Brit. J. Phil. Sci. **70** (2019), 153–178

A Simplicity Criterion for Physical Computation

Tyler Millhouse

ABSTRACT

The aim of this article is to offer a formal criterion for physical computation that allows us to objectively distinguish between competing computational interpretations of a physical system. The criterion construes a 'computational interpretation' as an ordered pair of functions mapping (i) states of a physical system to states of an abstract machine, and (ii) inputs to this machine to interventions in this physical system. This interpretation must ensure that counterfactuals true of the abstract machine have appropriate counterparts which are true of the physical system. The criterion proposes that rival interpretations be assessed on the basis of simplicity. Simplicity is construed as the Kolmogorov complexity of the interpretation. This approach is closely related to the notion of algorithmic information distance and draws on earlier work on real patterns.

doi:10.1093/bjps/axx046

Advance Access published on August 30, 2017

1 Introduction

1.1 Philosophical background

Pancomputationalism is the view that every physical system is a computational system (Piccinini [2007]). Varieties of pancomputationalism are primarily distinguished by the kind of computations physical systems implement. As Piccinini ([2007]) argues, the weakest of these views are plausible but uninteresting. For example, one might say that a rock implements a finite state machine (hereafter, an 'FSM') with a single state, no inputs, and no state transitions (Figure 1).[1] While this is a plausible model of rock behaviour, it tells us more about the flexibility of the FSM formalism than it does about the nature of physical computation. Stronger varieties of pancomputationalism hold that there is a large class of physical systems—for example, those of sufficient complexity—whose members individually realize many elaborate computations simultaneously. Of course, as computational systems proliferate and as the computations they implement become more elaborate, physical computation appears increasingly trivial.

This result would spell trouble for philosophical and scientific theories that invoke physical computation. For example, computational functionalism is the view that the brain is functionally organized as a computer and that mental states are computational states of the brain (Piccinini [2009]). Triviality arguments are one line of attack against computational functionalism and functionalism more broadly (Godfrey-Smith [2009]). Given the important explanatory role of mental states in psychology and cognitive science, it would be extremely unfortunate if implementing these states turned out to be a trivial feature of the brain. Generally, neither the proponents of triviality arguments nor the advocates of computational functionalism are prepared to abandon mentalistic explanation. Triviality arguments, then, are intended to serve as an informal *reductio* of computational functionalism. Hence, there is an intimate connection between strong pancomputationalism and the philosophy of mind.

There are two primary kinds of arguments in favour of strong pancomputationalism. For simplicity, I'll call these 'rock' and 'bucket' arguments.

Figure 1. An FSM plausibly implemented by a rock.

[1] A note on the FSM graph formalism: Nodes are states. Nodes with double circles are final or accept states (these can be ignored for the purposes of this article). Directed edges are state transitions, and labels on those edges indicate inputs that bring about that transition.

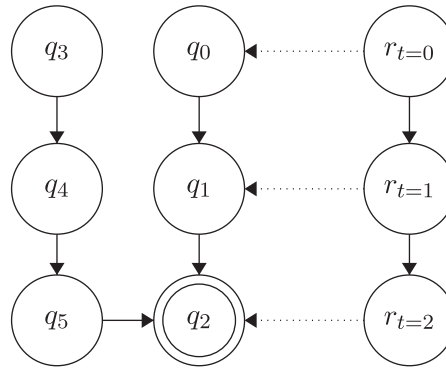

Figure 2. A mapping (dotted line) from a sequence of time-individuated states of a rock, $[r_{t=0}, r_{t=1}, r_{t=2}]$, to a sequence of machine states, $[q_0, q_1, q_2]$. This is meant to establish that the rock (over a particular time interval) implemented the computation described by the machine state sequence. The possible machine state sequence $[q_3, q_4, q_5, q_2]$ could just as easily have been mapped to the states of the rock by individuating another state, $r_{t=3}$.

The former, due to Putnam ([1988]), attempt to show that even extremely simple systems (like a rock) can rightly be said to implement a vast number of complex computations. The latter, attributed to Hinckfuss, attempt to show that highly complex dynamical systems (like a bucket of water warming in the sun) can rightly be said to implement countless sophisticated computations (Piccinini [2015a]). Though this distinction is not perfectly clear, the former attempt to show that the conditions on implementing a computation are surprisingly weak, while the latter attempt to show that these conditions, though strong, are met surprisingly often. In either case, we are left with an embarrassment of riches, for it turns out that far from being the unique province of minds and machines, sophisticated computation is ubiquitous to the point of triviality.

In general, triviality arguments attempt to establish a mapping between states of a physical system and states of an abstract machine as well as between inputs to the abstract machine and interventions in the physical system. For simplicity, I treat outputs as a subset of the possible states of an abstract machine. Trivializers take for granted that if one can establish such a mapping between a physical system and an abstract machine, the physical system implements that machine. For example, Putnam ([1988]) argues that a rock with temporally individuated states implements every inputless FSM (Figure 2) and that any system with certain input–output relations implements every FSM with the same input–output relations. To appreciate Putnam's argument, note that however many physical states (in the ordinary sense) occur between input and output, we can temporally individuate as many as we require. If the

number of intervening states is too large, we can simply map disjunctions of physical states to machine states.

It is unclear whether Putnam needed to treat FSMs with inputs as a special case.[2] Even with this concession, though, Putnam's position still undermines physical computation. Specifically, it trivializes the particular algorithms which implement functions. For example, there are different ways to completely and optimally search a graph, and these ways differ substantially both in how they traverse the graph and in their time and space complexity. Breadth-first search (BFS) uses exponentially more memory as the size of the graph grows, whereas iterative deepening search (IDS) requires only linearly more memory. For a given graph, however, they will both return the same (optimal) path to the goal node wherever such a path exists. Hence, a machine that accepted graphs, goal states, and initial states and returned paths would have the same input–output relations whether or not it employed BFS or IDS. This is true despite the fact that they make use of memory in markedly different ways. If mental states depend on the algorithms and not just the input–output mappings implemented by the brain, then we must resist Putnam's argument in order to save this condition from triviality. For example, this kind of condition is likely required to defeat the giant look-up table (or 'Blockhead') objection to computational functionalism (Block [1982]; Aaronson [2011]).

In response to Putnam ([1988]), Chalmers ([1996]) suggests an additional constraint on the kinds of mappings sufficient to establish genuine implementation. Putnam's argument relies on highly arbitrary mappings of physical states to machine states. This allows a physical system with appropriate inputs and outputs to trivially instantiate whatever intervening states occur between input and output. Importantly, Putnam only requires a mapping suitable for every computation considered individually. To see this, suppose that in the course of a particular computation, a machine never enters a particular state or fails to traverse every possible sequence of states (Figure 2). In such a case, Putnam-style mappings can (given different inputs or initial states) map different physical states to machine states in ways that each fail to capture counterfactual transitions allowed by the machine's transition function. For example, the FSM in Figure 2 can follow one of two paths to the accept state, q_2, depending on the initial state of the machine.[3] The rock will transition through $[r_{t=0}, r_{t=1}, r_{t=2}, \ldots]$ regardless. Hence, if we want to interpret the rock as taking the path beginning with q_3, we have to map the physical

[2] It seems Putnam ([1988]) wanted to avoid disputes about admissible kinds of inputs and output.

[3] Strictly speaking, different initial states are sufficient to distinguish otherwise identical FSMs as different machines. However, since the same problem can be created for a single machine by introducing inputs, I'll ignore this for simplicity.

sequence to a different sequence of machine states. Under neither mapping (considered individually) are all of the relevant counterfactuals true. Quite naturally, Chalmers proposes that we require that the mapping from physical to machine states be consistent with all the counterfactuals expressed in the machine's transition function. Hence, if states of a physical system are mapped to states of an abstract machine, then all the relevant counterfactuals must be true of the physical system under that mapping. Given this constraint, arbitrary mappings will not do. The physical states mapped to a particular machine state must support all possible transitions even when those transitions are not relevant to actual computations. In essence, Chalmers requires that for a system to have the computational states of a particular machine, it must first fully implement that machine.

This excludes some trivializing strategies, but problems remain. Noting one such problem, Chalmers ([1996]) argues that a system with a simple record of inputs (and a clock) could ensure that subsequent states could always be individuated based on prior inputs. For example, if the system will transition from state q_{10} to state q_{11} if and only if input x is entered, then whatever physical state comes next, we can safely map it to q_{11} if the system has kept a record of inputs since this record is sufficient to distinguish the resulting state from others. If x were not entered (and hence not stored), the subsequent state would differ at least insofar as the input record differed. To give this some physical meaning, we might imagine entering inputs to our rock-computer by scratching strings of characters into the surface in sequence. As we carve each character, the rock's physical state is changed and this new physical state can be mapped to whatever computational state is supposed to result from that input. For machine state sequences that proceed deterministically without input, we can revert to time-individuating states.

These concerns and others led Chalmers ([1996]) to develop a stronger model of computation which captures the combinatorial structure of computational states. Brown ([2012]), Scheutz ([2012]), Sprevak ([2012]), and others have critiqued this approach and Chalmers ([2012]) has responded. Piccinini ([2015b]) and others have offered accounts that reject a mapping approach. I cannot do justice to this debate here, but my primary project is to respond to triviality arguments by offering an alternative account of physical computation. In what follows, I will outline and independently motivate my alternative, but a comparative evaluation will require a separate treatment. For my purposes, I intend to grant that input records allow a profusion of counterfactual-supporting mappings. I will further grant *arguendo* that refined criteria designed to block these mappings universally fail. However, I will deny that triviality follows from these concessions and, instead, offer a formal criterion for evaluating competing mappings.

1.2 Information distance

Suppose that we received an email consisting entirely of zeros. Quite naturally, we might disregard the email as a mistake and move on. However, suppose that attached to the email was a key for decoding the message. The key works as follows. It assigns a plaintext character to each character in the ciphertext based on (at most) two factors: the character itself and its position in the message. Using this key, we decode the message and find that it is the sonnet shared by Romeo and Juliet upon their first meeting. In what sense does the original message contain the sonnet? Perhaps, like me, you think that the message only contains the sonnet in a very strained sense (if at all). What has gone wrong?

We might be tempted to say that it is the uniformity of the ciphertext (all zeros). More precisely, we might be concerned that the ciphertext has very low complexity compared to the complexity of the plaintext. Complexity, on certain accounts, is identified with information content, so the plaintext may carry more information than the ciphertext. This seems to rule out the possibility that the ciphertext 'contains' the plaintext. There is something to this concern, but it is an incomplete account of the problem. Suppose the ciphertext consisted of a random jumble of characters and was paired with a similar key. By 'a similar key', I mean that the key is contrived to assign the right character in the right position to recover the sonnet and that the key displays few if any regularities in how characters are assigned. For example, a simple key might assign 'A' to 'P' regardless of position, whereas the original key assigns letters to zero arbitrarily in order to achieve the desired ciphertext.[4] Compare:

Plaintext: 'If I p . . .'	Plaintext: 'If I p . . .'
Key:	Key:
1. If '0' in position 1, 'I'.	1. If 'b' in position 1, 'I'.
2. If '0' in position 2, 'f'.	2. If 'H' in position 2, 'f'.
3. If '0' in position 3, space.	3. If '2' in position 3, space.
4. If '0' in position 4, 'I'.	4. If '7' in position 4, 'I'.
5. If '0' in position 5, space.	5. If 'x' in position 5, space.
6. If '0' in position 6, p.	6. If '!' in position 6, 'p'.
7. . . .	7. . . .
Ciphertext: '00000 . . .'	Ciphertext: 'bH27x! . . .'

While it is less obvious (given the ciphertext alone) that the second key has been gerrymandered to return a particular ciphertext, direct comparison of the

[4] The former is known as a mono-alphabetic cipher. The latter is closer to a one-time pad which assigns a different key character to each character of the plaintext. The former are trivial to break, while the latter offer (in the technical sense) perfect secrecy (Katz and Lindell [2007]). See Footnote 8 for more details.

keys reveals that they have an isomorphic structure.[5] Unless we want to say that the second ciphertext contains more of the sonnet than the first despite this similarity, we should turn our attention to the keys themselves.

First, though, we should replace our intuitive notion of containment with a more rigorously defined notion, namely algorithmic information distance. The algorithmic information distance (hereafter, 'information distance') between two strings is defined as the length of the shortest program that can return one string given the other (Bennett *et al.* [1998]). This concept is intimately related to the concept of Kolmogorov complexity. The Kolmogorov complexity of a string is the length of the shortest program which could generate that string.[6] Whereas classical information theory concerns the average information required to communicate objects from a random source, Kolmogorov complexity allows us to quantify the information content of particular objects (Shannon [1948]; Bennett *et al.* [1998]).[7]

Hence, we can restate the problem with the message in terms of information distance. The problem is that the information distance between ciphertext and plaintext is high. This fact, in turn, explains the complexity of the key. If we understand the key as an algorithm for encrypting and decrypting the string, then the key simply is an algorithm for producing the ciphertext from the plaintext and vice versa. Because keys may have additional information beyond what is necessary for the particular task, the information distance between plaintext and ciphertext places a lower bound on the complexity of the key. Therefore, as the information distance between the ciphertext and plaintext increases, this lower bound on key complexity increases as well. As an illustration, compare the earlier keys with the following (trivial) key:

> Plaintext: 'If I p…'
>
> Key: For any character x at any position y, x.
>
> Ciphertext: 'If I p…'

In this case, the plaintext and ciphertext are identical, minimizing the information distance between them and allowing a markedly shorter key. This can be understood quite naturally as a result of minimizing the lower bound on

[5] The term 'gerrymandering' arose in the context of designing legislative districts. The term comes from the name of a Massachusetts governor who approved new legislative districts that furthered his political goals but were geographically and demographically absurd. The term seems especially apt for describing the kinds of manipulation involved in generating trivializing mappings or contrived encoding schemes.

[6] It is important to note that there may be no uniquely shortest program for generating a particular string. Hence, when I refer to 'the shortest program', I actually mean 'a shortest program'. I have preferred the former only due to the awkwardness of using the indefinite article with a superlative.

[7] For an extended discussion of how Kolmogorov complexity relates to Shannon information, see (Grünwald and Vitányi [2004]).

key complexity. The crucial point here is that as the information distance between the two strings grows, the decryption algorithm must increasingly carry out the hard work of communication.[8]

Before proceeding to the issue of physical computation, it would be good to get a feel for how information distance could help address other trivializing interpretations of physical systems. For example, suppose that a friend and I were on a walk and came across a sign that read, 'No Littering: Please Help Keep Our Parks Beautiful'. At the same time, we notice a line of ants marching along the edge of the sidewalk. After a moment, my friend announces that the line of ants carries the same message as the sign. To address my transparent incredulity, she explains that if we recorded the inter-ant distances in a sufficiently long segment of the line, we could (in principle) identify a decryption algorithm that accepts a string of inter-ant distances and returns strings of English letters. Using the notion of information distance, we can now formulate a preliminary response to my imaginative counterpart. We might say that the information distance between the string of inter-ant distances and the string of English letters on the sign is quite large. There is a simple one-to-one mapping between the figures painted on the sign and English letters (considered abstractly). The inter-ant distances, on the other hand, likely bear no straightforward relationship to English letters at all. Hence, while we can (in principle) recover the English text from the chain of ants, the contribution of the chain is likely much smaller than the contribution of the key.

2 Computers and Computation

2.1 Computational interpretations

Before I can introduce any criteria related to computational interpretations, it is essential to get clear on their nature. A computational interpretation of a continuous physical system, P, as an abstract discrete state machine, M, is an ordered pair of functions mapping (i) states of P to states of M, and (ii) inputs to M to possible interventions in P. We can call this interpretation I and its component functions S and C, respectively. Like Chalmers ([1996]), I require that the mapping support the right counterfactuals, in particular, those

[8] An exact parallel exists in more conventional models of encryption where the decryption algorithm is public and a key string is shared with whomever is permitted to decrypt the ciphertext. Ideally, the ciphertext should provide no information about the plaintext, that is, $P(\text{plaintext}) = P(\text{plaintext}|\text{ciphertext})$. This is called perfect secrecy. However, perfect secrecy requires an encryption string as long as the ciphertext. Otherwise, there will be some pattern in the ciphertext that (with a vast amount of classical computing) could be used to extract information about the plaintext. Hence, if we want to guarantee that our ciphertext is totally uninformative, but decryptable, we require a key that contains just as many bits as the ciphertext itself (Katz and Lindell [2007]).

specified by the abstract machine's transition function, δ. We can formalize this requirement as follows:

> For any machine states, m_i and m_j, any input, x, and any physical state, p_i, if $\delta(m_i,x) = m_j$, then if P were in p_i such that $S(p_i) = m_i$ and we did $C(x)$, the system would enter some physical state, p_j, such that $S(p_j) = m_j$.[9]

This formulation is more or less equivalent to Chalmers', but it makes explicit that the condition applies to the physical system given a particular mapping, and it omits talk of outputs which (as noted earlier) I construe as a subset of the states of the system.

2.2 A new criterion for implementation

The criterion I advocate acknowledges that many interpretations will satisfy the modest requirements specified above. Indeed, it does not propose to evaluate interpretations in a binary fashion. Instead, it offers a means of quantifying the relative goodness of proposed computational interpretations of any physical system. Interpretations assigned a smaller value count as better than those assigned higher values. For any physical system, P, and any pair of abstract machines, M_1 and M_2, if the optimal interpretation of P as M_1 is better than the optimal interpretation of P as M_2, then P more genuinely implements M_1 than it does M_2. The criterion is as follows:

> A physical system, P, implements a machine, M, to the extent that the simplest interpretation of P as M, $\text{argmin}_{I \in \mathcal{I}} \text{K}(I)$, is simple relative to the complexity of M, K (M). I is simple relative to M to the extent that it minimizes $\text{K}(I)/\text{K}(M)$.

Since our interpretation is simply a pair of functions, we can interpret the Kolmogorov complexity of I, K(I), as the length of the shortest program that computes S and C.[10] This amounts to a simplicity criterion for

[9] This formalization assumes an FSM, but it can easily be extended to more powerful models of computation. For Turing machines, the current state, head position, and tape contents constitute the current configuration of the machine (Sipser [2013]). The sequence of configurations is fully determined by the input, initial state, and transition function. S must map physical states to configurations in such a way that, given any sequence of input symbols, the system goes through the correct sequence of configurations.

[10] For details on how to understand the Kolmogorov complexity of functions (as opposed to, say, strings), see (Grünwald and Vitányi [2004]). In brief, it amounts to the number of bits required to specify a Turing machine that computes the relevant function. On this definition, K(f) will be equal to the length of the shortest program (or a shortest program if more than one exists) for computing f. Also, while it is more common to think of each Turing machine as computing a single function, we can imagine a machine that computes a piecewise function that has S and C as sub-functions. Inputs to S and C could be distinguished by a single bit (for example, '0…' might indicate a machine input and '1…' might indicate a physical state). This allows any information about relevant physical states to be shared across sub-functions.

interpretations—interpretations that are more complicated will require more elaborate programs to compute their component functions. Since M is a machine and since program length is computed for a particular universal Turing machine (UTM), $K(M)$ can be understood as the length of the shortest program which emulates M on the relevant universal machine.

The reason that the complexity of interpretations is discounted by the complexity of the machine is that it may be trivial to map even very complicated physical systems to simple machines. For example, consider a desktop computer and an inputless FSM with a single state (Figure 1). We can easily map the disjunction of all the states of the computer to the single state of the machine. In this case, I would be extraordinarily simple, but not tell us anything interesting about the desktop. However, interpreting a desktop as the machine that its designers intended to implement will require a more complex interpretation. Minimizing relative complexity requires only that more complex interpretations pay their way by relating physical systems to more complicated machines.

While this rationale seems reasonable, it underdetermines the formalization of relative complexity.[11] For example, why not $K(I) - K(M)$ or $K(I)/K(M)^2$? There are two things to say here. First, I am not wedded to the formalization of the criterion offered here. It may be that (in reflective equilibrium) we endorse a somewhat modified version. My core argument is that the complexity of I is centrally relevant to assessing claims about computer implementation and that *ceteris paribus* simpler interpretations are to be preferred. More specifically, when two competing interpretations attribute machines of similar complexity to a physical system, the simpler interpretation is to be preferred.[12] Second, though a fully rigorous defence would require a separate treatment, there are good reasons to believe that $K(I)/K(M)$ captures the value of interest. Suppose we had access to a physical system, P, and our reference UTM. Suppose further that we wanted to implement M using as few bits as possible. We can either implement M on our reference UTM, or we can implement I on our reference UTM, allowing us to use P as M. Thus, in a manner analogous to a compression ratio, $K(I)/K(M)$ represents the savings in bits relative to implementing M outright. In other words, it represents the degree of compression achievable given access to P.

Now consider the two proposed alternatives. Squaring the denominator would express a preference for attributing implementations of more complex machines since the denominator would scale polynomially (rather than

[11] I am indebted to the referees for noting this problem.

[12] In Section 2.4, I consider the related case of attributing the same machine to two different physical systems. I argue that the system we intuitively regard as implementing the machine affords a much simpler interpretation than the other (which we do not intuitively regard as implementing the machine).

linearly) with K (M). There may be some argument for this modification, but it's unclear what natural meaning the resulting criterion has. I compared the original version to a compression ratio. A compression ratio is the ratio of the uncompressed to the compressed file size. In this case too, one could choose to square the uncompressed file size, but it's unclear whether this is really telling us about compression in the sense we care about. Using such a measure, we could increase the apparent degree of compression by compressing a larger file by the same factor. The same problem exists for $\mathrm{K}(I) - \mathrm{K}(M)$, since the absolute value of the difference between compressed and uncompressed file sizes would increase as the files increase in size—even holding the compression ratio fixed. Hence, if we are interested in the degree of compression possible given access to a physical system, it seems best to employ the unmodified ratio.

Just as the aim of algorithmic information distance is to quantify the degree of similarity between two objects, the aim of this criterion is to quantify the degree of similarity between a physical system and an abstract machine. Both do so in terms of the number of bits we must employ to recover one structure from another—whether that be plaintext in ciphertext, or abstract machine in physical system. One might wonder whether a better criterion would explicitly invoke the information distance between P and M, $\max(\mathrm{K}(P|M), \mathrm{K}(M|P))$ (Bennett *et al.* [1998]). However, there are some important differences in the present case that make information distance inappropriate. The primary difference is that information distance is symmetric while implementation is not. In implementation, we might say that $\mathrm{K}(P|M)$ is allowed to be arbitrarily high since a vastly complicated physical system can implement a relatively simple machine.[13] $\mathrm{K}(M|P)$ will not do either since there are also elements of M that we needn't recover from P. For example, we needn't recover the accept states for M from P. I allows us to look up which states of M correspond to which states of P and which interventions in P correspond to which inputs to M. I assume that the details of M are already known. Thus, evaluating the complexity of I rather than the information distance between P and M is intended to restrict our measure of the difference between P and M to only those differences relevant in implementation.

Essentially, the criterion allows us to quantify the minimum amount of information required to use a physical system as a particular machine, abstracting away from practical limitations. This approach combines a standard

[13] Imagine using sheep and trained sheep dogs to implement a (non-restoring) AND gate. The dogs are trained to move the sheep from two areas to a third if and only if both of the original areas are occupied. Obviously, it would take a phenomenally complicated bit of code to recover a detailed physical description of the sheep–dog–field system from a description of an abstract machine that implements an AND gate. Nevertheless, an AND gate is a good way to describe the rules the dogs follow in moving the sheep. A program for mapping states of the sheep–dog–-field system to the states of the AND gate could simply place a threshold on the amount of mass in three spatial regions immediately above the surface of the field.

mapping account of implementation with the insights about compressibility found in work on real patterns (Dennett [1991]; Ladyman and Ross [2009]; Wallace [2014]). By requiring that machine states be defined over physical states, the criterion allows machine states in physical systems to avoid the charges of instrumentalism frequently levelled at the postulates of 'black box' predictive models. By invoking compressibility, the criterion offers a standard that (I will argue) can distinguish between competing interpretations and obviate the discovery of necessary and sufficient conditions for implementation.

Naturally, we will require some means of encoding physical states such that they can be passed as inputs to the relevant functions. There may be ways of (in effect) hiding interpretational complexity in our encoding by selecting particular encoding schemes. One avenue for trivialization might be to build a favourable individuation of states into the encoding of physical states itself. Under such an encoding, a complicated disjunction of physical states may get a single label, say p_{729}. It would then be trivial (assuming we've correctly designed the encoding) to map physical to machine states. Note this kind of manipulation depends on using a particular coarse-grained description of the continuous physical system. Ideally, our encoding would describe physical states at the level of fundamental physics and leave any further individuation of discrete states to *S*. Given this, it seems best to maintain that the canonical value of K(*I*) is given under an encoding of physical states at the level of fundamental physics.[14] If one is concerned about taking this kind of physical description as given, one might follow Wallace ([2014], p. 54) in framing implementation (or 'realization') as inter-theoretic. That is, the question is not how to reduce a computational model of a system to physics, but how to reduce computational models to physical models. On this view, the specification of physical states would just be the formalism (up to some limit of precision) given by a particular physical theory.

However, in either case, this level of detail is wildly impractical. Even if we maintain (as I do) that the canonical value of K(*I*) requires an encoding at the level of fundamental physics, for any practical purpose, we will require a relatively coarse-grained way of describing physical states. Any proposed coarse-grained encoding, however, must preserve the canonical ordering of K values for the interpretations under consideration. Naturally, it will be hard to prove that an encoding scheme satisfies this condition, but so long as there

[14] It's not clear whether or not requiring a fine-grained physical description is enough to rule out gerrymandered encodings. While I have not been able to identify any trivializing strategies that respect this constraint, it may ultimately be necessary to add an additional constraint. Given the nature of the problem, I suspect that we may need to require that fine-grained physical states be described economically. The tools from the literature on minimum message length and minimum description length may prove especially useful in precisifying this requirement (see especially Wallace [2005]).

are sufficient reasons to think that a particular coarse-grained encoding will not affect the order of K values, we can use it for practical purposes. It might be best to simply borrow conventions already in use for modelling in the physical sciences since these are both useful for predictive modelling and designed without an eye to favouring spurious interpretations (or any interpretations for that matter). As I will argue, the differences between the values of $K(I)$ for intuitive and trivializing interpretations are likely to be quite large. If this is correct, these differences will likely swamp any minor differences introduced by our choice of encoding scheme.

2.3 Kolmogorov complexity

In the next section, I will offer a preliminary defence of a simplicity criterion for computer implementation. First, though, it is worth heading off a specific objection to the use of Kolmogorov complexity in this context. I will not make a positive case for Kolmogorov complexity as a measure of simplicity other than to note that it has the advantage of measuring the information content of particular objects and, thus, is well suited to the task at hand. However, the Kolmogorov complexity of an object—such as a string—depends not only on the features of the object, but on our choice of a reference UTM on which the generating program must run. In other words, the length of the shortest program which implements our interpretation will vary with the UTM. Since this UTM is a matter of choice and since one can build arbitrary functions into the UTM, one can select a UTM that gives different values for $\text{argmin}_{I \in \mathcal{I}} K(I)$ or $K(M)$. Fortunately, if we restrict ourselves to additively optimal UTMs, then the value of K can only vary up to an additive constant across UTMs (Li and Vitányi [2008]).[15,16]

Given this variability, one might argue that the criterion merely pushes the problem back to the level of UTM selection. However, there is nothing wrong with relocating a problem so long as it is less problematic in its new location.

[15] Additively optimal UTMs are those that provide the guarantee discussed above (Li and Vitányi [2008], p. 103). Consider a UTM, U, that employs a two-part code. The first part of this code specifies a Turing machine, T. The second part specifies an input to T, i. Given this two-part code, U returns the same string, s, that T would return given i. Hence, for any input, i, the length of code required for U to return the relevant string is the length of the code required to specify T plus the length of i. The former is the additive constant bounding the difference between $K_U(s)$ and $K_T(s)$ for any Turing machine, T. If U required i to be repeated, ii, then $K_U(s)$ could instead be up to twice $K_T(s)$. See (Li and Vitányi [2008], p. 105) for more details.

[16] As an anonymous referee helpfully observed, there are ways of making our choice of UTM unique. For example, one could select whichever otherwise suitable UTM in the enumeration of Turing machines has the smallest state × symbol product. The state × symbol product is a way of measuring the complexity of Turing machines themselves. However, making the enumeration of Turing machines unique requires that we select a fixed formalism for representing Turing machines (Li and Vitányi [2008], p. 91). See (Shannon [1956]) and (Chaitin [1966]) for further discussion of this point.

Indeed, there are several disadvantages to gerrymandering at the level of UTM selection that do not occur at the level of interpretation selection. For one, it seems reasonable to require that a single UTM be used across contexts. If the same UTM is used to evaluate computational interpretations of numerous physical systems, then trivializers face the difficulty of selecting a UTM that favours all of their trivializing interpretations. This may seem like a merely practical difficulty, but note the key difference. Heretofore, trivializers have rested their arguments on the fact that trivializing interpretations can be had for arbitrary physical systems because of the nature of computational interpretation. They can no longer make this claim since trivializing interpretations are (at best) only favoured under certain UTMs. Hence, the trivializer must abandon the claim that trivializing interpretations are a general problem for physical computation rather than a problem that arises under particular choices of UTMs. In addition, it is far from clear that there exist UTMs that universally favour trivializing interpretations. Selecting a UTM that favours a particular trivializing interpretation will favour only that and similar interpretations (specifically, interpretations for which $K(I'|I)$ is low). Furthermore, there are good reasons to believe that in order to assign a low complexity to a complicated function, our reference UTM must itself be complicated (Li and Vitányi [2008], p. 113). As a result, there may be a relatively large class of UTMs (simple ones) that broadly favour good interpretations over bad ones. Hence, unless there is some deep similarity between bad interpretations, what Tolstoy ([2000], p. 3) said of families may be true of computational interpretations: 'Happy families are all alike; every unhappy family is unhappy in its own way'.

However, one might worry that even if no UTM broadly favours trivializing interpretations, we still have no principled reason to choose one UTM over another. The choice, one might argue, is a pragmatic one, reflecting the needs and interests of human beings but not reflecting anything metaphysically interesting about the nature of physical computation. In response, one could argue that selecting a UTM is just part of regimenting our ordinary concept of a computer—a concept that turns out to involve a notion of simplicity in need of precisification. If a particular UTM favours a trivializing interpretation, we might take that (in itself) as a reason to reject it. This kind of consideration may play some role in UTM selection, but I hope to subject our intuitions about implementation to wider reflection.

Crucially, our selection of a UTM can be constrained by applications outside the philosophy of computation. For example, Kolmogorov complexity is used as a measure of individual randomness (Kolmogorov [1965]) as well as a universal prior for formal models of inference (Solomonoff [1964]). If we stipulate that the same UTM must be used across these applications, then our choice of UTM will affect not only what we say about physical

computation, but what we say about randomness and inference as well. Given this, reframing the problem at the level of UTM selection not only forces trivializers to change the nature of their claim, it places outside constraints on the acceptability of computational interpretations. These constraints provide a basis for widening the scope of our reflection and bringing more diverse intuitions to bear on our selection of a reference UTM.

Rather than as a substantive constraint on interpretations, we should perhaps think of our chosen UTM as a precise way of expressing our considered judgements about simplicity in wide reflective equilibrium. To be sure, there are disputes about whether there really are objective standards of simplicity, and one's disposition toward my account may vary with one's position in these debates. Crucially, though, these problems are not unique to computation. They are extant problems in metaphysics and epistemology that afflict scientific realism as much as computational realism. Without resolving these debates here, I am willing to accept that computers share the same precarious ontological position as objects in any of the sciences.

2.4 The simplicity criterion

In this section, I will attempt to philosophically motivate the simplicity criterion by showing that it gives intuitively plausible answers to questions about which physical systems are computers. I will not attempt to prove any claims about the length of programs, but rather to provide plausible reasons to think that certain programs will be shorter than others. To this end, I will first develop an account of naturalness for the individuation of states in a physical system. In particular, I am interested in how we might carve up the state space of a continuous physical system into a finite set of discrete states.[17] The primary aim here is to discretize a continuous physical system (or a suitable coarse-graining thereof) since we intend to interpret that system as a discrete state machine. It is important to emphasize that this individuation is distinct from the project of coarse-graining physical systems discussed earlier. That coarse-graining is intended to supply a simplified model of P for the purpose of comparing different interpretations. The individuation of physical states I am discussing now is part of the project of specifying I for a given M. For different machines, it will be necessary to discretize (or further discretize) the space of physical states in different ways. For all interpretations under consideration, this space may be either the phase space of the system in terms of fundamental physics or an admissible coarse-graining thereof.

[17] This 'carving up' needn't be exhaustive since there are states of physical computers that are not computational states. A laptop may be in the physical state of being smashed to pieces, but this is not a computational state.

While discretization is essential to computational interpretations, it is not an end in itself, and the naturalness of a particular discretization alone does not determine whether it will best serve our interpretation. Minimally, we must individuate enough physical states to map those states to the states of our abstract machine. Moreover, we must individuate states such that the counterfactuals expressed in the machine's transition function hold of the states as individuated.[18] As I will argue, good interpretations (under the simplicity criterion) will be those for which there exists a natural individuation of states that supports the relevant counterfactuals.

An individuation (or discretization), D, is a function mapping states of a continuous physical system, P, to a finite set of discrete states. Implicitly or explicitly, this will involve the specification of equivalence classes of physical states. For example, suppose we had a knob that could rotate continuously through 100° of arc. We could divide this arc into ten smaller arcs and count knob positions within each of the smaller arcs as equivalent. Given this division, we could easily say which of the ten states the knob is in for any real-valued knob position between 0° and 100°. For D to be natural, the length of the shortest program that computes D, $K(D)$, must be short relative to the number of states it individuates. This is quite similar to the simplicity criterion itself. The program implementing our knob individuation will be very short since (in the worst case) we need only specify a list of nine cut-off points and check the input against them. Now consider a discretization that divides the arc into ten disjunctions of arcs. For example, suppose that the first state included knob positions between 0° and 6° and between 67° and 71°. Despite mapping the 100° arc to the same number of states, we must specify (implicitly or explicitly) additional cut-off points. Hence, the shortest program for implementing this kind of discretization will generally be longer than the shortest program for implementing the former.[19]

With this framework in place, we can see how causal regularities in physical systems affect the complexity of computational interpretations. Suppose for simplicity that instead of a continuous physical system, we were working with a relatively fine-grained discrete system. In this system, there are two knobs. Each knob has six different settings, and when we change the position of the

[18] In practice, we must also consider the feasibility of intervening in the physical system in order to enter inputs. For now, I'll simply assume that we face no practical limits in this regard since our physical limitations seem irrelevant to whether or not something is a computer. We would not want an alien with no fingers to discount our computers because of its difficulty using keyboards.

[19] A telling exception is when the disjunctions of arcs can be identified by some simple pattern. For example, suppose state one were 0° to 1°, 10° to 11°, 20° to 21° and so on for the other states as well. This may be more economical to represent than ten non-disjunctive states of random size and position.

first knob (P) the position of the second knob (Q) changes automatically according to some transition function. Suppose we wanted to map these knob states to the states of two bits, where setting the state of the first bit (p) changes the state of the second bit (q) according to the following transition function: $\{(p_1 \rightarrow q_1), (p_0 \rightarrow q_0)\}$. This mapping will be simple or complicated depending on the particulars of the knob system. Consider the following transition functions for the knob system:

$$A: (P_1 \rightarrow Q_1), (P_2 \rightarrow Q_2), (P_3 \rightarrow Q_3), (P_4 \rightarrow P_4), (P_5 \rightarrow Q_5), (P_6 \rightarrow Q_6)$$

$$B: (P_1 \rightarrow Q_2), (P_2 \rightarrow Q_5), (P_3 \rightarrow Q_1), (P_4 \rightarrow P_6), (P_5 \rightarrow Q_3), (P_6 \rightarrow Q_4)$$

A provides a number of opportunities for economy when mapping the knob system to the bit system. For example, we can specify the range of knob positions that count as each bit state quite simply, IF POSITION < 0; ELSE 1. Importantly, this is not only simple, it ensures that whenever knob P is in a state mapped to 1, so is knob Q and whenever knob P is in a state mapped to 0, so is knob Q. Hence, the bit system's transition function is respected by this mapping. With B, we must be more verbose about our mapping scheme for each knob. For example, IF P, THEN IF POSITION < 4, 0; ELSE 1; IF Q, THEN IF POSITION = 2 OR 5 OR 1, 0; ELSE 1. Of course, we shouldn't be misled by apparent disorderliness. Consider an alternative:

$$C: (P_1 \rightarrow Q_3), (P_2 \rightarrow Q_6), (P_3 \rightarrow Q_5), (P_4 \rightarrow P_6), (P_5 \rightarrow Q_1), (P_6 \rightarrow Q_4)$$

A mapping from the knob system to the bit system is quite simple given C, IF EVEN POSITION 0; ELSE 1. The point, noted earlier, is that a mapping between the states of a physical system and the states of an abstract machine will be simple insofar as there exists a natural individuation of the physical states under which the relevant counterfactuals hold. Intuitively, this is precisely the kind of system we should want an implementation criterion to favour.[20]

While I think this is sufficient to illustrate the general point, I would like to work through a specific case that has troubled opponents of triviality—the input record machine. Chalmers ([1996], p. 322) describes the machine as follows:

> Let us say a physical system contains an input memory if it has a subsystem that goes into a distinct state for every possible sequence of inputs (think of it as keeping a list of the inputs so far, perhaps). It then

[20] The matter of mapping inputs to interventions remains, but it is easy to see how the preceding reasoning extends to this case. Consider a system that requires one to perform different interventions to enter the same input depending on the present machine state. This conditional structure must be represented in the program implementing our mapping from inputs to interventions. By comparison, a system that allowed the same interventions across machine states would not require us to represent any such conditional structure. Hence, the more regularity there is in the interventions for each input, the more economically these interventions can be represented.

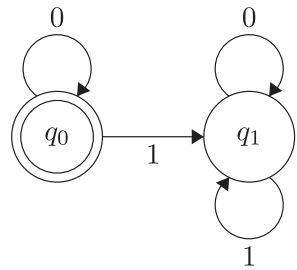

Figure 3. A finite state machine for recognizing the regular expression 0*. This machine will accept strings of any finite size consisting entirely of 0s.

> turns out that every physical system with an input memory and a dial implements every FSA [FSM] with the right input/output dependencies.

The state space of the input record is a tree of possible states with a depth equal to the size of the input memory and a branching factor equal to the length of the input alphabet. If each input character determines exactly one state transition, then it is easy to see how we can map the sequence of input record states traversed to the sequence of machine states traversed as the input is entered. However, if additional states are needed (as in a machine where a single input initiates a fully deterministic sequence of state transitions), then the dial (or clock) provides the physical differences necessary to support the individuation of the states within the deterministic sequence. Together, these two components allow the system to implement every FSM (or so the argument goes).

Fortunately, the simplicity criterion heavily penalizes such interpretations. That said, there are more immediate problems (as Chalmers acknowledges). FSMs can accept inputs of arbitrary size (Figure 3). For an input record of any size, there will be an FSM with an input that it cannot record, disrupting the mapping scheme. Another problem is that FSMs can loop indefinitely, so we'll need a reliable power source for our clock and a way to bend the laws of thermodynamics. However, these problems (as I'll discuss in a subsequent section) are not isolated to input recorders, and we will have to allow that physics places limits on the degree to which a physical system can approximate an abstract machine.

Even if we ignore these problems, there is ample reason to be suspicious of input record machines. Consider all the ways inputs to a simple hand calculator can result in the answer '4'. Every state in which the screen displays the numeral '4' is roughly the same (ignoring memory for simplicity) and sharply distinct from other states (such as the state where '5' is displayed). The same goes, *mutatis mutandis*, for the automaton implemented by the calculator. Many different sequences of inputs result in a single final state. Given the

sheer variety of input sequences that will result in the same final state, the states of the input recorder (unlike the calculator) will have little in common and few regularities which allow efficient individuation. Ironically, we may be able to economize on our representation of these disjunctive states by incorporating the calculator automaton itself into the program implementing our interpretation. In other words, the program could simulate the calculator, accept an input record as an input, and identify the ultimate state. While this approach saves the trouble of representing massive disjunctions of input states explicitly, it illustrates that all the hard work is being done by the interpretation itself—to the point that the interpretation includes the very system we wish to implement. As we saw earlier, the problem here is that the states of the input record do not have a natural individuation under which the calculator's counterfactuals hold.

Leaving aside the issue of inputs for simplicity, a program for S which simulates the target machine is very likely the worst-case scenario for the interpretation. After all, the program can simply extract the correct machine state sequence from a simulation of the machine and map a sequence of input or time-individuated physical states to the appropriate machine states. Again, it is telling that the upper bound on $K(I)$ is given by a program which simply simulates the target machine rather than relying in any interesting way on the behaviour of the physical system.

3 Some Lingering Concerns

3.1 Demarcation and the hard work of computation

As I just noted, in constructing a program to implement our interpretation of the input recorder as a simple hand calculator, we would likely find that our program would do all the work of the calculator. The reason for this is that if the program for I simulates M, then the program need not rely on P in any deep way. Where state changes in P have little in common with state changes in M, this approach represents a worst-case approach to computing the mapping between M and P. This point has also been made by Aaronson ([2011]). Instead of an input recorder, he is concerned with whether one might interpret a waterfall as a computer running a chess-playing algorithm. This bucket-style argument begins with the sheer complexity of the waterfall, then elicits the intuition that there must be some algorithm for using the waterfall as a chess-playing computer. Aaronson ([2011], p. 23, original emphasis) argues:

> Suppose we actually wanted to use a waterfall to help us calculate chess moves. How would we do that? In complexity terms, what we want is a *reduction* from the chess problem to the waterfall-simulation problem. That is, we want an efficient algorithm that somehow *encodes* a chess

> position P into an initial state $s_P \in S$ of the waterfall, in such a way that a good move from P can be read out efficiently from the waterfall's corresponding final state, $f(s_P) \in T$. But *what would such an algorithm look like*? We cannot say for sure—certainly not without detailed knowledge about f (i.e., the physics of waterfalls), as well as the means by which the S and T elements are encoded as binary strings. But for *any* reasonable choice, it seems overwhelmingly likely that any reduction algorithm would just *solve the chess problem itself*, without using the waterfall in an essential way at all!

Aaronson goes on to note that similar problems (trivial reductions) arise in the area of computational complexity. In assessing the time and space complexity of a program, it can be helpful to reduce one problem to another of known complexity. For example, suppose we want to know whether a particular problem, L, is NP-complete. We could ask whether a program could solve any NP problem, L', in NP time if it had access to an oracle that solves L. However, this wouldn't tell us anything about L since the program could just solve L' in NP time and ignore the oracle entirely. The lesson, Aaronson argues, is that to learn anything interesting about L from this exercise, we must find a program that can solve L' in less than NP time provided it has access to an oracle that solves L. Applying this insight to computer implementation, we could ask whether, given access to an oracle which simulates a physical system, we could lower the computational complexity of a program for simulating some machine.

While Aaronson's point seems entirely sound with respect to the kind of reductions he considers, my approach has a special advantage for the task at hand. Aaronson's approach can tell us whether a particular physical system could be used to do more computational work than could be done without it. This is a useful thing to know, but it does not tell us how the system accomplishes this work—that is, what kind of machine it is. Suppose that I built a machine, M_1, that solves a problem in exponential time that can be solved (using a different algorithm) in polynomial time. Suppose further that I built a different machine, M_2, that solves the problem in polynomial time. In each case, there would presumably be an efficient reduction of the problem to the problem of simulating either M_1 or M_2. After all, since both machines solve the problem, the reductions themselves could be identical and computationally trivial. Given this, we can see that although the criterion can tell us that both M_1 and M_2 are doing real work in solving the problem, it cannot tell us that M_1 or M_2 are different machines or what kind of machines they are.

An advantage of Aaronson's approach is that it offers a binary criterion for distinguishing trivial and substantive reductions. It would be advantageous if there were a similar criterion distinguishing trivial and substantive interpretations. Nevertheless, it may be that relative assessments are the best we can do.

Still, it is worth investigating whether something like Aaronson's approach can succeed for computational interpretations. Suppose that a trivial interpretation is one whose program is not shorter than the shortest program that could simulate the target machine. This (as it happens) would provide a natural interpretation of values of $K(I)/K(M)$ greater than one. As I noted earlier, a plausible worst-case scenario for an interpretation involves simulating M. If the shortest programs that implement I simulate M, then $K(I)$ must at least be equal to $K(M)$. Hence, a value of $K(I)$ less than $K(M)$ indicates that we are not dealing with this worst-case scenario, and more importantly, that we are relying on the physical system in more than a minimal way.

Nevertheless, there are some reasons to worry about the proposed demarcation. For automata with very simple transition functions, it may give the wrong answer. An interpretation has to represent quite a lot about the physical system (that $S(p_i) = m_i$, $C(x) = a_i$, and so on). This is something that a program specifying the machine could avoid, and even under auspicious circumstances, it may take more bits to accomplish than it would take to specify the machine directly. For example, even in the worst-case scenario, our program for I would have to include code for interpreting the physical states of the input record. There are a number of other complications here, but I'll leave these worries for another time. In any case, it is certainly too early to give up on a criterion for demarcating trivial and substantive interpretations.

3.2 Imperfect implementation

One problem for any approach to computer implementation is that physical systems can only imperfectly implement abstract machines. For example, there is nothing to prevent us from specifying an automaton that can go on changing states indefinitely. However, any physical system will be subject to the laws of thermodynamics and ultimately cease functioning. Still, there are good reasons to treat physical computers as if they will go on functioning forever. For one, the working lifespan of a computer is (with any luck) long relative to the time requirements of particular programs. While some programs have to run for hours or days, this is not brushing up against the working lifespan of most computers, much less the heat death of the universe. There are also very common types of automata (including UTMs) that technically require infinite memory. Any UTM-like machine with finite memory would fail to be genuinely universal. In fact, any machine with finite memory can be understood as an FSM and (strictly) has all the computational limitations pertaining thereto. That said, there are good reasons to interpret memory-limited physical systems as UTMs. First, because they are architecturally more similar to UTMs than to FSMs. Second, because they often have expandable memory. This latter property should be familiar to anyone who

has installed more RAM in order to run a demanding program. Hence, while memory at a time is limited, we can expand memory without radically changing the architecture of the system. Under the Turing machine formalism, there is a natural interpretation of expanded memory (longer tape), whereas the monadic treatment of states under the FSM formalism makes it harder to incorporate new memory without substantially revising the specification of the machine itself. Clearly, it is important to make room for idealizing assumptions in our account of computer implementation.

While it is beyond the scope of this article to fully develop a scheme for incorporating these assumptions, I believe a straightforward strategy holds a great deal of promise: simply incorporate a specification of the idealizing assumptions into our interpretation. On this approach, *I* becomes an ordered triple comprising two functions and a description of the necessary idealizing assumptions. Given an appropriate means of encoding these assumptions, interpretations that require fewer idealizing assumptions will be favoured over those that do not, all else being equal. Regardless of how it is accomplished, this result seems like the key desideratum of any account of imperfect implementation.

3.3 Practical difficulties

While I have already dealt with the difficulty of encoding fine-grained physical states, other practical difficulties remain. It is important to note that practicality need not be an aim of a theory of implementation. In fact, it would be unfortunate if the criterion relied in any essential way on practical concerns. The reason for this is that practical concerns are generally not universal. What is impractical for one kind of organism might be eminently practical for another. Tracking insects by listening for the echoes of our own shrieks is impractical for human beings, but as it turns out, this tells us more about the limits of our sensory and vocal apparatuses than it does about whether such echoes could carry useful information about the position of insects. Moreover, given that the nature of mental states is at stake (via computational functionalism), it would be strange to discover that whether or not an organism has a particular mental state turns on the practical limits faced by the human species. What we need is a theory that can distinguish in principle between different computational interpretations.

Of course, given that we make considered judgements about which systems implement which machines and given that we take these judgements as the foundation of our theory of implementation, it is important that we can reliably make judgements consistent with the criterion we endorse in reflective equilibrium. Given that we have based our theory on only the clearest and most central of these judgements, it is not necessary to suppose that we can

make reliable judgements in all cases, much less those which we ourselves regard as less obvious or less central. Fortunately, the simplicity criterion is well suited to this kind of practical use.

One of the main problems for any use of Kolmogorov complexity is the fact that K is not computable. That said, it is upper semicomputable, and computable approximations of K exist (Li and Vitányi [2008]). Of course, these theoretical limits on the computability of K do not mean that the value of K cannot be computed for any particular interpretation. The halting problem is also uncomputable, but that does not mean that we cannot prove that certain programs will or will not halt. Moreover, uncomputability does not entail that we cannot make justifiable claims (that fall short of proof) about the relative complexity of different interpretations.[21] This is especially clear when those interpretations differ in substantial ways—as I attempted to demonstrate in my earlier defence of the criterion.

While the ability to determine the relative value of K for different interpretations is important, there is also good reason to believe that the criterion will track differences in the practical value of physical computers. Earlier, I drew an analogy between computational interpretations and encryption–decryption algorithms. Just as these algorithms represent the competence one must have to fully use the code, computational interpretations represent the competence one must have to fully use a physically system as a computer.[22] Given this, it seems reasonable to believe that the simplicity of interpretations will correlate with the practical value of physical computers. In short, there is a reason that engineers prefer silicon chips to rocks or buckets of water, and this reason is closely related to the ease of interpreting these systems as computers.[23]

4 Conclusion

I have defended a formal simplicity criterion for physical computation. While the criterion may pose practical difficulties in ambiguous cases, it clearly disfavours classic trivializing interpretations. It also generally favours intuitively reasonable interpretations. The criterion has the additional advantage of

[21] As an anonymous referee observed, there are convincing reasons to believe that good compression algorithms, such as prediction by partial matching, will achieve near optimal compression for natural data.

[22] By 'fully use', I mean to distinguish the way hardware engineers use computers from the ways that other users do, such as software engineers or casual users.

[23] There are important differences between the representation of I needed to compute K(I) and the representation of I needed for human competence. For example, the minimal program for I may take a very long time to run even though its code is relatively short. The strategies employed by human engineers in understanding physical computers are likely to be seriously constrained by time and memory requirements. Even so, it is difficult to imagine a case where a trivializing interpretation would be easier to understand and apply than a conventional one. In this at least, an assessment of practical value agrees with an assessment under the proposed criterion.

admitting degrees of goodness in computational interpretations, while (i) providing a rigorous analysis of goodness and (ii) allowing us to rank the relative goodness of different interpretations. This criterion has intriguing implications for the philosophy of mind. If conscious or intentional states are computational states of the brain, then organisms may possess these states to different degrees. In many cases, differences between species will be best understood as differences in the kind of machine they implement or computations they carry out. However, within closely related species or in analysing particular cognitive modules, the differences may best be represented as differences in the degree to which two organisms implement a single machine. This would allow us to quantify differences in lower-level functionality that resist higher-level descriptions.

The implications of the criterion might go well beyond the philosophy of mind and the theory of computation. For example, it may also contribute to our understanding of ontological commitment in the special sciences. While my account draws on ideas from the literature on real patterns, it offers a more substantial account of what it means for a physical system to realize an abstract, higher-level model of its dynamics. Unlike black-box predictive models, the models I consider must support a mapping from physical states to states postulated by the model. The model then supplies a model of the dynamics of these higher-level states. Hence, these models are better understood as glossing the lower-level dynamics of a physical system than postulating unobserved states or entities. For this reason, models that permit such mappings can avoid the kind of instrumentalism that often casts ontological doubt on merely predictive models. For example, suppose we had two equally predictive high-level models of a physical system that differ in the entities and processes they propose. Which of these best cuts nature at the joints? If we can formalize these models as automata, we can use the simplicity criterion to answer this question in a way that does not reduce to a comparison of predictive value. While traditional model selection might be more appropriate to selecting the best predictive model, this alternative might provide a means of formalizing our intuitions about natural kinds and ontological commitment in the special sciences. Of course, this is all speculative and demands a separate and careful treatment. The primary result remains—that a simplicity criterion based on algorithmic information distance appears well suited to addressing triviality arguments and to providing a rigorous account of computation in physical systems.

Acknowledgments

I am indebted to Shaun Nichols, Scott Aaronson, Kathleen Kuo, David Chalmers, Jenann Ismael, Jonathan Weinberg, Juan Comensaña, Jason Turner, Arthur Breitman, and two anonymous reviewers for their comments

on earlier drafts of this article. Any remaining errors or omissions are entirely my own.

Department of Philosophy
University of Arizona
Tucson, AZ, USA
tylermillhouse@email.arizona.edu

References

Aaronson, S. [2011]: 'Why Philosophers Should Care about Computational Complexity', available at < arxiv.org/abs/1108.1791>.

Bennett, C. H., Gács, P., Li, M., Vitányi, P. M. and Zurek, W. H. [1998]: 'Information Distance', *IEEE Transactions on Information Theory*, **44**, pp. 1407–23.

Block, N. [1981]: 'Psychologism and Behaviorism', *The Philosophical Review*, **90**, pp. 5–43.

Brown, C. [2012]: 'Combinatorial-State Automata and Models of Computation', *Journal of Cognitive Science*, **13**, pp. 51–73.

Chalmers, D. [2012]: 'The Varieties of Computation: A Reply', *Journal of Cognitive Science*, **13**, pp. 211–48.

Chalmers, D. J. [1996]: 'Does a Rock Implement Every Finite-State Automaton?', *Synthese*, **108**, pp. 309–33.

Chaitin, G. J. [1966]: 'On the Length of Programs for Computing Finite Binary Sequences', *Journal of the ACM*, **13**, pp. 547–69.

Dennett, D. C. [1991]: 'Real Patterns', *The Journal of Philosophy*, **88**, pp. 27–51.

Godfrey-Smith, P. [2009]: 'Triviality Arguments against Functionalism', *Philosophical Studies*, **145**, pp. 273–95.

Grünwald, P. and Vitányi, P. [2004]: 'Shannon Information and Kolmogorov Complexity', available at < arxiv.org/abs/cs/0410002>.

Katz, J. and Lindell, Y. [2007]: *Introduction to Modern Cryptography*, Boca Raton, FL: CRC Press.

Kolmogorov, A. N. [1965]: 'Three Approaches to the Quantitative Definition of Information', *Problems of Information Transmission*, **1**, pp. 1–7.

Ladyman, J. and Ross, D. [2009]: *Every Thing Must Go: Metaphysics Naturalized*, Oxford: Oxford University Press.

Li, M. and Vitányi, P. [2008]: *An Introduction to Kolmogorov Complexity*, New York: Springer.

Piccinini, G. [2007]: 'Computational Modelling vs. Computational Explanation: Is Everything a Turing Machine, and Does It Matter to the Philosophy of Mind?', *Australasian Journal of Philosophy*, **85**, pp. 93–115.

Piccinini, G. [2009]: 'Computationalism in the Philosophy of Mind', *Philosophy Compass*, **4**, pp. 515–32.

Piccinini, G. [2015a]: 'Computation in Physical Systems', in E. N. Zalta (*ed.*), *The Stanford Encyclopedia of Philosophy*, available at < plato.stanford.edu/archives/sum2015/entries/computation-physicalsystems/>.

Piccinini, G. [2015b]: *Physical Computation: A Mechanistic Account*, Oxford: Oxford University Press.

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

Scheutz, M. [2012]: 'What It Is Not to Implement a Computation: A Critical Analysis of Chalmers Notion of Implementation', *Journal of Cognitive Science*, **13**, pp. 75–106.

Shannon, C. [1948]: 'A Mathematical Theory of Communication', *The Bell System Technical Journal*, **27**, pp. 379–423, 623–56.

Shannon, C. [1956]: 'A Universal Turing Machine with Two Internal States', in C. Shannon and J. McCarthy (*eds*), *Automata Studies*, Princeton, NJ: Princeton University Press, pp. 157–65.

Sipser, M. [2013]: *Introduction to the Theory of Computation*, Boston, MA: Cengage Learning.

Solomonoff, R. J. [1964]: 'A Formal Theory of Inductive Inference, Part I', *Information and Control*, **7**, pp. 1–22.

Sprevak, M. [2012]: 'Three Challenges to Chalmers on Computational Implementation', *Journal of Cognitive Science*, **13**, pp. 107–43.

Tolstoy, L. [2000]: *Anna Karenina*, New York: Random House.

Wallace, C. [2005]: *Statistical and Inductive Inference by Minimum Message Length*, New York: Springer.

Wallace, D. [2014]: *The Emergent Multiverse*, Oxford: Oxford University Press.