A Critical Survey of Some Competing Accounts of Concrete Digital Computation

Nir Fresco

School of History and Philosophy, The University of New South Wales, Sydney, Australia
Fresco.Nir@Gmail.com

Abstract. This paper deals with the question: what are the key requirements for a physical system to perform digital computation? Oftentimes, cognitive scientists are quick to employ the notion of computation *simpliciter* when asserting basically that cognitive activities are computational. They employ this notion as if there is a consensus on just what it takes for a physical system to compute. Some cognitive scientists in referring to digital computation simply adhere to Turing computability. But if cognition is indeed computational, then it is *concrete* computation that is required for explaining cognition as an embodied phenomenon. Three accounts of computation are examined here: 1. *Formal Symbol Manipulation.* 2. *Physical Symbol Systems* and 3. *The Mechanistic account*. I argue that the differing requirements implied by these accounts justify the demand that one commits to a particular account when employing the notion of digital computation in regard to physical systems, rather than use these accounts interchangeably.

Keywords: Concrete computation, Computability, Symbols, Semantics, Information Processing, Cognitive Systems, Turing Machines.

1 Introduction

All too often, cognitive scientists are quick to employ the notion of computation *simpliciter* when asserting basically that cognitive activities are computational. Unfortunately, it seems that a clearer understanding of computation is distorted by philosophical concerns about cognition. Some researchers in referring to *digital* computation simply adhere to Alan Turing's notion of *computability* when attempting to explain cognitive behaviour. Still, classical computability theory studies what functions on the natural numbers are computable, and not the spatiotemporal constraints that are inherent to cognitive phenomena.

Any analysis of cognitive phenomena that is based solely on mathematical formalisms of computability, is at best incomplete. It has been proven that Emil Post's machines, Stephen Kleene's formal systems model, Kurt Gödel's recursive functions model, Alonzo Church's lambda calculus, and Turing Machines (TMs) – are all extensionally equivalent. They all identify the same class of functions, in terms of the sets of arguments and values that they determine, as computable (Kleene 2002: pp. 232-233).

D.L. Dowe (Ed.): Solomonoff Festschrift, LNAI 7070, pp. 155–173, 2013.

However, c*oncrete digital computation* as it is actualised in physical systems seems to be a more appropriate candidate for the job of explaining cognitive phenomena.[1] It is not in vain that the reigning trends in contemporary cognitive science (whether it be connectionism or dynamicism) emphasise the embeddedness and embodiment of cognitive agents. This is one motivation for examining extant accounts of concrete computation, before we can make any sense of talk about 'cognitive computation', 'neural computation' or 'biological computation'.

There are many extant accounts of digital computation in physical systems on offer. Only three accounts are examined in this paper for lack of space.[2]

1. According to the *Formal Symbol Manipulation (FSM) account*, a physical system performs digital computation when it processes semantically interpreted (not just interpretable) symbols (Pylyshyn 1984: pp. 62, 72).
2. According to the *Physical Symbol Systems (PSS) account*, a physical system performs digital computation when it consists of symbols and processes operating on these symbols that designate other entities (Newell 1980: p. 157).
3. According to the *Mechanistic account*, a physical system performs digital computation if it manipulates input strings of digits, depending on the digits' type and their location on the string, in accordance with a rule defined over the strings (and possibly the system's internal states) (Piccinini and Scarantino 2011: p. 8).

No novel account of computation is offered here. The goal of this paper is to examine the conflict among well-known accounts and argue that they imply sufficiently distinct requirements for a physical system to compute to justify the demand that one commits to a particular account when employing the notion of concrete digital computation. Whilst the main driver here is cognitive science, this demand is unbiased. It applies just as well to biology, astronomy and any other science in which 'computation' is employed as explanans for some physical phenomenon. In the following three sections, I survey the FSM, PSS and Mechanistic accounts respectively. In the fifth section, I defend my argument for the non-equivalence of extant accounts of concrete computation.

2 The Formal Symbol Manipulation Account

According to this account digital computing systems are formal symbol manipulators. They manipulate symbol tokens which themselves are representations of the subject matter the computation is about, in accordance with some purely formal principles

[1] For the purposes of this paper, I shall remain neutral on whether cognition can indeed be *fully* explained computationally. Arguably, cognition involves the processing of information, and it is not entirely clear that information processing is equivalent to digital computation. This question can remain unanswered for now.

[2] These particular three accounts nicely demonstrate that extant accounts of computation are not only intensionally different, but also extensionally different, irrespective of their representational character. For a detailed analysis of Turing's account, Hilary Putnam & John Searle's trivialisation of computation, a reconstruction of Brian C. Smith's participatory account and the Algorithm Execution account see Fresco (2011).

(Scheutz 2002: p. 13). Although these manipulated symbols have both semantic and syntactic properties, only the latter are causally efficacious. Chief proponents of this account are Jerry Fodor (1975), Zenon Pylyshyn (1984; 1989) and John Haugeland (1985). Fodor asserts that "computations just are processes in which representations have their causal consequences in virtue of their form" (Fodor 1980: p. 68). Haugeland's well-known formalist's motto stated that "if you take care of the syntax, the semantics will take care of itself" (Haugeland 1985: p. 106). A computing system as an interpreted automatic formal system takes care of the syntax.

Furthermore, such systems are organised in three distinct levels: the semantic level, the symbolic level and the physical level (Pylyshyn 1989: pp. 58-59). This explanation framework of complex systems has some similarity to David Marr's tripartite model of complex systems: the computational level, the algorithmic level and the physical level (Marr 1982: p. 22). At the *semantic* level, symbolic expressions are transformed in the computing system in a way that coherently preserve their meaning and ensures that they continue to "make sense" when semantically interpreted. Marr's computational level, however, is a *function-theoretic* characterisation of the system in terms of the function it computes (i.e., its computational capacity). This computation may contingently involve the assignment of semantic contents. At the *symbolic* level the system operates in terms of *representations* and their transformations. Computing systems can operate at the semantic level only because of this middle level. Marr's algorithmic level also need not be aligned with the symbolic level, for his analysis is not necessarily committed to symbol-manipulation computation. At the physical level, the state transitions of the computing system correspond to some symbolic expressions and are connected by physical laws (Pylyshyn 1984: p. 58). This particular level is indeed analogous to Marr's physical level.

The FSM account identifies six key requirements for a physical system to perform digital computation.[3] The first requirement is that the system *be programmable to allow maximal plasticity of function*. In order to exclude such systems as mere calculators and interpreted automatic systems that are not formal (e.g., analogue computers), the class of computing systems is restricted to those that are programmable (Haugeland 1985: pp. 258-259). It is one of the foundational principles of computer science that the operations of digital computing systems be fully programmable (ibid: p. 126). Despite the rigidity of the physical structure of digital computing systems and the interconnections of their components, these systems are capable of maximal plasticity of function. This plasticity is enabled by their operation

[3] Pylyshyn argues that there is a missing requirement specifying what makes it the case that a symbol X represents, say, a particular daisy, rather than something else. The *computational theory of mind* has always been missing that part (Pylyshyn, personal communication). Specifically, he argues that the minimum function needed for this representation relation to obtain is that there be some causal or nomologically supported dependency between the daisy and X (Pylyshyn 2007: p. 82). However, it is not clear that conventional digital computing systems require that a similar causal relation obtain between a symbol and an external represented object for them to compute (a representation *internal* to the computing system, e.g., an instruction in memory, is not problematic).

being programmable to behave in accordance with any finitely specifiable function (Pylyshyn 1984: p. 53). It is also the basis for Turing's vision that a computer can (in principle) be made to exhibit intelligent activity to an arbitrary degree (thereby passing the Turing's test).

The second requirement is that the system operate using *internally represented rule-governed transformations of interpretable symbolic expressions*. As a formal system, by following the formal rules of transformation operating on symbolic expressions the semantic interpretation must make sense of those expressions. The computing system operates as a black box that automatically manipulates the symbolic tokens according to formal rules and when interpreted they make "sense in the contexts in which they're made" (Haugeland 1985: p. 106). The regularities of computing systems are rule-governed, rather than law-like. So any explanation of a computational process must make reference to what is represented by the (semantically interpreted) computational states and rules, rather than just to causal state transitions (Pylyshyn 1984: p. 57).

Moreover, it is a key property of computing systems that semantic interpretations of computational states must be consistent. Since computations follow a particular set of semantically interpreted rules, semantic interpretations of computational states cannot be given capriciously (ibid: p. 58), still these interpretations need not be unique. This is analogous to the rules of existential generalisations, universal instantiations etc. that apply to formulas in virtue of their syntactic form, but their salient property is semantical in that they are truth preserving (Fodor & Pylyshyn 1988: p. 29).

The third requirement is that the *computational states of the system must correspond to equivalence classes of physical states* such that their members are indistinguishable from the point of view of their function (Pylyshyn 1984: p. 56). There exists a primitive mapping from atomic symbols to relatively elementary physical states, and a mapping specification of the structure of complex expressions onto the structure of relatively complex physical states. The structure-preserving mapping is typically given recursively. This ensures that the relation between atomic symbols (e.g., 'A' and 'B'), and composite expressions (e.g., 'A&B'), is encoded in terms of a physical relation between constituent states that is functionally equivalent to the physical relation used to encode the relation between more complex expressions (e.g., 'A&B' and 'C') and their composite expression (e.g., '(A&B)&C'). Furthermore, the physical counterparts of the symbolic expressions and their structural properties cause the behaviour of the computing system. If you change the symbols, the system will behave differently (Fodor & Pylyshyn 1988: pp. 14, 17).

The fourth requirement is that the system *support an arbitrarily large number of representations* (Pylyshyn 1984: p. 62). Conventional computing systems' architecture requires that there be distinct symbolic expressions for each object, event or state of affairs it can represent (Fodor & Pylyshyn 1988: p. 57). This raises the question how so many semantically interpreted operations are possible if the number of expressions is arbitrary large. For a fixed number of expressions some sort of a lookup table could be implemented. However, this is not possible for an arbitrarily

large number of representations (Pylyshyn 1984: pp. 61-62). Instead, this capability is achieved by the fifth requirement.

The fifth requirement is that the system *be capable of capitalising on the compositional nature of expressions as determined by the constituent expressions and the rules used to combine them*. By supporting simple rules that operate on simple individual symbols the system is capable of an arbitrary large number of symbolic expressions. Complex expressions are realised and transformed by means of instantiating constituent expressions of representations (ibid). The semantics of composite symbolic expression is determined in a consistent way by the semantics of its constituents (Fodor & Pylyshyn 1988: p. 16). For instance, the semantics of 'the daisies in the vase on the table by the door' is determined by 'the daisies in the vase on the table', which is determined by 'the daisies in the vase'. Most of the symbolic expressions in computing systems as interpreted automatic formal systems are complexes, whose semantics is determined by their systematic composition Haugeland 1985: p. 96).

Finally, implicitly, the sixth requirement is that the system's *functional architecture include an accessible memory*. As in the idealised TM, a computing system must have a memory that allows writing of symbolic expressions and then reading them. This memory may consist of a running tape, a set of registers or any other storage media (Pylyshyn 1989: p. 56). The memory's capacity, its organisation and means of accessing it are properties of the specific functional architecture of the system. Most modern architectures are register-based, in which symbols and symbolic expressions are stored and later retrieved by their numeric or symbolic address (ibid: pp. 72-73). Although the set of computable functions does not depend on the particular system implementation of the memory, the time complexity of computation does vary (retrieving a particular string from a table could be, under certain conditions, be made independent of the number of strings stored and the size of the table) (Pylyshyn 1984: p. 97-99).

3 The Physical Symbol Systems Account

According to this account, championed by Allen Newell and Herbert Simon, digital computing systems are physical symbol systems.[4] They consist of sets of symbols, which are physical patterns that can occur as components of symbol structures or expressions (Newell and Simon 1976: p. 116). Computing systems also include a collection of processes that operate on these expressions to create, modify or destroy other expressions. Further, a physical symbol system is situated in a world of objects that is wider than just these symbolic expressions. The PSS account is indeed similar to the FSM account (as will be shown below), but there are also some differences that cannot be easily dismissed.

[4] The PSS hypothesis deals primarily with the intelligence of symbol systems and relates to minds and artificial intelligence. I will mostly limit my discussion to the PSS account of computation.

Newell and Simon (1976: p. 117) maintained that a physical symbol system is an instance of a Universal Turing machine (UTM). They discovered that UTMs always contain within them a particular notion of symbol and symbolic behaviour. Tautologically, physical symbol systems are universal (Newell 1980: p. 155). Their capacity to solve problems is accomplished by producing and progressively modifying symbol structures until they produce a solution structure, particularly by means of a heuristic search[5] (Newell and Simon 1976: p. 120). There are two basic aspects to each search, namely its object (i.e., what is being searched) and its scope (i.e., the set of objects within which the search is conducted). In computing systems each aspect must be made explicit in terms of specific structures and processes, since a system cannot search for an object that it cannot recognise (Haugeland 1985: p. 177). Computing systems (as all UTMs) solve problems mostly by using heuristic search, for they have limited processing resources.

The PSS account identifies seven key requirements for a physical system to perform digital computation. The first requirement is that the system *consist of a set of symbols and a set of processes that operate on them and produce through time an evolving collection of symbolic expressions*. At any given time, the system contains a collection of symbolic structures and processes operating on expressions to produce other expressions. Such processes are the creation, modification, reproduction and destruction of symbolic expressions through time (ibid: p. 116). These processes operate on and transform *internal* symbolic structures, or in other words the system executes computer programs that operate on data structures (Bickhard and Terveen 1995: p. 92).

The second requirement is that the system either *affect a designated object or behave in ways that are dependent on that object*. An entity (i.e., a symbol) X designates (i.e., is about or stands for) an entity (e.g., an object or a symbol) Y relative to a process P, if when X is P's input, P's behaviour depends on Y. Designation is grounded in the physical behaviour of P when its action could be at a distance if X (the input to P) stands for a distal object. This 'action at a distance' is accomplished by a mechanism of access (that is realised in physical computing systems) to three types of entities: symbol structures, operators[6] and roles in symbol structures. The set of processes includes programs, whose input could also be symbolic expressions. If an expression can be created at T_i that is dependent on an entity in some way, processes can exist in the system that at T_{i+1}, take that expression as input and behave in a way dependent on that entity. Thus, these expressions designate that entity (Newell 1980: pp. 156-157).

The third requirement is that the system *be capable of interpreting an expression, if it designates some process and given that expression the system can execute that process*. Interpretation is defined as the act of accepting an expression that designates a process as input and then executing that process (ibid: pp. 158-159). This is similar

[5] It is not clear that artificial digital computing systems (e.g., physical instantiations of UTMs) must use heuristic search as the only means for solving computational problems. Clearly, many algorithms are not based on any search mechanism, but rather a finite sequence of instructions to solve a particular problem.

[6] Operators are symbols that have an external semantics built into them (Newell 1980: p. 159).

to the process of indirectly executing computer programs by an interpreter program. The interpreter reads an expression E as input and if it is recognised as a program (or a procedure), rather than a data structure, it is then executed. This capability is necessary to allow the flexibility of UTMs to create expressions for their own behaviour and then produce that very behaviour. The total processes in the computing system can be decomposed to the basic structure of (control + (operators + data)) that is paradigmatic in all programming languages. The control continuously brings together operators and data to yield the desired behaviour.

The fourth requirement is *the existence of expressions that designate every process of which the machine is capable* (Newell and Simon 1976: p. 116). This requirement is self-explanatory and is necessary to support the full plasticity of behaviour of UTMs.

The fifth requirement is that the system be capable of distinguishing between some expressions as data and others as programs (Newell 1980: p. 166). This is a property of all UTMs that must be able to recognise some expressions as data when creating or modifying them at time T_i and then interpret them as programs at time T_j. The concept of universality, which is one of Turing's seminal contributions, unifies data and programs by way of the UTM taking programs of other (simulated) machines as data (as well as the inputs inscribed on the tapes of those simulated machines).

The sixth requirement is that the system have *a stable memory to ensure that once expressions are created they continue to exist until they are explicitly modified or deleted* (Newell and Simon 1976: p. 116). This requirement stems from the coupling of read/write operations in computing systems. Each of these operations requires its counterpart to be productive in affecting the system's behaviour. A read operation only retrieves expressions that were written to memory (and persisted). Conversely, a write operation of expressions, which are never subsequently read, is redundant (Newell 1980: p. 163).

Lastly, the seventh requirement is that the system *be capable of handling an unbounded number of expressions and realising the absolute maximal class of input/output functions using these expressions*. This requirement is weaker than the requirement for unbounded memory. The structural requirements for universality are not dependent on unbounded memory. Rather they are dependent on the system's capability to handle an unbounded number of expressions (Newell and Simon 1976: p. 116) and realise the absolute maximal class of input/output functions using these expressions (Newell 1980: p. 178).

4 The Mechanistic Account of Computation

According to the Mechanistic account, proposed by Gualtiero Piccinini (2007), digital computing systems are digit-processing mechanisms. They are mechanisms, which can be ascribed the function of generating output strings from input strings in accordance with a general rule (or map) that applies to all strings and depends on the input strings and (possibly) internal states for its application (ibid: p. 516). This account relies essentially on three conceptual elements: *I.* Medium independence of

the vehicles (digits) processed. They could be implemented in a variety of ways (such as mechanical components, electronic components, optical components etc.); *II.* The function of the system is to process those vehicles irrespective of their particular physical implementation; *III.* The operation of the system is performed in accordance with rules, which need not necessarily be algorithms or programs (as in the case of special purpose TMs or finite state automata, hereafter FSA).

Moreover, the mathematical notion of computation (i.e., computability) only applies directly to abstract systems, such as TMs or FSA, but not to physical systems. Computability is typically defined over strings of letters (often called symbols) from a finite alphabet (ibid: pp. 509-510). But not every process that is defined over strings of letters counts as computation (e.g., the generation of a random string of letters). To overcome this gap, Piccinini (2007: pp. 510-512) introduces the notion of a *digit* as the concrete counterpart to the formal notion of a *letter*. A digit is a stable state of a component that is processed by the mechanism.[7] Strings of digits (i.e., sequences of digits) can be either data or rules, so they are essentially the same kind of thing and differ only in the functional role they play during processing by the computing system (Piccinini and Scarantino 2011: pp. 7-8). Digits are permutable. Components that process digits of one type are functionally capable of processing digits of any other type.

The mechanistic account identifies four key requirements for a physical system to perform digital computation. The first requirement is that the system *process tokens of the same digit type in the same way and tokens of different digit types in different ways*. Under normal conditions, digits of the same type in a computing system affect primitive components of the system in sufficiently similar ways, thereby their dissimilarities make no difference to the output produced. For instance, two inputs to a XOR gate that are sufficiently close to a certain voltage (labelled type '1') yield an output of a voltage sufficiently close to a different specific value (labelled type '0'). However, that does not imply that for any two input types, a primitive component *always* yields outputs of different types. Two different inputs can yield the same computational output, such in the case of a NOR gate. Input types '1,1', '0,1' and '1,0' give rise to outputs of type '0'. Still, it is essential that the NOR gate yield different responses to tokens of different types, thus responding to input types '0,0' differently from other input types. Differences between digit types must suffice for the component to differentiate between them, so as to yield the correct outputs.

The second requirement is that the system *process all digits belonging to a string (of digits) during the same functionally relevant time interval and in a way that respects the ordering of the digits within that string*. When a computing system is sufficiently large and complex, there has to be some way to ensure synchronisation among all digits belonging to a particular string. The components of a computing system interact over time, and given their physical characteristics, there is only a

[7] In ordinary electronic computers digits are states of physical components of the machine (e.g., memory cells). In other cases, such as old punched card computers, strings of digits were implemented as sequences of holes (or lack thereof) on cards (Piccinini, personal communication).

limited amount of time during which their interaction can produce the correct result, which is consistent with the ordering of digits within strings. In primitive computing components and simple circuits it is mostly the temporal ordering of digits that is responsible for producing the correct result. So if, for example, digits, which are supposed to be summed together, enter an adder mechanism at times that are too far apart, they will not be added correctly (ibid: p. 513). In more complex components, processing of all digits belonging to a string must proceed in a way that also respects the spatial ordering of the digits within the string. Each digit in the sequence must be processed until we reach the last digit in the string. In some atypical cases the ordering of digits makes no difference to a computation (e.g., summing up all the numbers in an array or calculating the length of a sequence of symbols).

The third requirement is that *all the system's components that process digits stabilise only on states that count as digits*. Components can be in one of several stable states. In a binary computing system memory cells, for instance, can be in either of two stable states, each of which constitutes a digit. Upon receiving some physical stimulus (e.g., the pressing of a key), a memory cell enters a state on which it must stabilise. Memory cells stabilise on states corresponding to either of two digit types, typically labeled '0' and '1', that are processed by the computing system. If memory cells did not have the capacity to stabilise on one of these digit types, the memory would cease to function as such and the computer would cease to operate normally (ibid: p. 511).

The fourth requirement is that the *components of the system be functionally organised and synchronised so that external inputs, together with the digits stored in memory, be processed by the relevant components in accordance with a set of instructions*.[8] During each time interval, the processing components transform external input (if such exists) and previous memory states in a manner that corresponds to the transition of each computational state to its successor. The external input combined with the initial memory state constitute the initial string of a particular computation. Intermediate memory states constitute the relevant intermediate strings. Similarly, the output produced by the system (together with the final memory state) constitutes the final string. As long as the components of the system are functionally organised and synchronised so that their processing respects the well-defined ordering of the manipulated digits, the operation of the system can be described as a series of snapshots. The computational rule specifies the relationship that obtains between inputs and their respective outputs produced by modifying snapshots according to a set of instructions (ibid: pp. 509, 515).

5 Discussion

The literature contains many attempts to clarify the notions of computation *simpliciter* and digital computation, in particular. Matthias Scheutz, for example, has argued that

[8] Strictly, this requirement applies to systems that Piccinini dubs "fully digital" computing systems (Piccinini, personal communication). Other systems, which he dubs "input-output" digital computing systems, take digital inputs and produce digital outputs in accordance with a rule, but do not execute a step-by-step program (e.g., some connectionist networks).

there is no satisfactory account of implementation to answer questions critical for computational cognitive science (1999: p. 162). He does not offer a new account of concrete computation. Instead he suggests approaching the implementational issue by starting with physical digital systems progressively abstracting away from some of their physical properties until a (mathematical) description remains of the function realised. In a similar vein, David Chalmers (2011) also focuses on the implementational issue, only to offer a new mathematical formalism of computability that is based on combinatorial state automata (supplanting the traditional finite state automata). He too argues that a theory of implementation is crucial for (digital) computation to establish its foundational role in cognitive science. The motivation behind both Scheutz and Chalmers' efforts to clarify the notion of implementation is to block attempts by Putnam and Searle (and others) to trivialise computation (and undermine computationalism).

Other notable discussions of computation in cognitive science include David Israel (2002), Oron Shagrir (2006), Piccinini (2006, Piccinini & Scarantino 2011), Smith (2002, 2010) and the (long) list continues. Israel (2002) claims that often it seems that a better understanding of computation is hampered by philosophical concerns about mind or cognition. Yet "[o]ne would, alas, have been surprised at how quick and superficial such a regard [to computation] has been" (ibid: p. 181). Shagrir (2006) examines a variety of individuating conditions of computation showing that most of them are inadequate for being either too narrow or too wide. Although he does not provide a definitive answer as to what concrete computation is, he points out that neither connectionism nor neural computation nor computational neuroscience is compatible with the widespread assumption that digital computation is executed over representations with combinatorial structure.

Importantly, two uncommon examples of genuine attempts to explicate the notion of computation are Piccinini's and Smith's. Piccinini (2006) demonstrates how on various readings of computation, some have argued that computational explanation is applicable to psychology, but not, for instance, to neuroscience. Still, neuroscientists routinely appeal to computations in their research. Elsewhere, Piccinini examines the implications of different types of digital computation (as well as their extensions' relations of class inclusion) for computational theses of cognition (Piccinini and Scarantino 2011).

But as far as I am aware, nobody else in the literature has ever undertaken a more ambitious project than Smith to systematically examine the extant accounts of computation and their role in both computer science and cognitive science. In his 2002 "The foundations of computing", Smith lists the following six construals of computation: FSM, Effective Computability, Algorithm Execution, Digital State Machines, Information Processing and PSS. [9] His *Age of Significance* project (which is now long coming) aims to shed some light on the murky notion of computation, putting each one of these construals under careful scrutiny (Smith 2010). Surprisingly enough, Smith concludes that there is *no* adequate account of computation and *never*

[9] In an unpublished chapter from the Age of Significance, Smith adds the following construals: Calculation of a Function, Interactive Agents, Dynamics and Complex Adaptive Systems.

will be one. For computers per se are not "a genuine subject matter: they are not sufficiently special" (ibid: p. 38). Pace Smith, I do not believe that there is a compelling reason to reject *all* accounts of computation as inadequate, let alone to preclude the possibility of ever coming up with an adequate account. Still, I strongly agree that the accounts are different and many of them are indeed inadequate for explaining concrete computation.

My main argument here proceeds as follows:

- (P1) There are many accounts of digital computation at our disposal.
- (P2) These accounts establish different (but not all irreducibly different) requirements for a physical system to perform digital computation.
- (P3) Therefore, extant accounts of computation are non-equivalent.
- (P4) Cognitive capacities are sometimes explained by invoking digital computation terminology.
- (P5) When employing an equivocal interpretation, one needs to commit to an explicit interpretation (or account).
- Therefore, one needs to commit to an explicit account of computation when explaining cognitive capacities by invoking digital computation terminology. Specifically, any computational thesis of cognition is unintelligible without a commitment to a specific account of computation.

The truth of the first premise is evident in the philosophical literature (cf. Piccinini 2007; Shagrir 1999; and Smith 2002, 2010). In addition to the FSM, PSS and Mechanistic accounts examined here, there are also the Algorithm Execution account, the Gandy-Sieg account, the Information Processing account as well as others.

Similarly, premise four (at least) seems self-evident. Computationalists take premise four for granted (Fodor 1975, Pylyshyn 1984, Newell & Simon 1976, Marr 1982, van Rooij 2008) and so do some connectionists. *Radical* dynamicists do not subscribe to the computational theory of mind (Van Gelder & Port 1995, Thelen & Smith 1994), yet they reject it without committing to any particular account of computation proper. For they presuppose that digital computation is inherently representational. *Other* dynamicists do not deny that *some* aspects of cognition may be representational and be subject to a computational explanation.

Yet, this presupposition is unjustified. Digital computation (but not computationalism) could be explained without invoking any representational properties (barring internal representations) by appealing to causal or functional properties instead (see Fresco 2010 and Piccinini 2008a). As van Rooij (2008: p. 964) rightly points out, computation and computationalism have become associated with the symbolic tradition, but only sometimes with specific models in this tradition. Some accounts of concrete digital computation are indeed representational (cf. the reconstruction of Smith's participatory account in Fresco (2011) as well as the FSM and PSS accounts discussed above), but others need not be (cf. Copeland 1996, Chalmers 1994, the Mechanistic account discussed above). This simply reinforces the need to commit to a particular account of computation.

Moreover, premise five simply calls for disambiguation when there is an equivocation in terms. When some phenomenon is open to two interpretations or more, we should commit to one interpretation to avoid ambiguity. For instance, the concept depression has at least two typical meanings. In the sentence, "The great depression started in most countries in 1929 and lasted for a long time", it is clear that 'depression' means a long-term downturn in economic activity. On the other hand, in the sentence, "Long depression leads to making irrational decisions", 'depression' means something different. Similarly, when one asserts that hierarchical planning or linguistic tasks, for example, are computational, one ought to commit to a particular account of computation.[10] Is it in virtue of executing an algorithm, formally manipulating symbols, or implementing a TM that cognitive agents engage in hierarchical planning?

Furthermore, the commitment to a particular interpretation should be consistent to avoid further ambiguity. From the two sentences above it follows that irrational decisions were made in the countries that suffered the Great Depression in 1929. This conclusion would only validly follow from its premises, if 'depression' had the *same* interpretation in both premises. Otherwise, whilst this conclusion may be *plausible*, it does not follow. This is also known as the fallacy of equivocation. Similarly, if one explains a particular cognitive capacity in virtue of an explicit account of concrete digital computation, one has to consistently adhere to that account. An explanation of a linguistic task in virtue of formal symbol manipulation and then in virtue of algorithm execution ceases to be a coherent story, since they are not equivalent.

Importantly, not only are the extant accounts of concrete digital computation *intensionally* different, they are also *extensionally* different. These accounts offer different perspectives on what a physical computing system does. But rather than having the same *extension*, these accounts end up denoting different classes of computing systems. For example, the second requirement of the FSM account excludes computing systems that are neither program-controlled nor programmable[11]. For such systems do not follow semantically represented rules, instead the "rules" are hardwired. A physical symbol system is explicitly classified by Newell and Simon as an instance of a UTM (1976: p. 117). This classification is also derivable from the conjunction of the fourth and fifth requirements of the PSS account. Also, the FSM and PSS accounts exclude computing systems such as Gandy machines[12] and discrete neural networks, for they violate Turing's locality condition and do not necessarily operate on explicit symbolic expressions. The Mechanistic account, on the other hand,

[10] To be clear, *digital computation* is not ambiguous in the same sense that *depression* is. The aforementioned accounts offer specific proposals for how 'digital computation' should be understood, but they are still related by a more general sense of 'digital computation'. The example above is simply meant to emphasise the need for disambiguation.

[11] A special purpose TM is an example of a program-controlled system that is not programmable.

[12] A Gandy machine (introduced by Turing's student, Robin Gandy in 1980) can be conceptualised as multiple TMs working in parallel, sharing the same tape and possibly writing on overlapping regions of it.

is far less restrictive in terms of the systems it classifies as digital computing systems, including UTMs, and special purpose TMs, but also FSAs, discrete neural networks, primitive logic gates and even hypercomputers (see figure 1).

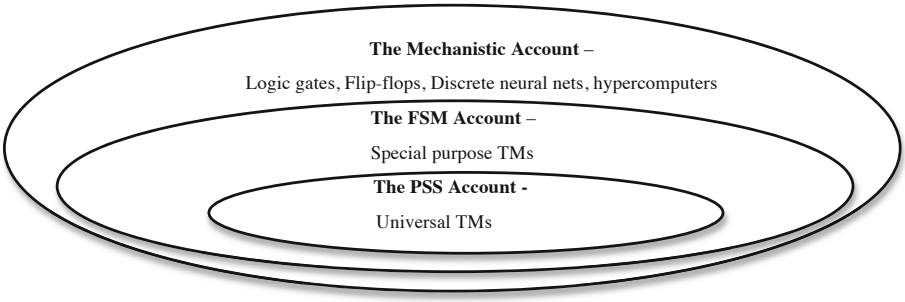

Fig. 1. With the exception of hypercomputers, UTMs are the most powerful and flexible computing systems in the class above (e.g., they can simulate any discrete neural net). Still, UTMs (and physical approximations thereof) do not exhaust all types of digital computing systems. The Mechanistic account is the broadest of the three accounts examined.

Prima facie, it might seem that premise two is self-defeating, but this is not the case. A possible consequence of *all* the requirements not being irreducibly different is some overlap between requirements of various accounts. Thus, the requirements that are implied by one account *could be* reduced to some of the other requirements.[13] And if all the requirements could be reduced to a coherent minimal set of key requirements, then this would constitute a *single* account of computation. Premise three would then no longer follow from the preceding premises. However, premise two suggests that although *some* of the requirements may overlap, not *all* of them do. For instance, the fourth key requirement of the Mechanistic account presupposes in some cases the existence of memory (whose cells stabilise on certain digits). Similarly, the sixth requirement of the PSS account and the sixth requirement of the FSM account both demand memory for storing and retrieving symbolic expressions. Still, some requirements of one account, such as spatiotemporal synchronisation of processing digits belonging to the same string (i.e., the Mechanistic account second requirement), cannot be reduced to any of the requirements of the competing accounts.

Possible challenges to my conclusion might be that some of the key requirements implied by different accounts could be synthesised or that one could simultaneously subscribe to two accounts or more. The first challenge may result in sidestepping the demand to commit to an explicit account. But even if that were the case, such a synthesis would simply yield a new (possibly adequate!) account of computation. The second challenge needs unpacking. It can be interpreted in one of two ways. Firstly, it could be interpreted as subscribing to more than one account simultaneously for

[13] An overlap among requirements clearly does not imply a reduction from one requirement to another. My intent here is to address a possible criticism to the effect that premise three would no longer follow as an intermediate conclusion from its preceding premises.

explaining different cognitive capacities respectively. I do not see that as a problem. There is still a need to commit to a particular account for each relevant cognitive capacity. But this could have some other consequences, such as explaining cognitive behaviour in a non-unified manner by resorting to a plethora of computational models. It will require a compelling account of how the different computational (cognitive) subsystems interrelate.

Secondly, the challenge could be interpreted as subscribing to several accounts simultaneously, since cognitive explanations by nature span multiple levels. This is consistent with Marr's (1982) tripartite analysis. For instance, we could hold that (1) cognitive computations are inherently representational. At the same time, we could also hold without being inconsistent that (2) these computations are constrained in terms of any one of the formalisms of computability, and lastly that (3) they occur in the brain, which is embodied and situated in the real world. This is all well and good. Still, as I have argued above, concrete computation (but perhaps not cognitive computation) could be explained without necessarily invoking any representational properties (e.g., by the Mechanistic account above or the Algorithm Execution account in Copeland 1996). If one wishes to commit to a representational account of digital computation, since *cognition* is representational, one should firstly justify *why* computation proper *is* representational. Also, subscribing to an account of concrete computation and to a formalism of computability simultaneously does not introduce any conflict.

Although some of the key requirements of the three accounts overlap, others do not, suggesting that there is sufficient dissimilarity between these accounts. For example, the conjunction of the fourth[14] and fifth requirements of the PSS account can be reduced to the FSM account's first requirement. The PSS account's fourth and fifth requirements amount to the universality property of soft-programmable computing systems that is achieved by means of symbolic expressions used either as data or as programs ensuring maximal plasticity of function. In addition, both the FSM account's fourth requirement and the first part of the PSS account's seventh requirement demand the capacity to handle an unbounded number of representations[15] (or symbolic expressions designating some entities).

Another seemingly important similarity between the FSM and PSS accounts (but not the Mechanistic account) is that computing systems engage in information processing at the *symbolic* level. For instance, Fodor and Pylyshyn (1988: p. 52) claim that "conventional computers typically operate in a 'linguistic mode', inasmuch

[14] The PSS account's fourth requirement demands the existence of expressions that represent every process of which the machine is capable to support the full plasticity of behaviour of the computing system. But it is not clear why it is necessary that *every* such expression exist. There could be a mismatch between the set of all functions and the set of all expressions representing them. For instance, some functions could be the serial invocation of several expressions (themselves representing other functions).

[15] Only the FSM account explicitly states that the unbounded number of representations is produced by means of compositionality.

as they process information by operating on syntactically structured expressions". As well, Newell and Simon's (1972: p. 870) fundamental working assumption is that "the programmed computer and human problem solver are both species belonging to the genus IPS" (that is information processing systems). The Mechanistic account, on the other hand, does not equate information processing and digital computation.[16] Still, it is not clear in what sense the information-processing characterisation of computing systems adds anything operative to the classification of certain physical systems as performing digital computation. This is stipulating that processing of information amounts to the production, modification and deletion of information.

A well-known non-semantic reading of information is based on Claude Shannon's concept of information (1948), but it is not clear what processing of Shannon Information amounts to. His theory dealt with information syntactically: whether and how much information is conveyed. Its basic idea is coding messages into a binary system at the bare minimum of bits we need to send to get our message across while abstracting from the physical media of communication. The amount of information conveyed is defined as the uncertainty (or entropy) associated with particular messages. The deletion and modification of information is only possible in the presence of noise. Noisy channels may displace information, but this is not the same as a deliberate deletion of information in computing systems, say to free up memory resources or reduce the size of a database. Although error detection and correction methods modify information to offset noise, symbolic expressions could be modified for many purposes other than error correction. The production of new information is more problematic, for the only source of new information, according to Shannon (1948: p. 12), is *uncertainty*.

Another possible non-semantic reading of information is based on *Algorithmic Information,* which was introduced by Ray Solomonoff and Andrei Kolmogorov. But even on this reading, it is not immediately clear what information processing amounts to in the context of concrete computation. The algorithmic information of a string X is defined as the length of the shortest program on a UTM that generates X as its output. Algorithmic information seems a more suitable candidate as the basis of an information processing characterisation of computation. For it is defined over algorithms, rather than over randomness of messages. Yet, the problem of processing algorithmic information remains, as it is invariant to the process of computation itself.

Importantly, as stated by Solomonoff, the actual value of Kolmogorov complexity of a string is incomputable, it can only be approximated (2009: pp. 6-7). This limitation prevents us from actually having a full description of all the possible optimal algorithms (that are also enumerable) to solve a specific problem (Calude 2009: p. 82, Calude et al, 2011). Still, a variation on algorithmic information theory, which is not based on UTMs but rather on Finite State Transducers, does allow us to compute the complexity of strings. This variation, however, comes at the cost of

[16] Instead, according to the Mechanistic account, information processing entails *generic* computation (the superclass of both digital computation and analogue computation) (Piccinini and Scarantino 2011: pp. 33-34). For information is a medium-independent concept. However, *digital* computation does not entail information processing, because although digits *could* carry information, they *need not* do so essentially.

Turing universality that does not apply to finite state transducers, since there is no universal transducer (Calude et al, 2011). Yet, algorithmic information theory will have a limited capacity to explain cases in which information is deleted and/or modified whilst the overall information complexity of the computing system does not decrease.

Other possible candidates for the *information processing* characterisation of computing systems alluded to by the FSM and PSS accounts are based on a semantic reading of information. Two main types of semantic information are *factual* information and *instructional* information. The former type is objective propositional information representing some facts or states of affairs, and arguably only qualifies as information if it is true (yet, this is a contentious claim). The latter type is not *about* facts or state of affairs, so it is not qualified alethically (Floridi 2009: pp. 35-36). Instructional information is conveyed either unconditionally (e.g., step 1: do this, step 2: do that) or conditionally (e.g., if X do this, otherwise do that). The subtleties of semantic information are not discussed further here for lack of space. However, since algorithms are finite sets of instructions, instructional information seems a plausible candidate as the basis of characterising digital computation as information processing.

Moreover, the Mechanistic account is grounded in *physical mechanisms* that perform computations, whereas both the FSM and PSS accounts are grounded in *symbolic* computation and *semantics*. Digits in the Mechanistic account are *not* symbols, but rather states of components (and are as physical as it gets). So, they have no representational character and their processing is independent of any (external) semantics. The second requirement of the FSM account, in contrast, emphasises that symbolic expressions are manipulated according to *formal* rules and must always be semantically interpretable even following numerous manipulations. The second requirement of the PSS account emphasises that symbols are manipulated in virtue of their semantics.

Incidentally, the semantics of symbols and their manipulation is a key difference between the PSS and FSM accounts. Although both accounts are based on the manipulation of symbols at the heart of the computational process, they diverge on how semantics enters this process. According to the FSM account's second requirement, symbols are formally manipulated *in virtue of* their syntax, but they are always semantically interpretable. It is a property of automatic formal systems that symbolic expressions continue to "make sense" when manipulated by truth-preserving rules. On this view, the formal manipulation of symbols based on their syntax is sufficient for the operation of the computing system. And the semantics of the manipulated symbolic expressions is epiphenomenal on their syntax.

However, the second requirement of the PSS account reveals that processes in computing systems are *causally* affected by the semantics of symbols. The behaviour of a process P (with X as its input) depends on a potentially distal entity Y, which is designated by X. The designation requirement is vague, for it leaves the ways in which a process depends on some entity unspecified. It might be the case that Newell and Simon took it for granted that symbols *symbolise* by definition and so they have not explicated where their external semantics comes from. If indeed *external* semantics is required for computation, then this gap is too big to be left unexplicated.

Internal access to symbols and expressions in a conventional digital computer is an assignment operation of, say, a symbol to some other internal entity and it is a primitive in its architecture (e.g., for memory retrieval). But there is no similar primitive for the external environment (Bickhard and Terveen 1995: pp. 93-94).

Additionally, the Mechanistic account emphasises the importance of synchronisation of processing digits belonging to the same string, whereas both the PSS and FSM accounts ignore temporal constraints of concrete computation. According to the second requirement of the Mechanistic account, with the growth in complexity of the computing system it becomes more crucial that digits belonging to the same string be processed in the same functionally relevant time interval. The other two accounts, while recognising the temporal aspects of concrete computation, do not explicitly mandate any temporal constraints on computing systems.

In sum, the above differences discussed as well as others clearly confirm that the extant accounts of concrete digital computation are not equivalent. The key motivation behind both the FSM and PSS accounts is advancing a substantive empirical hypothesis about how human cognition works, namely, that cognition is essentially a computational system of the specified sort. The Mechanistic account, on the other hand, has a different and less ambitious motivation. Rather than advancing an empirical hypothesis about cognition. Piccinini's goal in formulating his account is to provide a general characterisation of digital computing systems. He attempts to exclude as many paradigmatic cases of non-computing systems (such as planetary systems, digestive systems, mouse traps, etc.) as possible. At the same time, his account classifies many (but not too many) systems as performing digital computation. The FSM account is more restrictive and excludes any systems that are neither programmable nor program-controlled from the class of computing systems. The PSS account is even more restrictive, as it includes only UTMs (and physical approximations thereof) as genuinely computational.[17] Regardless of the (doubtful) representational character of computation presumed by the FSM and PSS accounts, they are simply too restrictive as accounts of concrete computation proper.

6 Conclusion

There is no question whether mathematical formalisms of computability are adequate analyses of *abstract* computation, but they are of the wrong kind to explain *concrete* computation. Any particular formalism does not specify the relationship between abstract and concrete computation. It is at the physical level that the algorithm (or more precisely, program) is specified and constrained by the implementing physical medium. So, stipulating that any complete account of a physical phenomenon must *also* consider its physical implementation, an explicit account of concrete computation has to be specified for a complete account of concrete computing systems.

[17] PSS yields a very restrictive class of computing systems that makes sense when considering cognitive systems. Since cognition exhibits substantial flexibility, it is unreasonable to assume that it is an instance of, say, a special purpose TM.

My main argument was that well-known accounts of concrete computation entail sufficiently distinct requirements for a physical system to compute, justifying the demand that one commits to a particular account when employing the notion of concrete computation. But despite the apparent straightforwardness of this argument, all too often its implied moral is surprisingly ignored by philosophers and cognitive scientists alike. The notions of computation *simpliciter* and digital computation, in particular, are employed without much awareness of what they mean exactly. At times, extant accounts are even used interchangeably as though they were equivalent (when they are not even extensionally equivalent). If we take cognition to be a physical phenomenon that can be explained computationally, we should state explicitly what we mean by (digital) computation. Otherwise, a computational thesis of cognition remains unintelligible.

Acknowledgments. Thanks are due to Eli Dresner, Gualtiero Piccinini and Frances Egan for providing helpful comments on a recent draft of this paper. I would also like to thank two anonymous referees for the Solomonoff 85th memorial conference for useful comments. I am grateful to Phillip Staines for detailed comments on various drafts of this paper. All these comments contributed to this final version, which, I trust, is much improved.

References

1. Bickhard, M.H., Terveen, L.: Foundational issues in artificial intelligence and cognitive science: Impasse and solution. Elsevier Scientific, Amsterdam (1995)
2. Calude, C.S.: Information: The algorithmic paradigm. In: Sommaruga, G. (ed.) Formal Theories of Information. LNCS, vol. 5363, pp. 79–94. Springer, Heidelberg (2009)
3. Calude, C.S., Salomaa, K., Roblot, T.K.: Finite state complexity. Theoretical Computer Science 412(41), 5668–5677 (2011), doi:10.1016/j.tcs.2011.06.021
4. Chalmers, D.: On implementing a computation. Minds and Machines 4, 391–402 (1994)
5. Chalmers, D.J.: A computational foundation for the study of cognition. Journal of Cognitive Science 12(4), 323–357 (2011)
6. Copeland, B.J.: What is computation? Synthese 108, 335–359 (1996)
7. Floridi, L.: Philosophical conceptions of information. In: Sommaruga, G. (ed.) Formal Theories of Information. LNCS, vol. 5363, pp. 13–53. Springer, Heidelberg (2009)
8. Fodor, J.A.: The language of thought. Harvard University Press, Cambridge (1975)
9. Fodor, J.A.: Methodological solipsism considered as a research strategy in cognitive science. Behavioral and Brain Sciences 3, 63–73 (1980)
10. Fodor, J.A., Pylyshyn, Z.W.: Connectionism and cognitive architecture: a critical analysis. Cognition 28, 3–71 (1988)
11. Fresco, N.: Explaining computation without semantics: keeping it simple. Minds and Machines 20, 165–181 (2010)
12. Fresco, N.: Concrete Digital Computation: What Does it Take for a Physical System to Compute? Journal of Logic, Language and Information 20(4), 513–537 (2011), doi:10.1007/s10849-011-9147-8
13. Gandy, R.: Church's thesis and principles for mechanisms. In: Barwise, J., Keisler, H.J., Kunen, K. (eds.) The Kleene Symposium, pp. 123–148. North-Holland, Amsterdam (1980)

14. Haugeland, J.: AI: the very idea. The MIT Press, Cambridge (1985)
15. Israel, D.: Reflections on Gödel's and Gandy's reflections on Turing's thesis. Minds and Machines 12, 181–201 (2002)
16. Kleene, S.C.: Mathematical logic. Dover, New York (2002)
17. Marr, D.: Vision: a computational investigation into the human representation and processing visual information. Freeman & Company, New York (1982)
18. Newell, A., Simon, H.A.: Human problem solving. Prentice-Hall, Englewood (1972)
19. Newell, A., Simon, H.A.: Computer science as an empirical enquiry: symbols and search. Communications of the ACM 19, 113–126 (1976)
20. Newell, A.: Physical symbol systems. Cognitive Science 4, 135–183 (1980)
21. Piccinini, G.: Computational explanation in neuroscience. Synthese 153, 343–353 (2006)
22. Piccinini, G.: Computing mechanisms. Philosophy of Science 74, 501–526 (2007)
23. Piccinini, G.: Computation without representation. Philosophical Studies 137, 205–241 (2008a)
24. Piccinini, G.: Computers. Pacific Philosophical Quarterly 89, 32–73 (2008b)
25. Piccinini, G., Scarantino, A.: Information processing, computation, and cognition. Journal of Biological Physics 37, 1–38 (2011)
26. Pylyshyn, Z.W.: Computation and cognition. The MIT Press, Cambridge (1984)
27. Pylyshyn, Z.W.: Computing in cognitive science. In: Posner, M.I. (ed.) Foundations of Cognitive Science, pp. 51–91. The MIT Press, Cambridge (1989)
28. Pylyshyn, Z.W.: Things and places: how the mind connects with the world (Jean Nicod Lectures). The MIT Press, Cambridge (2007)
29. Scheutz, M.: When physical systems realize functions. Minds and Machines 9, 161–196 (1999)
30. Scheutz, M.: Computationalism – the next generation. In: Scheutz, M. (ed.) Computationalism: New Directions, pp. 1–22. The MIT Press, Cambridge (2002)
31. Shagrir, O.: What is computer science about? The Monist 82, 131–149 (1999)
32. Shagrir, O.: Why we view the brain as a computer. Synthese 153, 393–416 (2006)
33. Shannon, C.E.: A mathematical theory of communication. Mobile Computing and Communications Review 5, 1–55 (1948)
34. Smith, B.C.: The foundations of computing. In: Scheutz, M. (ed.) Computationalism: New Directions, pp. 23–58. The MIT Press, Cambridge (2002)
35. Smith, B.C.: Age of significance: Introduction (2010), `http://www.ageofsignificance.org` (retrieved May 3, 2010)
36. Solomonoff, R.J.: Algorithmic probability - theory and applications. In: Emmert-Streib, F., Dehmer, M. (eds.) Information Theory and Statistical Learning, pp. 1–23. Springer Science+Business Media, NY (2009)
37. Thelen, E., Smith, L.B.: A dynamical systems approach to the development of cognition and action. The MIT press, Cambridge (1994)
38. van Gelder, T., Port, R.F.: It's about time: an overview of the dynamical approach to cognition. In: van Gelder, T., Port, R.F. (eds.) Mind as Motion. The MIT Press, Cambridge (1995)
39. van Rooij, I.: The tractable cognition thesis. Cognitive Science 32, 939–984 (2008)