On Implementing a Computation

DAVID J. CHALMERS
Department of Philosophy, Washington University, St. Louis, Missouri 63130, U.S.A.
(dave@twinearth.wustl.edu)

Abstract. To clarify the notion of computation and its role in cognitive science, we need an account of implementation, the nexus between abstract computations and physical systems. I provide such an account, based on the idea that a physical system implements a computation if the causal structure of the system mirrors the formal structure of the computation. The account is developed for the class of combinatorial-state automata, but is sufficiently general to cover all other discrete computational formalisms. The implementation relation is non-vacuous, so that criticisms by Searle and others fail. This account of computation can be extended to justify the foundational role of computation in artificial intelligence and cognitive science.

Key words. Computation, implementation, artificial intelligence, cognition, Turing machines.

1. Introduction

Much of modern cognitive science centers on the notion of computation, but many questions about computation remain unclear. What is it for a physical object to implement a computation? Does implementing an appropriate computation suffice for the possession of a mind? Does every cognitive system have a computational basis? These are among the most controversial questions in the foundations of cognitive science.

The mathematical theory of computation in the abstract is well-understood, but cognitive science and artificial intelligence go beyond the domain of abstract objects to deal with concrete systems in the physical world. The difficult questions about computation are largely questions about the relationship between these domains. How does the abstract theory of computation relate to a science of concrete, causal systems, and how might it help us explain cognition in the real world? To answer these questions, we need a bridge between the abstract and the concrete domains.

The crucial element of such a bridge is a theory of *implementation*. Implementation is 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. The key question that such a theory needs to answer is *what are the conditions under which a physical system implements a given computation*? Without an answer to this question, the foundational role of computation in cognitive science cannot be justified.

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

Minds and Machines **4**: 391–402, 1995.

appropriately. He argues, for instance, that his wall can be seen to implement the Wordstar program. If this is right, then it is hard to see how computation can play a foundational role in cognitive science. If there is no objective fact of the matter about whether a system implements a given computation, then we cannot appeal to the fact that a system implements a computation in explaining any of its objective properties. It would follow, for example, that unless a strong form of panpsychism is true, there is no computation such that implementing that computation suffices for the possession of a mind.

I will argue that there is no reason for such pessimism, and that a theory of implementation can be straightforwardly spelled out. Such an account can give perfectly objective criteria for when a physical system implements a given computation, such that any given computation will only be implemented by a small fraction of physical systems. With such an account in place, the path is cleared to justify the foundational role of computation in the science of the mind.

2. Implementation

When does a physical system implement a given computation? The short answer to this is straightforward.

> 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.

There is still a little vague. To spell it out fully, we must specify the class of computations in question. Computations are generally specified within some formalism, and there are many different formalisms: these include the formalisms of Turing machines, Pascal programs, cellular automata, and connectionist 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 example, it is generally useful to put restrictions on the way that inputs and outputs to the system map onto input 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.

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.

3. 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 decomposition 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 decomposition 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 decompositions and mappings for inputs and outputs, such that for every state-transition rule $([I^1, \ldots, I^k], [S^1, S^2, \ldots]) \rightarrow ([S'^1, S'^2, \ldots], [O^2, \ldots, O^l])$ of M: if P is in internal state $[s^1, s^2, \ldots]$ and receiving input $[i^i, \ldots, i^n]$ 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 1993 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. 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.

4. 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 account can be brought to bear to answer various questions bout computation and implementation. I will also address some potential objections to the account.

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 transi-

tions 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 not a flaw in the current account; it is precisely what we should expect. The system on my deck 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, but 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 intial 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).)

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) 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. At most, a semantic interpretation is sometimes imposed on the inputs and outputs, in order to interpret a Turing machine as computing a given arithmetical function, but this does not add any constraints on implementation. To implement a Turing machine, we need only ensure that the relevant formal structure is reflected in the causal structure of the implementation. Some Turing machines will 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 (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 computers 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.

5. Conclusion

An account of implementation is only half the story. The question remains open: *What is the relationship between computation and cognition*? This is an entirely different sort of question, but an account of implementation is vital in answering it. Without a bridge between the abstract domain of computation and the concrete domain of cognitive systems, such questions are out of our reach. With a bridge in place, however, there is the possibility of progress.

I discuss this matter at much greater length elsewhere (Chalmers 1994), but we can see very briefly the shape of an answer. According to the account I have given here, a computational description of a physical system is an *abstract specification of its causal organization*. To implement a computation is just to have a set of components that interact causally according to a certain pattern. The nature of the components does not matter, and nor does the way that the causal links between components are implemented; all that matters is the pattern of causal organization of the system. What do all implementations of a given computation have in common? Precisely their causal organization. Further, almost any causal organization we can imagine can be captured by a computational description. As long as a system's causal organization can be analyzed into a finite number of parts interacting in a finitely specifiable way, this organization can be abstracted into a CSA description, or perhaps a description within some other computational formalism.

We can therefore use the notion of causal organization as the central pivot in justifying the foundational role of computation in cognitive science. For example, the thesis of "strong artificial intelligence" holds that implementation of certain sorts of computation suffices for possession of a mind. With an account of implementation in place, this thesis can be justified in two steps. First, argue that implementation of a given computation suffices for possession of a given causal organization; second, argue that possession of the appropriate sort of causal organization suffices for the possession of a mind. The first step has been taken already. The second step is trickier, but it is not hard to imagine how it might go.

We can also use this account to justify the *explanatory* centrality of computation in cognitive science. It is independently plausible that the properties of a cognitive system that are most directly relevant to the explanation of its behavior and of its cognitive capacities are properties of its causal organization. Neurophysiological properties and the like are relevant, but only insofar as they play a role in determining the pattern of causal organization of the system. And on the account I have given, computation provides a perfect language for the description of causal organization. Descriptions of a system in computational terms are in effect descriptions of its causal organization, and are therefore descriptions of precisely the aspects of the system that are most relevant in the explanation of its cognitive capacities.

In this way, we can see why computation is such a valuable tool in cognitive science. Almost any theory of cognitive processes is ultimately a theory of causal organization, and computational descriptions are flexible enough to capture causal organization of almost any kind. Thus, computation is central in such diverse approaches to the modeling of cognition as the symbol-processing approach, connectionism, and artificial life. To embrace any one of these approaches is to make a bold empirical hypothesis, but to embrace the computational approach to the mind is not. If the mind can be understood in terms of a finite causal organization, it can be understood computationally. Computation is central to the

study of the mind precisely because it provides a flexible, almost universal framework for the expression and analysis of causal organization.

References

Chalmers, D. J. (forthcoming), 'Does a Rock Implement Every Finite-state Automaton?' *Synthese*.

Chalmers, D. J. (1994), 'A Computational Foundation for the Study of Cognition', Philosophy-Neuroscience-Psychology Technical Report 94-03, Washington University.

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

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

Haugeland, J. (1985), *Artificial intelligence: The Very Idea*. MIT 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 Science* **3**, 417–57.

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.

Turing, A. M. (1936), 'On Computable Numbers, with the Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society, Series 2* **42**, 230–65.