This is a pre-print of an article published in *Synthese*. The final authenticated (and significant reedited) version is available online at: https://doi.org/10.1007/s11229-019-02275-w

Does the Solar System Compute the Laws of Motion?

Douglas Ian Campbell & Yi Yang

The counterfactual account of physical computation is simple and, for the most part, very attractive. However, it is usually thought to trivialize the notion of physical computation insofar as it implies 'limited pancomputationalism', this being the doctrine that *every* deterministic physical system computes *some* function. Should we bite the bullet and accept limited pancomputationalism, or reject the counterfactual account as untenable? Jack Copeland would have us do neither of the above. He attempts to thread a path between the two horns of the dilemma by buttressing the counterfactual account with extra conditions intended to block certain classes of deterministic physical systems from qualifying as physical computers. His theory is called the 'algorithm execution account'. Here we show that the algorithm execution account entails limited pancomputationalism, despite Copeland's argument to the contrary. We suggest, partly on this basis, that the counterfactual account should be accepted as it stands, pancomputationalist warts and all.

Keywords. Counterfactual Account, Algorithm Execution Account, Physical Computation, Pancomputationalism, Cellular Automata

1. Introduction

The computational theory of mind (CTM) implies that *mentation* is a type of *computation*; that a person has the intentional and/or phenomenal mental states she has because of what her brain computes, while a rock lacks such mental states altogether because of what it does not compute (see, e.g., Fodor, 1975; Mcculloch & Pitts, 1943; Newell, 1980; Putnam,

1967; Rescorla, 2017).[1] But what is it for a brain to implement a given computation, or for a rock not to implement the selfsame computation? The answer to this question would take the form of *a theory of physical computation*—a theory that tells us how to 'read off' an object's computational properties, if any, from its physical properties; that lays out the necessary and sufficient conditions for a physical system to implement a given abstract computational system, or to compute a given logical or mathematical function. Developing and defending such a theory is of the first importance if the CTM is to be made viable.

One very elegant and *prima facie* plausible theory of physical computation is the so-called 'counterfactual account' (Piccinini, 2015, pp. 19–22, 2017). Let S be some physical system. Let I be a set of alternative possible *input states* that S can potentially be thrust into by an exogenous information source. Let O be a set of alternative possible *output states* that S could potentially evolve into. Then, under the counterfactual account, S is said to compute the function, f, iff there is some interpretative mapping function, M, under which, for any possible input state, $i \in I$, that S might be put into, if S were put into i then S's internal dynamics are such that it would reliably evolve into the output state, $M^{-1}(f(M(i))) \in O$. Here $M(i)$ maps the physical input state, i, onto a corresponding abstract input to f, while M^{-1}—M's inverse—maps f's abstract output back onto a corresponding physical output state of S.

The counterfactual account is vulnerable to an objection. It *trivializes* the notion of physical computation insofar as it implies what is called 'limited pancomputationalism', the doctrine that *every* deterministic physical system computes *some* (if not *every*) function (Piccinini, 2017).[2] To see why the counterfactual account implies this result, consider an example: the

[1] The CTM is a broad church which leaves room for different views as regards which particular types of mental states—intentional, or phenomenal, or both—are computational. These divisions within the CTM will not concern us here.

[2] Limited pancomputationalism is to be contrasted with *unlimited pancomputationalism*, which says that every physical system computes *every* function (perhaps modulo certain complexity constraints) (Piccinini, 2017). Arguments due to Putnam (1988) and Searle (1990) show that the so-called 'simple mapping account' (Godfrey-Smith, 2009) of physical computation implies unlimited pancomputationalism. The counterfactual account greatly improves on the simple mapping account in being invulnerable to Putnam's and Searle's arguments—see

solar system. The state of the solar system, as comprised by the locations, velocities, and masses of the sun and all its associated planets, moons, asteroids and comets, evolves in accordance with certain physical equations of motion. Does the solar system thereby compute these selfsame equations? The counterfactual account implies that it does. To see why, let I denote the set of possible physical states of the solar system at some point in time, *t0*. Let O denote the possible physical states of the solar system at some later point in time, *t1*. Let f denote the laws of motion. Finally, let M denote the interpretive mapping function which pairs the values of the various variables in these laws with the corresponding physical properties of the various bodies in the solar system. Then if the solar system were put into state $i \in I$ at *t0* (imagine God's hand reaching into the solar system and positioning the planets and moons appropriately), then it would, in consequence of its obeying the laws of motion, reliably evolve into the state $M^{-1}(f(M(i))) \in O$ at *t1*.[3] Hence the solar system computes function f by the lights of the counterfactual account. Q.E.D.

The point generalizes (Shagrir, 2006, p. 398). The counterfactual account implies that any physical system that obeys deterministic physical laws thereby 'computes' the functions implicit in these laws.[4]

Let's call this threat to the counterfactual theory of computation the 'threat from limited pancomputationalism'. One obvious possible response to the threat from limited pancomputationalism is to bite the bullet by accepting limited pancomputationalism. Some proponents of the counterfactual account have done this (Chalmers, 1996b, pp. 319–320, 1996a, p. 331; Putnam, 1967; Scheutz, 1999, p. 191). It is an approach we are ourselves

(Chalmers, 1996a), (Fresco, 2014, pp. 86–94) and (Piccinini, 2015, pp. 16–22). Putnam's and Searle's arguments for unlimited pancomputationalism will not concern us here.

[3] The solar system is too big for us puny human beings to manipulate its parameters. Hence the necessity of bringing 'God's hand' into the discussion in order to access the relevant counterfactuals. But, by the same token, a laptop computer would have the counterfactual dispositions it has even if it was on a planet whose only intelligent inhabitants were tiny ants too small and weak to press its keys.

[4] This trivialization result doesn't extend to *indeterministic* physical systems for the simple reason that the inputs to such systems underdetermine their outputs.

strongly sympathetic to. The various arguments typically offered against biting the bullet are, in our view, unpersuasive.

One such argument is that there is nothing in the counterfactual account 'that distinguishes computational explanation from ordinary causal or dispositional explanation' (Piccinini, 2015, p. 23). This argument should, we think, be met with a shrug. So what? Yes, the counterfactual account assimilates computational explanation under the broader umbrella of causal and dispositional explanation. For example, it implies that we would, by explaining why a person has the mental states she does in terms of what her brain computes, thereby ultimately be explaining her mental states in terms of the causal and dispositional properties of her brain-states. But what of it? This implication is one that many proponents of the CTM, ourselves included, embrace with open arms. (Chalmers, for example, writes that 'when computational descriptions are applied to physical systems, they effectively provide a formal description of the system's causal organization. The language of computation provides a perfect language in which this sort of abstract causal organization can be specified.' (1996b, p. 320). He could hardly be more explicit. Speaking of physical computation is a kind of shorthand for speaking of causal organization.)

A second argument against biting the bullet is that insofar as the counterfactual account implies limited pancomputationalism it defines 'physical computation' in a way that contradicts ordinary usage (Piccinini, 2015, pp. 12–13). There is a relatively clear-cut intuitive distinction between physical systems that do compute—things like desktop computers, iPhones and pocket calculators—and things that don't compute—like rocks, puddles of water, and planetary systems (Fresco, 2014, p. 42; Piccinini, 2007, p. 504; Shagrir, 2006). The counterfactual account is heedless of this intuitive distinction, and so it provides a radically defective analysis of our concept of physical computation. Or so goes the argument.

Again, however, this argument seems to us to badly miss the mark. Two projects need distinguishing: first, the project of analyzing the 'ordinary' concept of computation—of characterizing the conditions it is necessary and sufficient for a physical system to satisfy if it is to 'compute' (or be a 'computer') according to the everyday use of the word; and second, the project of analyzing the technical notion of computation that is in play within the CTM—the species of 'computation' that mentation supervenes upon if the CTM is true. The counterfactual account is, admittedly, flatly inadequate to the needs of the first project. But it nowise follows from this that it is also inadequate to the needs of the second project, and at least where the philosophy of mind (and our interests in this paper) is concerned, it is the second project that matters.

However, our benevolent attitude to the threat of limited pancomputationalism is not widely shared. Much ingenuity has gone into developing alternatives to the counterfactual account of physical computation that avoid its pancomputationalist implications. One such rival theory is Oron Shagrir's (2006) 'semantic account' of physical computation, which requires a physical system to be interpreted as computing a function by an intentional agent in order for it to count as genuinely computing. Another major rival to the counterfactual account is the 'mechanistic account', popularized by Gualtiero Piccinini (2007, 2015), which implies that only physical systems having associated teleological functions compute. A third rival theory has been developed by Jack Copeland (1996). Copeland's theory has been dubbed the 'Algorithm Execution Account' (AXA) by Nir Fresco (2014, p. 176). AXA tightens up the counterfactual account by adding extra necessary conditions to it. These extra conditions are intended to block certain types of physical systems from being classified as computers—hence avoiding limited pancomputationalism—but while still permitting entities that are stereotypically regarded as 'computing' to be so-classified.

Shagrir's semantic account is a poor fit with the CTM, since it presupposes mentation, in the form of the intentional states of the interpreting agent, in the course of defining physical computation, creating a vicious circle (Piccinini, 2004, p. 377; Shagrir, 2006, p. 413). We will say no more about the semantic account here, for this reason.

Piccinini's mechanistic account depends on the assumption that an adequate naturalistic theory of objectively existing teleological functions can be developed, since it is only if it is paired with such a theory that it can yield a verdict as to which physical systems compute and which don't. This assumption is, to put it mildly, a heroic one, given the longstanding, formidable difficulty of developing such a theory (Casini, 2017; Cummins, 2002; Davies, 2000; Dennett, 1978; Garson, 2017; Millikan, 1989; Peters, 2014; Searle, 1995, p. 15; Tolly, forthcoming).[5] Copeland's AXA, in contrast, makes no such bold assumptions. Copeland claims we need look no further than Turing's original description of the Turing machine (Turing 1936) in order to find a way of modifying the counterfactual account to obtain a theory that is immune from the threat of limited pancomputationalism (1996, p. 339). His amendments to the counterfactual account are modest, insofar as they draw only on notions that are part of classical computing theory (in particular, the notion of an *architecture* and of an *algorithm*).

The situation, then, is this. We believe the counterfactual account should be accepted 'as is', pancomputationalist warts and all; that advocates of the CTM should 'learn to stop worrying and love limited pancomputationalism', so to speak. First and foremost among the threats to our position is Copeland's AXA, which appears to avoid limited pancomputationalism very cheaply, by making only modest tweaks to the counterfactual account. Why 'learn to love' limited pancomputationalism if it can be avoided at such little cost? Our aim in the remainder of this paper is to dispense with this threat to our position. To

[5] Piccinini is alive to these difficulties and has developed a theory of teleological functions to address them (Piccinini, 2015, pp. 100–117). We are dubious of his theory's adequacy, but explaining why is a task for another occasion.

do this, we will show that AXA fails by its own lights, insofar as it entails limited pancomputationalism despite Copeland's protestations to the contrary. Our conclusion will be that AXA should be rejected. It should be rejected, not, of course, because it implies limited pancomputationalism, but rather because if one is going to accept a theory that implies limited pancomputationalism then one is better off accepting the counterfactual account, it being simpler.

The paper is organized as follows. In §2 we outline AXA. In §3—5 we show that it entails limited pancomputationalism, Copeland's denials notwithstanding. In §6 we anticipate possible replies. §7 wraps things up.

Before we start, a quick note on terminology. Copeland frames AXA in terms of what it takes for a physical system to *compute a certain function, f,* from input to outputs. His terminology is non-standard, with common practice, at least nowadays (Fresco, 2014; Piccinini, 2015, 2017), instead being frame theories of physical computation in terms of what it takes for a physical system to *implement an abstract computational system*. A computational system computes a function from inputs to outputs, but in addition to having input states and output states, it will typically have many internal states too (the role of which is, of course, to help mediate how inputs causally determine outputs). Framing a theory of physical computation in terms of the implementation of computational systems, as opposed to the computation of functions, has the great advantage of highlighting the importance of internal states, not just of input states and output states. But Copeland uses the terminology he uses, and so we will use it too, the only other option being to translate back and forth between two terminologies, which would make life difficult both for ourselves and for the reader. Thus it is that we will speak of physical systems 'computing functions', not of their 'implementing computational systems', in what follows.

2. AXA[6]

In framing AXA, Copeland draws his inspiration directly from Alan Turing. Turing famously characterized computation in terms of the operation of a Turing machine. The following four key aspects of a Turing machine can be distinguished:

1. *An architecture,* consisting of a 'dial' that can adopt any one of a certain set of *states*, and of a read/write head that traverses an infinite, bi-directional *tape* that is divided into *cells* in which *symbols* from a finite *alphabet* are inscribed. The cell over which the head is positioned during a given time-step is called the 'scanned cell', and the symbol in this cell is called the 'scanned symbol'.[7]
2. *An algorithm*, which takes the form of a *machine table*, constituted of a set of *instructions*, each of which specifies, for some given combination of a scanned symbol and a current dial state, which symbol the head should write in the scanned cell (erasing and overwriting its previous contents), where the head should be positioned during the next time-step (one cell to the left or one cell to the right of the cell currently being scanned), and which dial state it should be in during the next time-step.
3. *An input,* consisting of the sequence of symbols written in the tape's cells at step zero.
4. *An output,* consisting of the sequence of symbols written in the tape's cells if and when its dial eventually goes into a designated 'halting state', or of some portion of this sequence.

Copeland's suggestion is that these four fundamental ingredients of a Turing machine—its *architecture*, *algorithm*, *input* and *output*—are vital and indispensable elements *of computation*

[6] This description of AXA is necessarily condensed. For lengthier expositions, see Frisco (2014, pp. 176–179), (Scheutz, 1998) and, of course, Copeland (1996).
[7] Here we use the terminology of (Copeland, 2017).

in general. He holds that they are conjointly both necessary and sufficient for computation to be occurring. Of course, Copeland acknowledges that there are numerous classes of computing devices that have architectures dissimilar to a Turing machine's, that have algorithms that do not consist of a machine table, and that don't accept input and write output in the form of strings of symbols inscribed on a tape. But on Copeland's view, a device must have *some* architecture, *some* algorithm, and *some* way of accepting inputs and producing outputs if it is to truly compute. This applies to both *abstract* and to *physical* computing devices. Where abstract devices are concerned (Turing machines, Markov algorithms, register machines, finite state automata, cellular automata, and so forth), Copeland holds that no such device is genuinely computing unless it has some abstractly specified architecture, algorithm, input and output. And where physical devices are concerned, Copeland holds that no such device computes unless it is *isomorphic* with some abstract computing device.

These are the central tenets of AXA. Like the counterfactual account, AXA entails that a physical system must reliably convert input states into output states in accordance with some function in order to compute. But it deviates from the counterfactual account by imposing the additional requirement *that the system do this in a way that realizes some architecture and algorithm*.

Copeland frames AXA using the following terminology:

First, there is a thing he calls a 'SPEC'. This is simply an exhaustively detailed description of an abstract computing device's architecture and its algorithm. For example, the SPEC for a Turing machine would include a description both of the machine's general architecture, as constituted by its head, dial, tape, and machine table, and of the machine's algorithm, as constituted by the particular set of state transition rules in its machine table. Such a SPEC would contain all the information one would need in order to work out how the state of the

Turing machine would evolve, and what output it would produce, in response to its being provided with any particular input.

Second, as under the counterfactual account there is an interpretative mapping function, which Copeland calls a 'labelling scheme'. A labelling scheme, *L*, consists of a way of interpreting the states adopted by a physical entity, *e* in terms of the states adopted by the abstract computing device described by SPEC. In Copeland's words, it 'consists of two parts: (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, 1996, p. 338). The 'labels' borne by each state of the physical entity pair it with a corresponding state of the abstract computing device.

Third, Copeland says that the ordered pair, ⟨*e*, *L*⟩ is a 'model' of SPEC iff the labelling scheme, *L*, successfully maps *e*'s states onto the states of the abstract machine as described by SPEC.

Fourth, Copeland distinguishes between two kinds of models—*honest* and *nonstandard.* ⟨*e*, *L*⟩ is an 'honest model' of SPEC provided it meets two conditions. First, the labelling scheme, *L*, 'must not be *ex post facto*' (p. 350), which is to say, it must not have been 'cooked up' or gerrymandered after the fact so as to give the right results. In other words, it must be possible ahead of time, before *e* operates, to specify *L*, and to know that *e*'s states will be successfully mapped by *L* onto the abstract machine's states. Second, 'the interpretation associated with the model must secure the truth of appropriate counterfactuals concerning the machine's behaviour' (pp. 350-351). In other words, the labelling scheme should successfully map *e* onto the abstract device regardless of which inputs are provided to *e*.

A 'nonstandard model' is any model that fails to satisfy either of these two conditions.

Having introduced these terms, Copeland formulates AXA as follows:

> "Entity *e* is computing function *f* if and only if there exist a labelling scheme *L* and a formal specification SPEC (of an architecture and an algorithm specific to the architecture that takes arguments of *f* as inputs and delivers values of *f* as outputs) such that ⟨*e*, *L*⟩ is an honest model of SPEC." (p. 348)

Why, on Copeland's view, does AXA dispense with the threat of limited pancomputationalism? To help see why, let's return to the example of the solar system. As explained above, the solar system can be regarded as having an input, identified with its state at some time, *t0* (including the positions, velocities, and masses of all its moving bodies). Its output state can be identified with its state at some subsequent time, *t1*. But does the solar system also have an architecture and an algorithm? Copeland argues that it doesn't. He considers a possible algorithm for computing the laws of motion that 'requires a supporting architecture that makes available the operation of shifting the bits in a register one place to the right' (p. 339). Is the solar system a computing machine with this particular architecture and algorithm? He writes:

> To answer 'yes' is to suppose that the solar system is a register machine—is to suppose that the solar system consists of an interconnected structure of binary registers that are responsive to the shift operation and the various other operations demanded by the algorithm, for example binary addition and logical conjunction. Such a supposition no doubt strikes you as ludicrous. Perhaps those who seriously entertain the thought that the solar system computes will respond that it was never an algorithm like this one that they had in mind. Well and good. The foregoing account of computation presents them with a challenge: if they want to persist in the claim that the solar system is computing the function *f* then they must describe for us the solar system's computational architecture and detail the algorithm by which the solar system arrives at values of *f*. (ibid, p. 339)

Here Copeland lays down a challenge to his opponent which he believes cannot be met. We think it can be met, and in the next few sections we will explain how.

3. Elementary Cellular Automata, and Modified Elementary Cellular Automata

Copeland is surely correct in saying that the solar system doesn't have the architecture of a register machine or run an algorithm composed of operations (such as the 'shift' operation) supported by such an architecture. Nor does the solar system have the architecture of, say, a Turing machine. (Is there, in the solar system, any analogue of a Turing machine's head, with its capacity to read, write, and move along a tape? The answer, it appears to us, as it appears to Copeland, is pretty clearly 'no'.) However, it doesn't follow that there is no possible architecture, *A*, and algorithm, *G*, such that the solar system computes a function *f* using *A* and *G*. Register machines and Turing machines have relatively complex and richly featured architectures. Our strategy in what follows will be to focus instead on abstract machines having much simpler and less richly featured architectures—with the guiding thought being that this will greatly ease the task of showing that the solar system is isomorphic with the abstract machines in question. If an abstract machine doesn't come with a lot of bells and whistles, then we won't need to find analogues, in the solar system, of all these bells and whistles.

The abstract machines we will work with are a form of one-dimensional cellular automata that we will call 'modified elementary cellular automata' (MECA). As the name suggests, a MECA is a modified version of an 'elementary cellular automata' (ECA).[8] In this section, we will begin by describing what an ECA is, and then go on to describe the modifications that turn an ECA into a MECA.

An ECA consists of a finite sequence of *n* cells, arranged in a 'wraparound loop', such that for any *x* in the range 1…*n*, cell *x*+1 is 'to the right' of cell *x*, and cell *x*-1 is 'to the left' of cell *x*, but with the exception that cell 1 is 'to the right' of cell *n* and cell *n* is 'to the left' of cell 1.

[8] See (Wolfram, 2002) for a compendious discussion of ECAs.

An ECA's computation proceeds through a series of steps. At each step, each cell is in one of two states—either 0 or 1. The states of all the cells at step 0 (the 'starting step') comprise the ECA's *input*. There are countless different ways of defining an ECA's output, and for present purposes it doesn't matter which is chosen. For example, we might identify an ECA's output with the states of all its cells after (say) one thousand steps. Or we might divide the ECA's cells into a set of 'output-flagging' cells and a set of 'output-encoding' cells, and then identify the ECA's output with the sequence of 0s and 1s contained in its output-encoding cells during the first step when its output-flagging cells have entered a designated 'output ready' state.

The states of an ECA's cells at any step $s>0$ are determined by the states of the cells at step $s-1$ and by a set of *state transition rules*. The state transition rules are of the form $xyz\rightarrow u$, where each of x, y, z and u are either '0' or '1', and where such a rule is to be interpreted as saying that if any given 'target' cell is in state y at step s, and if its leftward neighbour is in state x at step s, and if its rightward neighbour is in state z at step s, then it (the target cell) is to go into state u in step $s+1$. So, for example, if $0\underline{1}1\rightarrow 0$ is one of the state transition rules, and if, at step s, there exists a series of three neighboring cells in states *0*, *1*, and *1* respectively, then at step $s+1$ the centermost of these three cells will change its state from *1* to *0*.

There are eight possible combinations of states that three neighboring cells can be in—namely, 000, 001, 010, 011, 100, 101, 110 and 111. An ECA's algorithm simply consists of a 'complete set' of eight state transition rules—one for each of these different combinations. $2^8 = 256$ algorithms of this kind are possible. One such algorithm, known as the '110 rule' (Wolfram, 1983, 2002), is as follows:[9]

[9] It is called the '110 rule' because the binary sequence '01101110' represents the number one-hundred-and-ten, or, in decimal, 110.

1<u>1</u>1→0

1<u>1</u>0→1

1<u>0</u>1→1

1<u>0</u>0→0

0<u>1</u>1→1

0<u>1</u>0→1

0<u>0</u>1→1

0<u>0</u>0→0

The 110 rule happens to be Turing complete (Cook, 2004), with the implication being that, for all their simplicity, ECAs do not lack for computational power.

ECA's rules are what might be called '3:1 rules' because they take the states of *only three* cells (the target cell itself, and its immediate leftwards and rightwards neighbours) into account when updating the target cell's state. Because such 3:1 rules only 'see' three cells, ECAs having such rules are quite restricted in how their states can evolve through time. However, it is trivial to define a variant of an ECA with rules that 'see' further, beyond the immediate neighbours of the target cell. For example, we might define an ECA which has 5:1 rules, of the form *vw*<u>*x*</u>*yz*→*u*. The rules of such an ECA would be capable of updating a target cell in a way that is sensitive, not just to the states of the target cell's own immediate neighbour cells, but also to the states of *their* neighbour cells too.

Recall that *n* is the total number of cells in the automaton. Obviously, no rule could see more cells of an ECA than this, there being no more cells to see. And so the most powerful types of rules that an ECA could be equipped with are what we might call '*n*:1 rules', rules capable of updating the state of any target cell in a way that is sensitive to the states of *all of*

the n cells in the automotan. (Such n:1 rules are 'omniscient', so to speak. None of the states of an automaton's cells would be hidden from them.)

One way of modifying an ECA is by giving it *n*:1 rules instead of 3:1 rules, as just described. Another way of modifying it is by equipping it with what we will call an 'index cell'. An index cell is simply a read-only cell that is permanently locked in a '*' state. Let an automaton's 'active cells' include all of its cells except its index cell. Whereas the index cell is always in a '*' state, the active cells will always be in either a '0' state or a '1' state. The index cell is easily distinguished from the active cells on this basis. Having distinguished the index cell from all the active cells in this way, we can then distinguish all the active cells *from each other* based on where they are situated with respect to the index cell. For instance, we might identify a certain active cell based on its being *exactly five cells to the left of the cell that is in the '*' state* (namely, the index cell).

Why incorporate an index cell within the architecture of an ECA? For one reason only: to allow its active cells to be distinguished from each other along the lines just described.

What we will call a 'Modified ECA', or MECA, is simply a variant of ECA which incorporates both of the modifications just outlined. First, it evolves in accordance with a set of *n*:1 rules. (Thus its rules 'see' the states of every cell in the automaton.) Second, it has an index cell. (Thus its *n* cells include one index cell in addition to *n*-1 active cells.)

4. The Solar System Simulator

We will now explain how a MECA can be programmed to simulate the solar system.

To start with, let's simplify by focusing on just two parameters of the solar system, the *x*-position and *y*-position of Earth relative to the sun. Consider Figure 1, which depicts the Earth and its orbit around the sun.

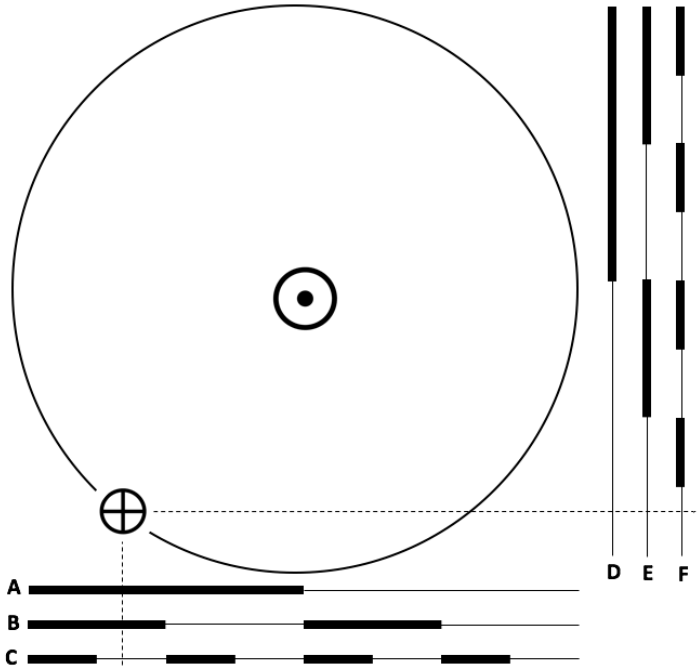

Figure 1. How the *x*- and *y*-positions of the Earth represent a binary number.

In Figure 1 the *x*-axis has three *rulers* associated with it, the A, B and C rulers. The *y*-axis likewise has three rulers associated with it—the D, E and F rulers. Each ruler is divided into a series of *positive* and *negative bars*. A positive bar of a ruler is indicated with a thick black line. A negative bar is instead indicated with a thin back line. Notice that the positive and negative bars of the A, B and C rulers form a binary tree, as do the positive and negative bars of the D, E and F rulers. Notice also that Earth's *x*-position falls within positive bars of the A and B rulers, but into a negative bar of the C ruler. We may therefore represent the Earth's *x*-position (admittedly, to very limited precision) as follows: $A^{+}B^{+}C^{-}$. We can likewise represent the Earth's *y*-position as $D^{-}E^{-}F^{-}$. Thus, in Figure 1 the Earth's *x*/*y* position is $A^{+}B^{+}C^{-}D^{-}E^{-}F^{-}$.

Let's define a mapping, $\mathcal{M}$, which maps each different ruler onto a different a cell of a MECA, and which maps negative bars to 0 and positive bars to 1. Let's suppose that $\mathcal{M}$ maps the A, B, C, D, E and F rulers onto a contiguous series of cells, the leftmost of which has the

index cell to its left. Then if these particular cells were in the following states, this would correspond, under $\mathcal{M}$, to the Earth being in the position depicted in Figure 1:

*	1	1	0	0	0	0

We now have an explanation of how the Earth's *x*- and *y*-positions in the solar system could be represented (to what is admittedly a very low degree of precision) within a MECA. Needless to say, this system of representation can be easily expanded to encompass other parameters of the solar system too. We can imagine a coordinate system (possibly centred on the sun's centre of gravity, although these details can be decided arbitrarily) whose *x*-, *y*- and *z*-coordinates all have such sets of rulers associated with them. This would allow the *x*-, *y*- and *z*-coordinates of each and every moving body in the solar system to be thought of as encoding a binary string. If we deepen the binary tree associated with each axis, by adding more rulers to it (with progressively shorter positive and negative bars), then this would solve the precision problem: the *x*-, *y*- and *z*-positions of each of the bodies could then be represented to arbitrarily high degrees of precision. Similar sets of rulers can be associated with other magnitudes too—such as with the velocities, masses, rotational axes, and rotational velocities of each of the solar bodies. If each and every one of these rulers is mapped onto a different cell of the MECA by $\mathcal{M}$, then the overall state of the MECA will represent a highly-detailed claim about the overall state of the solar system at a point in time.

In this way, any given state of the solar system can be mapped, by $\mathcal{M}$, onto a corresponding state of a MECA. But what of how the solar system's state *evolves over time*? We need to program the MECA so that if, during step *s* of its computation, its cells' states are mapped by $\mathcal{M}$ onto the solar system's state at time *t*, then in step *s*+1 of the computation, its cells' states

will instead be mapped by $\mathcal{M}$ onto the solar system's state at time $t+k$ (where k is a unit of duration—e.g., one second).

To see this can be accomplished, let's return to the simplified case where the Earth is the sole body in the solar system, and where only the x- and y-positions of the Earth are to be modelled by the MECA. At time t, the Earth is in the location represented by the following state of the MECA (which is to say, in the location depicted in Figure 1):

State 1.

*	1	1	0	0	0	0

Let's suppose that at time t+k, the Earth will instead be in the location represented as follows:

State 2.

*	0	1	1	0	0	0

Thus we require the MECA to be programmed with a set of rules that will, among other things, ensure that if it is ever in State 1 at one step, then it will be in State 2 during the next step. Rules that will advance the MECA from State 1 to State 2 are these:[10]

*11<u>0</u>000→1

110<u>0</u>00*→0

100<u>0</u>0*1→0

000<u>0</u>*11→0

00*<u>1</u>100→0

[10] These six rules merely tell the MECA what to do if it is in State 1. They are silent as to what the MECA will do if it is any other state. A complete set of state-transition rules, capable of fully determining the MECA's behaviour, would therefore need to include numerous other rules in addition to these six.

0*1<u>1</u>000→1

Notice that the x^{th} rule in this list is of the form $j \rightarrow k$, where j is a copy of State 1 rotated x-1 places to the left, and where k is the central digit of State 2 when it is rotated x-1 places to the left. Thus, it is trivial to generate rules that will take the MECA from State 1 to State 2 from information about State 1 and State 2 themselves.

This is a simple example, but the lesson generalizes. For any two successive states of the solar system, S1 and S2—no matter how complicated they might be—it is trivial to program a MECA so that if it is ever in state $\mathcal{M}$(S1) at one step in its operation, then its state will change to $\mathcal{M}$(S2) at the next step.

Now, suppose that we have a complete system of rulers associated with all of the solar system's significant parameters. Suppose, also, that we have defined some mapping function, $\mathcal{M}$, that pairs each one of these rulers with some corresponding cell in a MECA. Finally, suppose that we have equipped the MECA with a set of state transition rules, R, that are such that if the MECA is in state S at step s, then they will cause it to be in state $\mathcal{M}(f(\mathcal{M}^{-1}(\text{S})))$ at step s+1 (with f being the laws of motion). In other words, we have programmed the MECA so that if, at step 0, its cells' states accurately represent the state of the solar system at time t, then at step x its cells' states will accurately represent the state of the solar system at time $t+xk$.

This MECA, as so programmed, is what we will call the *Solar System Simulator* (SSS).

5. Why AXA entails limited pancomputationalism

Recall the challenge Copeland set for his foe: to 'describe for us the solar system's computational architecture and detail the algorithm by which the solar system arrives at values of f' (1996, p. 339). We are now in a position to meet this challenge. Copeland's AXA theory implies—we think—that the solar system is a computational implementation of SSS. If this is

right then the solar system's computational architecture is SSS's architecture, which is to say, the architecture of a certain MECA. Its algorithm is SSS's algorithm, which is to say, the set of rules, *R*, with which SSS has been programmed—rules that cause the states of its cells to evolve though time in a way that tracks the evolving state of the solar system.[11]

Copeland might resist this claim either by: (i) denying that SSS is an abstract computing device; or by (ii) denying that the solar system is a physical implementation of SSS. But neither of these options is viable, for reasons now explained.

On (i). SSS is a MECA. As such it accepts an input, in the form of the initial states of its cells, and produces an output, in the form of the states adopted by its cells after some fixed number of steps. It has a clearly defined computational architecture, consisting of the way in which the states of its cells are updated, step by step, under the control of a set of *n*:1 rules. Finally, it is programmed with an algorithm—the algorithm consisting of the particular set, *R*, of *n*:1 rules provided to it as a matter of fact. Since it has inputs, outputs, a computational architecture and an algorithm, it satisfies all the conditions for being an abstract computing device that Copeland himself lays down.

On (ii). The solar system satisfies AXA's conditions for being a physical implementation of SSS. Recall that AXA says the following:

> Entity *e* is computing function *f* if and only if there exist a labelling scheme *L* and a formal specification SPEC (of an architecture and an algorithm specific to the architecture that takes arguments of *f* as inputs and delivers values of *f* as outputs) such that ⟨*e*, *L*⟩ is an honest model of SPEC. (Copeland, 1996, p. 348)

[11] The solar system will doubtlessly have other computational architectures and algorithms too, there surely existing an endless variety of other abstract isomorphs of the solar system, in addition to SSS. But it suffices for our purpose of refuting Copeland to describe *just one* abstract computing device that the solar system physically implements.

Let *e* denote the solar system. Let *f* be the function computed by SSS, in the course of its simulating the motions of the bodies in the solar system. Let SPEC be a description of SSS's architecture and algorithm. Finally, let *L* be the mapping, $\mathcal{M}$, that maps each ruler we have associated with the solar system onto a corresponding cell of SSS (with the positive bars of a ruler being mapped to a 1, and the negative bars being mapped to a 0). According to AXA, *e* (the solar system) is computing the function *f* (the function computed by SSS) iff ⟨*e*, *L*⟩ is an honest model of SPEC, which to say, iff, ⟨*The solar system*, $\mathcal{M}$⟩ is an honest model of SSS's architecture and algorithm. Is it? Yes! It satisfies both the conditions Copeland specifies for being an honest model.

First, it is not *ex post facto*. It is perfectly possible to predict the motions of the solar bodies ahead of time, with high accuracy, (as, for example, NASA does when it sends space probes to Mars, and as even the ancients did when they successfully predicted eclipses). This being so, there is no barrier to our being able to program SSS, ahead of time, to forecast the future evolution of the solar system. One needn't have already observed the solar system over some period of time in order to be able to program SSS to simulate the solar system over that same period of time. SSS needn't be contrived only 'after the fact'.

Second, it is counterfactual supporting. Let God reach His hand into the solar system to give the planets and moons *these* starting locations and to impart *these* initial velocities (whichever locations and velocities He so chooses). $\mathcal{M}$ tells us how to set SSS's initial state so as to represent this new solar system configuration God has determined. Provided God now lets the solar system's state evolve in accordance with the equations of motion, without further divine interventions, then SSS's architecture and algorithm are such that its states will change, step by step, in a way that, under $\mathcal{M}$, accurately 'tracks' the evolving state of the solar system.

6. Possible rejoinders

How might Copeland reply to the above demonstration that the solar system is a physical computer even by the lights of his own theory, AXA? We are aware of only two avenues open to him, neither of which appears workable on careful examination.

Counterargument 1. In order for ⟨*The solar system*, $\mathcal{M}$⟩ to be an honest model of SSS, it must be the case that every element of SSS is mapped, by $\mathcal{M}$, onto a corresponding element of the solar system. We have seen how each of SSS's active cells will be mapped by $\mathcal{M}$ onto a corresponding ruler state of the solar system. But recall that SSS also has an extra read-only 'index cell' (which bears a '*' permanently). What element of the solar system is $\mathcal{M}$ to map the index cell onto? Copeland might contend that the index cell cannot be mapped onto any aspect of the solar system at all. If this was correct, then the solar system would not be an 'honest model' of SSS.

Reply. The solution to this problem is trivial. Because the state of the index cell is permanent and immutable, it can simply be mapped by $\mathcal{M}$ onto any permanent, immutable state of the solar system: e.g., its state of 'having a centre of gravity', or its state of 'either having eight planets or not'.

Counterargument 2. Copeland might attempt to evade our conclusion by pointing to a mismatch between the counterfactual dispositions of SSS and the counterfactual dispositions of the solar system. SSS is a MECA that has been provided with a particular algorithm—namely, *R*, an algorithm for simulating the solar system. But counterfactually, the same MECA might instead have been provided with a different algorithm instead—say, an algorithm for playing chess, or for forecasting the weather, or for simulating a nuclear explosion. The solar system, needless to say, doesn't support the same range of counterfactuals. It doesn't have parameters that, had their values been set differently than they actually have been, would have caused it to behave in accordance with the equations for chess playing, or the equations of weather forecasting, or the equations of exposition-simulation, instead of the equations of

orbital motion. In short, SSS is *programmable*, while the solar system is not. Copeland might claim, on this basis, that the solar system isn't a physical realization of SSS after all.

Reply. This approach won't fly. In arguing along these lines Copeland would be contradicting AXA, not defending it. Why so? Well, according to AXA all that we must do in order to show that the solar system computes the same function as that computed by SSS is find a labelling scheme, L, such that $\langle$*The solar system*, $\mathcal{M}\rangle$ is an honest model of SSS's *actual* architecture and algorithm. AXA does not require us to take any account of the *counterfactual algorithms* that a MECA might have been supplied with instead of the particular algorithm for simulating the solar system *that it was supplied with as a matter of fact* (the algorithm that transforms the MECA into the particular machine we are calling 'SSS'). We have just described such a labeling scheme—namely, $\mathcal{M}$. Hence the condition Copeland (1996) laid down has been met in full. In requiring us to take account of counterfactual algorithms, Copeland would be turning his back on AXA and proposing a new theory of physical computation in its place.

Now, perhaps Copeland would be prepared to bite this bullet. Perhaps he would be prepared to renounce AXA and propose a new theory—call it 'new-AXA'—that would require a physical entity to share *all the counterfactual dispositions* of an abstract computing device in order for it to qualify as computing the same functions as the abstract computing device.

But then he would confront a new problem. This problem concerns the fact that the distinction between a machine's architecture and its algorithm *is merely a matter of convention*, rather than being an objective feature of the machine in question. The architecture of a machine is that part of it which is considered to be immutable and set in stone by those who use the machine to compute, while the algorithm of a machine is that part of it which is considered to be re-configurable (Pylyshyn, 1984, pp. 93–95). If a particular switch within a physical machine is welded into its 'off' position then its being off is part of its architecture, not part of

its algorithm. But instead of welding it into the off position, we might padlock it into an off position, and throw away the key, to achieve the same effect. Or we might attach a 'do not touch' notice to it. Or we might merely make it a conventional rule, that is understood and obeyed by the machine's operators, that the switch is always to be left off. From the perspective of computational theory, these four ways of ensuring the switch must always be off are equivalent. They are all equally valid ways of ensuring that the switch's being off is a part of the machine's architecture, not of its algorithm.

Crucially, similar considerations apply to abstract machines. If you describe to me an abstract machine of the form $\langle\langle a,b,c\rangle,\langle d,e,\mathrm{f}\rangle\rangle$ where a, b and c are elements of the machine's (fixed) architecture and d, e and f are elements of the machine's (mutable and programmable) algorithm, then I can describe back to you a machine of the form $\langle\langle a,b,c,d\rangle,\langle e,f\rangle\rangle$ which differs from your machine just in the respect that d is now considered to be an architectural, rather than algorithmic, element. The two machines will have identical actual behaviour, but different counterfactual behaviour, because d's state can vary over counterfactuals in the first machine (wherein it is part of the algorithm), but not in the second (wherein it is instead part of the fixed architecture).

Why does this matter? Well, because we can do the same thing to SSS. We can redefine SSS so that the rules by which it is programmed are considered to be part of its architecture, not part of its algorithm. Let's call the abstract machine that we get when we do this, 'Firmware SSS'. Firmware SSS differs from SSS in that the program for simulating the solar system has, in effect, been 'hardwired' into it. Whereas SSS is a general purpose abstract computing device, that could have been programmed to do any of numerous things, but which has in fact been programmed to simulate the solar system, Firmware SSS is instead a single-purpose abstract computing device, that can do only one thing: namely, simulate the solar system. Because Firmware SSS's counterfactual dispositions are restricted to match the solar system's

counterfactual dispositions, new-AXA implies that the solar system is an honest model of Firmware SSS (if not of SSS itself). Thus, new-AXA implies that the solar system is a computer. This being so, Copeland can't evade pancomputationalism by abandoning AXA in favour of new-AXA.

7. Conclusion

The threat of limited pancomputationalism presents a supporter of the CTM with a pair of uncomfortable options. She must, so it seems, either abandon the counterfactual account of computation, despite this theory's very great elegance and simplicity, or bite the bullet and count the solar system and its ilk among the ranks of physical computing systems. Copeland attempts to thread a path between the two horns of this dilemma. He tightens up the counterfactual account by adding to it extra constraints drawn from classical computing theory. He claims that his resulting 'tweaked' version of the counterfactual account, AXA, avoids limited pancomputationalism. He challenges his opponent to refute him by showing how, according to AXA, the solar system could compute. This is a challenge we have met in the present paper. We conclude that Copeland's 'third way' is not viable: AXA offers no escape from limited pancomputationalism. Our own view is that the counterfactual account should instead simply be accepted 'as is', its pancomputationalist implications notwithstanding. There is, we say, no real cost to granting that the solar system computes, provided we at the same time lump it in with things like pocket calculators and iphones, that compute *only in a way that doesn't support mentation*. Does the solar system compute? Yes. Does it thereby think, or feel, or have any other mental states? No. It is like the human brain *in that* it computes, but not remotely similar *in what* it computes. Therein lies the reason why the brain is the seat of a mind while the solar system is not.

References

Casini, L. (2017). Malfunctions and teleology. *European Journal for Philosophy of Science*, *7*(2).

Chalmers, D. J. (1996a). Does a rock implement every finite-state automaton? *Synthese*, *108*(3).

Chalmers, D. J. (1996b). *The Conscious Mind: In Search of a Fundamental Theory*. Oxford: Oxford University Press.

Cook, M. (2004). Universality in Elementary Cellular Automata. *Complex Systems*, *15*, 1–40.

Copeland, B. J. (1996). What is computation? *Synthese*, *108*(3).

Copeland, B. J. (2017). Turing's great invention: the universal computing machine. In B. J. Copeland, J. Bowen, R. Wilson, & M. Sprevak (Eds.), *The Turing Guide*. Oxford University Press.

Cummins, R. C. (2002). Neo-teleology. In A. Ariew, R. E. Cummins, & M. Perlman (Eds.), *Functions: New Essays in the Philosophy of Psychology and Biology*. Oxford University Press.

Davies, P. S. (2000). Malfunctions. *Biology and Philosophy*, *15*(1), 19–38. https://doi.org/10.1023/A:1006525318699

Dennett, D. C. (1978). The abilities of men and machines. In *Brainstorms* (pp. 275–286). MIT Press.

Fodor, J. A. (1975). *The Language of Thought* (Vol. 87). Harvard University Press.

Fresco, N. (2014). *Physical Computation and Cognitive Science* (Vol. 12). Springer Publishing Company, Incorporated.

Garson, J. (2017). Against Organizational Functions. *Philosophy of Science*, *84*(5).

Godfrey-Smith, P. (2009). Triviality arguments against functionalism. *Philosophical Studies*, *145*(2), 273–295.

Mcculloch, W. S., & Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, *5*(4), 115–133.

Millikan, R. G. (1989). In defense of proper functions. *Philosophy of Science*, *56*(June).

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, *4*(2).

Peters, U. (2014). Teleosemantics, Swampman, and Strong Representationalism. *Grazer Philosophische Studien*, *90*(1).

Piccinini, G. (2004). Functionalism, computationalism, and mental contents. *Canadian Journal of Philosophy*, *34*(3).

Piccinini, G. (2007). Computing mechanisms. *Philosophy of Science*, *74*(4), 501–526.

Piccinini, G. (2015). *Physical Computation: A Mechanistic Account*. Oxford: Oxford University Press.

Piccinini, G. (2017). Computation in Physical Systems. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2017). Metaphysics Research Lab, Stanford University. Retrieved from https://plato.stanford.edu/archives/sum2017/entries/computation-physicalsystems/

Putnam, H. (1967). Psychological predicates. In W. H. Capitan & D. D. Merrill (Eds.), *Art, Mind, and Religion* (pp. 37–48). University of Pittsburgh Press.

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

Pylyshyn, Z. W. (1984). *Computation and Cognition*. MIT Press.

Rescorla, M. (2017). The Computational Theory of Mind. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Spring 2017). Metaphysics Research Lab, Stanford University. Retrieved from https://plato.stanford.edu/archives/spr2017/entries/computational-mind/

Scheutz, M. (1998). Do Walls Compute After All? — Challenging Copeland's Solution to Searle's Theorem Against Strong AI. In *Proceedings of the 9th Midwest AI and Cognitive Science Conference 1998* (pp. 43–49). AAAI Press.

Scheutz, M. (1999). When physical systems realize functions. *Minds and Machines*, *9*(2).

Searle, J. R. (1990). Is the brain a digital computer? *Proceedings and Addresses of the American Philosophical Association*, *64*(3), 21–37.

Searle, J. R. (1995). *The construction of social reality*. Free Press.

Shagrir, O. (2006). Why We View the Brain as a Computer. *Synthese*, *153*(3), 393–416.

Tolly, J. (forthcoming). Swampman: A Dilemma For Proper Functionalism. *Synthese*.

Wolfram, S. (1983). Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, *55*(3), 601–644. https://doi.org/10.1103/RevModPhys.55.601

Wolfram, S. (2002). *A new kind of science*. Wolfram Media. Retrieved from https://books.google.co.nz/books?id=dw_vAAAAMAAJ