Do Walls Compute After All?

Challenging Copeland's Solution to Searle's Theorem against Strong AI

Matthias Scheutz (mscheutz@indiana.edu)
Cognitive Science/Computer Science, Indiana University
Bloomington, IN 47404

Abstract

In this is paper I will briefly describe Searle's criticism of "strong AI" (which extends to computationalism in general) and review Copeland's version of what he calls "Searle's Theorem", a claim made by Searle that "for any object there is some description of that object such that under that description the object is a digital computer". Copeland's own diagnosis and his solution to the paradox posed by Searle's Theorem will then be examined more closely. An analysis of Copeland's definition of what it means to implement a computation will yield a Searle-like counterexample of computing (extending an idea advanced by Putnam): under a certain interpretation walls will, after all, compute. A brief discussion and assessment of the consequences of my counterexample will—contrary to one's expectation—provide an optimistic outlook for computationalism.

Introduction

Computation and implementation are mutually dependent concepts upon which the fundaments are built of what John Searle coined "strong AI"—the view that "the mind is to the brain, as the program is to the computer hardware" (Searle[84], p. 28). Searle, being a firm opponent of strong AI, presented various rebuttals of this doctrine, the most famous of which is his heavily debated "Chinese room" thought experiment (see, e.g., Searle[80] or Searle[84]). The argument rests on the assumptions that (1) programs are formal (syntactical), (2) minds have content (semantic content), and (3) that syntax itself is neither identical with nor sufficient by itself for semantics. It follows from these premises that programs are neither sufficient for nor identical with minds, thereby refuting strong AI. As conclusive as this may sound at first glance, assuming that the reasoning is valid, it can still be doubted that all the premises are true. And, as it turns out, much of the truth of the first assumption depends on how one interprets the notion "program" (see, e.g., Melnyk[96]).[1]

In his book *The Rediscovery of Mind*, Searle augmented the above arguments by another substantial claim (which now affects computation beyond its role in strong AI): physics is not sufficient for syntax. In other words, the physical properties of a system do not determine the system's syntactic properties (Searle[92], p. 210). Syntax has to be assigned to a physical system, and this assignment is arbitrary, hence observer relative.[2] It follows that "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[92], p.207). Thus, whether or not a physical system is "running 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*

[1] It seems that if minds are on a par with programs, i.e., if the term "mind" can be legitimately compared to the term "program", then minds would be static, formal objects, too. Otherwise, the first premise should really read "processes are formal", but that does not seem right anymore (for the program/process distinction see, e.g., Smith[96], p. 33-34).

[2] This argument, eventually, leads to multiple realizability of computation, and has been attacked for that very reason (see, e.g., Endicott[96]).

the formal structure of Wordstar." (Searle[92], p. 208-209)

In short: the notional pair "computation/implementation" is not suitable for describing minds and their relations to the physical systems "causing" them. This has not only fatal consequences for strong AI, but for any view maintaining that minds can be described computationally; in particular, computationalism, cognitivism, and various forms of functionalism are at stake. It, therefore, does not come as a surprise that Searle's attack against main stream cognitive science created hefty reactions, most of which tried to find faults in his reasoning.

In this paper, I will focus one of the attacks, namely Copeland's theory of implementation (Copeland[96]), which is especially interesting, because it deviates significantly from the standard "physical state - abstract state correlation" view (see, Chalmers[96,97], Melnyk[96], Endicott[96], et. al.). The problem with the latter is, as has been pointed out by the founder of functionalism, Hillary Putnam, that such a correspondence can always be found, if one "chooses" physical states cleverly.[3] Although some disagree with Putnam's proof (see, e.g., Chalmers[96]), I think that their arguments fail to get at the heart of Putnam's construction, but I will not be able to argue this here.[4] Instead, I will show that even Copeland's definition of "computing a function" can be tricked by a Putnam-like argument: systems can be shown to compute, to which one would normally not attribute any computational capacity whatsoever. Although one could argue that this rehabilitates Searle's theorem, I instead conclude that an adequate notion of implementation, which meets Putnam and Searle's challenges, is still missing.

Copeland's notion of implementation

Amongst the defenders of the "traditional theory of computation" is Jack Copeland, who reformulated both of Searle's above-mentioned theses as a theorem, calling it "Searle's Theorem". Given an intuitive account of what it means to implement a program (in general, a computation), this theorem turns out to be true. However, the proof relies essentially on non-standard interpretations of theoretical terms that are thought to describe a computer architecture, as Copeland points out. Exploiting this weak spot, one can block unwanted conclusions such as "the wall behind my back is right now implementing the Wordstar program" by requiring that interpretations of algorithms and their corresponding architectures be "standard". In the following, I will review Copeland's definition of "computing a function" and state Searle's theorem in Copeland's terms (the gist of its proof will be described in the next section).

The driving force of Copeland's *What is Computation?* (Copeland[96]) is to clarify and define what is means for an entity *e* to compute a function *f*, which eventually boils down to the following question: given a formal specification SPEC of an architecture (e.g., the blueprint of a PC) together with a specific algorithm for that architecture (e.g., an addition program written in 486 assembly language) which takes arguments of *f* as inputs and delivers values of *f* as outputs. Furthermore, given an entity *e* ("real or conceptual, artifact or natural" Copeland[96], p. 336). How can we say/determine that/whether *e* is a machine of the kind described by SPEC on which could "run" (i.e., a machine which computes the function *f*)? Notice that this question really requires an ontological as well as an epistemological answer, even if the latter will depend on the former.

Copeland finds a solution to bridge this gap in the notion "labeling scheme for *e*", which is a way of assigning labels (to parts of *e*) "that constitute a 'code' such that spatial or temporal sequences of labels have semantical interpretations" (Copeland[96], p. 338). Obviously, labels must be assigned for at least each of the constants in SPEC denoting parts of the architecture. A labeling scheme, then, consists of: (1) the designation of certain parts of the entity as label-bearers, and (2) the method for specifying the label borne by each label-bearing part at any given time (Copeland[96], p. 338).[5] Given a labeling scheme (which provides physical counterparts of *e* for formal objects in SPEC), one could attempt a truth-definition relating SPEC and *e* such that it

[3] In the appendix to his book *Representation and Reality* Putnam proved the following theorem: *every ordinary open system is a realization of every abstract finite automaton* ([Putnam88]), which I will use later to construct a counterexample to Copeland's version of Searle's theorem.

[4] I present a detailed analysis of all the pros and cons of Putnam's construction as well as a diagnosis why no "physical state - abstract state correlation" view is tenable in Scheutz[97].

[5] Note that nothing is said about the nature of the relation between labels and parts of *e.* It seems to me that one should at least require a functional correspondence.

is meaningful to ask if SPEC is true of *e*. Note that this truth-definition crucially depends on the vocabulary used for the formal specification SPEC. Copeland downplays the importance of the particular language of SPEC with the cryptic remark "For definiteness, let SPEC take the form of a set of axioms, although nothing in what follows turns on the use of the axiomatic method as opposed to some other style of formalisation" (Copeland[96], p. 337-338). However, his critique of Searle's Theorem assumes that a connective like "ACTION-IS" (which has to be interpreted in a particular way) be an essential part of SPEC. It seems to me that a serious truth-definition can only be provided if the language at hand is clearly defined and all necessary predicates and connectives (such as "ACTION-IS") together with their interpretation are completely determined, neither of which has been done in Copeland's paper. So, I will, too, pretend that the specifics of SPEC do not matter, and that a truth-definition can be provided once the language of SPEC has been fixed.

Assuming a truth-definition relating expressions ("formal axioms") of the language of SPEC using a labeling scheme *L* and an entity *e*, the notion "model of SPEC" can be defined:

Definition 1: Let SPEC be a Given a formal specification, *L* be a labeling scheme, and e be an entity. Then the pair <*e*,*L*> is *a model of* SPEC iff the formal axioms of SPEC are true of *e* under *L*.

Using the above definition, Copeland can define what it means for an entity to compute a function and state a precise version of Searle's Theorem (Copeland[96], p. 338-339):[6]

Definition 2: An entity *e* is computing function *f* iff there exists a labeling scheme *L* and a formal specifications SPEC (of an architecture and an algorithm specific to that architecture, which takes arguments of *f* as inputs and delivers values of *f* as outputs) such that <*e*,*L*> is a model of SPEC.[7]

Theorem 1: (Searle's Theorem) For any entity *e* (with a sufficiently large number of discriminable parts) and for any architecture-algorithm specification SPEC there exists a labeling scheme *L* such that <*e*,*L*> is a model of SPEC.

In more intuitive terms, this theorem states that every object can be "interpreted" as computing any function that any given computational architecture could specify. Obviously, something must be wrong with this result (otherwise computer dealers would be selling walls), the real challenge, however, is to find and explicate its defect.

Copeland's Analysis of Searle's Theorem

> *"[…]to compute is to execute an algorithm. More precisely, to say that a device or organ computes is to say that there exists a modelling relationship of a certain kind between it and a formal specification of an algorithm and supporting architecture. The key issue is to delimit the phrase 'of a certain kind'".* (Copeland[96], p. 335)

Since Copeland's notion of implementation depend essentially on a semantic interpretation of the relationship between a formal specification (i.e., the description of an architecture) and a physical object, one has to insure—as he points out in the quoted passage—that "unwanted interpretations" are excluded; the question is only *how*? To exclude them by *fiat* is certainly not a viable option, so a criterion has to be established to distinguish "benevolent" from "malicious" interpretations; otherwise "walls implement every computation". And this criterion can be found in the construction of the proof of Searle's Theorem (for details, see Copeland[96], p. 343-346).

Given a formal specification VNC (of a von Neumann computer, say), one would like to define a labeling scheme *J* such that a particular wall (the one right behind me, for example) under *J* is a model of VNC. To do this, one singles out parts of the wall that are supposed to correspond to "registers" in VNC, call them "wall states". Then one records the states of all registers in an actual von Neumann computer while it is running Wordstar for *n* computational steps and relates these to wall states. That is, for any two consecutive computational steps there will be two consecutive intervals of real-time such that the content of a particular register corresponds to a particular wall state (during the respective interval). It is easy to check that all formal states will correspond to wall states at any time during the interval [*t*,*t'*] (*t*<*t'*) under consideration. Furthermore, the axioms of VNC that

[6] There are other approaches that also use labeling schemes, but do not need a semantic interpretation to define a notion of computation, e.g., see Gandy[80].

[7] As a consequence of Searle's theorem, Copeland later requires in addition that the model be "honest".

describe "state transitions" (using the connective "ACTION-IS") will be mirrored by "wall transitions" under a certain interpretation of "transition". Hence, the wall will implement Wordstar during [*t*,*t*'].

There are obviously quite a few problems associated with this construction, and Copeland himself diagnoses three major shortcomings: (1) all computational activity occurred *outside* of the wall (by recording the activity within a CPU of a machine that *actually* performed the computation), meaning that the labeling scheme is constructed (from this record) *ex post facto*. (2) the labeling scheme involves unwanted temporal specificity (by limiting the wall's computational capacities to the time interval [*t*,*t*'], a necessary consequence of the *ex post facto* nature of the labeling scheme). And, finally, (3) the interpretation of "ACTION-IS" fails to support assertions about the counterfactual behavior of a real von Neumann computer.

All three problems point to the discrepancy between the "intended" interpretation of VNC and the "artificial" one that turned the wall into a von Neumann computer. Copeland, therefore, suggests that this is already the criterion we have been looking for: the definition of an entity computing a function has to be restricted to "honest" models, that is, to models that do not use non-standard interpretations of expressions in SPEC.

A Brief Analysis of Copeland's Objections

Let me not call into question Copeland's controversial assumption that computation depends on the right kind of interpretation of an object, which to some extent results from his attempts to subsume not only physical objects, but also conceptual ones under the category "computer". What still strikes me as a serious mistake, one that both Searle and Copeland's approach share, is to ignore the difference between "*f* can be implemented on *e*" and "*f* is running on *e*". The wording of definition 2 ("is computing") suggests that Copeland wanted to capture the notion "process" (that what is actually running on a computer), whereas the existential quantifiers in the *definiens* hint at the notion "program" instead of "process", at the potentiality of the entity to run a particular program.[8] A correct reading of definition 2 is crucial to Copeland's first and second objection, as they are at best objections under the "program" reading. Another, already mentioned criticism, that weakens Copeland's third objection, is the lack of a clear definition of what the minimal requirements of a potential language for SPEC are (it seems that, for example, the connective "ACTION-IS" or something equivalent should qualify). If formal state transitions are to be modelled using counterfactual supporting connectives, then this opens the door to all kinds of criticism regarding the nature and legitimacy of counterfactuals, a debate that in my opinion should not be part of a theory of implementation.

Copeland attempted to define very general notions of computation and implementation (reducing implementation to a logical modelling relation and computation to the presence of this relation between a formal specification and a description of an entity) that view all different kinds of systems as computers, and rightfully so: von Neumann computers, neural networks, Turing machines, or finite state automata, just to name a few. Unfortunately, this generality increases the difficulty to eliminate non-standard interpretations. Copeland mentions two necessary (but not necessarily sufficient) criteria that facilitate the assessment of an interpretation's nature:

> *"I suggest two necessary conditions for honesty. First, the labelling scheme must not be ex post facto. […] Second, the interpretation associated with the model must secure the truth of appropriate counterfactuals concerning the machine's behavior. Either of these two requirements suffices to debunk […] alleged problem cases."* (Copeland[96], p. 350)

However, as I will show in the next section, these criteria are not sufficient.

A Wall that Computes

Given Copeland's definition of computing a function (together with all other involved notions), my goal is to show that almost every system implements a finite state automaton. I will present the argument first (which—as already mentioned—extends an idea by Putnam[88], and then argue that it meets both criteria for "honesty". It follows that additional criteria are needed to single out "intended interpretations" (if this is possible at all).

[8] Maybe one has to introduce the further distinction "*f* can be implemented on *e*" versus "*f* is implemented on *e*" to distinguish between architectures cum algorithmic description from architectures without.

Theorem 2: Every ordinary open system *e* is a model of every finite state automaton.[9]

Proof: Let us start by defining the set of formal specifications SPEC. It is standard to define a finite state automaton (FSA) formally by a quintuple $<Q,,,q_0,F>$, where Q is the set of states, the input alphabet, the "transition function" from states and inputs to states, q_0 the start state, and F the set of final states.[10] All triples of can be viewed as instances of the axiomatic scheme $<q,i>q'$, where q and q' are states, i is an input, and '' is a primitive meaning "transits". This takes care of the "architecture part" of SPEC. The state table, as exhibited by , defines for each state in Q all possible transitions to other states depending on the current state and the current input (transitions can be made without reading input, too, so-called -transitions, or more transitions can be defined for the same input and state—in that case the machine is called "non-deterministic"—however, I will restrict myself to deterministic machines without -transitions). Starting in the single start state q_0, the automaton changes states according to its inputs and state table entries until it either reaches a final state (in which case it is said to "accept the input") or it ends up in the "trap" state (a state, from which it cannot make any other transition than remaining it this state for every possible input). Notice that determines what the actual state transitions are in the FSA. These particular transitions can be viewed as the algorithm "implemented" on a more "generic automaton" (i.e., the given FSA without a particular).[11]

Define a mapping f, then, from * into Q such that $f(w)=q$ if the FSA is in state q after having read w, for all strings w in * (this mapping can be obtained inductively from). Obviously, the FSA takes arguments of f as inputs and delivers values of f as outputs (in the sense that it ends up in the state, which is the output of the function). Hence, the second part of the requirements for SPEC is satisfied, too.

Now we need to exhibit a labeling scheme L (see definition 1) and an interpretation of '' such that SPEC is true under that scheme for every ordinary open system. 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 the FSA receives input from the "outside", I will treat inputs from now on (as far as the "model of the FSA" is concerned) as "input states" and call all other states "inner states". Since the only parts about the automaton specified in SPEC are its states (input and inner), we designate the boundary of *e* and its "inner" part 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 $[t,t']$ (within which the system will compute the function f) and let the boundary of *e* be the "input region". Note that the environmental conditions on the boundary throughout $[t,t']$ specify the input that *e* will receive. By the Principle of Noncyclical Behavior, "the state of the boundary of such a system is not the same at two different times" (see [Putnam88], p. 121). We need to define "physical states" for *e* and the boundary region of *e*, which can be related to the abstract states in the automaton. 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 *e*, respectively, for a given interval of real-time. To make them correspond to automata states in the right way (i.e., if $L(I)=i$, $L(S)=q$, and $(i,q)=q'$, then $L(S')=q'$ for the state S' following S), we define the labeling L from physical states onto abstract states inductively:
start by defining inductively an infinite sequence of consecutive intervals $T_0, T_1, T_2, \ldots$, where T_0 is $[t,)$ and T_k is the open interval [) of real-time (for $k>0$). Consider the interior of *e* during the interval T_0, call it S_0, and map it onto the start state q_0. Then, for every interval state S_k (defined by the interior of *e* during the interval T_k) corresponding to some q of the FSA do the following: first, define I_k to be the interval state of the boundary of *e* during the interval T_k. Let I_k correspond to the input state i of the FSA after k steps (*). If the FSA, being in state q reading input i, transits into state q', define the "successor state" S_{k+1} to be the interval state of

[9] For details about "ordinary open systems" see Putnam[88].

[10] Sometimes, if the automaton is also required to produce output, another component, the output alphabet , is added and is defined correspondingly as a function from states and inputs to states and outputs.

[11] One could also exclude F, since in a way final states will depend on all possible transitions. However, one can always take another "generic automaton" with a desired set F' different from F, if needs more or fewer final states.

the interior of e during the interval T_{k+1}. Note that e will always finish its "computation" of f within the interval $[t,t']$ independent of the length of the input.[12] This takes care of the second part of L.

To see that e is a model under the given labeling scheme L for SPEC we need to find an interpretation of '' under which every transition in the automaton modelled by $<q,i>(q')$ is true in e, in other words, $((q,i)(q'))]]=true$ in e. This amounts to showing that for all triples in :$(L(q),L(i))$ $(L(q'))$. Take to mean "causes" (or "follows nomologically" by the laws of physics, i.e., field theory given environmental conditions). (**)

If e is in state $L(q)$ and the input to e is $L(i)$, i.e., during the interval T_k the physical make-up of e 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[88], p. 122-123, for details) to determine that e will be in state $L(q')$ at the beginning of T_{k+1}. Given the boundary conditions during T_{k+1} (which correspond to the "next" input state and, thus, have to be provided), it can be determined that e will stay in state $L(q')$ throughout T_{k+1}. This concludes the proof that L and e are a model for SPEC under . Hence, we have shown that for every open system e and every finite state automaton described by SPEC a labeling scheme L can be found such that $<e,L>$ is a model of SPEC.

So far, it seems that yet another instance of Searle's Theorem has been proved (where the architecture-algorithm specifications are FSAs). To make this result interesting, it needs to be shown for "honest models", i.e., for the "intended" interpretation. In the above case, however, this seems rather problematic, since it is not clear at all what *the* standard model of a FSA is supposed to be. A FSA specifies a very general "architecture", a system with an input device (without further details as to the nature and structure of this system *cum* device) both of which can be in different states. So most objects, using a little imagination, are potential FSAs and in a way this is exactly what the theorem above shows. In fact, I would claim that every physical system that consists of different states and exhibits transitions between these states depending on some input to the system would count as a standard model. This implies that the above result could be strengthened to "honest model", if in addition Copeland's two criteria are met.

In the proof, I have marked two steps that are candidates for the application of Copeland's criteria. In step (*) reference is made to an input state which is undetermined, which essentially depends on the particular input that is presented to the FSA (i.e., the n-th input character). This is, of course, the trick that makes it possible to map the accidental boundary conditions of e at "run-time" to the n-th input character. Obviously, there is no systematic relation between inputs and boundary states. Note, however, that the reference to this character is uniquely fixed, not *ex post facto*, but for all possible inputs! So the first criterion does not apply.

Step (**) marks the second potential point of conflict, the interpretation of '' (the equivalent of Copeland's "ACTION-IS") as "causes". I think that Copeland would agree that all state transitions of e are caused, since they were just so defined (see Copeland[96], p. 353). The question is whether counterfactuals are supported, i.e., whether if the input had been $L(i)$ and the current state $L(q)$ at any time t, e would have changed to state $L(q')$ (according to the state table). I am not sure if counterfactual support can actually be 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). As far as the above construction is concerned, it does not really make sense to ask for counterfactual support, since L is only defined for a particular input string. Hence, if $L(i)$ and $L(q)$ were given, e would have changed to state $L(q')$ by definition of L. If we asked, on the other hand, whether e in state S with input I had changed to the successor state S' (for interval states S, S', I) without supplying L, then it is not even clear what this question means (despite the fact that, given the boundary conditions I and the state of the system S, its next state S' will be "caused"). So, one could say that the counterfactual requirement is vacuously satisfied.

[12] This means that the FSA could also "compute" infinite strings in a finite amount of time (actually, in an arbitrarily small time interval). Furthermore, the above construction even allows for "infinite automata" (that is, machines with a countably infinite set of state), see Scheutz[97] for more detail.

At this place I would like to mention a different kind of objection to the above theorem. One could claim that *e* does not compute very interesting functions, because the functions it computes are (too closely) dependent on the structure of the FSA. But *e* can really compute any function from strings to strings that every deterministic finite state transducer can compute. Recall that *e* computes a function *f* from * into *Q* such that *f*(*w*)=*q* (*per definitionem* in the above proof). One can easily turn this into a more interesting function by dividing *w* into an input and an output part separated by '#' (e.g., *w*=*x*#*y*). The FSA is then said to compute a function *g* from strings to strings such that *g*(*x*)=*y* iff *f*(*x*#*y*)=*q* for some final state *q*.

Conclusion

I have tried to show that even promising definitions, such as Copeland's, of what it means to "implement a computation" seem not precise enough to capture only intended cases, given the above result: walls, after all, compute. They might not be "honest" models of specifications of architectures such as "formal specifications of von Neumann machines", but they seem to be honest models of finite state automata, a common, often used computational model. The conclusion to be drawn from this example is not that Searle's theorem has been revived. Rather further evidence is provided—and that is, what I take Searle and Putnam's objections to argue for in the first place—for the need of a (better) theory of implementation.

References

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

Chalmers, D. J. (1997) "A computational Foundation for the Study of Cognition". (unpublished manuscript)

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

Endicott, R. P. (1996) "Searle, Syntax, and Observer Relativity", *Canadian Journal of Philosophy* v26, 101-122.

Gandy, R. (1980) "Church's Thesis and Principles for Mechanism". Proceedings of the Kleene Symposium (J. Barwise, H. J. Keisler and K. Kunen, eds.). New York: North-Holland Publishing Company.

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. (1997) "Facets of implementation". (unpublished manuscript)

Searle, J (1980) "Minds, Brains and Programs", *The Behavioral and Brain Sciences* 3, 417-424.

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.