2014

Objective computation versus subjective computation

Nir Fresco
Hebrew University of Jerusalem, nfresco@uow.edu.au

Publication Details

Fresco, N. (2014). Objective computation versus subjective computation. Erkenntnis: an international journal of analytic philosophy, *Online First*

Objective computation versus subjective computation

Abstract
The question 'What is computation?' might seem a trivial one to many, but this is far from being in consensus in philosophy of mind, cognitive science and even in physics. The lack of consensus leads to some interesting, yet contentious, claims, such as that cognition or even the universe is computational. Some have argued, though, that computation is a subjective phenomenon: whether or not a physical system is computational, and if so, which computation it performs, is entirely a matter of an observer choosing to view it as such. According to one view, which we dub bold anti-realist pancomputationalism, every physical object (can be said to) computes every computer program. According to another, more modest view, some computational systems can be ascribed multiple computational descriptions. We argue that the first view is misguided, and that the second view need not entail observer-relativity of computation. At least to a large extent, computation is an objective phenomenon. Construed as a form of information processing, we argue that information-processing considerations determine what type of computation takes place in physical systems.

Disciplines
Engineering | Science and Technology Studies

Publication Details
Fresco, N. (2014). Objective computation versus subjective computation. Erkenntnis: an international journal of analytic philosophy, *Online First*

Objective Computation versus Subjective Computation[#]

Nir Fresco[1]

Abstract. The question 'What is computation?' might seem a trivial one to many, but this is far from being in consensus in philosophy of mind, cognitive science and even in physics. The lack of consensus leads to some interesting, yet contentious, claims, such as that cognition or even the universe is computational. Some have argued, though, that computation is a subjective phenomenon: whether or not a physical system is computational, and if so, which computation it performs, is entirely a matter of an observer choosing to view it as such. According to one view, which we dub bold anti-realist pancomputationalism, every physical object (can be said to) computes *every* computer program. According to another, more modest view, some computational systems can be ascribed multiple computational descriptions. We argue that the first view is misguided, and that the second view need not entail observer-relativity of computation. At least to a large extent, computation is an objective phenomenon. Construed as a form of information processing, we argue that information-processing considerations determine what type of computation takes place in physical systems.

1. Introduction

Some maintain that the claim that computation is objective is either false or misguided. For example, both Hilary Putnam (1988) and John Searle (1990) have claimed, roughly, that every physical object can be *described* as implementing any number of programs and, hence, as computing every Turing-computable function. This claim leads to bold anti-realist pancomputationalism[1]. At least prima facie, it seems that if *every* physical object can be described as computing every Turing-computable function, then the phenomenon of physical computation is less interesting in its own right. Nevertheless, a better understanding of what computation is, and in

[#] This is a preprint of the article that appears in *Erkenntnis* (DOI: 10.1007/s10670-014-9696-8). It is reproduced with the permission of Springer-Verlag. The final publication is available at http://link.springer.com/article/10.1007/s10670-014-9696-8.

[1] Sidney M. Edelstein Centre for History & Philosophy of Science, Technology & Medicine. The Hebrew University of Jerusalem. Email: nir.fresco@mail.huji.ac.il

particular digital computation, helps show why some pancomputationalist arguments are wrong and that bold pancomputationalism is unfounded.

The observer-relativity of computation can also supposedly be derived from the less contentious view that multiple computational descriptions can be ascribed to a single computational system. For example, a basic AND-gate (under its conventional interpretation) could be just as well described as an OR-gate (Bishop 2009, p. 228; Shagrir 2001, p. 374; Sprevak 2010, pp. 268–269) (see Table 1). Which function does the gate compute then? "[T]here is nothing that decides between the two options. No physical, structural, or functional property [would help] decide [… in this matter]" (Sprevak 2010, p. 269). "[P]ositing processes and structures is necessary if an explanation is to be a computational explanation" (Dietrich 1989, pp. 130–131). Yet, "the precise function of the system-as-a-whole remains fundamentally observer- relative" (Bishop 2009, p. 228).

AND	In1	In2	Out
	1	1	1
	0	1	0
	1	0	0
	0	0	0

OR	In1	In2	Out
	0	0	0
	1	0	1
	0	1	1
	1	1	1

Table 1. Truth tables of an AND-gate and an OR-gate "mirroring" one another when logical 0s and 1s are swapped.

Whatever the "correct" account of digital computation is (Copeland 1996; Fresco and Wolf 2014; Piccinini 2007; Turing 1936), we argue here that, at least to a large extent, computation is an objective phenomenon. Nontrivial computation in memory-based systems is an objective phenomenon, whereas the boundaries of objectivity in memoryless system computation are blurry as is shown below. It would also seem that the cognitivist should resist the purported observer-relativity of computation. For if cognition is to be explained computationally, but computational

systems require a cognitive observer to determine what computation is performed, then the explanation is circular.

At least two related questions arise in relation to the purported subjectivity of computation and they should be addressed separately. The first one is whether computational descriptions are trivial or vacuous. The second question, which need not presuppose either *bold* or *weak* pancomputationalism (i.e., the thesis that every physical object computes at least one Turing-computable function), is what fixes the computational identity of a computational system. Indeed, some physical systems *may not* compute. But if *some* computational systems compute more than one Turing-computable function, what is the criterion for fixing the "correct" computational identity of the system? (cf. Table 1). We argue that the multiplicity of computational *descriptions* in itself does not imply that these descriptions are *incompatible* or that the computational identity of the system concerned is *unfixed*. Either of these implications would have been problematic for the objectivity of computation.

To ground the discussion, we begin (Section 2) by construing 'computation' as a form of information (or data) processing, and then reply to an important objection. In Section 3, we present the bold anti-realist pancomputational view and argue against it. In Section 4, we discuss the 'multiple computational identities' argument and examine the implications of this argument for genuine computational systems. Section 5 concludes the article.

2. Computation as a Form of Information Processing

2.1. Nontrivial versus trivial computation

According to the instructional information processing (IIP) account, *nontrivial* computation is the processing of discrete data in accordance with finite instructional information (possibly) through discrete state transitions. Digital computational systems send, receive and process data. *Trivial* computational systems process data, which need not be structured, by way of exercising a single

capacity. *Nontrivial* computational systems, on the other hand, process instructional information under the right conditions by way of being capable of systematically processing at least two distinct imperative instructions (Fresco and Wolf 2014).

This characterisation requires some unpacking. First, roughly speaking, a datum is the lack of uniformity between at least two distinct variables. Data are not defined in the alphanumeric sense of the word, but in the sense of either differences *de re* (i.e., as lack of uniformity in the real world) or differences *de signo* (i.e., as the lack of uniformity between at least two signals). For our purposes here, data are taken as differences *de signo*. Examples include a higher or lower charge in a battery, a variable electrical signal in a telephone conversation and even the presence or absence of noise on a communication line (Floridi 2011, pp. 85–86).

Second, instructional information is nonempty, well-formed and meaningful data that if satisfied through unambiguous action yield a particular outcome within a system in a given context. Put differently, well-formed and meaningful data carry instructional information *i* within a system *S iff* when *S* acts in accordance with *i* under the right conditions, *S* systematically performs an action in a way that depends on the structure of *i*. This shows that *S* can process two types of data: one is the "raw" data acted upon and the other is information that instructs *S* how the former has to be processed. The latter must be structured, whereas the former can be either structured or unstructured (for more details, see (Fresco and Wolf 2014)). Importantly, the structure of instructional information exists in concert with other "raw" data: at least some structure in the overall collection of input data is required for specific data to carry instructional information (Fresco and Wolf unpublished-a).

Third, we note that whilst a single datum seems, at first blush, unstructured, in the context of computational systems, it is structured. Indeed, when two or more data exist, there is the potential for a non-reflexive relationship obtaining amongst those data, whereas a single datum can only have reflexive relationships. Yet, a single bit is all that is required, for example, to issue either a stop or

resume command within *S*. And if a single bit can carry instructional information, on the definition above, it ought to be structured. We stipulate that a single datum is *trivially structured*. This sort of stipulation is akin to defining an entity with zero symbols as a string, namely the empty string: the simplest case is defined in such a way as to allow for consistent treatment (Fresco and Wolf unpublished-a).

Instructional information is prescriptive rather than descriptive: the execution of an action is neither true nor false. Hence, it cannot be straightforwardly qualified as true or false. One might argue that 'satisfied instructional information' is tantamount to 'true instructional information'. This claim, however, is misguided. The conflation of 'being-satisfied' and 'being-true' may certainly result from viewing the satisfaction of instructional information as singling out the set of possible worlds in which the relevant state of affairs obtains (Hamblin 1987, p. 152). For example, the instruction 'turn off the air conditioner', call it *Stop-AC*, is satisfied in those worlds in which the air conditioner is turned off. Yet, the air conditioner being turned off could be the result of a power outage, for example, rather than the recipient of *Stop-AC* executing the necessary action (turning off the AC) to achieve the particular outcome (the AC is stopped). Neither the instruction nor the action *Stop-AC* prescribes is straightforwardly alethically-evaluable. Rather, instructional information is either effective or ineffective depending on the contextual success criteria.

Computational systems that have at least two capacities (typically) require instructional information to single out the action to perform. "[T]he characteristic purpose of imperatives is to influence choice. It is futile to tell [...] anyone to do something [...] he cannot help doing" (Hamblin 1987, p. 145). The selection between these capacities is enabled by instructional information. A capacity is an action a system, whether abstract or physical, *can* reliably and systematically execute under the right conditions. Thus characterised, the distinction between *trivial* and *nontrivial* computational systems can now be articulated. A nontrivial computational system has at least two capacities and it selects between them by processing instructional information in a way that

systematically depends on the structure of that information (for a discussion on exceptions to this rule see (Fresco and Wolf unpublished-b)). A trivial computational system, on the other hand, has only one capacity: it is capable of only one action, and, therefore, need not process instructional information.

A critic might argue that *actions*, too, are observer-relative in a manner akin to computation[2]. Any given system can be viewed as performing any number of actions in several ways that go against our intuitive individuation of actions. There are four cases that result from the synchronic/diachronic and one-to-many/many-to-one distinctions. In the *diachronic/one-to-many* case, the same action (e.g., jumping or relaying high voltage only if all inputs are high voltages, call it RHV) that is performed at two different times could also be viewed as two different actions (first jump and second jump or first RHV and second RHV). In the *diachronic/many-to-one* case, an action (e.g., jumping or RHV producing low voltage) followed by another action (e.g., running or RHV producing high voltage) could instead be viewed as a single action (e.g., exercising or ANDing). (Similar examples can be constructed in the case of synchronic identity.)

The individuation of actions that potentially gives rise to an unlimited proliferation of actions has been discussed in the literature at length (for a discussion, see, e.g., Mackie (1997)). As is well-known, the act of a man moving his arm, thereby operating a pump, which pumps poisoned water in the house's water supply, thereby poisoning the inhabitants can be described as multiple or a single action(s). But actions can instead be individuated in terms of their results, and their spatiotemporal and causal properties can be determined by the properties of the results. By adopting an act-individuation principle according to which actions are identical *iff* they have the same results, the unlimited proliferation of actions can be avoided (Mossel 2001, pp. 258, 264). Poisoning the inhabitants is a *result* of the replenishing of the water supply, but an *effect*, rather than a result, of the man moving his arm. Similarly, the data processing actions underlying trivial and nontrivial

computations are individuated in terms of their data (and possibly information) processing results (e.g., low voltage output or transitioning into a new state).

Let us consider a few representative cases of trivial and nontrivial computational systems showing the delineation between classes of computational systems. Standard two-input, one-output Boolean gates are trivial computational systems, because they operate on *unstructured* data[3] and have only one capacity. Neither input bit of an AND-gate, for example, has any special role in determining the action of the gate. No ordering relation need be imposed on the data processed: swapping the order of the input bits still yields the same output. The AND-gate has a single capacity, which is to perform logical conjunction whenever two input data are present.

Next, we consider computational systems that process structured data. The input data to an n-bit adder, for example, are *structured*: the 0^{th} bits are separate from the 1^{st} bits and so on. It is, therefore, more computationally powerful than standard two-input, one-output Boolean gates. Still, it only performs trivial computation, as it only has a single capacity, which does not require instructional information to be exercised. An arithmetic logic unit (ALU), which is capable of more than one operation, say, addition and conjunction, is a nontrivial computational system, for it processes instructional information. Instructional information is required for the ALU to select between the two operations to perform. (In the next section, we qualify this characterisation of the ALU.) All the preceding cases are *memoryless* systems.

The next level of complexity is that of memory-based computational systems, such as flip-flops, finite state automata (FSAs), Turing machines (TMs) and physical instantiations thereof. These systems process instructional information in performing computation, but they are also capable of *retaining* their state (e.g., a flip-flop is the most basic computer memory cell) and their *present* state affects their *next* state. In memory-based computational systems, their memory model implies a structure to the data they process (Fresco and Wolf 2014). At each step of a TM computation, for example, the combination of the scanned symbol (being structured data) and the state of a

(deterministic) TM uniquely determines the capacity to be exercised next. This combination is instructional information, the processing of which leads to the execution of a capacity of a TM by moving from its current memory configuration to another. An FSA is different from a TM in that the former has a more restricted set of capacities. The contents of the FSA's input tape remains unchanged in the course of the computation, and the position of its *read-only* head shifts only to the right by exactly one position.

2.2. A reply to an objection

A critic might argue that such an account of nontrivial computation relies on a distinction between structured and unstructured data that is *itself* observer-relative. Consider a two-function logic unit (FLU) that performs either a logical AND or XOR. Its truth table is depicted in Table 2.

In1	In2	In3	Out
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

Table 2. A binary truth table of a two-function logic unit capable of AND or XOR.

Under this description (2-FLU_{AX}), the rightmost input-bit ('In3') is a control bit: if it is 0, perform XOR on the other input bits ('In1' and 'In2'); otherwise, perform AND on those bits. But, arguably, the very same truth table can describe a *single capacity* circuit, and under this description, the circuit is a memoryless trivial computational system. This, supposedly, shows that the IIP account fails to provide an observer-independent explanation of digital computation.

Our reply to this objection is that it shows that computational descriptions of *some* memoryless computational systems – rather than memory-based nontrivial computational systems – are observer relative. The same information – or data – processing operations may be *used* in different ways,

giving rise to different "computations" relative to different levels of abstraction (see Floridi 2011, Chapter 3).[4] The structuring of data is not an intrinsic property of data, rather it is imposed by the physical system processing the data. Nevertheless, the underlying computation at the *physical level* remains the same. This is akin to measuring the length of a piece of metal using either a yardstick or a metrestick. The two measurements yield different numerical results, but the objective physical length of the piece of metal remains the same.

It is simply the nature of memoryless computational systems that they can be described differently by appealing to a single abstract specification of the system, such as a truth table. Consider a truth table of an FLU capable of performing either AND or OR (rather than XOR as depicted in Table 2), call this description 2-FLU_{AO}. Under this description, the leftmost input bit ('In1') is a control bit (recall the observation above about structure in the overall collection of input data): if it is 0, perform AND on the other input bits ('In2' and 'In3'); otherwise, perform OR on those bits.

In1	In2	In3	Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	1	0	1
1	0	1	1
1	1	1	1

Table 3. A binary truth table of a two-function logic unit capable of AND or OR.

Here, too, the very same truth table can be used to describe a *single capacity* circuit that computes the 2-threshold function (2-THR): if there are two or more 1s as input, produce 1 as output; otherwise, produce 0 as output.

Prima facie, there is no reason inherent to the truth table description in Table 3 that privileges 2-FLU_{AO} over 2-THR, or vice versa, as the "correct" description of the computation the circuit performs. But, according to the IIP account, by adopting Occam's razor, we should opt for 2-THR:

structure should not be posited when it is not required. 2-THR provides a shorter description of the circuit than 2-FLU_{AO}. This is readily demonstrated by observing that the 2-FLU_{AO} description requires the introduction of additional (meta-)information, namely the structuring of the data and the identification of the control bit, whereas the description of the circuit as 2-THR does not require the introduction of any such information.

Many problematic cases of memoryless computational systems can be resolved elegantly by appealing to "shorter-length" descriptions rather than just to "simpler" descriptions, in the spirit of Occam's razor. The advantage of the former is the applicability of Kolmogorov Complexity for "large" enough systems in deciding between two given computational descriptions, such as 2-FLU_{AO} and 2-THR. The simplest amongst competing computational descriptions is determined by finding the one whose program-length is shortest (for a detailed discussion, see (Fresco and Wolf unpublished-b)). Occam's Razor demands that we adopt the simpler theory that posits the least structure unless there is evidence to the contrary. (Of course, there is no guarantee that the simpler explanation is always the correct one.)

These examples show that some *memoryless* computational systems can be described as performing *either* trivial or nontrivial computation. A single capacity circuit description is preferable over multiple-capacity descriptions, because the former is, generally, shorter. Clearly, some computational problems cannot be solved by memoryless computational systems, since they require stateful transitions and the storage of data from previous states. In such cases, for the implementing physical system to transition from one memory configuration to another where more than one possible transition exists, instructional information is required to induce the correct, systematic transition. (We elaborate more on this point in Section 4.2.)

FSAs and TMs, for example, are not subject to the same objection. The truth table describing *all possible input combinations* of an FSA is *infinite*, whereas a truth table describing, say, a two–output, one–output Boolean function is finite. An infinite truth table cannot be plausibly described

as a single capacity (otherwise, exercising that capacity would require traversing through infinitely many paths). It is the finite transition table of the FSA that describes its multiple capacities. For that reason, the IIP account classifies FSAs, TMs, flip-flops and conventional digital computers objectively as memory-based nontrivial computational systems.[5]

3. A Reply to Bold Anti-Realist Pancomputationalism

With a better understanding of digital computation as a form of information processing we can turn to examine the claims concerning the subjectivity of computation starting with bold pancomputationalism. The view described here should be distinguished from the computational view of the universe, according to which the universe not only has the *capacity* to implement any program, but it *actually* performs computations all the time (Fresco and Staines 2014). The former view is also known as anti-realist pancomputationalism, which is commonly used to argue against the Cognitivist position (Dodig-Crnkovic and Müller 2011, p. 154). This anti-realist version of pancomputationalism is the target of this article, since it claims that it is the observer's description of a physical object that makes it into a computational one.

3.1. Background

According to bold pancomputationalism, every physical object (can be said to) computes every Turing-computable function. In arguing against the Cognitivist position, Searle has pointed out that the very idea of multiple realisability is problematic for the cognitivist. Whilst the true threat of his argument is not pancomputationalism as such, but rather the observer relativity of computation, his argument can be taken to support bold anti-realist pancomputationalism. The Searlean Pancomputational Thesis may be stated as follows. For any sufficiently complex physical object *O* (that is, an object with a sufficiently large number of distinguishable parts) and for any arbitrary program specification *P*, there exists an isomorphic mapping *M* from some subset *S* of the physical states of *O* to the formal structure of *P* (Searle 1990, pp. 27–28). With the added assumption that

such an isomorphism is sufficient for *O* to realise *P*, Searle's result, if it were true, would imply that every sufficiently complex physical object computes every Turing-computable function.

How is this thesis justified? For any physical object to be "sufficiently complex" it simply needs to have a sufficiently large number of distinguishable parts. A big enough wall, for instance, would do the trick. That wall may then be physically described at a microscopic level specifying some movement pattern of some large enough number of molecules. Any arbitrary program, such as WordStar, has some specification that describes its formal structure and this structure can be implemented by any number of physical substrates as dictated by the principle of multiple-realisability. "The physics [of the implementing substrate] is irrelevant except insofar as it admits of the assignments of 0's and 1's and of state transitions between them" (Searle 1990, p. 26).

What remains to be seen is how *M* is fixed by the implementation of *P*. Here Searle was (mis)guided by Alan Turing's analysis of the human computor executing an algorithm. The TM goes through the steps of *P* based on the rules *P* provides for deriving the output symbols from the input symbols. This process is causal insofar as some physical properties of *O* (implementing some FSA or a TM) suffice for *O* to follow these steps (Searle 1990, pp. 32–33). *M* maps the states in *S* onto the abstract states of *P*. Since, by Searle's lights, *M* need not be a one-to-one mapping, it suffices that only the states in *S* partake in the mapping from *O* to *P*. The requirement that *O* have a sufficiently large number of distinguishable parts supposedly guarantees that such a subset *S* exists. Accordingly, a (large enough) wall is bound to have the right pattern of moving molecules that "mirrors" the state transitions of WordStar (or any other *P*) (Searle 1990, pp. 27–28).

In a similar vein, Putnam's Pancomputational Theorem can be stated as follows. There exists a physical theory *P* such that for any ordinary open system[6] *K*, an inputless FSA δ, $n > 0$ (the number of computational steps of δ) and for any (divisible) real-time interval *I*, *K* (describable in *P*) realises *n* computational steps of δ within *I*. According to Putnam, *K* realises *n* computational steps of δ within a given interval *I* of real time, if there exists a one-to-one mapping *F* from δ's state types

onto physical state types of ***K*** and a division of *I* into n subintervals such that for any two states q, p of δ the following condition holds: if q → p is a transition of δ from the computational step *j* to *j*+1 (for $0 < j < n$) and ***K*** is in state *F*(q) during the j^{th} subinterval, then ***K*** transitions into state *F*(p) in the $j+1^{th}$ subinterval (Scheutz 1999, pp. 166–167).

How is this theorem justified? Putnam argues that insofar as *K* has a sufficient number of internal states[7] it will be a realisation of any FSA. His example is an FSA that transitions between two computational states *A* and *B* producing the sequence *ABABABA*. The internal state of *K* is considered at seven points in time (e.g., 12:00, 12:01, … 12:06) yielding seven states s_1, s_2, … s_7. Each s_i is assumed to be caused by s_{i-1}. State *A* of δ may be defined as the disjunction $s_1 \vee s_3 \vee s_5 \vee s_7$ (and similarly for *B* and the remaining states of *K*). *K* then implements δ from 12:00 to 12:06 (inclusive). Crucial to his proof are two assumed principles. The first is the *principle of continuity*, which states that forces, such as electrical and gravitational fields, that may influence ***K*** are continuous. The second is the *principle of non-cyclical behaviour*, which states that there exists some natural clock that constantly stamps the system in such a way that ***K*** cannot have a duplicated physical state and enter the same (maximal) state twice. Accordingly, there are *only* one-to-one mappings of the physical states of ***K*** to the computational states of δ. For once *K* transitions out of a particular physical state it does not return to that state, due to forces that irregularly perturb ***K***. Putnam asserts that his proof can be generalised to apply to an arbitrary number of computational states and state transitions of an FSA, as well as for FSAs *with input*.

3.2. A Reply to Bold Anti-Realist Pancomputationalism

Neither Searle nor Putnam restricts the physical objects under consideration to computing only *some* Turing-computable functions. As long as the object has sufficient internal physical states it may implement any FSA. And since the physical states are chosen arbitrarily (there is no restriction on selecting atomic or subatomic particles), any "macroscopically small" object will do. We start

with two well-known objections. The first one is the failure to account for counterfactuals (cf. (Block 2002; Chalmers 1996; Chrisley 1994; Copeland 1996)). Inherent to Searle and Putnam's arguments are the lack of any constraints on physical state type formation (Scheutz 2012, p. 100) but also the *ex post facto* nature of their computational descriptions.

The physical states of Searle's wall or any "clocked" open system are chosen according to the computation *actually* performed. But counterfactuals are important to physical computational systems. (Nevertheless, some argue that it would be preferable to dispense with counterfactuals in an account of physical computation altogether (Scheutz 2012, p. 81).) Any particular sequence of state-transitions is only one (of possibly many) that could have occurred. A computational description should also allow all the *possible* computations, which the system *could have* performed, rather than just the one it *actually* performs. The more complex the computational system is, the more counterfactuals there are.

The second objection is an undesirable consequence of Searle's claim that syntax is *observer-relative*, rather than *intrinsic* to physics. The consequence is that even conventional digital computers do not compute. Logical 0s and 1s that are typically associated with the operation of physical computational systems are not intrinsic to the physics of the labelled states. There are no discrete binary states intrinsic to the physics of an iMac, iPhone or IBM's *Jeopardy!* player Watson. Binary labels are externally assigned according to whether some specific variables, such as voltage levels, degrees of magnetisation or the presence/absence of water droplets, fall within one predefined range or another. "If the truism that syntax is not intrinsic to physics implies that brains are not 'intrinsically digital computers' then by parity it implies that *no* entity is intrinsically a digital computer" (Copeland 1996, p. 356).

A better understanding of digital computation as a form of information (or merely data) processing gives rise to two other objections to bold pancomputationalism. Additionally, it motivates the computational objectivist view. The claim at the core of these objections is that a

finite physical object can only support a finite number of computations simultaneously. The result is that *bold* pancomputationalism cannot be true for finite physical objects. Two similar arguments support this claim and the first one can be summarised as follows.

1. A finite physical object has the capacity to store a finite amount of information[8].
2. Digital computation – as the processing of information – requires storage of information.
3. Simultaneous distinct computations (also) require the storage of distinct information.
4. (Therefore, a finite physical object can only support a finite number of distinct computations simultaneously.)

Since the locutions 'finite', 'storage of information' and 'simultaneously' play a key role in the ensuing discussion, they require some explication. The notion of finiteness is easiest to demonstrate in mathematics and more specifically in set theory. A set is finite if its cardinality is a positive integer. For example, the cardinality of {a, b, c, d, e} is 5. The set of natural numbers is *enumerable*, but its cardinality cannot be given as a positive integer. It is, therefore, infinite. Finiteness implies not only bounds but also minima and maxima. Whilst the description of a single TM is *finite*, its tape is *infinite*. The TM's tape, in principle, has no bounds – it can always be extended if needed. After a finite amount of time, the portion of the tape that has been used in the computation is *always* finite. When considering the physical realm, a region of space is finite if the set of all distances from its points to some fixed point is bounded.

Next, we consider the locution 'storage of information'. We use information storage systems every day: dictionaries, telephone directories and computer discs. Each discrete piece of information is stored separately, for example, as a dimple on the surface of a computer disc. Information is carried and transmitted by signals – or data, so, if it is to be stored physically, what needs to be stored is, first and foremost, the underlying data. Since a datum, by our definition, is the lack of uniformity between at least two distinct variables, the storing of data has to preserve this lack of uniformity. The most basic storage unit of discrete data is a classical bit. In an electronic

computer, it is physically realised by a bistable device stabilising on either logical 0 or 1. Very little information can be stored in a single bit, so larger amounts of information are stored as *collections* of bits.

We now proceed to examine the individual premises of the first argument in order. (The third locution is explicated below in conjunction with the third premise of the argument.) Putnam and Searle's arguments rely on physical microstates being used in the computations. The physical substrate of the object in question (be that matter or fields) serves as a storage medium of information. A physical object with 2^n possible states can store n classical bits of information.[9] "Space is […] literally just a storage space for information [… that] is naturally associated with matter" (Verlinde 2011, p. 5). According to some physicists, information is stored in discrete bits on surfaces, rather than in points of a discretised space: the information content of each particle is proportional to its surface area (cf. Vedral 2010, p. 190; Verlinde 2011, pp. 5–6).

In any finite physical system – that is, a system with an effectively finite-dimensional state space with some energetic constraint – only a finite amount of information can be stored (Hänggi and Wehner 2013). (From a quantum mechanical perspective, the amount of classical information that can be encoded in a system is known as Holevo's bound.[10]) Any finite region of space can only contain a finite amount of matter. And matter can only be subdivided into smaller parts up to a limit. There is some minimum size beyond which it cannot be further subdivided. This is compatible with Max Planck's idea of indivisible quanta (Planck 1914). Unless matter *is* infinitely divisible, which is certainly *logically* possible[11], and whatever the minimum size of the elementary particles is, it can only store a finite amount of information.

Construed as a form of information processing, digital computation requires storage of information. Here the distinction introduced above between memory-based and memoryless computational systems comes in handy. Computation in memoryless systems does not require the retention of state and, therefore, does not require a long-term storage of information. A two-input,

one-output AND-gate, for example, performs its computation without depending on a previous state. Trivial computations only require that the data processed be somehow physically realised by whichever particles are available as data storage media. (Trivial computations that operate on structured data require additionally that these particles be somehow structured appropriately.) However, nontrivial computations further require that instructional information be physically realised and be distinguished from "raw" data being operated on. Memory-based computations require additionally that their states be stored throughout the course of the computation.

Consider physical instantiations of a TM and an FSA. The former is the quintessential digital computational system storing and deleting information. Whatever computation the TM performs it uses its tape for storage. Perhaps, the TM is too easy an example. An FSA clearly computes, but its tape is unidirectional and it cannot store information on this read-only tape. Nevertheless, the FSA's tape is used for storing the input data and its states are storage media of instructional information. Anything that can be computed by a TM equipped with a finite tape can also be computed by an FSA with *sufficiently many* states. The FSA's states can be used to store information that the TM would store on its read/write tape.

Bold pancomputationalism entails that every physical object computes every Turing-computable function and there are *infinitely many* TMs, each corresponding to a different algorithm for computing a function using a different table of instructions. This brings us to the third premise and to the third locution: 'simultaneous'. It may certainly be true that 'most conventional digital computers instantiate distinct programs simultaneously' (where at least some portions of these programs do not overlap; more is said about this in Section 4). But the truth of this claim depends on what 'simultaneous instantiation' means. One possible interpretation is that a general-purpose computer has multiple distinct programs stored in its memory and each program is executed at some point in time but not necessarily *all at once*. 'Simultaneous instantiation', accordingly, could mean that all these distinct programs are executed on the computer over its lifetime (possibly, *ad*

infinitum), provided that it is powered indefinitely. On this interpretation, less information storage is required. Any particular program requires information storage during its execution, and upon its termination – the information can be discarded.

Nonetheless, bold pancomputationalism implies that every physical object implements infinitely many distinct programs simultaneously. If we accept the three premises above, then we should also accept the conclusion that follows and reject bold pancomputationalism. Consider a universal TM (UTM) that has infinitely many heads and infinitely many (enumerable) tapes (instead of a single infinite tape), each of which stores a single TM. Such an *idealised* UTM can, arguably, support infinitely many distinct programs simultaneously. On the other hand, a *finite physical* object cannot support infinitely many distinct programs (or TMs) even given unbounded time (though arbitrarily many computations may then be possible by manufacturing more tape when necessary).

It seems that either way the bold pancomputationalist might respond to this last assertion, she faces serious challenges. On the one hand, despite using classical physics, she might grant that matter is indeed not infinitely divisible, so there is some minimum unit of information storage in Nature. But, then, the argument continues by claiming that there can still be infinitely many TMs all operating on the same set of data, perhaps, even cooperatively to a single end. In other words, (possibly infinite) storage space could be saved by having those TMs write to overlapping "physical regions". But, for one thing, even by extending the tape of a single TM, the *very same* TM description characterises a computation of the same function over a larger domain (Brown 2012, p. 62). And bold pancomputationalism implies that infinitely many Turing-computable functions are computed simultaneously. Why is there a reason to assume that *all* these infinitely many TMs operate on the same set of data? Neither Searle nor Putnam restricts the set of data that physical objects supposedly operate on. Besides, why should all possible inputs to these infinitely many TMs be excluded? These infinitely many inputs also have to be physically stored as data for the computation.

Moreover, the bold pancomputationalist should also account for the storage of the infinitely many TMs' tables of instructions. Every TM clearly has a finite description, but infinitely many of them exist. If these (infinitely many) tables of instructions are not somehow stored in the physical object performing the computation, then the information about every particular TM algorithm can be present only in a complete description of all the possible trajectories of the particles engaged in the computation (Brown 2012, p. 63). In other words, bold pancomputationalists ought to provide a complete listing of what each TM would do under every possible circumstance (cf. the counterfactual objection above). However, that, too, would unavoidably require that a finite physical object store an infinite amount of information.

Suppose, on the other hand, that the bold pancomputationalist rejected the idea that there exists some indivisible particle. That would mean that somehow a finite physical object has infinitely many particles, each of which can be an information storage unit. But, then, she would have to account for the structural stability of the "physical system", as a functionally related set of "parts", performing the computation. The structural stable "physical system" should be able to recover an equilibrium state upon being disturbed by any of the admissible perturbations. (Putnam's pancomputational theorem, for one, is explicitly defined in terms of open systems that are not shielded from external forces.) At some point, subdividing a physical system destroys its property of *being structurally stable*. Clearly, resorting to infinitesimally small particles as information storage units unavoidably reaches that point.

Our second argument can be summarised as follows.

1. A finite physical object has a finite amount of energy.
2. Digital computation dissipates energy in order to process information.
3. Simultaneous distinct computations require the dissipation of distinct energy.
4. (Therefore, a finite physical system can only support a finite number of distinct computations simultaneously.)

The first premise is motivated by Einstein's mass–energy relationship in special theory of relativity and Planck's observation on the speed of light. "Since […] energy radiation is propagated in the medium with a finite velocity […], there must be in a finite space a finite amount of energy" (Planck 1914). Light propagates at a finite speed anywhere and it cannot propagate instantaneously. Otherwise, if one assumes that the propagation of energy is variable, one gets a contradiction: some parts of the system would receive energy before others via a non-constant light speed (Shour 2008). Similarly, from the mass–energy relationship it follows that matter confined within a finite volume with a standing structure has a finite amount of energy associated with it. In this context, too, it is posited that light has a finite speed of propagation (Bohm 2006, p. 12).

The justification of the second premise proceeds in two steps: for irreversible and reversible computations. An operation is logically reversible if its inputs can always be inferred from its outputs. Conversely, a logically irreversible operation is such that in producing an output loses information about the history of the operation (i.e., not all inputs may be inferred from the output). The ultimate limits of the real-time speed of computational systems (as well as their miniaturisation) are governed by unavoidable heat increase through their energy dissipation. There is a minimum thermodynamic cost to any computation. This cost is the sum of the energy involved in providing the extra bits required in the course of a computation plus the destruction (by erasure) of the "garbage" bits produced.[12] This erasure operation, and, similarly, the merging of two computational paths, is the irreversible part of the computation. According to Landauer's principle, only this irreversible part of the computation dissipates heat. As it turns out, there are upper and lower bounds on the ultimate limits of the thermodynamic cost of effective computations (Landauer 1961; Li and Vitanyi 1992).

Most conventional computational systems are irreversible. Consider, for instance, an AND-gate or an OR-gate that lose information about the history of computation when the output produced is a logical 0 or 1, respectively. The loss of information in logically irreversible operations is

accompanied by energy dissipation to the surrounding environment as heat. Still, even reversible computation is accompanied by some minimum heat dissipation when garbage information is erased. Importantly, any finite physical system that computes all infinitely many Turing-computable functions will ultimately run out of storage space. That is, of course, assuming that information is stored *discretely* and that the physical media used for storage is not infinitely divisible. Since our concern is *digital* computation, this assumption is not implausible.

From the three premises above it follows that a finite physical system can only support a finite number of distinct computations simultaneously. The third premise is justifiable by the first law of thermodynamics: inevitably, computation, as a thermodynamic process, decreases the internal energy of the physical object whilst some heat is dissipated to the surrounding environment. The upper limit to the number of possible concurrent computations (also) depends on the energy footprints of the particular computations performed. Still, bold pancomputationalists might argue that *multiple* computational tasks can somehow be performed efficiently using the "same" energy dissipation, thereby lowering the total energy dissipation. However, they ought to show how arbitrary finite physical objects in nature minimise energy dissipation by way of efficient heat exchange between different physical objects, preventing heat loss, etc.

This shows that the bold pancomputationalist can argue for either weak or moderate pancomputationalism. Searle's Pancomputational thesis and Putnam's Pancomputational theorem may threaten the objectivity of computation even if the number of relativistic computational descriptions of a given system is neither maximal nor infinite. According to weak pancomputationalism, every physical object (can be said to) computes at least one Turing-computable function. Many physical objects can possibly be described as performing *trivial* computation, for example, as computing Boolean functions on n inputs. (Such computations can be performed by memoryless systems and do not require information storage for their computation.) Furthermore, every physical object can be described as implementing a *single-state* FSA.

Nevertheless, the more interesting – but hard to justify – hypothesis would be that these objects are nontrivial memory-based computational systems with multiple states.

The lack of appropriate constraints on physical state type formation simply reinforces the question of what counts as a legitimate physical state in (possibly competing) computational descriptions (Scheutz 2012, p. 104). Many objects can be described as implementing a two-state FSA as long as the physical states describable by the physical theory can be straightforwardly and reliably mapped onto one of these two state types. Fewer objects can be described as implementing a three-state FSA and so on. According to moderate pancomputationalism, every physical object (can be said to) computes more than one Turing-computable function. In the next section, we address the matter of multiple computational descriptions given to a single (computational) system and what it takes for a system to compute more than one Turing-computable function.

Moreover, it is one thing to describe the physical system concerned in an *arbitrary* and *ex post facto* manner. However, it is questionable whether Searle's pancomputational thesis and Putnam's pancomputational theorem can be upheld in a form of a reliable law-like generalisation. A computational state-transition has to be reliable and yet open systems are susceptible to external influences where even the slightest perturbation affects the system's state.

By way of concluding this section, we briefly respond to a possible objection to the two arguments above: the conception of information being used in these arguments is objective and non-semantic.[13] Such conception, so the objection continues, is very different from the semantic conception of instructional information used in Section 2. Supposedly, the same physical system can be given an infinite number of informational descriptions in the sense of what instructional information is processed.

Our brief reply to this objection is twofold. First, explaining nontrivial digital computation in terms of instructional information, which is a semantic notion, does not entail either 1) that there is no underlying quantitative/non-semantic conception of information in play or 2) that instructional

information cannot be formalised.[14] For one thing, instructional information is underpinned by structured data. Structured data *can* be quantified and shown to be different from unstructured data. Second, even though instructional information (e.g., in a given data set $\{d_1, d_2, d_3\}$) can be any one of the data being processed (i.e., any of d_1, d_2, d_3, d_1d_2, d_1d_3 or d_2d_3), it does not follow that there exists an infinite number of informational descriptions of the computational system processing that information (in our case there are at most six such descriptions). Besides, very few of those would be coherent descriptions in consideration of all the valid input/output mappings of the computational system concerned.

4. Multiple Computational Identities

4.1. Background

Let us now examine the putative, more modest, claim for the observer-relativity of computation stemming from the multiplicity of computational identities of genuine computational systems. The argument for multiple computational identities can be summarised as follows.

1. Some computational systems simultaneously implement multiple computations.
2. In any given context, the computation performed by a computational system is determined by a single syntactic structure, which is the underlying task.
3. The underlying *task* is at least partially semantically individuated.
4. (Therefore, some computation is at least partially semantically individuated relative to one particular task that the computational system performs.)

The gist of the argument is that some physical systems implement more than one syntactic structure simultaneously. It appears in (Shagrir 2001) and is originally aimed at computational *cognitive* systems. To determine which syntactic structure constitutes the system's computational identity, some other constraint has to be invoked. Oron Shagrir argues that this constraint is semantic: the system's computational identity is affected by the content of its computational states.

We focus here solely on the argument's implications for the alleged observer-relativity of computation and, in particular, on the first premise. An AND-gate can be said to implement either conjunction or disjunction (in negative logic) (Bishop 2009, p. 228; Shagrir 2001, p. 374; Sprevak 2010, p. 269). Similarly, a NAND-gate can be reinterpreted as a NOR-gate and a XOR-gate can be reinterpreted as an XNOR-gate by reversing the standard interpretation of logical 0s and 1s. Whilst the first premise clearly does not entail either bold or weak pancomputationalism, it does invite the question whether the computational identity of the system is observer-relative.

It should be noted that not all those who endorse the view described here subscribe to the identity conditions of computational systems being observer-relative. Shagrir, for one, claims that none of the premises of the argument above makes the identity conditions of computational systems observer-relative. Rather, he claims that "a physical system may simultaneously implement more than one syntactic structure" and it can safely be "assume[d] that all these implemented structures are intrinsic [to the system]" (Shagrir 2001, p. 379). On the other hand, Mark Bishop claims that the computational function implemented by the physical system must be contingent on the observer-determined computational-to-physical state mapping used (2009, p. 228). In a similar vein, Mark Sprevak claims that even an "appeal to the larger system in which [a computational] unit is embedded does not help to determine whether [… it computes] AND or OR" (2010, p. 269).

4.2. A Reply to Multiple Computational Identities

A clarificatory question is in order before we proceed. Multiple syntactic structures introduce an observer-relativity problem only if 1) they are somehow incompatible, and 2) none of them is ontically or epistemically privileged over the others. As already suggested above, computation being an observer-relative phenomenon does not follow from a single computational system lending itself to more than one description. Arguably, standard conventional digital computers (e.g., those using x86 compatible processors) instantiate distinct assembly language programs and higher-level language programs simultaneously. Why would some constraint be required to determine the

computational identity of the system?

The argument for multiple computational identities *does not* centre on a computational system simultaneously implementing a program *P* in, say, Java or C++, as well the same program compiled into, say, assembly (call it *P'*). Thus described, every conventional soft-programmable computer certainly implements multiple programs, such as *P* and *P'*, simultaneously. Such description is akin to a multi-level description of a UTM performing some computation. At one level of abstraction, it may be described as executing a particular program on some specific input. At another level, it may described as computing, for example, the square root of 80 to 10 decimal places. However, these two descriptions do not qualify as two distinct functions computed by the UTM. In that sense, the AND-gate being amenable to performing disjunction is a red herring. The argument is better served by considering a multi-head/multi-tape UTM that implements many distinct programs (i.e., specific TMs) simultaneously. Provided that these programs imply different behaviours and that none of them is ontically or epistemically privileged over the others, the computational identity of the system is supposedly observer-relative.

What, then, determines the computational identity of the system concerned? Shagrir lists three different ways to describe what a physical computational system, *S*, does.

1. A physical description, for example, in terms of the volts flowing through *S*.
2. A syntactic description of the abstract structure that *S* implements, for example, in terms of a truth table description of *S*' input/output mappings.
3. A semantic description interpreting the state types of the abstract structure of *S* (e.g., interpreting the symbols '0' and '1' as representing numbers).

On the received view, the computational identity of *S* is determined by its syntactic structure, yet Shagrir argues that it does not follow from that view that semantic content has no impact on individuating *S*' computational identity (2001, p. 373). Similarly, Sprevak argues that "representational content is a necessary condition that does crucial work in determining

computational identity" (2010, p. 261) of physical computational systems, whereas an abstract "Turing machine operates on syntactic entities" (2010, p. 269). But it seems curious to insist on semantic interpretation as being necessary for establishing the computational identity of *S* when there could be multiple distinct interpretations. Of course, none of these interpretations would change the underlying physical working of *S*. "[T]he computational identity of [the system] is the same if we interpret the '0' and '1' as representing numbers, colored hats or shapes" (Shagrir 2001, p. 373).

Rather than *S*' computation being determined by some *single syntactic structure* (being "the underlying task" according to the second premise above), it is determined by *all* the syntactic structures that *S* implements. None of these syntactic structures is either ontically or epistemically privileged over the others. Whilst a UTM is an abstract computational system, it serves us well here to make our point. Consider again a multi-head/multi-tape UTM, *U*, that implements many distinct programs simultaneously. A complete description of *U* at any point in its computation can be specified by giving the state of its controllers, the contents of all the tapes, and the position of the heads on the tapes. Call such a complete description the Instantaneous Description (ID) of *U*. A directed graph consisting of all the possible IDs can be used to represent every possible computation performed by *U* where each vertex in the graph is in one-to-one correspondence with an ID (labelled ID_x). A directed edge is added from vertex ID_x to vertex ID_y when the configuration specified by ID_y follows from ID_x via a single move of *U*. A single vertex (labelled ID_0) represents the starting configuration of *U*. Any path starting at ID_0 that continues through the graph to some vertex ID_i with the halt state corresponds to a valid computation *U*. (For a full description of how such configuration graphs are constructed, see, for example, (Adriaans and van Emde Boas 2011, p. 7) and (Fresco and Wolf 2014).)

A physical conventional computer can be similarly represented by such a configuration graph that can be used to determine the computational identity of the computer at any given time. Each

vertex of the corresponding configuration graph represents the total memory configuration of the computer (including the contents of *all* its data storage units) and the state of all its CPUs. (The initial state of the computer is represented by the vertex labelled ID_0.) Viewed this way, the processing of instructional information results in the computer moving from one memory configuration to another. The loading of a program into the computer's memory affects its memory configuration, even if the program is not executed. A description of a computational system that appeals to its configuration graph depends not only on the syntactic structures implemented by the system but also on its physical makeup.

At the physical level, computation as information/data processing is a systematic manipulation of dynamically enforced microphysical state correlations. This systematic manipulation means that the way information is manipulated is itself sensitive to the dynamical mechanisms that produce these correlations (Bokulich 2013, p. 40). Structured systems arise through a systematic correlation of effective *degrees of freedom*. A degree of freedom is an independent parameter that specifies the state of the system at a given time. A single point particle has three degrees of freedom, whilst its microphysical *state space* (displaying both the generalised coordinates and the canonically conjugate momenta of the particle) is six-dimensional. Structure reduces degrees of freedom. A molecule can do less than its composite atoms can (e.g., by limiting the momentum of some of the atoms in a molecule) (Bokulich 2013, pp. 32–33).

The objectivity of the computational identity of the system is fixed by degrees of freedom at the physical level and in memory-based systems the configuration graph can be used to determine this identity during runtime. The effective causal dynamical structures that process information are observer-independent facts about the computational system that *allow* an observer to interpret a particular system as performing a given computation. The correlation of one microphysical state of the system with another allows inferring one microphysical state from another correlated microphysical state. In memory-based computational systems, multiple semantic interpretations can

be ascribed to a single system only if they are homomorphically equivalent to its configuration graph. That is, not only do input/output data need to be fully specified by the semantic interpretation concerned, but also every possible transition from one memory configuration to another. The configuration graph representation does not apply to *memoryless* computational systems though, such as two-input, one-output Boolean gates.

Nevertheless, the semantic interpretations that can be ascribed to trivial memoryless computational systems are also restricted by the data processing operations performed at the physical level. A conventional AND-gate can only be used as a device performing logical conjunction on input data due to the particular structure of the gate that limits the degrees of freedom of the underlying microphysical states. The same AND-gate can be semantically reinterpreted as performing logical disjunction. But this is only because the same microphysical state correlation obtained in an OR-gate when the input and output data are interpreted in negative logic and $\neg(p \wedge q)$ is equivalent to $\neg p \vee \neg q$. However, a conventional AND-gate *cannot* be similarly reinterpreted as performing NAND or XOR.

The semantic interpretation does not *fix* the computational identity of the system concerned, but rather gives the computation a different name. A logical disjunction may very well be called conjunction instead (and vice versa), but the computational identity of the Boolean gate is fixed by the underlying data processing operations. The corresponding truth table of the gate is merely mirrored, whereas the "black box" does not change, only how the input and output data are interpreted. "There is [...] an abstraction involved in naming an operation and using it on account of 'what it does' while completely disregarding 'how it works'" (Dijkstra 1972, p. 11).

Whilst the Cryptographer's Constraint may not apply in the case of Putnam's pancomputational theorem, it does apply as a constraint on the semantic interpretations that can be ascribed to some computational system. According to the Cryptographer's Constraint, the larger (or more complex) a string of text (or a system) to be decoded is, the fewer nontrivial interpretations of the text there are

(Dennett 1989, p. 136, fn. 6). This constraint does not apply to Putnam's pancomputational theorem because of the posited principle of continuity (Chrisley 1994, p. 419 , fn. 9). But it does apply when considering nontrivial versus trivial computational systems.

The distinction drawn in Section 2 between trivial and nontrivial computation elucidates the increasing complexity of such computational systems. Two-input, one-output Boolean gates are memoryless and their data processing operations are amenable to a simple description. Yet, as the complexity of the computational system increases "the chances that there is more than one meaningful interpretation for that [… system] decreases drastically" (Chrisley 1994, p. 419 , fn. 9). Nontrivial memory-based computational systems are far more complex in that they have multiple capacities and require some form of information to determine the next state-transition depending on the current state and input read. The more complex the configuration graph of the system is, the fewer (interesting) coherent interpretations of the computation there are.

Lastly, a critic might argue that any instance of an NP-complete problem can be translated/reduced to an instance of another NP-complete problem in polynomial time (Hopcroft et al. 2001, p. 447 ff). And this shows that *any* NP-complete problem can be semantically reinterpreted as *any* other NP-complete problem. However, we argue that multiple semantic interpretations can be correctly ascribed to a single computational system only if they are homomorphically equivalent to its configuration graph. The homomorphic equivalence requirement is, strictly, more restrictive than what is required of a reduction of one NP-complete problem, *N*, to another, *M*. The latter requires that the input(s) to and output(s) of the algorithm for solving *N* be translated to the corresponding ones in *M* (e.g., translating Boolean clauses in *N* to vertices on graphs for *M*). But such reduction does not require, for example, that the instance of *N* and instance of *M* be of exactly the same input size.

5. Conclusion

We conclude that, at least to a large extent, computation is an objective phenomenon. Construed as a form of information/data processing, the effective dynamics of the computational system have to maintain the structures that realise the information. The structure of the data and the system processing them guarantees the correlation amongst the effective microphysical states. We have argued that information-processing considerations determine what type of computation takes place in physical systems. Some genuine computational systems may indeed be semantically interpreted as performing "different" computations. This is certainly the case for some trivial computational systems. But, in the case of nontrivial computation, it becomes increasingly harder to ascribe them multiple coherent interpretations as their complexity increases. Besides, the computational identity of the system remains fixed even though all these semantic interpretations are equally "correct". We have also argued above that bold anti-realist pancomputationalism does not hold in finite physical objects. The result was weakening pancomputationalism to the claim that a finite physical system can only support a finite number of computations simultaneously.

Acknowledgments

I am grateful to Marty Wolf for extremely useful remarks on a previous draft of this paper as well as many discussions on issues raised herein. I thank Graham White for his comments on an early draft of this paper. Several anonymous referees have contributed important insights into the revision process thereby significantly improving this latest version. This research was supported by a Research Fellowship from the Sidney M. Edelstein Centre for History & Philosophy of Science, Technology & Medicine. Part of this research was conducted while I was a visiting fellow at the School of Humanities & Languages, University of New South Wales, Australia. I gratefully acknowledge their support. The usual disclaimer applies: any remaining mistakes are my sole responsibility.

Notes:

[1] Some have called this view *unlimited pancomputationalism* claiming that it is the strongest version of pancomputationalism (Piccinini 2012). But the strongest version of pancomputationalism is the view that the universe is a computer (Dodig-Crnkovic and Müller 2011, p. 153).

[2] I thank an anonymous referee for this critique.

[3] There are four two-input, one-output gates that process structured input data.

[4] For a similar argument in the context of defending the mechanistic account of computation see (Dewhurst 2014).

[5] It may be argued that there exist some weaker FSAs and TMs (e.g., a single-state FSA) for which an indeterminacy argument can be constructed thereby showing that the IIP account cannot treat all these systems uniformly as memory-

based nontrivial computational systems. For a detailed discussion of the memory model in such systems in a generalised meta-computational space see (Fresco and Wolf 2014).

[6] The expression 'ordinary open systems' refers to systems that are open to influences, such as gravitational and electromagnetic forces, and constantly change.

[7] An *internal state* of K is defined as a set of K's maximal states (a maximal state is a complete specification of the values of *all* the relevant variables of K) for some given real-time interval (e.g., from 13:00 to 13:04) such that these maximal states correspond to the automaton's states in a single run of δ.

[8] This argument can be reformulated in terms of data, rather than information, since the underlying data are those used to carry information. For ease of exposition it is formulated in terms of information.

[9] Interestingly, although quantum mechanics provides for a significant improvement over classical mechanics in the efficiency of communication, arguably, "the transmission of n quantum bits cannot serve to communicate more than n classical bits of information" (Brassard 2003, p. 1611).

[10] I thank an anonymous referee for emphasising the quantum mechanical perspective.

[11] If matter and time *are* infinitely divisible, then it is possible to perform infinite computations in finite space and time.

[12] Garbage information is an intermediate memory that is used to keep track of the history of the computation going from state s_x to s_y but is not otherwise used for the computation itself.

[13] A related critique might be that the information a physical state carries is observer-relative. But the critic will have to concede that the sum of all actual or possible attributions *is* finite.

[14] cf. Peter Corning's (2001, pp. 1280–1281) proposed framework for quantifying "control information" in relation to its capacity to control and utilise available energy and matter in or by a cybernetic system.

References

Adriaans, P., & van Emde Boas, P. (2011). Computation, Information, and the Arrow of Time. In S. B. Cooper & A. Sorbi (Eds.), *Computability In Context* (pp. 1–17). Imperial College Press. http://www.worldscientific.com/doi/abs/10.1142/9781848162778_0001.

Bishop, J. M. (2009). A Cognitive Computation Fallacy? Cognition, Computations and Panpsychism. *Cognitive Computation*, *1*(3), 221–233. doi:10.1007/s12559-009-9019-6

Block, N. (2002). Searle's arguments against cognitive science. In J. Preston & M. Bishop (Eds.), *Views into the Chinese room : new essays on Searle and artificial intelligence* (pp. 70–79). Oxford: Clarendon Press.

Bohm, D. (2006). *The special theory of relativity*. London: Routledge.

Bokulich, P. (2013). The Physics and Metaphysics of Computation and Cognition. In V. C. Müller (Ed.), *Philosophy and Theory of Artificial Intelligence* (Vol. 5, pp. 29–41). Berlin, Heidelberg: Springer Berlin Heidelberg. http://www.springerlink.com/index/10.1007/978-3-642-31674-6_3.

Brassard, G. (2003). Quantum Communication Complexity. *Foundations of Physics*, *33*(11), 1593–1616. doi:10.1023/A:1026009100467

Brown, C. (2012). Combinatorial-State Automata and Models of Computation. *Journal of Cognitive Science*, *13*(1), 51–73.

Chalmers, D. J. (1996). Does a rock implement every finite-state automaton? *Synthese*, *108*(3), 309–333. doi:10.1007/BF00413692

Chrisley, R. L. (1994). Why everything doesn't realize every computation. *Minds and Machines*, *4*(4), 403–420. doi:10.1007/BF00974167

Copeland, B. J. (1996). What is computation? *Synthese*, *108*(3), 335–359. doi:10.1007/BF00413693

Corning, P. A. (2001). "Control information": The missing element in Norbert Wiener's cybernetic paradigm? *Kybernetes*, *30*(9/10), 1272–1288. doi:10.1108/EUM0000000006552

Dennett, D. C. (1989). *The intentional stance*. Cambridge, Mass.: MIT Press.

Dewhurst, J. (2014). Rejecting the Received View: Representation, Computation, and Observer-Relativity. In *The 7th AISB Symposium on Computing and Philosophy: is computation observer-relative?*. University of London, UK.

Dietrich, E. (1989). Semantics and the computational paradigm in cognitive psychology. *Synthese*, *79*(1), 119–141. doi:10.1007/BF00873258

Dijkstra, E. W. (1972). Structured programming. In O. J. Dahl, E. W. Dijkstra, & C. A. R. Hoare (Eds.), (pp. 1–82). London, UK, UK: Academic Press Ltd. http://dl.acm.org/citation.cfm?id=1243380.1243381

Dodig-Crnkovic, G., & Müller, V. C. (2011). A Dialogue Concerning Two World Systems: Info-computational Vs. Mechanistic. In G. Dodig-Crnkovic & M. Burgin (Eds.), *Information and Computation* (pp. 149–184). World Scientific.

Floridi, L. (2011). *The philosophy of information*. Oxford: Oxford University Press.

Fresco, N., & Staines, P. J. (2014). A Revised Attack on Computational Ontology. *Minds and Machines*, *24*(1), 101–122. doi:10.1007/s11023-013-9327-1

Fresco, N., & Wolf, M. J. (unpublished-a). Information Processing and the Structuring of Data.

Fresco, N., & Wolf, M. J. (unpublished-b). Objective Computational Descriptions and Kolmogorov Complexity.

Fresco, N., & Wolf, M. J. (2014). The instructional information processing account of digital computation. *Synthese*, *191*(7), 1469–1492. doi:10.1007/s11229-013-0338-5

Hamblin, C. L. (1987). *Imperatives*. New York, NY: Basil Blackwell.

Hänggi, E., & Wehner, S. (2013). A violation of the uncertainty principle implies a violation of the second law of thermodynamics. *Nature Communications*, *4*, 1670. doi:10.1038/ncomms2665

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). *Introduction to automata theory, languages, and computation*. Boston: Addison-Wesley.

Landauer, R. (1961). Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, *5*(3), 183–191. doi:10.1147/rd.53.0183

Li, M., & Vitanyi, P. M. B. (1992). *Mathematical theory of thermodynamics of computation*. Amsterdam, The Netherlands: Centre for Mathematics and Computer Science.

Mackie, D. (1997). The Individuation of Actions. *The Philosophical Quarterly*, *47*(186), 38–54. doi:10.1111/1467-9213.00045

Mossel, B. (2001). The Individuation of Actions. *Australasian Journal of Philosophy*, *79*(2), 258–278. doi:10.1080/713659226

Piccinini, G. (2007). Computing Mechanisms. *Philosophy of Science*, *74*(4), 501–526. doi:10.1086/522851

Piccinini, G. (2012). Computation in Physical Systems. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy*. http://plato.stanford.edu/archives/fall2012/entries/computation-physicalsystems/

Planck, M. (1914). *The theory of heat radiation*. New York: Maple Press: P. Blakiston's Son & Co.

Putnam, H. (1988). *Representation and reality*. Cambridge, Mass.: The MIT Press.

Scheutz, M. (1999). When Physical Systems Realize Functions... *Minds and Machines*, *9*(2), 161–196. doi:10.1023/A:1008364332419

Scheutz, M. (2012). What it is not to Implement a Computation: a critical analysis of Chalmers' notion of implementation. *Journal of Cognitive Science*, *13*(1), 75–106.

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

Shagrir, O. (2001). Content, Computation and Externalism. *Mind*, *110*(438), 369–400. doi:10.1093/mind/110.438.369

Shour, R. (2008). Isotropy, entropy, and energy scaling. *eprint arXiv:0805.1715*.

Sprevak, M. (2010). Computation, individuation, and the received view on representation. *Studies in History and Philosophy of Science Part A*, *41*(3), 260–270. doi:10.1016/j.shpsa.2010.07.008

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2-42*(1), 230–265. doi:10.1112/plms/s2-42.1.230

Vedral, V. (2010). *Decoding reality: the universe as quantum information*. Oxford: Oxford University Press.

Verlinde, E. (2011). On the origin of gravity and the laws of Newton. *Journal of High Energy Physics*, *2011*(4). doi:10.1007/JHEP04(2011)029