suggested that on a definition like this, almost any physical system can be seen to implement every finite-state automaton. He argues for this conclusion by demonstrating that there will almost always be a mapping from physical states of a system to internal states of an FSA, such that over a given time-period (from 12:00 to 12:10 today, say) the transitions between states are just as the machine table say they should be. If the machine table requires that state A be followed by state B, then every instance of state A is followed by state B in this time period. Such a mapping will be possible for an inputless FSA under the assumption that physical states do not repeat. We simply map the initial physical state of the system onto an initial formal state of the computation, and map successive states of the system onto successive states of the computation.

However, to suppose that this system implements the FSA in question is to misconstrue the state-transition conditionals in the definition of implementation. What is required is not simply that state A be followed by state B on all instances in which it happens to come up in a given time-period. There must be a reliable, counterfactual-supporting connection between the states.

Given a formal state-transition $A \rightarrow B$, it must be the case that *if* the system were to be in state A, it would transit to state B. Further, such a conditional must be satisfied for *every* transition in the machine table, not just for those whose antecedent states happen to come up in a given time period. It is easy to see that Putnam's system does not satisfy this much stronger requirement. In effect, Putnam has required only that certain weak material conditionals be satisfied, rather than conditionals with modal force. For this reason, his purported implementations are not implementations at all.

(Two notes. First, Putnam responds briefly to the charge that his system fails to support counterfactuals, but considers a different class of counterfactuals—those of the form "if the system had not been in state A, it would not have transited to state B". It is not these counterfactuals that are relevant here. Second, it turns out that Putnam's argument for the widespread realization of inputless FSAs can be patched up in a certain way; this just goes to show that inputless FSAs are an inappropriate formalism for cognitive science, due to their complete lack of combinatorial structure. Putnam gives a related argument for the widespread realization of FSAs with input and output, but this argument is strongly vulnerable to an objection like the one above, and cannot be patched up in an analogous way. CSAs are even less vulnerable to this sort of argument. I discuss all this at much greater length in Chalmers (forthcoming b).)

What about semantics? It will be noted that nothing in my account of computation and implementation invokes any semantic considerations, such as the representational content of internal states. This is precisely as it should be: computations are specified syntactically, not semantically. Although it may very well be the case that any implementations of a given computation share some kind of semantic content, this should be a *consequence* of an account of computation and implementation, rather than built into the definition. If we build semantic considerations into the conditions for implementation, any role that computation can play in providing a foundation for AI and cognitive science will be endangered, as the notion of semantic content is so ill-understood that it desperately needs a foundation itself.

The original account of Turing machines by Turing (1936) certainly had no semantic constraints built in. A Turing machine is defined purely in terms of the mechanisms involved, that is, in terms of syntactic patterns and the way they are transformed. To implement a Turing machine, we need only ensure that this formal structure is reflected in the causal structure of the implementation. Some Turing machines will certainly support a systematic semantic interpretation, in which case their implementations will also, but this plays no part in the definition of what it is to be or to implement a Turing machine. This is made particularly clear if we note that there are some Turing machines, such as machines defined by random sets of state-transition quintuples, that support no non-trivial semantic interpretation. We need an account of what it is to implement these machines, and such an account will then generalize to machines that support a semantic interpretation. Certainly, when computer designers ensure that their machines implement the programs that they are supposed to, they do this by ensuring that the mechanisms have the right causal organization; they are not concerned with semantic content. In the words of Haugeland (1985), if you take care of the syntax, the semantics will take care of itself.

I have said that the notion of computation should not be dependent on that of semantic content; neither do I think that the latter notion should be dependent on the former. Rather, both computation and content should be dependent on the common notion of *causation*. We have seen the first dependence in the account of computation above. The notion of content has also been frequently analyzed in terms of causation (see e.g. Dretske 1981 and Fodor 1987). This common pillar in the analyses of both computation and content allows that the two notions will not sway independently, while at the same time ensuring that neither is dependent on the other for its analysis.

What about computers? Although Searle (1990) talks about what it takes for something to be a "digital computer", I have talked only about computations and eschewed reference to computers. This is deliberate, as it seems to me that computation is the more fundamental notion, and certainly the one that is important for AI and cognitive science. AI and cognitive science certainly do not require that cognitive systems be computers, unless we stipulate that all it takes to be a computer is to implement some computation, in which case the definition is vacuous.

What does it take for something to be a computer? Presumably, a computer cannot merely implement a single computation. It must be capable of implementing many computations—that is, it must be *programmable*. In the extreme case, a computer will be universal, capable of being programmed to compute any recursively enumerable function. Perhaps universality is not required of a computer, but programmability certainly is. To bring computers within the scope of the theory of implementation above, we could require that a computer be a CSA with certain parameters, such that depending on how these parameters are set, a number of different CSAs can be implemented. A universal Turing machine could be seen in this light, for instance, where the parameters correspond to the "program" symbols on the tape. In any case, such a theory of computers is not required for the study of cognition.

Is the brain a computer in this sense? Arguably. For a start, the brain can be "programmed" to implement various computations by the laborious means of conscious serial rule-following; but this is a fairly incidental ability. On a different level, it might be argued that learning provides a certain kind of programmability and parameter-setting, but this is a sufficiently indirect kind of parameter-setting that it might be argued that it does not qualify. In any case, the question is quite unimportant for our purposes. What counts is that the brain implements various complex computations, not that it is a computer.

3 Computation and cognition

The above is only half the story. We now need to exploit the above account of computation and implementation to outline the relation between computation and cognition, and to justify the foundational role of computation in AI and cognitive science.

Justification of the thesis of computational sufficiency has usually been tenuous. Perhaps the most common move has been an appeal to the Turing test, noting that every implementation of a given computation will have a certain kind of behavior, and claiming that the right kind of behavior is sufficient for mentality. The Turing test is a weak foundation, however, and one to which AI need not appeal. It may be that any behavioral description can be implemented by systems lacking mentality altogether (such as the giant lookup tables of Block 1981). Even if behavior suffices for *mind*, the demise of logical behaviorism has made it very implausible that it suffices for specific mental properties: two mentally distinct systems can have the same behavioral dispositions. A computational basis for cognition will require a tighter link than this, then.

Instead, the central property of computation on which I will focus is one that we have already noted: the fact that a computation provides an abstract specification of the causal organization of a system. Causal organization is the nexus between computation and cognition. If cognitive systems have their mental properties in virtue of their causal organization, and if that causal organization can be specified computationally, then the thesis of computational sufficiency is established. Similarly, if it is the causal organization of a system that is primarily relevant in the explanation of behavior, then the thesis of computational explanation will be established. By the account above, we will always be able to provide a computational specification of the relevant causal organization, and therefore of the properties on which cognition rests.

3.1 Organizational invariance

To spell out this story in more detail, I will introduce the notion of the *causal topology* of a system. The causal topology represents the abstract causal organization of the system: that is, the pattern of interaction among parts of the system, abstracted away from the make-up of individual parts and from the way the causal connections are implemented. Causal topology can be thought of as a dynamic topology analogous to the static topology of a graph or a network. Any system will have causal topology at a number of different levels. For the cognitive systems with which we will be concerned, the relevant level of causal topology will be a level fine enough to determine the causation of behavior. For the brain, this is probably the neural level or higher, depending on just how the brain's cognitive mechanisms function. (The notion of causal topology is necessarily informal for now; I will discuss its formalization below.)

Call a property P an *organizational invariant* if it is invariant with respect to causal topology: that is, if any change to the system that preserves the causal topology preserves P. The sort of changes in question include: (a) moving the system in space; (b) stretching, distorting, expanding and contracting the system; (c) replacing sufficiently small parts of the system with parts that perform the same local function (e.g. replacing a neuron with a silicon chip with the same I/O properties); (d) replacing the causal links between parts of a system with other links that preserve the same pattern of dependencies (e.g., we might replace a mechanical link in a telephone exchange with an electrical link); and (e) any other changes that do not alter the pattern of causal interaction among parts of the system.

Most properties are not organizational invariants. The property of flying is not, for instance: we can move an airplane to the ground while preserving its causal topology, and it will no longer be flying. Digestion is not: if we gradually replace the parts involved in digestion with pieces of metal, while preserving causal patterns, after a while it will no longer be an instance of digestion: no food groups will be broken down, no energy will be extracted, and so on. The property of being tube of toothpaste is not an organizational invariant: if we deform the tube into a sphere, or replace the toothpaste by peanut butter while preserving causal topology, we no longer have a tube of toothpaste.

In general, most properties depend essentially on certain features that are not features of causal topology. Flying depends on height, digestion depends on a particular physiochemical makeup, tubes of toothpaste depend on shape and physiochemical makeup, and so on. Change the features in question enough and the property in question will change, even though causal topology might be preserved throughout.

3.2 The organizational invariance of mental properties

The central claim of this section is that most mental properties are organizational invariants. It does not matter how we stretch, move about, or replace small parts of a cognitive system: as long as we preserve its causal topology, we will preserve its mental properties.

An exception has to be made for properties that are partly supervenient on states of the environment. Such properties include knowledge (if we move a system that knows that P into an environment where P is not true, then it will no longer know that P), and belief, on some construals where the content of a belief depends on environmental context. However, mental properties that depend only on internal (brain) state will be organizational invariants. This is not to say that causal topology is irrelevant to knowledge and belief. It will still capture the *internal* contribution to those properties—that is, causal topology will contribute as much as the brain contributes. It is just that the environment will also play a role.

The central claim can be justified by dividing mental properties into two varieties: psychological properties—those that are characterized by their causal role, such as belief, learning, and perception—and phenomenal properties, or those that are characterized by way in which they are consciously experienced. Psychological properties are concerned with the sort of thing the mind *does*, and phenomenal properties are concerned with the way it *feels*. (Some will hold that properties such as belief should be assimilated to the second rather than the first class; I do not think that this is correct, but nothing will depend on that here.)

Psychological properties, as has been argued by Armstrong (1968) and Lewis (1972) among others, are effectively defined by their role within an overall causal system: it is the pattern of interaction between different states that is definitive of a system's psychological properties. Systems with the same causal topology will share these patterns of causal interactions among states, and therefore, by the analysis of Lewis (1972), will share their psychological properties (as long as their relation to the environment is appropriate).

Phenomenal properties are more problematic. It seems unlikely that these can be *defined* by their causal roles (although many, including Lewis and Armstrong, think they might be). To be a conscious experience is not to perform some role, but to have a particular feel. These properties are characterized by *what it is like* to have them, in Nagel's (1974) phrase. Phenomenal properties are still quite mysterious and ill-understood.

Nevertheless, I believe that they can be seen to be organizational invariants, as I have argued elsewhere. The argument for this, very briefly, is a *reductio*. Assume conscious experience is not organizationally invariant. Then there exist systems with the same causal topology but different conscious experiences. Let us say this is because the systems are made of different materials, such as neurons and silicon; a similar argument can be given for other sorts of differences. As the two systems have the same causal topology, we can (in principle) transform the first system into the second by making only gradual changes, such as by replacing neurons one at a time with I/O equivalent silicon chips, where the overall pattern of interaction remains the same throughout. Along the spectrum of intermediate systems, there must be two systems between which we replace less than ten percent of the system, but whose conscious experiences differ. Consider these two systems, N and S, which are identical except in that some circuit in one is neural and in the other is silicon.

The key step in the thought-experiment is to take the relevant neural circuit in N, and to install alongside it a causally isomorphic silicon back-up circuit, with a switch between the two circuits. What happens when we flip the switch? By hypothesis, the system's conscious experiences will change: say, for purposes of illustration, from a bright red experience to a bright blue experience (or to a faded red experience, or whatever). This follows from the fact that the system after the change is a version of S, whereas before the change it is just N.

But given the assumptions, there is no way for the system to *notice* these changes. Its causal topology stays constant, so that all of its functional states and behavioral dispositions stay fixed. If noticing is defined functionally (as it should be), then there is no room for any noticing to take place, and if it is not, any noticing here would seem to be a thin event indeed. There is certainly no room for a thought "Hmm! Something strange just happened!", unless it is floating free in some Cartesian realm.[3] Even if there were such a thought, it would be utterly impotent; it could lead to no change of processing within the system, which could not even mention it. (If the substitution were to yield some change in processing, then the systems would not have the same causal topology after all. Recall that the argument has the form of a *reductio*.) We might even flip the switch a number of times, so that red and blue experiences "dance" before the system's inner eye; it will never notice. This, I take it, is a *reductio ad absurdum* of the original hypothesis: if one's experiences change, one can potentially notice in a way that makes some causal difference. Therefore the original assumption is false, and phenomenal properties are organizational invariants. This needs to be worked out in more detail, of course. I give the details of this "Dancing Qualia" argument along with a related "Fading Qualia" argument in (Chalmers, forthcoming a).

If all this works, it establishes that most mental properties are organizational invariants: any two systems that share their fine-grained causal topology will share their mental properties, modulo the contribution of the environment.

3.3 Justifying the theses

To establish the thesis of computational sufficiency, all we need to do now is establish that organizational invariants are fixed by some computational structure. This is quite straightforward.

An organizationally invariant property depends only on some pattern of causal interaction between parts of the system. Given such a pattern, we can straightforwardly abstract it into a CSA description: the parts of the system will correspond to elements of the CSA state-vector, and the patterns of interaction will be expressed in the state-transition rules. This will work straightforwardly as long as each part has only a finite number of states that are relevant to the causal dependencies between parts, which is likely to be the case in any biological system whose functions cannot realistically depend on infinite precision. (I discuss the issue of analog quantities in more detail below.) Any system that implements this CSA will share the causal topology of the original system. In fact, it turns out that the CSA formalism provides a perfect formalization of the notion of causal topology. A CSA description specifies a division of a system into parts, a space of states for each part, and a pattern of interaction between these states. This is precisely what is constitutive of causal topology.

If what has gone before is correct, this establishes the thesis of computational sufficiency, and therefore the the view that Searle has called "strong artificial intelligence": that there exists some computation such that any implementation of the computation possesses mentality. The fine-grained causal topology of a brain can be specified as a CSA. Any implementation of that CSA will share that causal topology, and therefore will share organizationally invariant mental properties that arise from the brain.

The thesis of computational explanation can be justified in a similar way. As mental properties are organizational invariants, the physical properties on which they depend are properties of causal organization. Insofar as mental properties are to be explained in terms of the physical at all, they can be explained in terms of the causal organization of the system.[4] We can invoke further properties (implementational details) if we like, but there is a clear sense in which they are not vital to the explanation. The neural or electronic composition of an element is irrelevant for many purposes; to be more precise, composition is relevant only insofar as it determines the element's causal role within the system. An element with different physical composition but the same causal role would do just as well. This is not to make the implausible claim that neural properties, say, are entirely irrelevant to explanation. Often the best way to investigate a system's causal organization is to investigate its neural properties. The claim is simply that insofar as neural properties are explanatorily relevant, it is in virtue of the role they play in determining a systems causal organization.

In the explanation of behavior, too, causal organization takes center stage. A system's behavior is determined by its underlying causal organization, and we have seen that the computational framework provides an ideal language in which this organization can be specified. Given a pattern of causal interaction between substates of a system, for instance, there will be a CSA description that captures that pattern. Computational descriptions of this kind provide a general framework for the explanation of behavior.

For some explanatory purposes, we will invoke properties that are not organizational invariants. If we are interested in the biological basis of cognition, we will invoke neural properties. To explain situated cognition, we may invoke properties of the environment. This is fine; the thesis of computational explanation is not an *exclusive* thesis. Still, usually we are interested in neural properties insofar as they determine causal organization, we are interested in properties of the environment insofar as they affect the pattern of processing in a system, and so on. Computation provides a general explanatory framework that these other considerations can supplement.

3.4 Some objections

A computational basis for cognition can be challenged in two ways. The first sort of challenge argues that computation cannot *do* what cognition does: that a computational simulation might not even reproduce human behavioral capacities, for instance, perhaps because the causal structure in human cognition goes beyond what a computational description can provide. The second concedes that computation might capture the capacities, but argues that more is required for true mentality. I will consider four objections of the second variety, and then three of the first. Answers to most of these objections fall directly out of the framework developed above.

But a computational model is just a simulation! According to this objection, due to Searle (1980), Harnad (1989), and many others, we do not expect a computer model of a hurricane to be a real hurricane, so why should a computer model of mind be a real mind? But this is to miss the important point about organizational invariance. A computational simulation is not a mere formal abstraction, but has rich internal dynamics of its own. If appropriately designed it will share the causal topology of the system that is being modeled, so that the system's organizationally invariant properties will be not merely simulated but *replicated*.

The question about whether a computational model simulates or replicates a given property comes down to the question of whether or not the property is an organizational invariant. The property of being a hurricane is obviously not an organizational invariant, for instance, as it is essential to the very notion of hurricanehood that wind and air be involved. The same goes for properties such as digestion and temperature, for which specific physical elements play a defining role. There is no such obvious objection to the organizational invariance of cognition, so the cases are disanalogous, and indeed, I have argued above that for mental properties, organizational invariance actually holds. It follows that a model that is computationally equivalent to a mind will itself be a mind.

Syntax and semantics. Searle (1984) has argued along the following lines: (1) A computer program is syntactic; (2) Syntax is not sufficient for semantics; (3) Minds have semantics; therefore (4) Implementing a computer program is insufficient for a mind. Leaving aside worries about the second premise, we can note that this argument equivocates between programs and implementations of those programs. While programs themselves are syntactic objects, implementations are not: they are real physical systems with complex causal organization, with real physical causation going on inside. In an electronic computer, for instance, circuits and voltages push each other around in a manner analogous to that in which neurons and activations push each other around. It is precisely in virtue of this causation that implementations may have cognitive and therefore semantic properties.

It is the notion of implementation that does all the work here. A program and its physical implementation should not be regarded as equivalent— they lie on entirely different levels, and have entirely different properties. It is the program that is syntactic; it is the implementation that has semantic content. Of course, there is still a substantial question about how an implementation comes to possess semantic content, just as there is a substantial question about how a *brain* comes to possess semantic content. But once we focus on the implementation, rather than the program, we are at least in the right ball-park. We are talking about a physical system with causal heft, rather than a shadowy syntactic object. If we accept, as is extremely plausible, that brains have semantic properties in virtue of their causal organization and causal relations, then the same will go for implementations. Syntax may not be sufficient for semantics, but the right kind of causation is.

The Chinese room. There is not room here to deal with Searle's famous Chinese room argument in detail. I note, however, that the account I have given supports the "Systems reply", according to which the entire system understands Chinese even if the homunculus doing the simulating does not. Say the overall system is simulating a brain, neuron-by-neuron. Then like any implementation, it will share important causal organization with the brain. In particular, if there is a symbol for every neuron, then the patterns of interaction between slips of paper bearing those symbols will mirror patterns of interaction between neurons in the brain, and so on. This organization is implemented in a baroque way, but we should not let the baroqueness blind us to the fact that the causal organization— *real*, physical causal organization—is there. (The same goes for a simulation of cognition at level above the neural, in which the shared causal organization will lie at a coarser level.)

It is precisely in virtue of this causal organization that the system possesses its mental properties. We can rerun a version of the "dancing qualia" argument to see this. In principle, we can move from the brain to the Chinese room simulation in small steps, replacing neurons at each step by little demons doing the same causal work, and then gradually cutting down labor by replacing two neighboring demons by one who does the same work. Eventually we arrive at a system where a single demon is responsible for maintaining the causal organization, without requiring any real neurons at all. This organization might be maintained between marks on paper, or it might even be present inside the demon's own head, if the calculations are memorized. The arguments about organizational invariance all hold here —for the same reasons as before, it is implausible to suppose that the system's experiences will change or disappear.

Performing the thought-experiment this way makes it clear that we should not expect the experiences to be had by the *demon*. The demon is simply a kind of causal facilitator, ensuring that states bear the appropriate causal relations to each other. The conscious experiences will be had by the system as a whole. Even if that system is implemented inside the demon by virtue of the demon's memorization, the system should not be confused with demon itself. We should not suppose that the demon will share the implemented system's experiences, any more than it will share the experiences of an ant that crawls inside its skull: both are cases of two computational systems being implemented within a single physical space. Mental properties arising from distinct computational systems will be quite distinct, and there is no reason to suppose that they overlap.

What about the environment? Some mental properties, such as knowledge and even belief, depend on the environment being a certain way. Computational organization, as I have outlined it, cannot determine the environmental contribution, and therefore cannot fully guarantee this sort of mental property. But this is no problem. All we need computational organization to give us is the *internal* contribution to mental properties: that is, the same contribution that the brain makes (for instance, computational organization will determine the so-called "narrow content" of a belief, if this exists; see Fodor 1987). The full panoply of mental properties might only be determined by computation-plus-environment, just as it is determined by brain-plus-environment. These considerations do not count against the prospects of artificial intelligence, and they affect the aspirations of computational cognitive science no more than they affect the aspirations of neuroscience.

Is cognition computable? In the preceding discussion I have taken for granted that computation can at least *simulate* human cognitive capacity, and have been concerned to argue that this counts as honest-to-goodness mentality. The former point has often been granted by opponents of AI (e.g. Searle 1980) who have directed the fire at the latter, but it is not uncontroversial.

This is to some extent an empirical issue, but the relevant evidence is solidly on the side of computability. We have every reason to believe that the low-level laws of physics are computable. If so, then low-level neurophysiological processes can be computationally simulated; it follows that the function of the whole brain is computable too, as the brain consists in a network of neurophysiological parts. Some have disputed the premise: for example, Penrose (1989) has speculated that the effects of quantum gravity are noncomputable, and that these effects may play a role in cognitive functioning. He offers no arguments to back up this speculation, however, and there is no evidence of such noncomputability in current physical theory (see Pour-El and Richards (1989) for a discussion). Failing such a radical development as the discovery that the fundamental laws of nature are uncomputable, we have every reason to believe that human cognition can be computationally modeled.

What about Gödel's Theorem? Gödel's theorem states that for any consistent formal system, there are statements of arithmetic that are unprovable within the system. This has led some (Lucas 1963; Penrose 1989) to conclude that humans have abilities that cannot be duplicated by any computational system. For example, our ability to "see" the truth of the Gödel sentence of a formal system is argued to be non-algorithmic. I will not deal with this objection in detail here, as the answer to it is not a direct application of the current framework. I will simply note that the assumption that we can see the truth of a Gödel sentence requires that we have the ability to determine the consistency or inconsistency of the relevant formal system, and there is no reason to believe that we have this ability in general. (For more on this point, see Putnam 1960, Bowie 1982 and the commentaries on Penrose 1990.)

Discreteness and continuity. An important objection notes that the CSA formalism only captures *discrete* causal organization, and argues that some cognitive properties may depend on continuous aspects of that organization, such as analog values or chaotic dependencies.

A number of responses to this are possible. The first is to note that the current framework can fairly easily be extended to deal with computation over continuous quantities such as real numbers. All that is required is that the various substates of a CSA be represented by a real parameter rather than a discrete parameter, where appropriate restrictions are placed on allowable state-transitions (for instance, we can require that parameters are transformed polynomially, where the requisite transformation can be conditional on sign). See Blum, Shub and Smale (1989) for a careful working-out of some of the relevant theory of computability. A theory of implementation can be given along in a fashion similar to the account I have given above, where continuous quantities in the formalism are required to correspond to continuous physical parameters with an appropriate correspondence in state-transitions.

This formalism is still discrete in time: evolution of the continuous states proceeds in discrete temporal steps. It might be argued that cognitive organization is in fact continuous in time, and that a relevant formalism should capture this. In this case, the specification of discrete state-transitions between states can be replaced by differential equations specifying how continuous quantities change in continuous time, giving a thoroughly continuous computational framework. MacLennan (1990) describes a framework along these lines. Whether such a framework truly qualifies as *computational* is largely a terminological matter, but there it is arguable that the framework is significantly similar in kind to the traditional approach; all that has changed is that discrete states and steps have been "smoothed out".

We need not go this far, however. There are good reasons to suppose that whether or not cognition in the brain is continuous, a discrete framework can capture everything important that is going on. To see this, we can note that a discrete abstraction can describe and simulate a continuous process to any required degree of accuracy. It might be objected that chaotic processes can amplify microscopic differences to significant levels. Even so, it is implausible that the correct functioning of mental processes *depends* on the precise value of the tenth decimal place of analog quantities. The presence of background noise and randomness in biological systems implies that such precision would inevitably be "washed out" in practice. It follows that although a discrete simulation may not yield precisely the behavior that a given cognitive system produces on a given occasion, it will yield plausible behavior that the system *might* have produced had background noise been a little different. This is all that a proponent of artificial intelligence need claim.

Indeed, the presence of noise in physical systems suggests that any given continuous computation of the above kinds can never be reliably implemented in practice, but only approximately implemented. For the purposes of artificial intelligence we will do just as well with discrete systems, which can also give us approximate implementations of continuous computations.

It follows that these considerations do not count against the theses of computational sufficiency or of computational explanation. To see the first, note that a discrete simulation can replicate everything *essential* to cognitive functioning, for the reasons above, even though it may not duplicate every last detail of a given episode of cognition. To see the second, note that for similar reasons the precise values of analog quantities cannot be relevant to the explanation of our cognitive *capacities*, and that a discrete description can do the job.

This is not to exclude continuous formalisms from cognitive explanation. The thesis of computational explanation is not an exclusive thesis. It may be that continuous formalisms will provide a simpler and more natural framework for the explanation of many dynamic processes, as we find in the theory of neural networks. Perhaps the most reasonable version of the computationalist view accepts the thesis of (discrete) computational sufficiency, but supplements the thesis of computational explanation with the proviso that continuous computation may sometimes provide a more natural explanatory framework (a discrete explanation could do the same job, but more clumsily). In any case, continuous computation does not give us anything fundamentally new.

4 Other kinds of computationalism

Artificial intelligence and computational cognitive science are committed to a kind of computationalism about the mind, a computationalism defined by the theses of computational sufficiency and computational explanation. In this paper I have tried to justify this computationalism, by spelling out the role of computation as a tool for describing and duplicating causal organization. I think that this kind of computationalism is all that artificial intelligence and computational cognitive science are committed to, and indeed is all that they need. This sort of computationalism provides a *general* framework precisely because it makes so few claims about the *kind* of computation that is central to the explanation and replication of cognition. No matter what the causal organization of cognitive processes turns out to be, there is good reason to believe that it can be captured within a computational framework.

The fields have often been taken to be committed to stronger claims, sometimes by proponents and more often by opponents. For example, Edelman (1989) criticizes the computational approach to the study of the mind on the grounds that:

> An analysis of the evolution, development, and structure of brains makes it highly unlikely that they could be Turing machines. This is so because of the enormous individual variation in structure that brains possess at a variety of organizational levels. [...] [Also,] an analysis of both ecological and environmental variation, and of the categorization procedures of animals and humans, makes it highly unlikely that the world (physical and social) can function as a tape for a Turing machine. (Edelman 1989, p. 30.)

But artificial intelligence and computational cognitive science are not committed to the claim that the brain is literally a Turing machine with a moving head and a tape, and even less to the claim that that tape is the environment. The claim is simply that some computational framework can *explain* and *replicate* human cognitive processes. It may turn out that the relevant computational description of these processes is very fine-grained, reflecting extremely complex causal dynamics among neurons, and it may well turn out that there is significant variation in causal organization between individuals. There is nothing here that is incompatible with a computational approach to cognitive science.

In a similar way, a computationalist need not claim that the brain is a von Neumann machine, or has some other specific architecture. Like Turing machines, von Neumann machines are just one kind of architecture, particularly well-suited to programmability, but the claim that the brain implements such an architecture is far ahead of any empirical evidence and is most likely false. The commitments of computationalism are more general.

Computationalism is occasionally associated with the view that cognition is rule-following, but again this is a strong empirical hypothesis that is inessential to the foundations of the fields. It is entirely possible that the only "rules" found in a computational description of thought will be at a very low level, specifying the causal dynamics of neurons, for instance, or perhaps the dynamics of some level between the neural and the cognitive. Even if there are no rules to be found at the cognitive level, a computational approach to the mind can still succeed. Another claim to which a computationalist need not be committed are "the brain is a computer"; as we have seen, it is not computers that are central but computations).

The most ubiquitous "strong" form of computationalism has been what we may call *symbolic computationalism*: the view that cognition is computation over representation (Newell and Simon 1976; Fodor and Pylyshyn 1988). To a first approximation, we can cash out this view as the claim that the *computational* primitives in a computational description of cognition are also *representational* primitives. That is to say, the basic syntactic entities between which state-transitions are defined are themselves bearers of semantic content, and are therefore *symbols*.

Symbolic computationalism has been a popular and fruitful approach to the mind, but it does not exhaust the resources of computation. Not all computations are symbolic computations. We have seen that there are some Turing machines that lack semantic content altogether, for instance. Perhaps systems that carry semantic content are more plausible models of cognition, but even in these systems there is no reason why the content must be carried by the systems' computational primitives. In connectionist systems, for example, the basic bearers of semantic content are *distributed* representations, patterns of activity over many units, whereas the computational primitives are simple units that may themselves lack semantic content. To use Smolensky's term (Smolensky 1988), these systems perform *subsymbolic* computation: the level of computation falls below the level of representation.[5] But the systems are computational nevertheless.

Note that the distinction between symbolic and subsymbolic computation does not coincide with the distinction between different computational formalisms, such as Turing machines and neural networks. Rather, the distinction divides the class of computations within each of these formalisms. Some Turing machines perform symbolic computation, and some perform subsymbolic computation; the same goes for neural networks. (Of course it is sometimes said that all Turing machines perform "symbol manipulation", but this holds only if the ambiguous term "symbol" is used in a purely syntactic sense, rather than in the semantic sense I am using here.)

Both proponents and opponents of a computational approach have often implicitly identified computation with symbolic computation. A critique called *What Computers Can't Do* (Dreyfus 1972), for instance, turns out to be largely directed at systems that perform computation over explicit representation. Other sorts of computation are left untouched, and indeed systems performing subsymbolic computation seem well-suited for some of Dreyfus's problem areas. The broader ambitions of artificial intelligence are therefore left intact.

On the other side of the fence, Fodor (1992) uses the name "Computational Theory of Mind" for a version of symbolic computationalism, and suggests that Turing's main contribution to cognitive science is the idea that syntactic state-transitions between symbols can be made to respect their semantic content. This strikes me as false. Turing was concerned very little with the semantic content of internal states, and the concentration on symbolic computation came later. Rather, Turing's key contribution was the formalization of the notion of *mechanism*, along with the associated *universality* of the formalization. It is this universality that gives us good reason to suppose that computation can do almost anything that any mechanism can do, thus accounting for the centrality of computation in the study of cognition.

Indeed, a focus on symbolic computation sacrifices the universality that is at the heart of Turing's contribution. Universality applies to entire classes of automata, such as Turing machines, where these classes are defined syntactically. The requirement that an automaton performs computation over representation is a strong further constraint, a semantic constraint that plays no part in the basic theory of computation. There is no reason to suppose that the much narrower class of Turing machines that perform symbolic computation is universal. If we wish to appeal to universality in a defense of computationalism, we must cast the net more widely than this.[6]

The various strong forms of computationalism outlined here are bold empirical hypotheses with varying degrees of plausibility. I suspect that they are all false, but in any case their truth and falsity is not the issue here. Because they are such strong empirical hypotheses, they are in no position to serve as a *foundation* for artificial intelligence and computational cognitive science. If the fields were committed to these hypotheses, their status would be much more questionable than it currently is. Artificial intelligence and computational cognitive science can survive the discovery that the brain is not a von Neumann machine, or that cognition is not rule-following, or that the brain does not engage in computation over representation, precisely because these are not among the fields' foundational commitments. Computation is much more general than this, and consequently much more robust.[7]

5 Conclusion: Toward a minimal computationalism

The view that I have advocated can be called *minimal computationalism*. It is defined by the twin theses of computational sufficiency and computational explanation, where computation is taken in the broad sense that dates back to Turing. I have argued that these theses are compelling precisely because computation provides a general framework for describing and determining patterns of causal organization, and because mentality is rooted in such patterns. The thesis of computational explanation holds because computation provides a perfect language in which to specify the causal organization of cognitive processes; and the thesis of computational sufficiency holds because in all implementations of the appropriate computations, the causal structure of mentality is replicated.

Unlike the stronger forms of computationalism, minimal computationalism is not a bold empirical hypothesis. To be sure, there are some ways that empirical science might prove it to be false: if it turns out that the fundamental laws of physics are noncomputable and if this noncomputability reflects itself in cognitive functioning, for instance, or if it turns out that our cognitive capacities depend essentially on infinite precision in certain analog quantities, or indeed if it turns out that cognition is mediated by some non-physical substance whose workings are not computable. But these developments seem unlikely; and failing developments like these, computation provides a general framework in which we can express the causal organization of cognition, whatever that organization turns out to be.

Minimal computationalism is compatible with such diverse programs as connectionism, logicism, and approaches focusing on dynamic systems, evolution, and artificial life. It is occasionally said that programs such as connectionism are "noncomputational", but it seems more reasonable to say that the success of such programs would vindicate Turing's dream of a computational intelligence, rather than destroying it.

Computation is such a valuable tool precisely because almost any theory of cognitive mechanisms can be expressed in computational terms, even though the relevant computational formalisms may vary. All such theories are theories of causal organization, and computation is sufficiently flexible that it can capture almost any kind of organization, whether the causal relations hold between high-level representations or among low-level neural processes. Even such programs as the Gibsonian theory of perception are ultimately compatible with minimal computationalism. If perception turns out to work as the Gibsonians imagine, it will still be mediated by causal mechanisms, and the mechanisms will be expressible in an appropriate computational form. That expression may look very unlike a traditional computational theory of perception, but it will be computational nevertheless.

In this light, we see that artificial intelligence and computational cognitive science do not rest on shaky empirical hypotheses. Instead, they are consequences of some very plausible principles about the causal basis of cognition, and they are compatible with an extremely wide range of empirical discoveries about the functioning of the mind. It is precisely because of this flexibility that computation serves as a *foundation* for the fields in question, by providing a common framework within which many different theories can be expressed, and by providing a tool with which the theories' causal mechanisms can be instantiated. No matter how cognitive science progresses in the coming years, there is good reason to believe that computation will be at center stage.

References

Armstrong, D.M. 1968. *A Materialist Theory of the Mind*. Routledge and Kegan Paul.

Block, N. 1981. Psychologism and behaviorism. *Philosophical Review* 90:5-43.

Blum, L., Shub, M., and Smale, S. 1989. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions, and universal machines. *Bulletin (New Series) of the American Mathematical Society* 21(1):1-46.

Bowie, G. 1982. Lucas' number is finally up. *Journal of Philosophical Logic* 11:279-85.

Chalmers, D.J. (forthcoming a). Absent qualia, fading qualia, dancing qualia. Chapter 7 of Chalmers (forthcoming c).

Chalmers, D.J. (forthcoming b). Does a rock implement every finite-state automaton? Submitted to *Synthese*.

Chalmers, D.J. (forthcoming c). *Toward a Theory of Consciousness*. MIT Press.

Dietrich, E.S. 1990. Computationalism. *Social Epistemology*.

Dretske, F. 1981. *Knowledge and the Flow of Information*. MIT Press.

Dreyfus, H. 1972. *What Computers Can't Do*. Harper and Row.

Edelman, G.M. 1989. *The Remembered Present: A Biological Theory of Consciousness*. Basic Books.

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

Fodor, J.A. 1987. *Psychosemantics: The Problem of Meaning in the Philosophy of Mind*. MIT Press.

Fodor, J.A. and Pylyshyn, Z.W. 1988. Connectionism and cognitive architecture. *Cognition* 28:3-71.

Fodor, J.A. 1992. The big idea: Can there be a science of mind? *Times Literary Supplement* 4567:5-7 (July 3, 1992).

Gibson, J. 1979. *The Ecological Approach to Visual Perception*. Houghton Mifflin.

Harnad, S. 1989. Minds, machines and Searle. *Journal of Experimental and Theoretical Artificial Intelligence* 1:5-25.

Haugeland, J. 1985. *Artificial intelligence: The Very Idea*. MIT Press.

Lewis, D. 1972. *Psychophysical and theoretical identifications*. Australasian Journal of Philosophy 50:249-58.

Lucas, J.R. 1963. Minds, machines, and Gödel. *Philosophy* 36:112-127.

MacLennan, B. 1990. Field computation: A theoretical framework for massively parallel analog computation, Parts I - IV. Technical Report CS-90-100. Computer Science Department, University of Tennessee.

Nagel, T. 1974. What is it like to be a bat? *Philosophical Review* 4:435-50.

Newell, A. and Simon, H.A. 1981. Computer science as empirical inquiry: Symbols and search. *Communications of the Association for Computing Machinery* 19:113-26.

Penrose, R. 1989. *The Emperor's New Mind: Concerning computers, minds, and the laws of physics*. Oxford University Press.

Penrose, R. 1990. Precis of *The Emperor's New Mind*. *Behavioral and Brain Sciences* 13:643-655.

Pour-El, M.B., and Richards, J.I. 1989. *Computability in Analysis and Physics*. Springer-Verlag.

Putnam, H. 1960. Minds and machines. In (S. Hook, ed.) *Dimensions of Mind*. New York University Press.

Putnam, H. 1967. The nature of mental states. In (W.H. Capitan and D.D. Merrill, eds.) *Art, Mind, and Religion*. University of Pittsburgh Press.

Putnam, H. 1988. *Representation and Reality*. MIT Press.

Pylyshyn, Z.W. 1984. *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press.

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

Searle, J.R. 1984. *Minds, brains, and science*. Harvard University Press.

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

Searle, J.R. 1991. *The Rediscovery of the Mind*. MIT Press.

Smolensky, P. 1988. On the proper treatment of connectionism. *Behavioral and Brain Sciences* 11:1-23.

Turing, A.M. 1936. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2* 42: 230-65.

Notes

[1]I take it that something like this is the "standard" definition of implementation of a finite-state automaton; see, for example, the definition of the description of a system by a probabilistic automaton in Putnam (1967). It is surprising, however, how little space has been devoted to accounts of implementation in the literature in theoretical computer science, philosophy of psychology, and cognitive science, considering how central the notion of computation is to these fields. It is remarkable that there could be a controversy about what it takes for a physical system to implement a computation (e.g. Searle 1990, 1991) at this late date.

[2]See Pylyshyn 1984, p. 71, for a related point.

[3]In analyzing a related thought-experiment, Searle (1991) suggests that a subject who has undergone silicon replacement might react as follows: "You want to cry out, 'I can't see anything. I'm going totally blind'. But you hear your voice saying in a way that is completely out of your control, 'I see a red object in front of me"' (pp. 66-67). But given that the system's causal topology remains constant, it is very unclear where there is room for such "wanting" to take place, if it is not in some Cartesian realm. Searle suggests some other things that might happen, such as a reduction to total paralysis, but these suggestions require a change in causal topology and are therefore not relevant to the issue of organizational invariance.

[4]I am skeptical about whether phenomenal properties can be explained in wholly physical terms. As I argue in Chalmers (forthcoming c), given any account of the physical or computational processes underlying mentality, the question of why these processes should give rise to conscious experience does not seem to be explainable within physical or computational theory alone. Nevertheless, it remains the case that phenomenal properties *depend* on physical properties, and if what I have said earlier is correct, the physical properties that they depend on are organizational properties. Further, the explanatory gap with respect to conscious experience is compatible with the computational explanation of cognitive processes and of behavior, which is what the thesis of computational explanation requires.

[5]Of course there is a sense in which it can be said that connectionist models perform "computation over representation", in that connectionist processing involves the transformation of representations, but this sense is to weak to cut the distinction between symbolic and subsymbolic computation at its joints. Perhaps the most interesting foundational distinction between symbolic and connectionist systems is that in the former but not in the latter, the computational (syntactic) primitives are also the representational (semantic) primitives.

[6]It is common for proponents of symbolic computationalism to hold, usually as an unargued premise, that what makes a computation a computation is the fact that it involves representations with semantic content. The books by Fodor (1975) and Pylyshyn (1984), for instance, are both

premised on the assumption that there is no computation without representation. Of course this is to some extent a terminological issue, but as I have stressed in 2.2 and here, this assumption has no basis in computational theory and unduly restricts the role that computation plays in the foundations of cognitive science.

[7]Some other claims with which computationalism is sometimes associated include "the brain is a computer", "the mind is to the brain as software is to hardware", and "cognition is computation". The first of these is not required, for the reasons given in 2.2: it is not computers that are central to cognitive theory but computations. The second claim is an imperfect expression of the computationalist position for similar reasons: certainly the mind does not seem to be something separable that the brain can load and run, as a computer's hardware can load and run software. Even the third does not seem to me to be central to computationalism: perhaps there is a sense in which it is true, but what is more important is that computation suffices for and explains cognition. See Dietrich (1990) for some related distinctions between computationalism, "computerism", and "cognitivism".