Implementation: Computationalism's Weak Spot

Matthias Scheutz (matthias.scheutz@univie.ac.at)
Insitut für Wissenschaftstheorie und Wissenschaftsforschung
Universität Wien
Sensengasse 8/10, A-1090 Wien, Austria

Introduction

Computationalism, the claim widely held among cognitive scientists that mental states are computational states, has recently come under heavy attack from various directions. Some (especially adherents of connectionism and dynamic systems people") hold that mind cannot be explained in terms of computation.[1] Others believe that crucial aspects of the intuitive notion of computation are still not well understood and, hence, not reflected in formal definitions of computation.[2] Even the acceptance of standard notions of computation (such as "Turing-computability"), however, leaves one with an explanatory gap: how are abstract "computations" linked to concrete physical systems? It has been pointed out that a reasonable *theory of implementation*, which specifies the link between computations and "computers", is still missing—hence, "walls *implement* the Wordstar program" and every ordinary open system *implements* every finite state machine, at least so it is claimed (by Searle 1992 and Putnam 1988, respectively).

My goal in this paper is to show that Searle and Putnam's arguments are based on the same criticism: as long as physical states (of a given physical system) can be chosen freely, one can always relate these physical states to computational states (of an arbitrary computation) in such a way that the physical system can be viewed as implementing that computation. From this I will conclude that unless a better notion of implementation is provided that avoids state-to-state correspondences between physical systems and abstract objects (be they computations or certain kinds of formal theories) computationalism will remain in bad shape.

Searle's Critique of Computationalism

In his recent book *The Rediscovery of Mind*, Searle (1992:210) augments his famous "Chinese room" thought experiment (e.g., Searle 1984) by another line of argument claiming that physical properties do not suffice to determine a system's syntactic properties. Syntax has to be assigned to a physical system, and this assignment, Searle claims, is arbitrary: "If computation is defined in terms of the assignment of syntax then everything would be a digital computer, because any object whatever could have syntactical ascriptions made to it" (Searle 1992:207). In other words, whether or not a physical system is implementing a program depends solely on one's interpretation of that system:

> *"On the standard definition […] of computation it is hard to see how to avoid the following results: 1. For any object there is some description of that object such that under that description the object is a digital computer. 2. For any program and for any sufficiently complex object, there is some description of the object under which it is implementing the program. Thus for example the wall behind my back is right now implementing the Wordstar program, because there is some pattern of molecule movements that is isomorphic with the formal structure of Wordstar. But if the wall is implementing Wordstar then*

[1] E.g. see van Gelder 1995.

[2] Smith 1996, for example, argues that notions such as "Turing computability" cannot account for computational practice and its rich semantic involvement.

if it is a big enough wall it is implementing any program, including any program implemented in the brain." (Searle 1992:208-209)

The core of Searle's view is thus: the standard notion of "computation" is not suitable for describing minds, since one can always find a suitable interpretation of a computation under which a given system implements this computation.

Although Searle himself does not flesh out these intuitions about the "arbitrariness" of implementation, they can be formulated quite succinctly, once one has adopted a few definitional conventions (such as those from Copeland 1996, which I will use in the following, though in somewhat modified form). First, a computational framework has to be specified which is general enough to encompass standard notions of computation (theoretical as well as practical). Following Copeland, we assume a formal theory T specifying an architecture (e.g., the blueprint of a PC) together with an algorithm α for that architecture (e.g., an addition program written in 486 assembly language) which takes arguments of a function f as inputs and delivers values of f as outputs. To relate this formal specification to physical objects, we have to define a mapping from terms of the theory T to (spatio-temporal) parts of the physical system, which themselves have to be identified and clearly described. Copeland calls such a mapping a "labeling scheme", which consists of (1) the designation of certain parts of the physical system as label-bearers, and (2) the method for specifying the label borne by each label-bearing part at any given time (Copeland 1996:338).[3] Once a labeling scheme has been provided, a truth-definition for T can be attempted (given, additionally, an interpretation of other logical as well as non-logical constants). This definition will be different for different architectures, but it will in general contain something like a "causes" connective, say '$\emptyset$' (or "ACTION-IS" in Copeland's words), whose semantics have to be specified, too. Assuming such a truth-definition for T, the notion "model (of T)" can be defined:

Definition 1: Given a formal theory T, a labeling scheme L, and a physical system S, the pair $<S,L>$ is *a model of* T iff all sentences of T are true of S under L. A physical system S that is a model of T (under L) is said to "implement" T.

Using this definition, we can define what it means for a physical system to implement a function and state a precise version of Searle's Theorem (see Copeland 1996:338-339):[4]

Definition 2: A physical system S implements a function f iff there exists a labeling scheme L and a formal theory T (e.g., a specification of an architecture and an algorithm specific to the architecture that takes arguments of f as inputs and delivers values of f as outputs) such that $<S,L>$ is a model of T.[5]

[3] Copeland uses the term "entity" instead of "physical system", since he wants to allow all kinds of systems as computers: "real or conceptual, artifact or natural" (Copeland 1996:336). I will, however, use "physical systems" instead, since in this paper I am solely concerned with the "physical aspect" of implementation.

[4] Note that definition 2 deviates in a crucial aspect from Copeland's, who uses "is computing" for "implements", since I believe that Copeland fails to distinguish between "implementing a function" and "computing a function" (i.e., "program" and "process", see, for example, Smith 1996:33-34).

[5] As a consequence of Searle's theorem, Copeland later requires for his own account that the model in addition be "honest" (where "honest models" are models under standard interpretations of the language of T, non-standard models are excluded). It can be shown, however, that there are even honest models in Copeland's sense that violate our intuitions about computation (see Scheutz 1998).

Theorem 3: (Searle's Theorem) For any physical system *S* (with a sufficiently large number of discriminable parts) and for any formal theory *T* (e.g., of an architecture-algorithm specification) there exists a labeling scheme *L* such that <*S*,*L*> is a model of *T*.[6]

So, what it all boils down to, if one wants to establish a physical system's computational power, is—according to Searle's Theorem—to exhibit the "right" labeling scheme. This, in turn, requires that the "right" parts of *S* be singled out as label bearers in a systematic manner. One method for defining such parts (which does not even depend on the peculiarities of the system under scrutiny) can be derived from the construction introduced by Putnam that I will describe in the next section.

Putnam's Critique of Functionalism

In his book *Representation and Reality,* Putnam (1988, especially chapter 5) launched a severe attack on functionalism arguing that computational descriptions could not account for the richness of mental states, not even a combination of physical and computational states would suffice. Furthermore, his objections extend to "physical realizations of a psychological theory", a notion that can be defined as follows:

Definition 4: A *physical system S realizes a psychological theory T* (in the language *L*=<*Q*,∅,…> where *Q* is a finite set of "theoretical terms", call them "*T*-terms", standing for mental states, '∅' is the theoretical pendant to "causes", and "…" indicates additional primitives) if there exists a 1-1 mapping *f* from *T*-terms onto physical state types of *S* such that the following holds: for all *q*, *p* in *Q*, if *q*∅*p* is a theorem in *T* and *S* is in state *f*(*q*), then this will "cause" *S* to change into state *f*(*p*). If *S* realizes *T*, then *S* is said to be *a model of T.*

Given definition 4, Putnam claims that "if *any* physical state is allowed as a possible 'realizer' for any '*T*-term' in a psychological theory, then […] psychological theories will just have *too many* realizations" (Putnam 1988:100).

He gives particular credence to this claim by proving that *every ordinary open system is a realization of every abstract finite automaton* (without input and output) (see the appendix of 1988:121-125), since an abstract finite automaton seems to capture essential aspects of the notion of "psychological theory" as used in definition 4. The proof of this counterintuitive result hinges crucially upon a very "liberal" formation of physical states/state types, namely arbitrary disjunctions of "maximal states":

> *"In physics an arbitrary disjunction (finite or infinite) of so-called 'maximal states' counts as a 'physical state', where the maximal states (in classical physics) are complete specifications of the values of all the field variables at all the space-time points".* (Putnam, 1988, p. 95)

Regions in phase-space, that is, sequences of real-time intervals during the "live-time" of the physical system *S* (e.g., all maximal states of *S* from to 12:00 to 12:01 on February 02, 1998), are then taken to be *physical states* that correspond to sequences of abstract states determined by the machine table of the automaton. Finally, state types are defined as the union of all physical states corresponding to a single state of the automaton.[7] Since "sequences" of state transitions (=computational steps in the abstract) are crucial to Putnam's construction, this will have to be reflected in a definition of what it means for a physical system to realize a finite state automaton (FSA):

[6] This theorem corresponds to Searle's second claim, for a version of his first, see Copeland 1996.
[7] For details of the construction, see the appendix of Putnam 1988.

Definition 5: A physical system S realizes n computational steps of a FSA (without input and output) within a given interval I of real-time if there exists a 1-1 mapping f from automata states onto physical state types of S and a division of I into n subintervals such that the following holds: for all q, p in Q, if $q \varnothing p$ is a transition in the automaton from the k-th to the $k+1$-th computational step ($1 <= k < n$) and S is in state $f(q)$ during the k-th subinterval, this will “cause” S to change into state $f(p)$ in the k+1-th subinterval.

Given this definition, we can formulate Putnam’s result, which I will christen “Putnam’s Realization Theorem”:

Theorem 6: (Putnam’s Realization Theorem) For every ordinary open system S, for every finite state automaton M (without input and output), for any number n of computational steps of the automaton M, and for every real-time interval I (divisible into n subintervals) S realizes n computational steps of M within I.

Although this version might provoke objections regarding the “order” of the involved quantifiers, these flaws can be repaired (exploiting the expressive power of set theory), but I will not be able to get into details here. More interesting than the actual phrasing of the theorem is the striking structural similarity of definitions 4 and 5. In fact, Putnam notices at some point, that the notion “psychological” theory in his (and Lewis’) account simply *replaces* the computational formalism in the standard version of functionalism (see Putnam 1988:99). And, indeed, it makes no difference to Putnam’s argument if one speaks of the realization of a psychological theory or of a computation as long as both involve states and transitions between them (in the former case “mental states”, in the latter “computational”), as I will show in the next section.

A Proof of Searle’s Theorem Using Putnam’s Construction

Theorem 6 is stated in a way that does not seem well-suited to present a serious challenge to computationalism. After all, the finite state machine, having neither input nor output capabilities, lacks any kind of interaction with its environment (see, for example, Chalmers 1996:318). This defect, however, can be repaired and Putnam’s construction itself extended to prove Searle’s Theorem.[8] In the following, I will assume a formal theory T whose language contains a connective ‘$\varnothing$’ together with a non-empty, finite set of constants Q and another finite set of constants I such that if i is in I and p, q are in Q, then $(p,i)\varnothing q$ is a well-formed formula (where Q can be viewed as a set of mental or computational, or whatever states and I as a set of inputs, but nothing hinges upon either intuition). For the present purpose of challenging computationalism, or functionalism in general, this form will be sufficient.

Theorem 7: Every ordinary open system S is a model of every formal theory T (in the language $L=<Q,I,\varnothing,\ldots>$ in the above sense).[9]

[8] In a way, this is a somewhat more general version of Searle’s theorem, since it does not only apply to computations, but to all kinds of “formal objects”. In the case of computation, the requirements for the language are either met directly, or states and inputs should be definable within the theory (for any reasonable theory of computation). Also, something like the connective ‘$\varnothing$’ will have to be part of it, too (see Copeland 1996:341).

[9] Intuitively, open systems are not shielded from environmental influences. Together with the “Principle of Non-cyclic Behavior” (also assumed to be true of open systems by Putnam), open systems can never assume the same physical state at different times. That way there will be “enough” different physical states that can be mapped onto the same automaton state in Putnam’s construction (for details about “ordinary open systems” see Putnam 1988).

Proof: First, we need to exhibit a labeling scheme L for S and an interpretation of '$\varnothing$' such that T is true under that scheme for S. Part 1 of the labeling scheme asks us to specify parts of the entity, i.e., of an open system, as label-bearers. In order to account for the fact that T has constants for inputs besides constants for states, we designate the boundary of S (for inputs) and its "inner" part (for the other states) as bearers of labels. For part 2, a method has to be exhibited for specifying the label borne by each label-bearing part at any given time—this is where things get tricky.

Consider an arbitrary interval of real-time $[t,t']$ and let the boundary of S be the "input region". Note that the environmental conditions on the boundary throughout $[t,t']$ specify the input that S will receive. By Putnam's Principle of Noncyclical Behavior (see footnote 9), "the state of the boundary of such a system is not the same at two different times" (Putnam 1988:121). We need to define "physical states" for S and the boundary region of S, which can serve as designations of the constants from I and Q. The physical states, call them *interval states*, will be defined (analogous to Putnam) as sets of values of all field parameters at all points within the boundary or at the boundary of S, respectively, for a given interval of real-time. These interval states need to be grouped together (using set union) to form state types such that each state type corresponds exactly to one constant.

We first define a mapping from interval states onto constants and then construct the labeling from this mapping: start by defining inductively an infinite sequence of consecutive intervals Int_0, Int_1, Int_2, …, where Int_0 is $[t,\frac{t+t'}{2})$ and Int_k is the open interval $[t+\sum_{j=1}^{k}\frac{t'-t}{2^j}, t+\sum_{j=1}^{k+1}\frac{t'-t}{2^j})$ of real-time (for $k>0$). Map the interior of S during the interval Int_0, call it P_0, onto the constant (q_0 in Q, say) designating the state in which the system described by T starts out. Then, for every interval state P_k (defined by the interior of S during the interval Int_k) corresponding to some p of Q do the following: first, define I_k to be the interval state of the boundary of S during the interval Int_k. Let I_k correspond to input i from I after k steps (*). If the system described by T in state p transits into state q upon input i, define the "successor state" P_{k+1} to be the interval state of the interior of S during the interval Int_{k+1}. Now form physical state types by taking the union of all interval states P that correspond to a single constant (for all constants from I and Q). (**) For the resulting physical state types a 1-1 and onto labeling function from constants can be defined, which takes care of the second part of L. (***)

To see that S is a model under the given labeling scheme L for T we need to find an interpretation $[]_L$ of '$\varnothing$' such that $[(p,i)\varnothing q]_L$ *true* in S. Take $[\varnothing]_L$ to mean "causes" (or "follows nomologically" by the laws of physics, i.e., field theory given environmental conditions). (****)

If S is in state $L(p)$ and the input to S is $L(i)$, i.e., during the interval Int_k the physical make-up of S is given as well as its boundary conditions, then by the laws of physics it would be possible for a mathematically omniscient being (a Laplacian supermind, see Putnam 1988:122-123 for details) to determine that S will be in state $L(q)$ at the beginning of Int_{k+1}. Given the boundary conditions during Int_{k+1} (which correspond to the "next" input state and, thus, have to be provided), it can be determined that S will stay in state $L(q)$ throughout Int_{k+1}. This concludes the proof that L and S are a model for T under $[]_L$. $\therefore$

Before I discuss critical points of the construction (indicated by asterisks) as well as consequences of this proof, I would like to point out two interesting features: (1) the interval of real-time was used to fix the reference of constants, and (2) its choice is completely arbitrary, hence any interval will do, which implies that (3) the system can change states arbitrarily "fast". In fact, physical systems will not only realize finite state automata, but also (countably) infinite state automata with a (countably) infinite input language (if T is taken to be a description of an automaton and the finiteness restriction on Q and I is dropped).

Another, still untouched issue is the question what kind of function the system *S* implements? To answer this, define a function *f* from *I** into *Q* such that *f*(*w*)=*q* if the state *q* is reached after having read *w* from q_0 for all strings *w* in *I** (this mapping can be obtained inductively from '∅'). Obviously, *T* takes arguments of *f* as inputs and delivers values of *f* as outputs, in Copeland's sense, hence, *S* implements *f*.

A Brief Analysis of Possible Objections to the Above Construction

I have marked four steps in the above proof that are open to criticism—I will now address them starting with the least difficult.

Step (***) would be criticized by Chalmers, since the labeling scheme does not include states of the automaton that are not reached in the particular run (see Chalmers, 1996:315). This requirement, however, can be satisfied by taking the "subinterval" [*t*,*t'*] of [*t*,*t''*] (for *t''*>*t'*) and mapping all unreached states onto physical states within this interval.

In step (*) reference is made to an input state which is undetermined and essentially depends on a particular input (i.e., the *n*-th input character). This is, of course, the trick that makes it possible to map the accidental boundary conditions of *S* at "run-time" to the *n*-th input character. Obviously, there is no systematic relation between inputs and boundary states, which a "good" theory of implementation should provide. Note, however, that the reference to this character is uniquely fixed for all possible inputs, and that it is not *ex post facto* assignment (which is, for example, Copeland's critique of Searle's construction, see Copeland 1996:348).

Step (****) marks a very critical point, the interpretation of '⑦' (the equivalent of Copeland's "ACTION-IS") as "causes". This can be considered one of the most problematic questions in general, since there are many different possible interpretations of '⑦'. I think that Copeland would agree with the claim that all state transitions of *S* are *caused*, since they were just so defined (see Copeland 1996:353). The question is whether counterfactuals are supported, i.e., whether the input had been *L*(*i*) and the current state *L*(*q*) at any time *t*, *S* would have changed to state *L*(*q'*) (according to the state table). I am not sure if counterfactual support can be directly required as a criterion, since even real systems under "different environmental conditions" will not support counterfactuals (e.g., a PC will stop working correctly if it is exposed to a strong magnetic field). Somehow it has to be specified when transitions obeying counterfactuals can be required and expected. In the end, this boils down to a notion of "normal operation", which is problematic in its own ways as pragmatic agreements on the physical theory, background assumptions, etc. enter the picture (e.g., see Hardcastle 1995).

As far as the above construction is concerned, one could argue that it does not really make sense to ask for counterfactual support, since *L* is defined every possible input string. Hence, if *L*(*i*) and *L*(*q*) were given, *S* would have changed to state *L*(*q'*) by definition of *L*. If we asked, on the other hand, whether *S* in state *P* with input *I* had changed to the successor state *P'* (for interval states *P*, *P'*, *I*) without supplying *L*, then it would not even be clear what this question means (despite the fact that, given the boundary conditions *I* and the state of the system *P*, its next state *P'* will be "caused"). So, one could say that the counterfactual requirement is vacuously satisfied.

There are objections, however, to the underlying assumption that input states can be arbitrarily defined. Chrisley (1994:415) points out that "for computational purposes, inputs and outputs are characterized in terms of their intrinsic properties." Inputs and outputs cannot successfully be defined as in the above construction in a "post-hoc" manner as the boundary of a physical system without at the same time losing *the predictive computational understanding of the system*, because it is not known in advance which input state is going to become which boundary state. According to Chrisley (1994) this predictive nature of computational system is one of its characteristic properties.

There is certainly something true about the predictive power of "genuine" computational descriptions, but the details of Chrisley's objection would have to be elaborated more in order to assess its intellectual weight with respect to the above construction. For now, one could answer that the system is just so defined that *all* boundary information has to be available throught each interval in order to predict the boundary condition at the beginning of the next interval. Given the environmental influences during that interval in turn, it is possible to determine the next state, but again, only knowing all the environmental conditions. The reverse direction is certainly not true, but it cannot even be asked meaningfully because computational states only have correlates via *L*, and *L* in turn needs complete information about the development of the evironment of *S* during the computation period. Thus, the system is lacking the predictive power of computational descriptions if *L* is not supplied—but that seems like a platitude, since no system without a labeling, i.e., a correlation between formal and physical descriptions, can be used to make predictions. The question remains, however, if it is possible at all to have this kind predictive power for descriptions at the level of fields. It seems that there would be too many different possible instantiations of one and the same computational state, so that it would even be difficult to argue that genuine computational systems can "predict" the states of fields by virtue of their future computational states.

Another objection concerning the involved notion of causation, brought forward by Chrisley, is that, being a physical notion of causation, it is too "weak" in that it would view events causally connected that "in everyday life and science other than mathematical physics we would not take to be causally related" (Chrisley 1994:414). Chalmers too rejects the "physical notion of causation" and requires that the transition be "reliable" regardless of environmental influences (see Chalmers 1996:313, where he argues that Putnam's construction fails to guarantee the reliability of state transitions). Melnyk as well as Copeland argue for a state transition relation that satisfies certain counterfactuals (see Melnyk 1996:398 and Copeland 1996:351).

Besides the fact that a counterfactual based notion of causation is problematic in its own right, the interpretation of '⑦' as a physicist's notion of causation seems in my view appropriate in the above construction, because at the level of physical fields *no other notion of causality can be successfully employed* (see also Putnam 1988:97). A different notion of causality would also require a different level of description, thus changing the problem at hand. In fact, the ordinary language notion of causation might have even more problems connected to it than the "physical" (e.g. regarding the description of real world systems), since it presupposes an ontological division of the world that phycisists (as well as philosophers) might not be committed to.

Step (**), finally, is the one I find most critical: the formation of physical state types. As I have already mentioned above, these types are legitimate physical states by the physical theory used to the describe the system.. Yet, it seems that something about them is very "unnatural". One would like *to restrict type formations to types that have "something" in common* (something other than what is described by the constant in T, which they realize).

The philosophical strategy of this kind of "disjunctive type formation" is to turn the *multiple realizability argument* against itself: while the multiple realizability argument was invented to show that one and the same computational/psychological description can have very different kinds of physical realizations that need not have anything interesting in common, it is used here to show that *too many* different physical systems can be viewed as sharing the same computational/psychological properties. This problem on one hand, known to philosophers as the "disjunction problem", and "multiple realizability" on the other are two sides of the same coin, the coin called functionalism. Being dual principles, they can be used to defend and defeat functionalism at the same time; in other words, they are useless to establish either point. Functionalism, and in particular computationalism, could only be defended, if one can find individuation criteria for physical state types in a non-circular manner—this is the challenge for cognitive science.

Discussion

One lesson to be learned from Putnam's construction is that as long as there are no criteria that distinguish "natural" from "unnatural" physical state types, the door will be open to unintended labelings of parts of a physical system that are not really "parts" (in a common sense understanding of "part"). I am not sure if it is possible to formulate such criteria in general at all. It might be necessary to involve "high level" objects, concepts, and/or properties to describe common features of states at the "low level" description of physical systems, as used in the above construction. In other words, what is common to all these physical states might be exactly what we expected them to reveal as being common to them. If this is so, then without already assuming common properties (at a higher level of description), these states cannot be shown to have anything in common (at a low level of description). As a consequence, every approach to implementation that tries to establish a mapping between physical state types and abstract states is doomed to fail, if the physical state types that are to correspond to computational states are not given.

Interestingly enough, most arguments that are thought to refute the Putnam/Searle theorems implicitly or explicitly assume and defend such a "state-based" theory of implementation (e.g., Chalmers 1996 or Melnyk 1996) without making explicit their assumptions about the formation of physical state types. One could assume that a semantic approach (such as Copeland's) would be a way of out of this dilemma. As the above construction shows, however, it is no better off because it, too, depends on the formation of physical state types to specify parts of a physical system as bearers of labels.

Apart from their impact on computationalism, the above considerations also reveal the weak grounds upon which the whole foundational enterprise of computer science is still built. Note that these seem to be solely foundational difficulties, since computational practice is certainly not impressed by Searle's "computing walls"—otherwise computer dealers would be selling them. I think that for a general theory of implementation a very different approach is needed, one that does not depend on state-to-state correspondences, but which exploits descriptions of certain properties of concrete systems and abstract formalisms in such a way that a state-to-state correspondence results as a consequence of the account.[10] In my view, a "good" theory of implementation should deliver a substantive (i.e., constrained) link between the concrete and the abstract, not depend upon it.

References

Chalmers, D. J. 1996. Does a Rock Implement Every Finite-State Automaton? *Synthese* 108:310-333.

Chrisley, R. L.: 1994, Why Everything Doesn't Realize Every Computations. *Minds and Machines* 4:391-402.

Copeland, B. J. 1996. What is Computation? *Synthese* 108:335-359.

Hardcastle, V. 1995. Computationalism. *Synthese* 105:303-317.

Melnyk, A. 1996. Searle's Abstract Argument Against Strong AI. *Synthese* 108:391-419.

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

Scheutz, M. 1998. Do Walls Compute After All? – Challenging Copeland's solution to Searle's Theorem Against Strong AI. In *Proceedings of the Ninth Midwest AI and Cognitive Science Conference*: AAAI Press.

Scheutz, M. 1999. When Physical Systems Realize Functions…. Forthcoming in *Minds and Machines*.

Searle, J. 1984. *Minds, Brains and Science*. Cambridge, Massachusetts: Harvard University Press.

Searle, J. 1992. *The Rediscovery of Mind*. Cambridge, Massachusetts: MIT Press.

Smith, B. C. 1996. *The Origin of Objects*. Cambridge, Massachusetts: MIT Press.

Van Gelder, T. 1995. What Might Cognition Be, If Not Computation? *Journal of Philosophy* 91:345-381.

[10] For an account of "implementation" that does not rely on a state-to-state correspondence between physical and computational states see Scheutz 1999.