Minds & Machines (2008) 18:379–401
DOI 10.1007/s11023-008-9111-9

An Analysis of the Criteria for Evaluating Adequate Theories of Computation

Nir Fresco

Received: 27 August 2007 / Accepted: 1 July 2008 / Published online: 15 July 2008
© Springer Science+Business Media B.V. 2008

Abstract This paper deals with the question: What are the criteria that an adequate theory of computation has to meet? (1) *Smith's answer*: it has to meet the empirical criterion (i.e. doing justice to computational practice), the conceptual criterion (i.e. explaining all the underlying concepts) and the cognitive criterion (i.e. providing solid grounds for computationalism). (2) *Piccinini's answer*: it has to meet the objectivity criterion (i.e. identifying computation as a matter of fact), the explanation criterion (i.e. explaining the computer's behaviour), the right things compute criterion, the miscomputation criterion (i.e. accounting for malfunctions), the taxonomy criterion (i.e. distinguishing between different classes of computers) and the empirical criterion. (3) *Von Neumann's answer*: it has to meet the precision and reliability of computers criterion, the single error criterion (i.e. addressing the impacts of errors) and the distinction between analogue and digital computers criterion. (4) *"Everything" computes answer*: it has to meet the implementation theory criterion by properly explaining the notion of implementation.

Keywords Cognition · Computation · Computationalism · Computers · Implementation · Practice · Subject · Matter · Theory · Turing machines

Introduction

There's a widespread tendency to compare minds to computers and to explain minds in computational terms. However, I maintain that a deeper understanding of

N. Fresco (✉)
School of History and Philosophy, The University of New South Wales, Sydney, NSW, Australia
e-mail: fresco.nir@gmail.com; nir.fresco@optusnet.com.au

computation is required beforehand. According to proponents of computationalism,[1] minds are computers, i.e., mechanisms that perform computations. In my view, the main reason for the controversy about whether computationalism is accurate in its current form, or how to assess its adequacy is the lack of a satisfactory theory of computation. Before a critical debate regarding the relation between computations and minds can take place some preliminary groundwork is needed.

The purpose of this paper isn't to offer a theory of computation. It's rather meant to resist Smith's discouraging claim that no such theory is possible. His project begins as a search for a comprehensive theory of computation, which is able to do empirical justice to practice and cognitive justice to the computational theory of mind. A rigorous commitment to the three criteria outlined below ultimately leads him to recommend a radical overhaul of our traditional conception of metaphysics. The focus of philosophical discussions concerning computation in the second half of the 20th century shifted from the disciplines of logic and mathematics into cognitive sciences and philosophy of mind, primarily related to computational theories of mind. However, Smith asserts that what was lacking throughout these philosophical discussions was a foundational investigation of the nature of computation itself.

In what follows, I argue that Smith's criteria are inadequate and over demanding. These criteria have not only led him to reject existing theories of computation as inadequate, but also to pre-empt any venture to provide a satisfactory theory. By presenting the competing answers I show that there are acceptable alternatives to Smith's view, which allow for future theories of computation to be put forward again. My aim is not to nominate the 'correct' answer, but to point out the criteria that are inadequate, and to emphasize those that are mandatory for candidate theories of computation.

Evidently, the first answer advocated by Smith is the primary one examined in the paper. It states that an adequate theory of computation has to meet strict criteria. It has to do justice to both computational practice and the computation theories of mind. Firstly, every satisfactory theory of computation should be able to account for the computational systems that made Silicon Valley famous. It has to distinguish models from implementations, analyses from simulations etc. Secondly, computation became an essential ingredient in cognitive sciences and philosophy of mind. A candidate theory has to take its consequences for computationalism into account.

The second answer advocated by Piccinini states that an adequate theory of computation has to do justice to the practices of computer scientists and computability theorists. He emphasizes the importance of clearly distinguishing between things that compute and things that don't. He also maintains that sufficient attention should be given to failure to compute correctly as part of a satisfactory account. Piccinini offers his own account of computation, namely the *mechanistic account of computation.* According to this account computation doesn't presuppose representation or semantic content, unlike many accounts in the philosophical literature (cf. Fodor 1975; Pylyshyn 1989). Hence, in his opinion, it is ideal for grounding the comparison and assessment of computational theories of mind.

[1] Throughout this paper I use computationalism and the computational theory of mind interchangeably to denote the same thing.

The third answer advocated by Von Neumann is an attempt to outline the logical foundations of computation. His approach was characterized by the application of mathematical and logical methods to the foundations of computation. Von Neumann compared the logical aspects of computers with living organisms and the organization of the central nervous system. The analogue—digital distinction criterion isn't unique to theories of computation; living organisms also exhibit this principle.

The fourth answer advocated by Scheutz, Putnam and others states that a satisfactory account of computation is underpinned by an adequate theory of implementation. Views like those of Putnam and Searle imply a very loose notion of computation so that almost everything can be deemed to be computing. Searle's notorious wall and Putnam's realization theorem of finite automata suggest that even a *rock* can be claimed to be computational. Scheutz suggests that Putnam's realization theorem emphasizes the need for a *theory of implementation*. He offers an account that appeals to function realization, rather then the standard concept of physical state-to-computational state correspondence.

The answers presented in this paper can be found in relevant literature, but they are not exhaustive by any means. Readers, who are interested in the foregoing question, are invited to consider this paper as a starting point for further research. Supplementary answers could be taken into account like those advocated by Church and Turing, Gandy, Copeland, Fodor and Pylyshyn and others as well as theories of quantum or molecular computations.

Smith's Answer

According to the first answer an adequate theory of computation has to meet the empirical criterion, the conceptual criterion and the cognitive criterion (Smith 1996, pp. 14–17; 2002). Smith claims that the extant construals fail to meet either a single criterion or a combination of criteria. In his view, questions like what computers are or what computation is require tackling other questions of metaphysical nature. Any attempt to characterise computers as universal, programmable, rule following etc. inevitably appeals to higher order properties. And these properties are of the wrong metaphysical kind to be candidates for what is distinctive or characteristic about computation. He asserts that not only must an adequate account meet the three criteria above and include a theory of semantics; it must also include a theory of *ontology*. It is not just intentionality that is at stake, in his view, but so is metaphysics. The most serious problems that stand in our way of developing an adequate account of computation are as much ontological as they are semantical.

The formal symbol manipulation construal, for instance, is usually defined as manipulation of symbols in a way that is independent of their interpretation. Thus, some may think that it needn't rely on any semantic foundations. In his opinion, this is simply mistaken because it is only independent at the level of the phenomenon. But, at the ontological level this construal is dependent on semantics—it is defined in terms of interpretation of symbols. Symbols must have a semantic character, i.e., have actual interpretations, so that there is something substantive for their formal manipulation to proceed independently of. Without it, the formal symbol manipulation construal would simply be vacuous. Smith claims that the same applies to all extant construals of computation.

The Empirical Criterion

This criterion dictates the need to be compliant with the extant computational practice. It means that any such a theory should be capable of explaining a program like the Open Office Writer. It should account for its construction, maintenance and everyday use (Smith 1996, p. 5; 2002, p. 24). The empirical criterion "does justice" to computational practice by keeping the analysis grounded in real world examples of computers. The computer (as well as the Internet) revolution demonstrates again and again its ability to evolve, expand and adjust beyond the alleged constraints of any computational theory. This criterion serves to question the legitimacy of all extant theoretical perspectives. In this context, *Silicon Valley* is nominated as the gatekeeper to decide whether in practice something may be deemed computational. An adequate theory of computation must make a substantive *empirical* claim about what Smith calls *computation in the wild*, which is the body of practices, techniques, machines, networks etc. that revolutionized the last decades (Smith 1996, pp. 5–6; 2002, pp. 24–25).

The Conceptual Criterion

This criterion dictates the need to repay all the intellectual debts, in the sense that any such theory clearly ought to explain underlying concepts like: compiler, interpreter, algorithm, semantics etc. With that in mind, we should understand what the theory says, its origins and its implications (Smith 2002, p. 24). The conceptual criterion, which is no more than a meta-theoretical constraint on any theory, is especially crucial in the computational case for two main reasons. The first reason is that many candidate theories of computation rely on important notions such as interpretation, representation and semantics with no proper explanation of these notions. The second is that there's a widespread tendency to resort to computation as a possible theory of exactly those very 'disobedient' notions. The end result is thus a conceptual circularity that deprives candidate theories of their explanatory power (ibid).

The Cognitive Criterion

This criterion dictates the need to provide solid grounds for the computational theory of mind, often known as *computationalism,* the thesis that regulates the traditional fields of artificial intelligence and cognitive science (ibid). This criterion is also a meta-theoretical constraint on the form of any candidate theory of computation. In the present context, computationalism has potential epistemological consequences depending on the theory of computation one chooses to endorse. If the computational theory of mind were true then a theory of computation would apply not only to computing in general, but also at the meta-level to the process of theorizing. In other words the theory's claims about the nature of computation would apply to the theory *itself* (i.e. the product). So if computationalism was true, then upon judging a candidate theory of computation and finding it to be adequate or not, there will be supposedly no reason to trust the conclusion. The reason for that is that the presumed meta-theory is conceptually inadequate. In sum this criterion directly translates to the following questions:

1. What computational theory of mind would be generated?
2. What form theories in general would take, on such a model of mind?
3. What would the candidate theory of computation in question look like?
4. Would the resulting theory of computation hold true of computation in the wild?
5. Would mentation and theorizing be computational as well?

None of those commits in advance to computationalism being true or false (Smith 1996, pp. 6–8; 2002, pp. 25–28).

Asking too Much and too Little

Smith rightly asks that candidate theories do justice to computational practices or to what he calls *computation in the wild*. Elsewhere he claims that there's a big gap between the theory and the practice, which the theory won't be able to overcome. Smith calls it the explanatory gap. True, any such theory has to account to some extent for the computer practices that have penetrated every aspect of our lives in recent years. Indeed, a candidate theory shouldn't be fully abstract and detached from computation in the wild. But the technological gap, which is boosted by the computer revolution and the Internet revolution, doesn't force us to give up in advance simply because the theory is supposedly left far behind. An adequate account needn't capture and explain all the aspects of *every* existing technological breakthrough in computer science, artificial intelligence, molecular computers etc.

Computation is a highly diversified and fluid concept, which may not be fully explained by a strict and definite theory. Instead a more flexible—context based account should be sought. Wittgenstein's (2001, pp. 52–56) rejection of general explanations and definitions based on sufficient and necessary conditions in his discussion about language and games may be very well applicable to computation. Rather than looking for one essential core to account for all the computation practices and defining a clear boundary, a disjunctive account can be given. A candidate theory can account for Turing machines, desktop computers and compilers. This theory can be limited to do justice to the basic computing devices only or be extended to account for more complex devices such as parallel and distributed computers, high speed network elements, expert systems, molecular computers etc. Every such computing device or computation process will be accounted for based on the context and whether it exhibits sufficient family resemblance to the concept of computation. This however isn't to say that such a theory would be so loose as to accept *any* device, since this may result in an account that attributes computation to any physical system (e.g. Searle's wall).

Smith (1996, pp. 8–9; 2002, pp. 28–31) also claims that primary candidate theories fail to meet the empirical criterion being incapable of making sense of current systems and even much less when the new generation is concerned.

Eventually Smith (2002, p. 51) makes the strong claim that it's not only that we don't currently have any satisfactory theory of computation, but also that we'll always fail to provide an adequate theory. I believe that this assertion is too hasty. The empirical criterion may be indeed hard to meet, but insufficient to dismiss any attempt to provide an adequate theory. There's nothing that prevents us from refuting a particular theory of computation, and providing a new account, which addresses the weaknesses of its predecessor. Precisely as Popper (2002a, pp. 124–125; b, pp. 9–10) suggested a good scientific theory is subjected to falsification under the appropriate conditions. A scientific theory should undergo genuine tests in an attempt to refute it. This method of elimination ensures that only the fittest theories survive. Scientific theories are tentative solutions to problems, which can never be justified, and theories of computation are no different. Computer science is by definition of a scientific nature and so are theories of computation. When an existing theory of computation is falsified, its weakness should be addressed by a new candidate theory.

Smith's cognitive criterion is no more convincing than his empirical criterion. Whether the claims of computationalism are true or not isn't meta-theoretically relevant to a candidate theory of computation. Regardless of the very critical debate regarding the legitimacy of computationalism, its claims needn't dictate any meta-level constraints on the nature of the theory of computation. If anything it should rather be the other way around. Any proponent of computationalism has to show why minds work the same way that computers do. Smith accurately claims that if computationalism were found to be true, there would be significant epistemological consequences for the process of theorizing itself. But if we take his view seriously, then computationalism can't be an adequate theory. To put it simply, according to computationalism, the mind is a computer (whether it is a digital computer or any other). Smith (2002, p. 51) claims that we'll always fail to provide an adequate theory of computers. From these two premises it follows that proponents of computationalism will always fail to provide an adequate computational theory of the mind.

Smith argues that if computationalism were true, then a theory of computation would apply not only to computing, but also to the process of theorizing. So unless one nails down the reflexive implications of the candidate theory of computation on theorizing itself, and examines this theory from a reflexively consistent standpoint, one will be incapable of judging whether it's adequate. If computationalism were false, then this criterion would become irrelevant anyway. But if it were true, then the cognitive criterion would a-priori pre-empt any attempt to produce a theory of computation. The result would be that a theory, which deals with computers, is reflexive and deals with minds as well. The cognitive process of theorizing itself may thus be said to be computational. The best approach to deal with Smith's argument will be to simply avoid the trap. The burden of addressing the supposedly reflexive characteristic of the theory lies on *computationalism* rather on any *theory of computation*. According to computationalism minds are computers in whatever way that computers are computers. Hence, proponents of the computationalism ought to address any reflexive implications of any particular theory of computation.

In my opinion the five questions, which underlie the cognitive criterion, don't contribute much from a meta-theoretical perspective when assessing the adequacy of a candidate theory. In reply to the first question: "what computational theory of mind would be generated?" my response is that if anything, it should be rephrased to pose a *difficulty* for candidate theories of computation. The following case, for instance, may be a potential obstacle for a candidate theory of computation. Assume that computationalism is true and human beings are indeed computers. And suppose that a particular theory of computation leads to the conclusion that human beings can't consciously follow an algorithm. Such a theory declares that no computer (of any kind), which works by following rules unconsciously, could consciously follow rules. An inevitable result would be that we have to give up either *this particular theory of computation* or *computationalism.*

Clearly, we follow many rules unconsciously. Even obeying traffic rules eventually becomes an almost programmed task, when we find ourselves driving everyday without even realizing how many driving related decisions we make unconsciously. But undoubtedly this doesn't mean that we don't consciously follow rules as well. When you walk along a country track, you step over little stones, tree branches and makes adjustments for various obstacles in your path of which you have no conscious awareness. Further, there is abundant evidence that indicates that people are capable of following rules consciously. This invites the question whether one should reject the candidate theory of computation or simply dismiss computationalism as false.

The particular theory of computation in this case declares that computers follow rules unconsciously and are incapable of consciously following rules. But such an analysis anthropomorphizes computation by introducing concepts like consciousness into the discourse, whereas theories of computation ought to be solely committed to capture the essence of *computation*. Evidently, the imminent outcome is a cross reference between the theory of computation and computationalism. Any 'careless move' in theorizing about computation immediately affects computationalism. And likewise any deviation in the way computationalism explains consciousness changes this particular theory of computation. Unfortunately, Smith maintains that a theory of computation has to be judged based on its consequences for computationalism rather then vice versa. The preferred approach should be such that candidate theories of computation remain detached from anthropomorphic explanations. It ought to explain necessary notions like algorithms, implementation, complexity, error handling etc. in logical and mathematical terms. Furthermore, one may also simply conclude that a particular theory of computation leads us to reject computationalism altogether[2] (e.g. people are undeniably capable of following algorithms consciously, thus they can't be computers of the sort mentioned above).

In reply to the second question: "what form theories in general would take, on such a model of mind?" my response is: it shouldn't matter. The reasons for that were already outlined above. The same applies for the third question. In reply to the fourth question: "would the resulting theory of computation hold true of computation in the wild?" my response is no different than my criticism of the

[2] It may easily turn out that some phenomena like consciousness can't be explained in computational terms as its essence can't be fully captured by simply appealing to algorithmic processes or information processing accounts and the likes. However, certain kinds of cognitive processes and capacities such as learning, inferring, calculating etc. may still be given computational accounts. This may lead to providing a weaker version of computationalism rather then dismissing it altogether.

empirical criterion. In reply to the last question: "would mentation and theorizing be computational as well?" my response is: trivially, yes. Mentation and theorizing should be explained in terms of minds and cognitive processes. Theorizing is only made possible due to our thinking faculty and is as such a direct derivative of the mind. In consequence, if computationalism were true, then mentation and theorizing would be likewise computational.

It appears to me that Smith is asking *too much and too little* with regard to potential theories of computation. The foregoing criteria constrain any candidate theory and leave very slim chances of providing an adequate theory. On the other hand, I believe that he has overlooked other essential criteria, which should be seriously considered. For instance, the miscomputation criterion and what I call the dichotomy criterion. The former criterion dictates that any theory of computation has to explain miscomputation as an inevitable feature of computation. I shall elaborate more on this criterion in the answers that follow. The latter criterion dictates that things that compute should be clearly distinguished from things that don't.

It may be argued that the dichotomy criterion is too strong since computation is a graded concept. According to this line of argument, different devices and mechanisms range at different points on the scale. Some paradigmatic examples like UTMs, digital computers, multiple-processing computers etc. clearly perform computations and are located at one extreme end of the scale. Other examples like digestive systems, walls, toasters etc. don't perform computations, and are located at the opposite end of the scale. Whereas in the middle ground one may find mechanisms such as lookup tables, Ethernet cards, finite state automata etc. that aren't always clear cut cases. However, this criterion has to be methodically followed by any candidate theory of computation. Setting the goal high enough regarding what constitutes *performing computation* and what doesn't may achieve better results. A theory of computation, which clarifies a larger number of computing mechanisms, is ceteris paribus better than one that accounts for fewer. If it were able to achieve that, borderline cases might be left in the grey area. We may still end up with borderline cases of computing mechanisms. But this approach has a better chance of producing a broader version of a theory of computation.

Smith (2002, pp. 50–51) concludes that there's no distinct ontological category of computation, one that will be the *subject matter* of a deep and explanatory theory. The things that Silicon Valley calls computers do not form a coherent intellectually delimited class. In his opinion, computers turn out in the end to be rather like cars. They are objects of personal, social and economical importance, but not in themselves, the focus of an enduring intellectual inquiry. Computers aren't "as philosophers would say, a natural kind" (Smith 2002, p. 51). Computers are indeed like cars in that they are material objects, which occupy space, but the latter don't necessarily require an adequate theory, computers do. It is generally clear cut whether an arbitrary object is deemed a car (or a vehicle) and what is involved in the operation of cars. Computers, on the other hand, are not always as clear cut, even though anybody will acknowledge that a personal desktop computer *is* a computer. It is not always so well understood what constitutes computation, which is the defining essence of computers. Computers are hardly a natural kind, since they are inanimate man-made objects.

But exactly because we don’t understand what computation is, an intellectual inquiry is called for; one that will seek to explain concepts like: algorithm, implementation, interpretation, compilation and so forth. It may also be true that things that *Silicon Valley calls computers* do not form a coherent intellectually delimited class. And if that were the case, then the best thing to do would be dismissing the requirement that something only computes if *Silicon Valley* so claims. Computation is a distinct subject matter, and this is probably why there’s been an ongoing debate about computation and computers during the last century. Computer manufacturers build them although computation is not yet thoroughly understood. An object that has wheels and can take us from point A to point B is usually considered to be a car or some kind of a vehicle.

The challenges that computer scientists face can’t be dismissed, but contrary to Smith, I maintain that computation does constitute a *distinct subject matter*, which calls for deeper research and analysis. He concentrates so much on “doing justice” to the practice (or computation in the wild) and to computationalism, that he neglects essential meta-theoretical constraints. And those ‘neglected’ constraints are the ones, which are necessary to providing an adequate theory of computation. The following answers can be examined as alternatives to Smith’s criteria, and illuminate what he has overlooked.

Piccinini’s Answer

Piccinini (2007) offers an account of computation without representation. He maintains that computation has to be explained in *mechanistic* terms in a way that is analogous to engineering. He proposes a mechanistic account of computation, which doesn’t presuppose semantic content of computational states and processes. Rather, it states that the capacities of a computing mechanism are due to the organization of its sub components and their corresponding functions. This account appeals to mechanistic explanations, and endorses the distinction between successful computations and miscomputations.

The notion of mechanistic explanation applies to computers as computing mechanisms whose function is computing. Furthermore, it matches the language and practices of computer scientists and computability theorists and thus meets the empirical criterion, which is put forward by Piccinini. Likewise, in his opinion this account successfully meets the remaining criteria: the objectivity criterion, the explanation criterion, the right things compute criterion, the miscomputation criterion and the taxonomy criterion. According to this answer an adequate theory of computation has to meet the aforementioned six criteria.

The Objectivity Criterion

This criterion dictates that an adequate theory of computation ought to identify computations as *a matter of fact*. Piccinini asserts that some philosophers (like Searle and Putnam) have suggested that computational descriptions are vacuous, because any system may be described as performing any computation. So allegedly there is no further fact of the matter as to whether one computational description is more accurate than another.

Computer practitioners appeal to empirical facts about the systems they study, design and implement to determine which computations are performed by which mechanisms (or components). They apply computational descriptions to concrete mechanisms in a way entirely analogous to other credible scientific descriptions (e.g. physicists who use empirical descriptions to explain natural phenomena). Moreover, Piccinini argues that many psychologists and neuroscientists are trying to understand which computations are performed by minds and brains. They do so by appealing to empirical evidence about the systems they study (ibid, pp. 502–504).

The Explanation Criterion

This criterion dictates that an adequate theory of computation should explain the behaviour of computing mechanisms. It ought to explain how program execution relates to the general notion of computation. Inner computations may explain outer behaviours of computers. Normally the outer behaviour of ordinary digital computers is explained by appealing to the programs they execute. The literature on computational theories of mind contains explanations, which appeal to the computations performed by the mind. And it also contains assertions that cognitive processes should be explained in terms of program execution (ibid, p. 504). Traditionally, computational explanations have been translated or reduced to explanations by program execution. Piccinni however resists this one-to-one translation. He gives music boxes and automatic looms as examples of mechanisms, which operate by executing programs, but do not perform computations (ibid, p. 517).

The Right Things Compute Criterion

This criterion dictates that a candidate theory of computation need to only encapsulate the mechanisms and devices that actually *compute*. Such a theory should entail that paradigmatic examples like digital computers, Turing machines, and finite state automata, compute (ibid, p. 504). On the other hand, an adequate theory of computation ought to *exclude* non-computing mechanisms and systems. Such a theory should entail that paradigmatic examples like planetary systems, digestive systems, Hinck's pail[3] and Searle's walls don't perform computations. Digital computers, Turing machines, and finite state automata perform computations and constitute the *subject matter* of computer science. To the extent that the assumptions of computer science practitioners ground the success of their science, they ought to be respected (ibid, pp. 504–505).

[3] Ian Hinckfuss presented the problem case (known as 'Hinck's pail') to attack the functionalist theory of mind in a discussion at the Australasian Association of Philosophy Conference, Canberra, 1978. He described a pail of spring water in which at the micro level a vast complexity of things is going on. At the molecular level an even more complex activity is required to sustain the micro level 'things' in the water. Some may argue that this underlying complex activity might realize a human program for a brief period (Copeland 1996, p. 336).

The Miscomputation Criterion

This criterion dictates the requirement that an account of computation addresses the fact that a mechanism can miscompute, i.e. a computation may go wrong. A mechanism *M* is said to be miscomputing in case computing a function *F* on input *I*, where $F(I) = O^1$, but *M* outputs O^2, where $O^1 \neq O^2$. An adequate theory of computation should explain how it's possible for a physical system to miscompute. This requirement plays an important role in computer science and in computation in the wild. Computer science practitioners devote a large portion of their time and efforts to avoid miscomputations and coming up with the appropriate ways to prevent them (ibid, p. 505).

The Taxonomy Criterion

This criterion dictates the requirement that any adequate theory of computation distinguishes between capacities of different classes of computing mechanisms. For instance, logic gates, which are a very low level component in computers, can perform only trivial operations on pairs of bits. More sophisticated calculators, which are non-programmable, can compute a finite number of functions for inputs of bounded size. And ordinary digital computers can in-principle compute any function on any input until they run out of memory. If we choose an account like that of Cummins, who claims that computing amounts to program execution, then we can hardly distinguish computing capacities of UTMs and digital computers from non-universal Turing machines and finite state automata. And finite state automata aren't characterized by computer scientists as executing programs. The difference between computing mechanisms, which execute programs, and those that don't is important to computer practice and according to Piccinini it should also make a difference to theories of mind (ibid, pp. 505–506).

The Empirical Criterion

This criterion dictates the need to account for computational practice and existing computational systems and applications. This criterion is distinct from the other criteria proposed by Piccinini. The former criteria are explicit in his paper (ibid, pp. 501–506) whereas this criterion is only implicit. Piccinini emphasizes the importance that computational practice plays in an adequate account of computation, is a way similar to Smith's empirical criterion. However, Smith asserts that this criterion questions the legitimacy of all the theoretical perspectives and nominates Silicon Valley to decide whether in practice something can be deemed computational. In short, he presents this criterion to undermine the likelihood of producing an adequate theory of computation. Piccinini is only implicitly committed to a narrower conception of doing justice to the body of practices. He claims that the existing computational practice, computing applications, computing systems etc. need to be properly taken into account.

An Adequate Alternative

Though some of Piccinini's criteria require some fine-tuning, I believe that they serve as an adequate alternative to those defended by Smith. The former presents decisive criteria that serve to discriminate between a satisfactory theory of computation and an unsatisfactory one. This is clearly not to say that I necessarily accept the mechanistic account of computation that he proposes in his paper (Piccinini 2007). The criteria above deal with essential characteristics of computation and address the need to account for miscomputation as well. The objectivity criterion dictates that whether a mechanism, device, or any other objects perform computation is a *matter of fact*. Performing computation is an empirical fact similar to the functional role of the heart as a blood pump or the photosynthesis process in plants to produce glucose. Scientists from various disciplines resort to empirical studies to explain different phenomena, and so do computer scientists when dealing with computations. An adequate theory of computation thus ought to provide a suitable framework according to which attributing computation to any system isn't a trivial matter.

It is likewise crucial for such a candidate theory to show why certain systems perform computations whereas others simply don't perform any. Some philosophers assert that almost any system, which is complicated enough, realizes a function etc., can be deemed performing computations (e.g. Searle's wall implementing the WordStar program, Scheutz's systems that realize a function, etc.). In my opinion, views like these trivialize the notion of computation and consequently theories of computation become trivial. If everything can be deemed a computer, then computational explanations become pointless and lose any philosophical interest.

The miscomputation criterion is yet another important feature of any adequate theory of computation. Computational systems are susceptible to miscomputations that result in an abnormal behaviour, which generally speaking may end in one of two ways. The system can 'handle' the miscomputation and resume its normal functioning (perhaps losing some output in the process), in a best case scenario. Or it can malfunction and stop functioning completely, in a worst case scenario. These miscomputations are known in computer science as *bugs* or *faults* and are likely to be present in both hardware and software systems. Though many might expect software (as well as hardware) to be *bug free* it is hardly ever the case.[4]

To briefly explain, software's life cycle includes, among others, phases of design, cutting code and testing. Theoretically, if the testing phase is long and rigorous enough, the software should be bug free. But in practice even with sufficient testing there are still many bugs lurking in the corner. There are two fundamental types of bugs: logical bugs and assumption oriented bugs. The former type of bugs can usually be attributed to human error, introduced during the software-coding phase. The assumption-oriented bugs are those 'pieces of code' that developers implement as part of the design assumptions. Some assumptions are made, for instance, in an

[4] This is not to say that every program inevitably contains bugs, but it rather refers to the more complex programs, which can be found in commercial use, for instance. Clearly, a trivial program comprised of a single line of code, which prints '*Hello World*', will be most likely *bug free*.

attempt to deal with extreme conditions encountered during program runtime (e.g. system running out of memory). Miscomputations are almost an inherent part of any computation process and this is why practitioners of computer science spend so much time attempting to handle as many potential miscomputations as practical.

Modern programming languages such as *C++* and *Java* contain built-in mechanisms to handle such miscomputations, whereas older languages like *C* and *Pascal* don't. The classic *C* approach to this lack of built-in mechanism is using return codes. Each function returns a value indicating success or failure and accordingly every function must check the return code of every function call it makes. *C++* and *Java* in contrast have a built-in mechanism, which is called *exception handling*, for handling these errors. The basic function of exception handling is to transfer control to an exception-handler when an error occurs. If a failure code is encountered then the program invokes its error handling code and resumes its normal function as long as it's possible. Miscomputations are likely to manifest themselves in practical applications and computational systems and therefore mustn't be ignored by any candidate theory of computation.

It may be argued that miscomputations are analogous to misrepresentations. But such a claim presupposes that computation resembles representations to start with. Dretske (1998, pp. 65–70) claims that to misrepresent is to say or mean that *P* when *P* is not the case. It is the power to represent something as being so, when it is not so. The capacity to correctly represent how things stand in the world is of paramount importance, but only insofar as the representation in question is the sort of thing that can get things wrong. Telling the truth is a virtue, only if one is capable of lying in the first place. Only if a system has the capacity to misrepresent, does it have the power to get things right, something approximating *meaning*. But according to Dretske's account of misrepresentation, computers only derive *the capacity to represent or misrepresent* from humans, who already have the full range of intentionality. The computers capacities to represent are merely reflections of our minds.

Piccinini (2007, p. 505) argues that though miscomputation is analogous to misrepresentation according to Dretske, it's not the same. A computer may compute correctly or incorrectly regardless of whether it *represents or misrepresents* anything. A painting, on the other hand, may represent correctly or incorrectly regardless of whether it *computes or miscomputes* anything. A computer may fail to perform its functions in a variety of ways (e.g. running out of memory, a hardware malfunction, etc.). Miscomputations may occur regardless of whether these functions represent anything external to the computer or not. Some miscomputations may result in a complete halt of the computation process, in which case the end result isn't *incorrect* but simply *non-existent*. And clearly, *nothing* can neither represent nor misrepresent.

In sum, Piccinini arrives at an impressive set of criteria, which paves the path for theorists of computation. However, I believe that some aspects of his answer need to be slightly refined. For instance, in his opinion, finite state automata are paradigmatic computing mechanisms (ibid). But finite state automata are merely abstract descriptions of programs. A finite state automaton captures the basic elements of an abstract machine: it reads in a string, and depending on the input and the way the machine was designed, it outputs *true* or *false*. Finite state automata are highly useful practical abstractions, because they retain sufficient flexibility to perform computational tasks. Yet, the hardware requirements for *building* them are abstracted. They are analogous to the blueprints of an architect, which are plans for a home or other structure in such detail as to enable workmen to construct it from the print. They capture all the relevant details, but they are stripped of any physical realization.

Von Neumann's Answer

Back in the late 40s Von Neumann (1948) claimed that we were very far from possessing a proper logical—mathematical theory of automata. He was correct then and to some extent his claim is still resonating today. Von Neumann argued that formal logic deals with rigid, all-or-none concepts and has very little contact with the continuous concepts of the real and complex numbers. His motivation for announcing the need for such a theory was the unlikelihood of constructing automata of a much higher complexity than the ones, which existed then, without it. The high reliabilities and error checking are crucial when dealing with high-speed computing mechanisms. An exhaustive study, which takes them into account, and a non-trivial theory of computation are certainly called for. According to this answer an adequate theory of computation has to meet the precision and reliability criterion, the single error criterion and the analogue—digital distinction criterion.

The Precision and Reliability Criterion

The result of complex computation performed by computing mechanisms may depend on a sequence of a billion steps and has the characteristic that every step actually matters or, at least, may matter with a considerable probability. This is the most specific and most difficult characteristic of computing mechanisms (ibid, pp. 291–292). In dealing with modern logic the important thing is whether a result can be achieved in a finite number of elementary steps or not. The *number* of steps, which are required, is hardly ever a concern. In formal logic any finite sequence of correct steps is, as a matter of principle, as good as any other. On the other hand, when it comes to computing mechanisms the thing, which matters, is not only whether it can reach a certain result in a finite number of steps at all, but also how many such steps are needed (i.e. what computer science refers to as efficiency).

There are a couple reasons for that. The first reason is that computing mechanisms are constructed in order to reach certain results in certain orders of magnitude pre-assigned durations. The second reason is that componentry employed in computing mechanisms has in every individual operation a non-zero probability of malfunctioning (ibid, pp. 303–304). Any step is as important as the whole result and any error can damage the entire result. Computing mechanisms not only have to perform a billion or more steps in a short time, but in a considerable part of their procedure they are permitted not even a single error (ibid, p. 292).

The Single Error Criterion

Von Neumann compares the error handling of computing mechanisms to that of living organisms. He asserts that the organism itself, without any significant external intervention, corrects any malfunction, which occurs in it. The system must, therefore, contain the necessary arrangements to diagnose errors, as they occur in order to minimize their effects, and to correct or block the component at fault. Error handling in computing mechanisms on the other hand is treated entirely different. In actual practice every effort is made to detect any error as soon as it occurs. An attempt is then made to isolate the erroneous component as fast as possible. The basic principle of nature in dealing with errors is to make their effect as harmless as possible and to apply correctives, if required. However, when computing mechanisms are concerned an immediate diagnosis is required so an attempt is made to ensure that errors become as conspicuous as possible. This way intervention and correction can be applied immediately after diagnosis.

A computing mechanism could be designed so that it's able to operate almost normally in spite of a limited number of errors. However, as soon as the mechanism has begun to malfunction it will most likely go from bad to worse and only rarely restore itself. Therefore, it is essential that an intervention be made immediately after an error occurs. The error-diagnosing techniques that are employed in practice are based on the assumption that the computing mechanism contains only one faulty component. Since this is the case, iterative subdivisions of the mechanism into its sub components allows us to determine which portion contains the single fault. As soon as this assumption is invalidated and a possibility exists that the mechanisms may contain several faults, these powerful—dichotomic methods of diagnosis are lost. A better built in error handling mechanism is then called for (ibid, pp. 305–306).

The Analogue—Digital Distinction Criterion

All computing mechanisms fall into two main classes in a way, which is immediately obvious. This classification is into analogy and digital machines.[5] An analogue computing mechanism is based on the principle that numbers are represented by continuous physical quantities. Such quantities might be, for instance, the intensity of an electrical current or the size of an electrical potential. An entire aggregate of currents and electrical potentials serves as a 'black box' into which two currents are fed to produce a current whose numerical magnitude is equal to their product. These computers were fast, operated simultaneously, and had inherently limited accuracy due to the noise level. The guiding principle concerning this type of computers is the "signal to noise ratio". Hence, the question, which has to be asked, is how large are the uncontrollable fluctuations of the mechanism, which constitute the noise, compared to the significant signals, which express the numbers on which the machine operates? The critical problem of any analogue

[5] Von Neumann refers to analogue computers as analogy machines or analogy automata. I choose to use the more common term analogue to avoid the debate about the analogue–analogy comparison.

computation is how low it can keep the relative size of the uncontrollable noise level (ibid, pp. 292–293).

A digital computing mechanism is based on the method of representing numbers as aggregates of digits. These computers represent quantities by discrete states, operate serially, and have inherently unlimited accuracy. The basic operations of a digital mechanism are usually the four species of arithmetic: addition, subtraction, multiplication, and division. Prima facie one might mistakenly think that a digital computing mechanism possesses absolute precision. Even if the operation of each component produces only fluctuations within its pre-assigned tolerance limits, errors eventually creep in.

If a digital mechanism is built to handle ten digits numbers only, it will have to disregard the last ten digits of a twenty digits number, which is the product of multiplying two 10-digit numbers. The necessity of rounding off the product introduces in a digital computing mechanism qualitatively the uncontrollable noise of its analogue counterpart. Only the error in the former isn't a random variable like the noise in the latter. The important difference between digital and analogue computing mechanisms lies in the ability to reduce the fluctuations. The computational noise level can be reduced in digital computing mechanisms in an increasingly easy manner comparing to analogue mechanisms (ibid, pp. 294–296).

"Everything" Computes Answer

Views like those of Putnam and Searle imply a very loose notion of computation so that almost everything can be deemed to be computing. Searle's notorious wall and Putnam's realization theorem of finite automata suggest that even a *rock* can be claimed to be computational (Chalmers 1996). The fourth answer advocated by Scheutz, Putnam and others states that a satisfactory account of computation is underpinned by an adequate theory of implementation. However, existing accounts of computation are inadequate due to lack of a satisfactory theory of implementation. Scheutz (1999) asserts that the notion of implementation is construed as realization of functions, rather then the standard concept of physical state to computational state correspondence. According to this answer an adequate theory of computation has to meet the implementation theory criterion.

The Implementation Theory Criterion

According to Putnam's realization theorem (1992, pp. 121–125) every ordinary open system is a realization of every abstract finite automaton. For the purposes of the current discussion my assumption is that by "every ordinary open system" he essentially means every physical object.[6] This assumption is substantiated by the fact that Putnam (1992, p. XV) asserts that there is a sense in which every physical

[6] It can also be interpreted as a sufficiently complex physical object similar to Searle's (1990) thesis that any physical system can be seen to implement any computation, so that even the wall behind him might be seen as implementing the Wordstar program.

system implements every computation. Furthermore he argues that the computer analogy doesn’t answer the question what the nature of mental states is (ibid, p. XI). Based on this view every physical object can be viewed as implementing every program, even a rock (Chalmers 1996). As Scheutz (1999, p. 162) claims, Putnam’s theorem shows that every account of computation, which lacks an adequate theory of implementation, is built on weak grounds. In consequence, a satisfactory theory of implementation is required to answer essential questions like: ‘What computation does a given physical system implement?’

Scheutz (ibid, pp. 162–163) suggests tackling computation from a practical point of view, i.e. by looking at existing applications and systems that are designed, implemented and used by people. Rather then asking how abstract computations relate to physical systems, it should be the other way around. This approach will result in a more restricted notion of function realization based on the physical constraints (e.g. measurability, feasibility, error range, etc.). One of the defining characteristics of the standard notion of computation is that it is independent from the physical system that realizes it. This means that the same computation can be executed on a range of different physical systems. The motivation for appealing to behavioural descriptions of concrete systems instead of the abstract levels is that this approach ensures that the close ties to the concrete world are maintained. Computational systems are physically situated in the world and by appealing to the abstract levels we tend to lose sight of this fact.

Scheutz (1999) claims that Putnam’s theorem and Searle’s wall have a tremendous impact on the foundations of computer science. But these foundational difficulties seem to be limited to the theoretical level as the computational practice doesn’t seem to be holding back. To avoid such difficulties a different approach to an implementation theory is called for. Such a theory needn’t depend on state-to-state correspondence, but rather one that exploits descriptions of certain properties of concrete systems and abstract computations. It should appeal to the link between the concrete and the abstract, while emphasizing the practical constraints.

The infamous duo *computation—implementation* has to make way for the “realization of a function” notion, i.e. what it means for a physical system *S*, which is described by a theory *P*, to realize a function *F*. It is inevitable that more and more constraints like input/output, abstract time/real time, range of errors, etc. are factored into the equation. The time constraint, for instance, seems to be overlooked by standard notion of computation in favour of computational steps. However, every input to a physical system occurs in real time, and hence the realized function has to have a time parameter attached to it.

In Scheutz’s view (1999, p. 190) such an approach doesn’t require a notion of physical states, but it directly determines the function, which is realized by a physical system. It doesn’t even need to rely on a particular computational formalism like Turing machines or finite state automata, but can be related to any of them via the functions that these abstract computations give rise to. This way, Putnam’s theorem and Searle’s wall no longer pose a threat because state-to-state correspondence mappings can be avoided. Furthermore, taking this approach is a result of practical considerations of functions, which are realized by systems that we can *recognize and use* as computers. Scheutz’s theory of implementation also implicitly presupposes the *empirical criterion*. He maintains that computation should be defined in terms of an abstraction over the physical properties determining the functionality of a physical mechanism.

The Empirical and Cognitive Criteria Revisited

Scheutz's version of the empirical criterion is weaker than the one outlined by Smith. The former maintains that the approach he takes in his theory of implementation reflects the computational practice. Computer practitioners define computation in terms of the functions realized by concrete systems, rather then appealing to abstract computations independently of real life computer systems. Smith asserts that the empirical criterion precludes any satisfactory theory of computation. In contrast, Scheutz uses the empirical criterion in a constructive manner when theorizing about implementation, which consequently affects subsequent theories of computation too. The computational practice guides Scheutz to give up the standard notion of state-to-state correspondence, and come up with a substitute notion of implementation, which is based on functions realization.

Scheutz also acknowledges that the notions of computation and implementation have implications for the theory of mind. He ties the cognitive descriptions of the brain directly to the *level of description of computation* (ibid, pp. 191–192). The class of functions, which are realized by computers, is effectively what it means for something to compute. If brain activities can be adequately explained at this level of description, then computationalism may be true. However, if they can only be explained in levels that are lower than those applicable in computation, then we'll have to either give up *computationalism* or the *current notion of computation*. Similar to Smith's claim above, if computationalism is true, then the *current notion of computation* will have to change to be described as the class of functions, which are realized in a lower level. Otherwise, computationalism is false, because the brain can't be described at the same mechanical level of computers.

Everything Computes?

A very loose concept of computation emerges by accepting Putnam's *realization theorem* approach, Searle's wall (which implements the Wordstar program) or Scheutz's functions realization notion of implementation. The resulting notion of computation is philosophically uninteresting. If in a way everything can be deemed to perform computations, then theories of computation become trivial.[7] Putnam (1992, pp. 121–125) proves a theorem stating that every ordinary open system is a realization of every abstract finite automaton. Thus, in a sense every physical system implements every computation, since the system's physical states can be mapped onto the computational states of the corresponding automaton.

[7] Computationalism also becomes trivial, since if everything computes, then it's trivially true that minds compute as well. This is exactly what Searle and Putnam have tried to show by arguing that (almost) everything computes.

Searle (1990) brings the wall behind him as an example of an object that implements the Wordstar program, and similarly the same may be claimed for *every object*. He also maintains that brains are digital computers because everything is a digital computer (ibid, p. 5). Even Scheutz, who offers a new theory of implementation, admits that in a way, every system, which realizes a function, could be seen as a computer, namely a computer computing that very function (Scheutz 1999, p. 191). However, he claims that most of those computers are not useful for us, because we can't influence their inputs and outputs, for instance. Hence, they do not qualify as computers in a practical sense.

Discussion

Smith claims that computation is intrinsically intentional, which also prompts him to formulate the *cognitive criterion*. He maintains that the misconception of computation as being entirely abstract and formal hides the semantic character of computation. Thus, any adequate theory of computation has to rely on a solid *theory of semantics and intentionality*. Furthermore, he claims that there are many *ontological* questions at the foundations of computation that must be answered beforehand. Computation doesn't constitute a *distinct ontological* category. What qualifies as computers according to Silicon Valley doesn't form a coherent intellectually delimited class. By showing that extant theories of computation fail to meet the above criteria, Smith maintains that any candidate theories are condemned to failure. His demand for a conceptual meta-theoretical constraint isn't unjustified. Theories that explain a phenomenon need to address the underlying concepts that are at the core of this phenomenon. A theory of computation thus needn't presuppose that notions like algorithm, implementation, process etc. are self-explanatory, but rather it needs to address them.

Arguably, as Smith claims computers don't form a coherent intellectually delimited class. However, our goal needn't be providing a theory of computers, but rather a theory of computation. The most sophisticated commercially available computer today will be rendered obsolete in some years to come. Similarly, computers, which were deemed powerful at some point in the past, are no longer suitable to run today's programs. New architectures, stronger CPUs, high volume memory chips, etc. make computers much more powerful and robust. The thing, which constantly changes so rapidly, as Smith identifies is technology. For one to be able to determine whether an arbitrary object is indeed a computer, one has to do examine it from the prevailing technological perspective.

Though commercially available computer platforms may be ordinary digital computers at present, in the near future they may just as well be molecular computers, DNA computers, Quantum computers and so on. Computer theorists should channel their efforts to provide an adequate theory of *computation* and let technology dictate what *computers are*. As Agassi (1985) asserts that one may claim that since theory should guide practice, the engineer should by right be the boss of the technician. But, theory and practice always mix and so every technician is a bit of an engineer, and vice versa. Moreover, techniques offer less room for rational debates of abstract matters. Techniques are easily tested by implementation, so it is easy to falsify claims about them. Therefore, there seems to be no need for a unified theory of technology for successful application of various techniques, and *technology* has no need for abstraction. Similarly, there's no need for a unified theory of computers as technological artefacts, but for a theory of computation.

Smith pushes it further and proposes two criteria that are extremely hard to meet. Though the empirical criterion isn't unexpected, he takes it to the extreme. Not only does a theory of computation have to do justice to real life computation by explaining programs like the Open Office Writer, but it should also give rise to reconstructing computational practice. The analysis has to remain grounded in real-world examples, in computation-in-the-wild. The scope of a satisfactory theory should be sufficiently broad to account for physical mechanisms, implementations, architectures, programs, processes, algorithms, languages, networks, interactions, behaviours, interfaces etc. Moreover, it also has to account of the design, maintenance and even the way we use such systems. Such a theory is no longer confined to scientific and philosophical domains, but rather extends over to other domains like economics, social sciences and even ethics. Smith claims that the computer revolution adapts, expands, and in general outstrips our theoretical grasp. This is exactly what his empirical criterion accomplishes: destroying all chances of producing an adequate theory of computation.

This is not to imply though that there is no justification for adopting a *subtler* version of the empirical criterion. Piccinini's implicit empirical criterion implies that existing practices, computing applications and other real life examples need to be properly *taken into account*. Assumptions and empirical facts of computer science practitioners, which ground the success of their science, have to be respected by an adequate theory of computation. Theorists of computation may not have the 'luxury' of being completely detached from practice, but they certainly needn't account for any particular use of any particular program. Scheutz is another good example of how the empirical criterion can be put into good use. The implementation theory criterion reflects computational practice by revisiting the standard correlation between computation and implementation. Rather then appealing to a top down approach that begins in the abstract level and progresses down to the concrete, Scheutz's starting point is real life examples of computers and applications. In his view, an implementation theory should appeal to the link between the concrete and the abstract, while emphasizing the *practical constraints*. This eventually leads him to abandon the common state-to-state correspondence in favour of the function realization analysis. Even Von Neumann's classic view, which is reflected in the third answer, 'does justice' to practice. By emphasizing the single error criterion, he points out that in practice every effort is made to detect errors as soon as they occur. Error handling can be easily neglected, if a theory of computation is completely detached from practice. But this serves to show that considerations of practice can be taken into account in a constructive manner, without renouncing future candidate theories.

Smith's cognitive criterion is also too difficult to meet. Whilst proponents of the computational theory of mind should obviously heed the theory of computation, Smith believes it should instead be the other way around. Theorists of computation need to consider the potential consequences of candidate theories of computation for computationalism. An adequate theory of computation should apparently be an intelligible foundation for the formulation of the computational theory of mind. Moreover, when considering this criterion in conjunction with his claim that we will never have an adequate theory of computation, the result is surprising. If we take the main claim of computationalism to be that minds are *computational* systems, and given that there will never be an adequate *theory of computation*, it follows that there will never be an adequate computational theory of mind. In my opinion, this defeats the purpose of introducing the cognitive criterion in the first place.

Acknowledging the paramount role of theories of computation in computationalism is by no means unique to Smith. Each one of the competing answers presented in this paper addresses the relevance of computation to cognition or to computationalism in particular. Piccinini argues that the difference between computing mechanisms that execute programs and those that don't is important not only to computer science, but also to theories of mind. It is also crucial to distinguish different kinds of computational descriptions, because some might be relevant to explaining the behaviour of computers or minds by appealing to their computations.

Piccinini asserts that his mechanistic account allows systems to be described as rule following. Thus, it is suited to formulate explanatory theories of rule-following systems like the computational theory of mind. By individuating computing mechanisms and the functions they compute, determining whether minds are computing mechanisms becomes a matter of whether they have the relevant functional properties. Even the earlier view of Von Neumann emphasizes the similarities of computing mechanisms to the human central nervous system. He compares nerve impulses to binary digits due to the binary digit's nature of being an all-or-none affair. The influence of muscular contractions, which are induced by nerve impulses, on the blood stream is analogue-like. In his opinion, living organisms are very complex-part digital and part analogue mechanisms.

Scheutz (1999, p. 192) argues that if brains are best described at a lower than mechanical level of description, which is crucial to a theory of mind, then either minds are not computational (if computational is taken to mean 'mechanical') or a different notion of computation is called for. The lack of a reasonable notion of implementation renders not only computer science meaningless but also computationalism. Moreover, if natural cognitive systems do essentially exploit 'non discrete magnitudes', 'quantum effects', etc., then they are *essentially non-digital systems*. And in that case they cannot be described solely in terms of functions realized by digital systems.

Conclusions

I believe that the competing answers, which were outlined above, show that there are acceptable alternatives to Smith's view. This is sufficient to re-establish the need for an adequate theory of computation. Obviously, other answers ought to be considered and I don't necessarily argue that any of the above answers is the one we should adopt. Each of the answers is susceptible to some legitimate criticism. The mechanistic account of Piccinini, in its current form, excludes analogue computation, which is clearly a valid type of computation. Von Neumann argued that all computers fall into one of two main classes: analogue and digital. This classification excludes the case of hybrid computers, for example (and evidently more recent examples like molecular computers). The fourth answer discusses the need for a solid theory of implementation. But it allows for a loose notion of computation in terms of which given enough complexity almost any object may be deemed computational.

The four answers discussed above are of varying levels of severity. Two of the three criteria, which are included in the first answer advocated by Smith, are extremely hard to meet. Thus according to that answer an adequate theory of computation is implausible. The criteria, which are included in Piccinini's answer, are easier to meet by an adequate theory of computation (he even proposes such a theory). The third answer advocated by Von Neumann suggests criteria, which are even easier to meet by a broader range of theories of computation. The implementation theory criterion, which is supported by the last answer, is highly achievable, and yields a loose notion of computation. A key criterion, which an adequate theory of computation has to meet according to most of the aforementioned answers, is the empirical criterion. Whilst Smith explicitly proposes this criterion and takes it as a reason for rejecting future theories of computation, it is only implicit in the other answers. Piccinini, Von Neumann and Scheutz attribute significant importance to computational practice, but don't use the empirical criterion against any potential candidate theories of computation.

Smith concludes with a negative claim, I will opt for a positive one. Smith claims that we will never have a theory of computation, and he raises some valid points that require further discussion. However, I maintain that there is no compelling reason to renounce every attempt to provide an adequate theory of computation. The answers above clearly show that there is yet work to be done. Former theories of computation, which were refuted, simply pave the path for a more adequate theory of computation.

Acknowledgments An earlier version of this paper was presented at the 2007 AAP conference in Armidale, Australia. I'm greatly indebted to Phillip Staines, whose trenchant comments and insights made this paper appreciably better. I'm grateful to Joseph Agassi, who thought me what philosophy is all about and that conflict is not to be feared. I also thank Gualtiero Piccinini for his comments on the latest version of this paper. Many thanks to Calanit Sabaz for her constant support and feedback.

References

Agassi, J. (1985). *Technology: Philosophical and social aspects*. Dordrecht: Reidel.
Chalmers, D. J. (1996). Does a rock implement every finite-state automaton? *Synthese, 108*, 309–333.
Copeland, J. B. (1996). What is computation? *Synthese, 108*, 335–359.
Dretske, F. I. (1988). *Explaining behaviour*. Cambridge: The MIT Press.
Fodor, J. A. (1975). *The language of thought*. Cambridge: Harvard University Press.
Piccinini, G. (2007). Computing mechanisms. *Philosophy of Science, 74*, 501–526.
Popper, K. R. (2002a). *The poverty of historicism*. London: Routledge Publishing.

Popper, K. R. (2002b). *The logic of scientific discovery*. London: Routledge Publishing.

Putnam, H. (1992). *Representation and reality*. The MIT Press.

Pylyshyn, Z. W. (1989). Computing in cognitive science. In M. I. Posner (Ed.), *Foundations of cognitive science* (pp. 51–91). Cambridge: The MIT Press.

Scheutz, M. (1999). When physical systems realize functions. *Minds and Machines, 9*, 161–196.

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

Smith, B. C. (1996). *On the origin of objects*. Cambridge: The MIT Press.

Smith, B. C. (2002). The foundations of computing. In M. Scheutz (Ed.), *Computationalism: New directions* (pp. 23–58). Cambridge: The MIT Press.

Von Neumann, J. (1948). The general and logical theory of automata. In A. H. Taub (Ed.), (1963), *Collected works* (Vol. V, pp. 288–328). London: Pergamon Press.

Wittgenstein, L. J. (2001). *Philosophical investigations: 50th Anniversary commemorative edition*. Blackwell Publishing.