Computational versus Causal Complexity

Matthias Scheutz (mscheutz@cse.nd.edu)
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
USA

Abstract

The main claim of this paper is that notions of implementation based on an isomorphic correspondence between physical and computational states are not tenable. Rather, "implementation" has to be based on the notion of "bisimulation" in order to be able to block unwanted implementation results and incorporate intuitions from computational practice. A formal definition of implementation is suggested, which satisfies theoretical and practical requirements and may also be used to make the functionalist notion of "physical realization" precise. The upshot of this new definition of implementation is that implementation cannot distinguish isomorphic bisimilar from non-isomporphic bisimilar systems anymore, thus driving a wedge between the notions of causal and computational complexity. While computationalism does not seem to be affected by this result, the consequences for functionalism are not clear and need further investigations.

Keywords: computation, implementation, computational complexity, causal complexity, realization, functionalism, functional architecture, computationalism, cognitive science

1. Introduction

A widespread idea of what it means for a physical system to implement a computation is that there be a functional correspondence between physical states and computational states. The exact properties and restrictions of this correspondence differ from author to author [Chalmers (1996), Melnyk (1996), Cummins (1989), Copeland (1996), McLennan (1994), et al.], but the general idea is that "the causal structure of the physical system mirrors the formal structure of the computation" [Chalmers (1994), p. 392]. This view of implementation that computational formalisms capture the causal structure of their implementing systems is what ultimately makes computation such an attractive candidate for cognitive explanations. To put it crudely, if the claim—central to the research program called *computationalism*—that *mental states are computational states* is true, then it suffices to study the computations that give rise to cognitive functions without having to pay attention to the underlying "hardware". The idea that the mental can be studied independently from the physical belongs without any doubt to the most influential *discoveries* of the last century. It is at the heart of two ambitious empirical research programs, which set out to reveal the secrets of cognition (artificial intelligence and cognitive science), and gave rise to functionalism, today's most commonly held view in the philosophy of mind. Yet, a tacit assumption underwriting both computationalism and functionalism is that the notion of implementation (or "realization" as it is often called by

functionalists) is unproblematic, i.e., that it is known how to implement computations. A viable definition of implementation is therefore of crucial importance for both, the philosophy of mind and AI/cognitive science.

While the notion "implementation" is heavily used in computer science (and the same is true of "realization" in philosophy), formal definitions of "implementation" are rare. Chalmers (1994, 1996) is among the few who provide an *explicit* and (to some extend) formally elaborated definition of implementation, which is intended to make our intuitions about implementation precise while avoiding the pitfalls that others have exploited to argue that these very intuitions are misguided [e.g., see Searle (1992), or Putnam (1988)]. Hence, I will examine Chalmers' notion of implementation as representative of others to see if it, as it promises, provides "a basis for the theory of implementation in general" [Chalmers (1994), p. 396], and as a consequence for computationalism and functionalism.

The structure of this paper is as follows: in section 2, I introduce Chalmers' notion of implementation and argue that it fails in two respects: 1) it does not give a clear account of *how* computations *mirror* the causal structure of physical systems; and more importantly, 2) it fails at giving a tenable account of the "simpler" computations that physical systems in Chalmers' view can implement simultaneously with a given computation. I use a simple physical system consisting of two switches, a light bulb, and a battery to demonstrate my objections to Chalmers' definition under which the simple, finite, deterministic switch-system can be seen to implement complex, infinite, and non-deterministic computations.

In the subsequent sections 3 and 4, I analyze the shortcomings in detail and propose a modification of Chalmers' definition that can block unwanted implementation results by restricting the class of computations that can be implemented by a given physical system. I then argue that this restricted notion of implementation is still not sufficient. I show that to be in accordance with intuitions from computational practice (as well as functionalist theories in philosophy) the notion of implementation needs to be based on the notion of *bisimilarity* (instead of that of *isomorphism*). This notion, however, separates the notion of "computational complexity" from the notion of "causal complexity", both of which are viewed as identical by isomorphism-based notions of implementation.

Implementation based on bisimilarity, besides being able to block unwanted implementations and account for implementations known from computational practice, also suggests a way to define the functionalist notion of realization, which is then discussed in section 5.

Finally, section 6 explores some of the implications of the shift from "computational states" to "computational sequences" for computationalism and functionalism. While the prospects do not seem discouraging at all for the former, as computationalists have always favored computational processes over computational states, the consequences for functionalists are not clear and might force us to reconsider some of our intuitions about the relation between functional and physical descriptions.

2. Chalmers' Definition of Implementation

Chalmers' basic conception of how a computation is connected to the physical is that "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" (1994, p. 396)—i.e., $f(\rightarrow)$="reliably causes" (where $\rightarrow$ is the formal state transition relation in the computation, an automaton, for example, and f the mapping that establishes the correspondence between computational and physical states).

While this view seems to imply that physical systems must have a unique causal structure, Chalmers at the same time holds that "there is no canonical mapping from a physical object to the computation it is performing." (1994, p. 397). He believes that physical systems can have *multiple* causal structures depending on the groupings of their physical states (where groupings of physical states may correspond to different levels of description of the system). The apparent contradiction is easily resolved by requiring that states of the implemented computation are to be in isomorphic correspondence to grouped physical states instead of the physical states themselves (according to some specified grouping). That way Chalmers can make the further claim that "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" [Chalmers (1994), p. 397].

Chalmers (1994) provides two definitions of implementation, an informal and a (more) formal one, which he holds equivalent. Since a precise definition of the mathematical concept of isomorphism (the mathematical term expressing structural identity, i.e., "mirroring") is needed later for the analysis of these definitions, I shall review it already at this point:

Definition 1: [Isomorphism] Let $M_1=\langle D_1,R_1\rangle$ and $M_2=\langle D_2,R_2\rangle$ be two structures with domains D_1 and D_2, respectively, where relation R_1 is defined over $D_1\times D_1$ and relation R_2 is defined over $D_2\times D_2$. These structures are then said to be *isomorphic* if there exists a *bijective* function f from D_1 to D_2 such that for all $x,y\in D_1$ the following two conditions hold:

$$[\text{iso}\Rightarrow] \quad R_1(x,y) \Rightarrow R_2(f(x),f(y))$$
$$[\text{iso}\Leftarrow] \quad R_1(x,y) \Leftarrow R_2(f(x),f(y))$$

Chalmers' first (informal) definition, call it '*Ch1*', reads as follows:

> "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 states-types related by a corresponding causal state-transition relation." [Chalmers (1994), p. 392]

Having explicated the overall structure of "implementation" in *Ch1*, Chalmers spells out the details in a second definition, call it '*Ch2*', in which he uses a finite state automaton (FSA) representative for other computational formalisms:

> "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 $f(o')=O'$." [Chalmers (1994), p. 393]

Note that *prima facie* there are two differences between both definitions (together referred to by '*Ch*'): for one, *Ch2* does not require the existence of a grouping of physical states into state types as does *Ch1*. Instead, the grouping is (implicitly) established by the correspondence mapping *f*: all those physical states are grouped together that are mapped onto the same automaton state under *f*.

The second difference is a consequence of the first: whereas the mapping in *Ch1* is bijective (i.e., *one-to-one* and *onto*) and established from computational states to (grouped) physical states, the mapping in *Ch2* is only surjective (i.e., *onto*) to allow for groupings of physical states and thus established from physical states to computational states.

Chalmers, although never explicitly, seems to suggest that it is possible to obtain an isomorphic mapping *f** (from grouped physical states to computational states) from *f* by collecting all those states *s* to form a grouped state, or "state type" s_i that are mapped to the same automaton state S_i according to *f*: "The state-types can be recovered, however: each $[s_i]$ corresponds to a set $\{s|f(s)=S_i\}$ for each $S_i\in M$. From here we see that the definitions are equivalent. The causal relations between physical state-types will precisely mirror the abstract relations between formal states." [Chalmers (1994), p. 393]

Indeed, *f** (defined as the mapping $f^*(s_i)=S_i$ for each $S_i\in M$) is *one-to-one* (because the physical state types have just been so defined) and *onto* (ensured by the "for every state transition relation"-clause in the definition). Therefore, the two definitions are equivalent with respect to the mapping. Yet, neither ensures that the mapping be *isomorphic* (since [iso⇒] does not necessarily hold for either definition); in fact, *f** might not even be homomorphic for the very same reason.[1] A straightforward remedy would require [iso⇒] in *Ch*, but this would be counterproductive to Chalmers' goal of viewing all kinds of "simpler computations" as being implemented by the system simultaneously, since it would limit all implemented computations to exactly one.

To illustrate the problems with *Ch*, I shall consider a simple physical system *P*, which can be described in a physical theory (i.e., circuit theory), whose physical states can be easily specified. *P* consists of a battery, two switches, and a light bulb, which are connected by copper wires (see Figure 1).

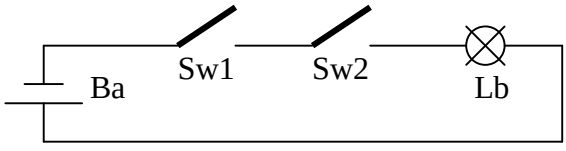

Fig. 1 The simple physical system *P* consisting of a battery, two switches, and a light bulb.

Input to *P* consists of pressing either switch (the states are named '1d' for "*Sw1* downwards", and '2d' for "*Sw2* downwards"), where each switch can only be pressed down (i.e., *pressed once*).[2] The internal states of *P* are the states of *Sw1* and *Sw2* ('uu' for "both switches up", 'dd' for "both switches down", 'du' for "switch 1 down, switch 2 up", 'ud' for "switch 1 up, switch 2 down"). Finally, output produced by *P* are the states of *Lb* which is either lit or not lit ('+' for "light on", "-" for "light off").

Now consider the following automaton $M_1=\langle Q,\Sigma,\Gamma,\delta,q_0,F\rangle$, where Q={A,B,C,D} is the set of inner states, Σ={a,b} the input alphabet, Γ={0,1} the output alphabet, $\delta=\{\langle\langle A,a\rangle,\langle B,0\rangle\rangle,\langle\langle A,b\rangle,\langle C,0\rangle\rangle,\langle\langle B,b\rangle,\langle D,1\rangle\rangle,\langle\langle C,a\rangle,\langle D,1\rangle\rangle\}$ the transition function from inner and input states to inner and output states, q_0=A the start state, and *F*={D} the set of final states.[3] The automaton is depicted (in the standard fashion) as a graph in Figure 2, where nodes denote states and edges denote transitions between states, both labeled accordingly (using the format "input/output"). In the following I will use graphs to represent automata instead of the more tedious mathematical notations.

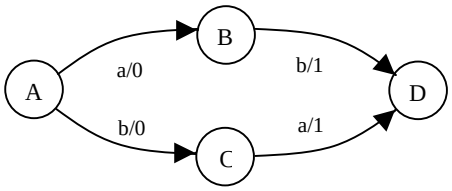

Fig. 2 The automaton M_1 with inputs from {a,b} and outputs from {0,1}.

It is easy to see that *P* implements M_1 according to *Ch2*; just map "uu" to 'A', "du" to 'B', "ud" to 'C', "dd" to 'D', "1d" to 'a', "2d" to 'b', "+" to '1', and finally "-" to '0'. The resulting mapping *f* obviously satisfies all conditions of the definition, because it supports what Chalmers calls "strong conditionals": "If a system is in state *A*, then it will transit into state *B*, however it finds itself in the first state" [Chalmers (1996), p. 316]. Both of Chalmers' requirements for counterfactual support, that the transition in the physical system be *lawful* and *reliable*, are satisfied by *P* (according to circuit theory).

Note that *f*, being a bijection, establishes an isomorphism between the given physical and the automata states (i.e., inner, input and output states) with respect to causal and computational transitions. Such "isomorphic" automata are unique (up to renaming of their states) and have the same number of states and state transitions (because their sets of states are finite). I shall call such an isomorphic automaton "the characteristic automaton *M* of the physical system *S*" (denoted by 'M^S') as it *truly captures and reflects the entire causal structure of the physical system for the given set of physical states* by retaining the mere structural properties of the physical system while abstracting of the particular physical qualities of the physical states. Obviously, *S* implements M^S for every physical system *S*.[4] The following is an easy corollary:

Corollary 2: S implements *M* iff *S* implements M^S and M^S implements *M* (where "implements" between two automata M_1 and M_2 is to be understood in Chalmers' sense as "there exists a mapping *g* from states of M_1 onto states of M_2 such that for every state

transition relation $(S_2,I_2)\rightarrow(S_2',O_2')$ of M_2 such that $g(S_1)=S_2$ and $g(I_1)=I_2$, there exists a state transition relation $(S_1,I_1)\rightarrow(S_1',O_1')$ of M_1 such that $g(S_1')=S_2'$ and $g(O_1')=O_2'$ ").

Hence, the characteristic automaton of a physical system can be used to study the set of automata that are implemented by the system (instead of the system itself).

To see now that [iso⇒] does not necessarily hold in *Ch*, consider the following automaton M_2 (which is "missing" the transition from C to D in M_1): it too is implemented by *P* (as M_1 obviously implements M_2 according to Corollary 2). Note, however, that while M_1 implements M_2, the converse is not true (even though both automata have the same number of states and a bijection can, therefore, be established between them). Hence, [iso⇒] does not hold between M_1 and M_2, and thus M_2 can be viewed as being *less complex* than M_1. The implied complexity measure is directly based on *Ch2*: an automaton *M* is *at least as complex as* an automaton *M*' if and only if *M* implements *M*'. *M* is *more complex* (or *M*' is *less complex*), if in addition *M*' does not implement *M*. Note that the *most complex* computation a system can implement under this notion of complexity is the one isomorphic to the causal structure of the system itself, i.e., the characteristic automaton (under a grouping of the original states into singletons).

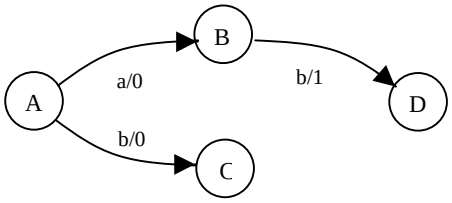

Fig. 3 A (non-characteristic) automaton implemented by *P*.

The next example illustrates how physical systems can implement "less complex automata" according to Chalmers' definition by using groupings of physical states: instead of pressing each switch individually in *P*, one could view them as "tied together" (i.e., they can only be pressed together), resulting in a simpler "one-switch system" as shown by its characteristic automaton M_3:

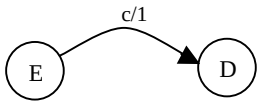

Fig. 4 A one-switch system depicted by its characteristic automaton M_3.

To see that M_1 implements M_3, group M_1's inner and input states in the following way: c:={a,b} and E:={A,B,C} (i.e., each element of c gets mapped onto 'c' and each element of E gets mapped onto 'E', the rest stays the same). Note that M_2 also implements M_3 under this mapping, but M_3 implements neither M_1 nor M_2, since it has fewer states, which makes it impossible to map states of M_3 *onto* states of M_1 or M_2. In general it is true for *Ch2* that no physical system with *n* physical states can implement a machine with more that *n* states. Note also that each of the machines M_2 and M_3 have

some physical "interpretation", which relates them to *P* (as they should if they are supposed to be implemented by *P*): M_2 would correspond to a case in which, for whatever reason, it is not possible to press switch 1 (corresponding to 'a') anymore after switch 2 (corresponding to 'b') has been pressed (e.g., because switch 2 blocks access to switch 1 once it has been pressed). In case of M_2, we could imagine that these two switches are micro-switches (like the ones used on computers to pre-set certain parameters) which are so close together that they cannot be pressed individually (unless you use a screwdriver with a very fine tip, say).

Before moving on, I would like to point out that neither *Ch1* nor *Ch2* impose any restriction on groupings; arbitrary groupings of states are permitted as is apparent from the unrestricted existential quantification in *Ch1* (i.e., the "there exists *a grouping of physical states of the system into state types*"-clause) and the unconstrained mapping from physical states to computational states in *Ch2* (again, only existentially quantified: "[..] there is a mapping *f* that [..]"). This comes indeed as a surprise, especially because Chalmers (1996) introduces *Ch2* against the backdrop of Putnam (1988), who exploits "unnatural state type formations" to argue that every system can be viewed as implementing every computation. While Chalmers' definition of implementation can block Putnam's conclusion by requiring that the state transitions be reliable and counterfactual supporting (a justified move pointing to a severe shortcoming in Putnam's argument), he purposefully seems to ignore the issue of state type formation, as he literally writes: "Some object to [Putnam's] argument on the grounds that it requires 'unnatural' physical states, involving arbitrary disjunctions. It is difficult to make this objection precise [..] I will not pursue this line, as I think the problems lie elsewhere" [Chalmers (1996), p. 312].

While many objections advanced to counter Putnam's argument on various different grounds are in my view justified (Chalmers, Chrisley, et al.), I think that Putnam's argument can be repaired and patched up in such a way that the formation of physical states remains its only weakness (although a demonstration of how this could be done will have to be left for another occasion). So, contrary to Chalmers, I am convinced (with Putnam) that the problem is *exactly the grouping of physical states* and this can be best illustrated using the following three examples, each of which points to a different aspect of what can go wrong if *unconstrained* groupings of physical states are allowed.

Take, for example, automaton M_4 (in figure 5): it *does not* reflect the causal structure of *P* anymore, for it effectively suggests that either *Sw1* or *Sw2* could be pressed twice to turn the light on, but this is not possible in *P*. Yet, M_1 implements M_4 under the grouping F:={B,C} (and thus *P* implements M_4 as well, since M_1 is *P*'s characteristic automaton).[5] Note that state F could be interpreted as "one switch down". But as the example demonstrates the physical system does not really support such a state, since it matters *which switch* is in "down position". A state like F blurs an essential distinction expressed by the state transitional structure of M_1: there are different transitions going into and coming out of B and C for the very reason that each switch can only be pressed down once. There are only two possible sequences of pressing the switches, either switch 1 is pressed first and then switch 2, or switch 2 is pressed first followed by switch 1. Because the switch pressed first determines which switch is left to be pressed (to turn the light on),

a state like "one switch down" is an *illegitimate abstraction* in that it abstracts over details that are an essential part of the causal structure of the system. It is because of this illegitimate grouping F of states C and B, that it appears possible, at a more abstract level, to turn the light on by pressing any switch twice (in addition to the two sequences that turn the light on according to M_1). Note that even if switches could be pushed up in addition to be pressed down the problem would not vanish, as it would still not be possible to turn the light on by pressing and pushing only one switch.

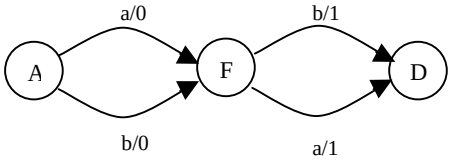

Fig. 5 A "simpler" automaton that is wrongly implemented by *P*.

Another example exploiting unrestricted groupings is automaton M_5, which is also implemented by *P* under the grouping E:={B,C,D} (see figure 6). While in the above case automaton M_4 wrongly suggested two additional possibilities to turn the light on, M_5's implementation is truly an unwanted result, because the simpler machine M_5 should not be able to exhibit *more complex behaviors* than its implementing machine M_1 (otherwise the notion of "simpler" is rendered absurd).[6] Yet, M_5 suggests that after having pressed switch 1 or 2 once, one can keep pressing both of them indefinitely.

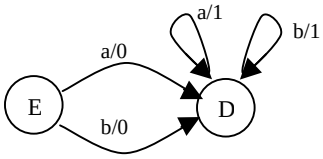

Fig. 6 Another "simpler" automaton wrongly implemented by *P*, which exhibits more complex behavior than its implementing system.

Finally, a deterministic machine (like M_1) can be even turned into a non-deterministic machine (like M_6 under the grouping G:={A,C} and H:={B,D}—see Figure 7): pressing switch 1 alone might or might not turn the light on.

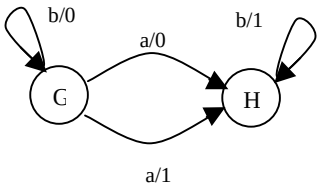

Fig. 7 A "simpler" non-deterministic automaton implemented by *P*—the light might or might not come on if the switch represented by 'a' is pressed.

The last three machines (M_4, M_5, and M_6) showed that if no restrictions are imposed on groupings of physical states, then simple, finite, deterministic physical systems (such

as the switch system *P*) can possibly be seen to implement complex, infinite, and non-deterministic computations, none of which reflects the causal structure of the switch system anymore. The only way such unwelcome implementations could be prohibited (while retaining the overall structure of Chalmers' definition of implementation) is by imposing restrictions on the grouping of physical states. Yet, it is not clear how this could be achieved in a definition like *Ch2* (using the "functional approach"). Formally, it is easier to incorporate a restriction into *Ch1* (where groupings of physical states are already *separated* from the mapping) by restricting the existential quantifier to "there exists a φ-grouping..." where φ specifies the constraints. In either case, the *exact nature* of the constraints will have to be specified if the definitions are to be of any (practical) use.

3. Critique

The above difficulties with Chalmers' notion of implementation have, in my view, three independent sources: 1) his conflating the conceptually distinct steps (or operations) of grouping physical states into types and specifying (or establishing) the correspondence mapping itself (in *Ch2*), 2) his conception of *simpler computation*, and 3) his view that physical systems implement simpler computations *simultaneously*. I will address these issues in reversed order.

It is certainly true and in accordance with computational practice to view—as Chalmers does—a standard desktop computer, say, as implementing many computations at the same time. However, by saying that "the system on my desk is currently implementing all kinds of computations, from EMACS to a clock program, and various subcomputations of these." [Chalmers (1994), p. 397] one *does not* look at different groupings of all physical states (of the system) that can be set in correspondence with the clock or Emacs program; rather one considers on a *subset* of all physical states in the computer, i.e., those states that exhibit the right kind of correspondence to the computational ones (e.g., the memory locations in which the respective programs are stored). Take again the switch system *P* and consider the following "subsystem" *P'* of *P*, which consists of the battery, one switch, and the light bulb (see Figure 8).

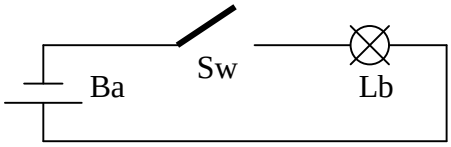

Fig. 8 The subsystem *P'* of system *P* consisting of a battery, one switch, and a light bulb.

P' is a subsystem of *P* in the sense that it has the same causal structure as *P* if one of the two switches in P has already been pressed. In other words, the characteristic automaton $M^{P'}$ of *P'* is a "subautomaton" of the characteristic automaton M^{P} of *P*. To be precise, $M^{P'}$ is isomorphic to either the transition (B,b)→(D,1) or the transition (C,a)→(D,1) of M^{P} (i.e., M_1) depending on which switch of *P* is being considered to correspond to "Sw" of *P'* (switch 2 in the former case, switch l in the latter). Note that

the states of $M^{P'}$ are a subset (or, more generally, are in 1-1 correspondence with a subset) of the states of M^P, and the transitions of $M^{P'}$ are a subset (or, more generally, are in 1-1 correspondence with a subset) of the transitions of M^P.

In general, I would claim, it is this idea of focusing on a subset of the set of all physical states of a system together with (some) transitions between them that usually underwrites the notion of implementation in computational practice and also in cognitive science: in computer science, this is apparent from the fact that programs have "locations" in memory associated with them, which in turn correspond to a subset of physical states of the computer; in cognitive science, it is underlined when people speak of cognitive functions such as "motor control", "object recognition", "memory recall", "speech production", etc. and *where* in the brain these functions are *implemented* (i.e., in which cortical areas).

Chalmers' conception rather seems to correspond to cases, in which programs implement *virtual machines*: in such a case, one would group states of the implementing machine that implement states of the virtual machine together and view the group as corresponding to a single state of the virtual machine. For example, if a PC-simulator is implemented on a MAC computer, then a computational state in the implemented (virtual) CPU of the PC will correspond to a sequence of states in the *implementing* CPU on the MAC. Note, however, that even in such a scenario *Ch2* is not satisfactory, as the sets of states in the implementing machine that correspond to states of the virtual machine might not be disjoint (as is required by *f* being a function in *Ch2*).

As a result, one has to distinguish (at least) two notions of complexity: one that corresponds to "being a subautomaton" (when one concentrates on subsets of states and transitions of the original automaton), and another that is based on groupings of physical states. The former is quite unproblematic and seems to be in accordance with some aspect of Chalmers' implicit notion of complexity:

Definition 3: An FSA *M*' is said to be *at most as complex as* an FSA *M*, if there exists a one-to-one mapping from states of *M*' to states of *M* such that [iso⇒] holds with respect to their state transitions.[7]

Corollary 4: Every FSA *M*' which is at most as complex as some FSA *M* (according to definition 3) is implemented by every physical system *P* that implements *M* (in the sense of *Ch2*).

Proof: Let *M*' and *M* be FSAs such that *M*' is at most as complex as *M*. Since *P* implements *M*, there exists a surjective mapping *f* from physical states of *P* to computational states of *M* such that [iso⇐] holds between causal transitions and state transitions. To show that *P* implements *M*' according to *Ch2*, it suffices to exhibit a surjective mapping *g* from *M* to *M*' such that [iso⇐] holds between the state transitions of the two FSAs, since the composition of *f* and *g* will be a surjective function between physical states and computational states of *M*' such that [iso⇐] holds between causal transitions in *P* and state transitions in *M*'. Since *M*' is at most as complex as *M*, there exists a mapping *h* from states of *M*' to states of *M* such that [iso⇒] holds with respect to their state transitions. Hence, the inverse h^{-1} of *h* is a surjective, one-to-one mapping

from those states in *M* that are in the range of *h* to all states in *M'* such that [iso⇐] holds. To extend h^{-1} to get the required mapping *g*, one simply maps those states of *M* that are not in the domain of h^{-1} to some states of *M'*. Note that they can be mapped to arbitrary states in *M'*, since [iso⇐] will not be affected by any such mapping. This last step essentially makes use of the fact that states can be arbitrarily grouped according to *Ch2*.

The second notion of complexity presupposes a notion of "legitimate" grouping of physical states (one must not allow arbitrary groupings as demonstrated in the previous section). Even if one agrees with Chalmers that physical systems can have *multiple* causal structures depending on the grouping of their physical states, it seems that the notion of implementation should not be made accountable for which of these groupings are legitimate. If I understand Chalmers' idea of grouping physical states right, that is, if it is supposed to correspond to the intuition that the same physical system (i.e., the same spatio-temporal region) can be described at different levels of abstraction ("at a different causal grain" as he puts it), then one would ultimately want the physical theory that delivered the *physical states* in the first place to delimit the set of all possible groupings: a grouping of states that is excluded for whatever physical reason should not enter the picture again *via* implementation (as with *Ch*). The notion of implementation should rather be used to show *that* (and maybe *how*) a certain abstraction can be *implemented*, i.e., related to something *less abstract* [see also ch. 4 of Agre (1997)]. Groupings of states, therefore, have to be separated from the correspondence mapping.[8]

This is, however, not to say that general arguments to delimit the set of possible groupings (as part of a "general theory of abstraction", for example) are excluded *apriori*. In fact, there are ways to restrict the set of all "reliable and counterfactual supporting" transitions between *grouped states* by looking only at the transitions between the given states.

Recall automaton M_4: grouping states B and C together to form the grouped state F was problematic, because state F made is formally possible to reach state D from A using input 'a' twice. In M_4 it is possible to enter F on transition (A,a)→(F,0) and leave F on transition (F,a)→(D,1) (i.e., enter F *via* member state B and leave F *via* member state C, where "member state" refers to a state part of the grouping of a grouped state). But that is physically impossible in *P* (one switch cannot be pressed twice to turn the light on). The *prima facie* problem with state F seems to be that not all of its member states share the same transitions coming into and going out of F. Obviously, if all member states of a grouped states have the same transitions coming into and going out of them, then grouping them together is not going to alter the causal structure. Turning this implication around, however, would be too restrictive: there are (formally) legitimate groupings where not every member state shares every transition entering or leaving the grouped state. It seems sufficient for member states of a group to be connected in such a way as to guarantee a path (i.e., sequence of transitions) from every member state, through which the group state can be entered, to every member state, through which the group state can be left.[9]

To see this, imagine an extension *P** to system *P* with an additional button *Bu* that can be pushed any number of times. Pushing the button will exchange the position of the

two switches: if one switch is up and the other one down, after pushing *Bu* the first one will be down and the second one up. If both switches are up or both are down, pushing *Bu* will have no effect (Figure 9 depicts the characteristic automaton of this system, where 'c' stands for "pushing button *Bu*").

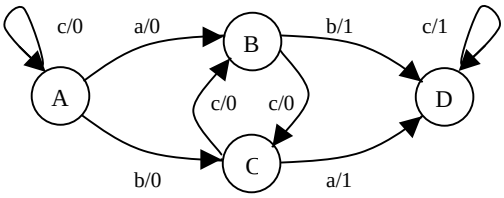

Fig. 9 The characteristic automaton M^{P*} of an extension P^* of P (by an additional button *Bu*, which—if puhsed—switches the position of both if they are different); 'c' indicates "pushing *Bu*".

Now, consider the relation between M_4 and P^*. According to definition 3, M_4 can be seen to be less complex than M_4^*, which is obtained from P^* by grouping states B and C to form state F. Hence, if M_4^* reflects (part of) the causal structure of P^*, then M_4 will. It is easy to check that every member state of F through which F can be entered (i.e., B and C) is connected to some member state through which F can be left (i.e., B and C) by a sequence of transitions (i.e., (B,c)→(C,0) and (C,c)→(B,0)).[10] Therefore, M_4^* and M_4 both reflect (part of) the causal structure of P^*; in both systems it is possible to enter F via member state B and leave F via member state C. The difference between M_4^* and M_4 is that M_4 is ignorant of transitions caused by pressing *Bu*. But this difference does not matter if one is merely interested in a state like "one switch down" (i.e., F) from which it is possible to turn the light on by pressing any switch (without having to know *all* the details about *what else* needs or needed to be done, e.g., pushing the button if one entered through B and left through C).

While the above requirement about the relation between member states of a group has to be satisfied *regardless* of the physical nature of the involved states, additional constraints on groupings coming from the respective physical theory itself will have to be taken into account. Certain groupings that would seem legitimate from a formal point of view, might still violate physical principles and laws. The groupings of physical states in M_3, for example, in which both switches are viewed as being pressed together as if they were only one switch, might not be permissible if the switches are so far apart that "pressing them together" is impossible or does not make sense according to the physical theory (e.g., the theory of relativity). Obviously such restrictions cannot be determined at a mere formal level, but will depend on physical properties of the physical system, i.e., on *implementation details* that are abstracted over by mere structural descriptions.

There is yet another problem with Chalmers' view of "simpler" that has not been addressed so far: only computations with (at most) the same number of states and fewer transitions as a given computation *C* can count as simpler than *C*. Such a complexity measure is problematic, however, because there are (complex) automata with a large number of states that compute very *simple functions* while other (simple) automata with only few states compute very *complex functions* (here the terms "simple" and "complex

function" are to be understood in the sense of standard complexity theory). For example, every fully connected *n*-state FSA implements every FSA with up to *n* states according to *Ch2* (yet it cannot be implemented by any other FSA of *n* states).[11] This automaton is therefore more complex (according to *Ch2*) than all the other *n*-state FSAs, yet intuitively it does not seem right to say that such the FSA has a *more complex causal structure* (as every state can be reached from every other state with every input).

There are many examples from computational practice showing that even computations with more computational states than physical states of a given physical system can be implemented by such a system. The most obvious examples are probably redundant computations or redundant data structures, where "redundant" means that these "sub"-computations or data structures could be somehow eliminated without affecting the overall computation. If an optimizing compiler, for example, detects that two variables have the same value at all times in a given computation *C*, it will map both onto the same machine register on a given machine *M*. Or, in another scenario, the compiler determines that the respective values of two variables are only needed once during some non-overlapping parts of the overall computation, and thus maps both variables to the same memory location. In both cases two different computational states are set in correspondence with one physical state (by virtue of the compiler). According to *Ch2*, however, *M* does not implement *C*, while *C* compiled by a non-optimizing compiler (which does not result in fewer states) would count as being implemented. Or to turn it around, suppose a machine *M*' only has a certain number of registers such that only the optimized version *C*' of *C* can be run on *M*', in which the redundant operations have been eliminated. Then *M*' implements *C*', but does not implement *C* according to *Ch2*. At the same time, *M*' is running a program that behaves as specified by *C* in *every computational aspect*.[12]

Additional examples are data structures of various types such as "bit set", "bit vector", "character", "byte", etc. which are all conceptually distinct, but can be (and are on some machines) implemented in the same physical region using the same physical states. Just consider a particular location in main memory, in which a bit vector of 8 bits is stored at a given time. At some later time, a "character", which happens to have "the same physical representation", is stored in the same location...

The above implementation results according to *Ch* are clearly in conflict with the uses the notion of implementation is put to in practice. In fact, computer scientists speak of "implementation" even in cases where computations are defined over infinite data structures (such as infinite streams), which because of their infinite nature have infinitely many computational states.[13] Other examples demonstrating implementations of architectures with much larger state spaces than those of physical computers are various forms of sparse matrices (where a matrix with more elements than states in the computer's memory can be fit into memory using subtle "implementation techniques") or virtual memory systems (that manage to maintain only those portions of the virtual memory in "physical memory", which are accessed by the current computational process). While I do not want to endorse the view that scientific terms have to be in agreement with their practical usage, in the case of implementation it is practice that

determines the range of application a formal definition of implementation has to account for. After all, the above examples of computations can indeed be *implemented*!

4. A Revised Notion of Implementation

Chalmers intended his notion of implementation to block "unwanted" implementation results (such as Putnam's) and at the same time to account for the fact that "within every physical system there are numerous computational systems" (1994, p. 397) of some complexity, where the complexity of a computation is defined in terms of "implementation". Furthermore, he wanted to provide a very general notion of implementation, which suits the needs of cognitive science to describe the causal structure of physical systems computationally. Unfortunately, his notion does not respect intuitions about implementation from computational practice (as apparent from the examples of "unwanted" and "unaccounted" implementations in sections 2 and 3) and furthermore results in an unsatisfactory notion of "computational complexity".

The former difficulties can be remedied easily: 1) groupings of physical states have to be separated from the correspondence mapping, and 2) the notion of "subautomaton" has to be used as a complexity measure instead of the groupings of physical states (note that "implementation" is now defined *relative* to groupings of physical states):

Definition 5: Let *S* be a physical system and *GS* be a φ-grouping of its physical states into types. Then *S implements* an FSA *M* iff *M* is at most as complex (in the sense of Definition 3) as M^{GS} (the characteristic automaton with respect to *GS*).

If we take FSA descriptions to be the most abstract characterizations of physical organizations with respect to causality, then this definition of implementation implies a natural way to compare physical systems with respect to their causal complexity: a physical system *P* is *at most as complex as* a physical system *P'* if every FSA *M* implemented by *P* is at most as complex (in the sense of Definition 3) as some FSA *M'* implemented by *P'* (in symbols: $P <=_c P'$).

Definition 5 is more restrictive than Chalmers' as can be seen from the following theorem:

Theorem 6: Let *S* be a physical system and *M* an FSA. *S implements M* according to C*h2*, if *S* implements *M* according to definition 5. On the other hand, there exists a physical system and an FSA *M implemented* by *S* according to *Ch2*, which is not *implemented* by *S* according to definition 5.

Proof: For the first part observe that *Ch2* does not impose any restrictions on groupings, hence *S* implements M^{GS} for every φ-grouping *GS* according to *Ch2*. Using corollary 4 it follows that *S* also *implements* every FSA *M'* which is at most as complex as M^{GS}, i.e., *S implements* every FSA *M* according to C*h2*, which is implemented by *S* according to definition 5. For the second part, consider the switch system *P*, which implements M_4 according to *Ch2*, yet there is no φ-grouping *GS* (according to circuit theory) such that *P*

implements M_4 according to definition 5 (the argument that there is no legitimate grouping of states corresponding to M_4 was given in section 2).

While definition 5 eliminated unwanted implementations by being more restrictive than *Ch2*, both definitions are *too restrictive* in that they require a physical system to implement only computations with at most the same number of computational states as there are physical states in the system. Yet, as mentioned before, there are situations where it makes perfect sense to view certain computations as being implemented by a system even though the system has fewer physical states than there are computational states. Accounting for such implementations means to relax the constraints on the correspondence mapping and allow single physical states to be related to (possibly) many computational states. In other words, implementation will have to be viewed as a relation! The upshot of this move is that one has to abandon isomorphism, since it is only defined for functions... . Fortunately, there is a substitute that expresses exactly the idea of "mirroring" under these relaxed conditions, the notion of "bisimilarity":[14]

Definition 7: [Bisimilarity] Let I and O be two finite sets (e.g., the sets of input and output states, respectively) and let $M_S = \langle S, \rightarrow_S \rangle$ and $M_T = \langle T, \rightarrow_T \rangle$ be two structures with domains S and T, respectively, where the relation $\rightarrow_S$ is defined over $S \times I \times S \times O$ and relation the $\rightarrow_T$ is defined over $T \times I \times T \times O$. These structures are then said to be *bisimilar* (symbolically $M_S =_{\text{bisim}} M_T$) if there exists a non-empty relation R (called "bisimulation") between S and T such that the following four conditions hold:

1. If $R(s,t)$ and $(s,i) \rightarrow_S (s',o)$, then there exists $t' \in T$ such that $R(s',t')$ and $(t,i) \rightarrow_T (t',o)$
2. If $R(s,t)$ and $(t,i) \rightarrow_T (t',o)$, then there exists $s' \in S$ such that $R(s',t')$ and $(s,i) \rightarrow_S (s',o)$
3. For every $s \in S$ there exists a $t \in T$ such that $R(s,t)$
4. For every $t \in T$ there exists a $s \in S$ such that $R(s,t)$

Definition 7 can be easily applied to the above implementation problem: one structure will be the description of the physical system with its physical states and the relation "reliably transits" (a causal notion), while the other structure will be the FSA with its transition relation. It is ensured that for any physical state s and any causal transition from s there will be a computational state c related to s and a computational transition from c related to the causal transition in the physical system such that the physical state s' resulting from the causal transitions is again related to a computational state c' resulting from the computational transition, and vice versa. Thus, systems that are related according to definition 7 will not only have the same input-output behavior, but should intuitively also have all "computational sequences" (or *computational processes*, if you will) for all possible inputs in common, where "computational sequence" is defined as follows:

Definition 8: [Computational sequence] A *computational sequence* (in a system, physical or abstract) is a finite sequence $\langle x_1, x_2, ..., x_n \rangle$ of pairs $x_i = \langle u_i, v_i \rangle \in (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$,

where Σ is the input alphabet, Γ the output alphabet, ε the "empty string", and x_i the label of a transition in the system.[15] The *input of* and the *output produced by a computational sequence* $<x_1, x_2,..., x_n>$ are the strings $u_1u_2...u_n$ and $v_1v_2...v_n$, respectively.

Definition 8 formalizes the intuition that a computational sequence, i.e., *how* an output is obtained from a given input, is defined and completely determined by state transitions (i.e., the inputs and outputs that occur at each transition). The following corollary is an immediate consequence of definition 8:

Corollary 9: Any two bisimilar systems (and as a consequence, any two isomorphic systems) have identical computational sequences.
Proof: Let $M_S = \langle S,\rightarrow_S\rangle$ and $M_T = \langle T,\rightarrow_T\rangle$ be two bisimilar structures, let R be a bisimulation for M_S and M_T, and $<x_1, x_2,..., x_n>$ be an arbitrary computational sequence. Then there are two cases: either there exist $s_1, s_2,..., s_{n+1}\in S$ such that $<s_i,u_i>\rightarrow_S <s_{i+1},v_i>$ where $x_i=<u_i,v_i>$ for all $i <= n$ or such s_i do not exist. In the first case, by clause (3) of definition 7 there exists a $t_1\in T$ such that $R(s_1,t_1)$, and consequently by clause (1) a t_2 such that $<t_1,u_1>\rightarrow_T <t_2,v_2>$. By repeatedly applying clauses (3) and (1) using an induction argument one can furthermore show that $<t_i,u_i>\rightarrow_T<t_{i+1},v_i>$ for all $i <= n$ (where $R(s_i,t_i)$ for all $i <= n+1$). The same argument works in the reverse direction using clauses (2) and (4) of definition 8. In the second case, in which appropriate s_i (or t_i) do not exist, there cannot be appropriate t_i (s_i) either, otherwise the argument for the first case above could be used to show that s_i (t_i) must exist, yielding a contradiction. Hence, M_S and M_T have the same computational sequences.

Corollary 9 shows that as far as computational sequences are concerned it does not matter what the computational states are or how many different computational states are involved in a particular computation for two systems to have all possible computational sequences in common (i.e., *to be equally complex from a computational point of view*!). All that matters about computational states is their role in the whole network of transitions, what transitions come into a state and what transitions leave the state, where transitions are determined by their labels, i.e., their inputs and outputs (this description is reminiscent of functionalist views on functional states—see the next section). Hence, two systems with a different number of "inner" states that are bisimilar will be indistinguishable from a computational point of view (i.e., with respect to their possible computational sequences).

On the other hand, computational sequences can distinguish physical systems that do not have the same computational sequences in common despite the fact that both systems might show the same input-output behavior. There are basically two possibilities of how any two computational sequences on the same input producing the same output can differ: either 1) they are of different length, or 2) if they have the same length, they differ on at least one particular pair. Systems that differ with respect to at least one computational sequence cannot have the same "computational structure" (as determined by their computational sequences), they compute some input-output pairs differently. Such systems then do not even have to differ with respect to *what* they compute. What

matters is that they differ in *how* they compute it for them to be separable by computational descriptions (together with an appropriate notion of implementation).

The following notion of implementation, then, incorporates the notion of bisimilarity into definition 5 and can, therefore, answer the question *what* a system computes and *how* it computes it without having to require that physical states and computational states are in a particular functional correspondence:

Definition 10: [Implementation revised] Let *S* be a physical system and *GS* be a φ-grouping of its physical states into types. Then *S implements* an FSA *M* iff *M* is bisimilar to some automaton at most as complex (in the sense of Definition 3) as the smallest automaton bisimilar to M^{GS} (the characteristic automaton with respect to *GS*).

Definition 10 effectively states that in order to be "implementable on a physical system" computations must share one crucial aspect with the physical system: the state transitional structure of the computation must be at most as complex as the causal transitional structure of the implementing physical system with respect to computational sequences. While (infinitely) more automata are viewed as implemented by a given physical system according to definition 10 than according to definition 5, unwanted implementations are still excluded. For example, the switch system *P* does not implement any of M_4, M_5, or M_6 according to definition 10 while implementing them according to *Ch2*. This shows that definition 10 and *Ch2* are essentially different. They agree on some implementations (e.g., on all implementations with respect to definition 5), but differ on others. Yet, definition 10 captures some of the intuitions that Chalmers might have had about groupings of physical states. For example, it is now possible to establish formally that a virtual machine *VM* (i.e., an abstract description of a computational architecture) is implemented on a given physical system *S* (where different sequences of transitions in *S* correspond to only one transition in *VM* or more than one transition in the *VM* corresponds to only one transition in *S*).

More interestingly, a bisimulation implicitly defines a *minimal* set of possible groupings, namely those that result in bisimilar automata: just consider for a given FSA *M* the class of all bisimilar FSAs that have fewer states than the given automaton. Every FSA in this class will define a particular legitimate grouping of states of *M* while preserving all computational sequences of *M*.[16] Note that the set of groupings obtained by looking at bisimilar automata is a subset of the set of physically allowable groupings (which in turn is a subset of all possible groupings): additional possible groupings (i.e., groupings that do not result in bisimilar automata) will have to be defined by the underlying physical theory.

While definition 10 could correct the shortcomings of definition 5, there is a price to be paid for relaxing implementation constraints: consider two non-isomorphic systems that have identical computational sequences for all possible inputs (as in figure 10), call the left one 'M_p' and the right one '$M_{p"}$'. Recall that such systems, being bisimilar, are identical not only with respect to *what* they compute, but also with respect to *how* they compute it. In other words, they are indistinguishable (or *equally complex*) from a computational point of view. While computational descriptions according to definition 5

were able to tell non-isomorphic bisimilar systems apart, computational descriptions according to definition 10 cannot distinguish between *isomorphic* and *non-isomorphic* bisimilar systems anymore!

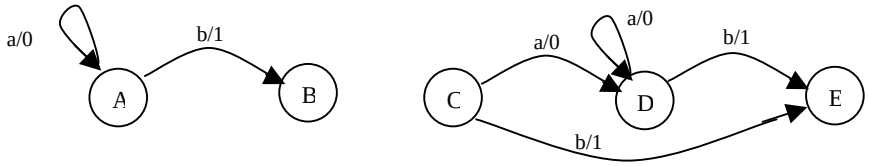

Fig. 10 Two bisimilar, but non-isomorphic automata. The bisimilarity *R* is the relation {<A,C>,<A,D>,<B,E>}.

Note that physical systems *P* of which M_p is the characteristic automaton are causally less complex than physical systems *P*" of which $M_{p"}$" is the characteristic automaton, yet both systems have the same computational complexity. To see that the relationship between causal complexity and computational complexity is quite complex, consider the automaton $M_{p'}$' (and its corresponding physical systems *P'*) obtained from *M*" by removing the transition from C to E: $P <_c P' <_c P$", yet $M_p =_{bisim} M_{p"}$", but neither $M_p =_{bisim} M_{p'}$' nor $M_{p'}' =_{bisim} M_{p"}$". If the number of states of the smallest automaton/automata in each equivalence class of bisimular automata is taken to be the measure of their computational complexity (e.g., compare this to Chaitin's notion of computatational complexity), then we get that $M_{p'}$ is computationally more complex than $M_{p"}$ (while its implementing systems are causally less complex than the implementing systems of the computationally simpler automaton). In short: the computational complexity and the causal complexity of a physical system can be different.

5. Implementation and Functionalism

Definition 10 opens an interesting perspective on functionalist theories of mind that will allow us to view the relation between functionalist descriptions and physical systems in a new light. First, however, note that definition 10 seems to be some sort of intermediary between a behaviorist and a functionalist view on implementation: while the behaviorist only requires of a system that it get the input-output function (i.e., the behavior) right to be able to say that it "implements" this function/behavior, the functionalist imposes (among others) constraints on inner states, namely that inputs cause the system to assume certain inner states (which in turn cause it to produce certain outputs). Thus, to say that a system implements a functionalist description is to require that it get the mapping of the inner states right in addition to the input and output mapping. Usually, these "inner states" are assumed to be *multiply realizable*, therefore the mapping has to be a many-to-one mapping from physical states to functional states. Yet, inner states are viewed by functionalists as intrinsically relational states, being mutually defined by all states in the functional architecture. It seems to me that this conception of states solely defined in terms of their relations to each other is what contradicts isomorphism-based definitions of implementation and thus prevents the functionalist from having an homo/iso-morphism-based notion of implementation.

To see this, consider, for example, the following automaton, which has two inner states ‘E’ and ‘O’ standing for “even” and “odd”. Depending on whether the number of ‘1’s that the automaton has seen so far is even or odd, it outputs either ‘a’ or ‘b’, respectively.

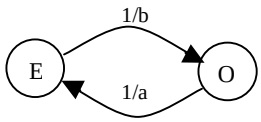

Fig. 11 The even-odd transducer with two inner states.

A functionalist account of what it means to be in state E would look like this [e.g., see Block (1995)]:

> Being in E $=_{def}$ Being an x such that $\exists P\, \exists Q$ [x is in P $\wedge$ (if x is in P and gets input ‘1’, then it goes into Q and outputs ‘b’) $\wedge$ (if x is in Q and gets input ‘1’, then it goes into P and outputs ‘a’)].[17]

Since it is only claimed that there has to be an arrangement of physical states that corresponds to the functional states in a way that preserves inputs and outputs as well as transitions between states, it is possible for one physical state to serve as the instantiation of more than one functional state. In the previous example this would amount to P=Q, i.e., E and O having the *same* (physical) realizer while being *distinct* functional states. Therefore, the correspondence between physical systems and functional architectures cannot be based on a mapping between physical states and functional states to preserve causal transitions. Rather, causal transitions will have to be preserved in a less restrictive way allowing functional states to have possibly multiple realizers and physical states to be the realizers of possibly multiple functional states. But this means that implementation, from a functionalist point of view, has to be viewed as some sort of “bisimilarity” between functional and physical architectures.

Obviously, this conclusion can be prevented by simply requiring that $\exists P\, \exists Q(P \neq Q \wedge$ [...]). The question then is on what grounds this move could be justified as it seems to make a particular *physical claim*, namely that P and Q have to be distinct. Furthermore, a particular kind of “supervenience claim” seems to be implied (e.g., that there can be no mental difference without a physical difference), a claim that in my view should not be introduced into the functionalist notion of realization [Papineau (1995), p. 240 uses a different argument to make the same point “there is nothing puzzling about the idea of a state that fails to supervene on the physics of the brain, but is nevertheless realized by some physical feature of the brain”].

One philosophical consequence is that the notion of implementation of a computation (in the sense of definition 10) and that of a functional architecture coincide at least with respect to the way physical states are related to computational or functional states, respectively.[18] Of course, the functionalist notion of implementation (or “realization”, as it is often called) would need to be made more precise to allow for a formal comparison.[19] But even without a formal notion of realization it is possible to see that

functionalist accounts will fail at capturing the causal complexity of physical systems if computationalist accounts fail in this respect.

The question then remains whether computational descriptions do actually fail to capture the causal structure of physical systems. In the light of the above, there seem to be two possible answers depending on one's metaphysical stance on "the causal structure of a physical system": if the causal structure of a physical system (i.e., its causal complexity) is determined by its various physical states *and* their causal relations among each other, then computational, and as a consequence functionalist descriptions will not be able to cover and capture *all* aspects of the causal structure. If, on the other hand, the causal structure of a physical system is solely determined by its various causal transitions, where physical states only play a subordinate role (i.e., in defining and fixing causal relations), then computational descriptions will be able to capture the causal structure of those physical systems that implement them.[20] And *mutatis mutandis*, functionalist descriptions will capture the causal structure of physical systems *realizing* them (modulo an appropriate notion of "functional realization").

6. Discussion and Conclusion

The notion of implementation suggested in definition 10 sheds new light on the relation between computations and physical systems by relaxing conditions about the correspondence of physical and computational states. Rather than establishing a rigid mapping between physical and computational states, definition 10 requires only a rigid correspondence between computational and causal *paths*. In other words, every computational sequence corresponds to a causal sequence, and every causal to a computational sequence—computations and (subsystems of) physical systems have to agree on their transitions.

This shift from states to transitions has various philosophical implications: for one, computational states are rendered inappropriate as a measure of the *causal complexity* of a physical system if the causal complexity of a physical system is defined by its physical states and all possible causal transitions between them. As shown at the end of section 4, such a complexity measure drives a wedge between the *causal* and the *computational complexity* (as induced by the notion of "smallest bisimilar automaton") of physical systems. Consequently, this raises questions about the adequacy of its underlying notion of causal complexity as well as the appropriateness of intuitions about computations mirroring the causal structure of their implementing physical systems. Furthermore, possible implications of the conceptual separation of causal and computational complexity for functionalism and in particular computationalism need to be investigated.

It seems to me that computationalism is unaffected by the notion of implementation suggested in definition 10. While it is true that computations cannot single out the particular causal structure of a physical system (the brain, say) according to the suggested notion of implementation, they are still appropriate to describe all possible computational, i.e., causal sequences of their implementing system. This should be sufficient if the cognitively relevant aspects of physical systems are the ones preserved under bisimilarity (and, hence, open to computational descriptions).[21] In fact, I suspect

that computationalists, while usually speaking of computational *states*, have always tacitly taken the notion of computational *process* (instead of computational state) to underwrite their paradigm. If my suspicion is correct, then the above notion of implementation should not come as a surprise to computationalists nor should it disturb their endeavor, as computationalism really is the claim that *cognitive processes are computational processes*.

The same is not quite as clear in the case of functionalism. Mental states cannot always be viewed as processes (what does it mean to have a belief in terms of processes?). Furthermore, some views of how functional descriptions are related to physical systems might turn out to be problematic, in particular those views that base functional realization on some notion of "supervenience", for it might no longer be true that the physical realization of mental properties (necessarily) implies their supervenience [as Kim (1998), p. 23 suggests]. Or it might even turn out that the notion of bisimilarity used in definition 10 is still too strong to account for how functional states relate to the physical (in that it requires all transitions of two systems to match up perfectly) and would have to be further weakened. A satisfactory answer to these questions, however, will require precise notions of "functional architecture" and "realization of a functional architecture", none of which are available to my knowledge as of today.

Notes

[1] The usage of the term "mirror" is indeed quite confusing in Chalmers' paper: whereas one would expect "mirror" to mean something like "is isomorphic to" (as "mirrors" usually indicates sameness in structure), neither *Ch1* nor *Ch2* imply *structural sameness*, since both only require [iso⇐] (i.e., that "formal states related by an abstract state-transition relation are mapped onto physical states-types related by a corresponding causal state-transition relation"), but not the other direction [iso⇒] (i.e., that "physical states-types related by the corresponding causal state-transition relation have to be mapped onto formal states related by an abstract state-transition relation"). This would suggest that "mirrors" means [iso⇐]. Yet, there are other places in the text, where "mirror" seems to suggest only [iso⇒], as in the quote preceding this footnote.

[2] That switches can only be pressed once is not essential to the argument, but it allows for a simpler presentation of the material.

[3] Note that δ is not defined for all transitions, thus making the automaton formally non-deterministic. While this is a mere formality (as the automaton is still deterministic) it can be accounted for by requiring that in cases where the transitions is not defined for some input (e.g., $\langle B,a\rangle$ and $\langle C,b\rangle$) the automaton stay in its current state and output nothing (i.e., $\langle B,\varepsilon\rangle$ and $\langle C,\varepsilon\rangle$ respectively) or go into a "dead states" from which no other state can be reached. See, for example, Hopcroft and Ullman (1979).

[4] If not stated otherwise, "implement" will always be understood in the sense of *Ch2* in sections 2 and 3. Furthermore, the term "states" will always refer to input, inner, and output states collectively.

[5] To see that *P* implements M_4 according to *Ch2*, consider, for example, the transition $(F,a)\rightarrow(D,1)$. Then there exists a state *S* in *P* such that $f(S)=F$. Obviously, S=ud as with S=du there are no transitions with a to D. The only possible input is "1d", upon which *P* reliably transits into state "dd" and outputs "+", i.e., $(f(ud),f(1d))$ *causes* $(f(dd),f(+))=(F,a)\rightarrow(D,1)$. Similar arguments work for the other three transitions.

[6] By "more complex behaviors" I intend to point to the fact that M_5 computes an infinite input-output function, namely $(a\cup b)^n(a\cup b\}\rightarrow 0^n1$ for all natural numbers *n* (using the standard notation for regular expressions), while the input-output function computed by M_1 consists only of the two pairs <ab,01> and <ba,10>.

[7] This notion respects intuitive ideas about the complexity of a system (such as considering only part of all computational states of a system, which is possible in *Ch2* only if *f* is allowed to be a partial mapping) as well as those about the complexity of behaviors: according to this complexity measure, *M* can emulate

every behavior of *M*' (i.e., there exists a surjective mapping *h* between the output of *M* and *M*' such that every behavior of *M* will correspond to at least one behavior of *M* under *h*). The last section showed that this is not true of *Ch2*.

[8] In this respect *Ch1* was already on the right track: the existential quantifier ranges over mappings *and* over groupings of physical states, yet this conceptual separation was (formally) abandoned in *Ch2*.

[9] Note that this requirement of having member states connected in certain ways is still a rough cut. In general there will be additional restrictions on the kinds of permissible transitions within a group, since transitions within groups that affect inputs and outputs of the overall system in a way that "matters" to the overall behavior of the system must not be allowed.

[10] Reflexive transitions, i.e., transitions of the form $(X,y) \rightarrow (X,z)$ for states X, are not required, since X can be reached from X without any transition.

[11] This is quite informal. To make this claim precise, one has to take the number of different input and output states into consideration.

[12] Note that compilers do not *change* the computational description, rather they attempt to *map it* to the given machine architecture.

[13] Obviously, at any given time only a finite number of the infinitely many computational states can be implemented in a physical system. This raises the question whether "implemented" is the right term to describe the relationship between a program using infinite streams, for example, and what actually runs on the computer. Without going into details here, I would claim that it is by virtue of *how* these computations are implemented (in an intuitive sense), e.g., using delay mechanisms, etc. that we are justified in saying *that* they are implemented (in a strict sense).

[14] See, e.g., Barwise and Moss (1996), p. 37, who regard bisimilar automata as "almost isomorphic", i.e., "the same from the outside".

[15] The empty string (or empty character) is included to model transitions that do not depend on (relevant) input or that do not produce (relevant) output (i.e., so-called ε-transitions).

[16] While there might not be a unique smallest automaton, one can still take the number of states of some smallest automata as a measure of the complexity of the input-output behavior of *M* and conclude that in order to show such and such behavior it is necessary to have *at least* that many states. As a consequence using definition 10, the same applies to the number of physical states of physical systems implementing *M*.

[17] Note that the existential quantifiers could be viewed as ranging over properties or as picking out particular physical states of the system.

[18] In fact, it is hard to see how both notions could differ at all if the set of functional states is taken to be finite (as functional descriptions would then be reduced to FSA descriptions, thus making definition 10 applicable).

[19] There are a few promising attempts to formalize the functionalist notion of realization [e.g., Kim (1998), or David (1997)].

[20] It seems to me, however, that this move would first and foremost require a reconsideration of the notion of physical state and the role it plays in physical descriptions and explanations.

[21] Put differently, if all that matters is "how a function is computed" (as it is common in computational practice), then a computational description of the causal transitional structure of the implementing system will capture the relevant aspects.

References

Agre, P. (1997), *Computation and Human Experience*. Cambridge, Cambridge University Press.

Barwise, J., and Moss, L. (1996), *Vicious Circles*, CSLI Lecture Notes, Cambridge University Press.

Block, N. (1996) 'What is Functionalism?' In *The Encyclopedia of Philosophy Supplement*, Macmillan.

Chalmers, D. J. (1994), 'On Implementing a Computation', *Minds and Machines* 4, 391—402.

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, 403—420.

Cummins, R. (1989), *Meaning and Mental Representation*, Cambridge, MA, MIT Press.

David, M. (1997), ‘Kim’s Functionalism’, In *Philosophical Perspectives, 11, Mind, Causation, and World*.

Hopcroft and, J. E. and Ullman, J. D. (1979), *Introduction to Automata Theory, Languages, and Computation*. Massachusetts, Addison-Wesley Publishing Company.

Kim, J. (1998), *Mind in a Physical World*, Cambridge, MA, MIT Press.

MacLennan, B. J. (1994), ‘Words Lie in Our Way’, *Minds and Machines* 4, 421—437.

Melnyk, A. (1996), ‘Searle’s Abstract Argument Against Strong AI’, *Synthese* 108, 391—419.

Papineau, D. (1995), ‘Arguments for Supervenience and Physical Realization’. In *Supervenience*,. Savellos, E. and Yalçin, Ü. (eds.), Cambridge University Press.

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