What is Computation: An Epistemic Approach*

Jiří Wiedermann[1] and Jan van Leeuwen[2]

[1] Institute of Computer Science of AS CR, Prague, Czech Republic
jiri.wiedermann@cs.cas.cz
[2] Dept. of Information and Computing Sciences, Utrecht University, The Netherlands
J.vanLeeuwen1@uu.nl

"How can one possibly analyze computation in general? The task seems duanting, if not impossible." Y. Gurevich [14]

Abstract. Traditionally, computations are seen as processes that transform information. Definitions of computation subsequently concentrate on a description of the mechanisms that lead to such processes. The bottleneck of this approach is twofold. First, it leads to a definition of computation that is too broad and that precludes a separation of entities that, according to prevailing opinions, do perform computation from those which don't. Secondly, it also leads to a 'machine-dependent' notion of computation, complicating the identification of computational processes. We present an alternative view of computation, viz. that of a knowledge generating process. From this viewpoint, computations create knowledge within the framework of 'more or less' formalized epistemic theories. This new perception of computation allows to concentrate upon the meaning of computations – what they do for their designers or users. It also enables one to see the existing development of computers and information technologies in a completely new perspective. It permits the extrapolation of the future of computing towards knowledge generation and accumulation, and the creative exploitation thereof in all areas of life and science. The flux of our ideas on computation bring challenging new problems to the respective research, with wide connotations in the field of artificial intelligence, in cognitive sciences, and in philosophy, epistemology and methodology of science.

1 Introduction

Why do we compute? What do we compute? These two seemingly innocent questions were recently posed by Samson Abramsky in his contribution to the book commemorating the hundredth anniversary of Alan Turing's birth [1] in 2012. These questions can be made more concrete, e.g., why are we using computers? What are we computing with them? What is the meaning of computations performed with the help of computers? Here, and also in the sequel, we do not just have numerical computations ('computations with numbers') in mind, but

* This work was partially supported by RVO 67985807 and the GA ČR grant No. P202/10/1333.

G.F. Italiano et al. (Eds.): SOFSEM 2015, LNCS 8939, pp. 1–13, 2015.
© Springer-Verlag Berlin Heidelberg 2015

any computations performed by whatever kind of computer. Of course, there are numerous replies possible and each of us will have an answer why he/she is making use of computations. However, what we are after is not a subjective answer pertinent to some specific use of commonly used computers. We want to have an answer that is grounded in some systematic theory, as part of a deeper understanding of the notion of computation, applicable to whatever use of whatever sort of computers. So far we do not seem to have a satisfactory answer obeying the latter conditions. This is related to fact that from the viewpoint of computer science, or computability theory for that matter, we in fact do not know quite well what computation 'is', in general. This is very unsatisfactory, since computation is the central notion in many scientific disciplines. The reasons for this unsatisfying state of affairs have accumulated during the past few decades.

Namely, computation is no longer what it used to be a one or two decades ago. Up until the end of the nineteen eighties no expert was bothered by the question what computation was. The answer was clear – computation is what is described by the generally accepted mathematical model of computation: the Turing machines, or any computational model equivalent to it [11,22].

With the advent of new computing technologies, networking, and advances in physics and biology, computation became understood as a far broader, far more common, and far more complex phenomenon than was modeled by means of Turing machines. In fact, it has become harder and harder to see these newer notions of computation through the lens of Turing machines (cf. [28]). Examples include biologically inspired models, physically inspired models and, last but not least, 'technologically enabled' models such as the Internet. One has to consider non-numerical computational models and devices, but also computations done on paper or by heart, proofs, computation with real numbers, continuous computations, geometrical constructions using compass and ruler, etc. The question is then, what is computation? What device performing computations is the 'right' one? How can computation be defined in such a way that every computational device realizes it in its specific way? Is there anything that all these computations have in common?

The scientific community, especially in informatics, physics and philosophy, has, of course, reacted to these new trends. (Un)surprisingly, instead of agreeing on a joint view of computation, the community has split into several opinion groups. For example, Frailey [12] maintains the radical position that computation is realized by whatever process. Other computer scientists, like Bajcsy [4] and Rosenbloom [16], define computation as a process which transforms information. Other researchers require additional properties, or state that computation is about symbol manipulation (e.g., Fortnow [11], Denning, [9], Conery [8] or philosopher Searle [3]). Still others, - like A.V. Aho [2], or Searle (again) [18] - require that there must be some computational model supporting the computation. Fredkin [13] has put it like this: *The thing about a computational process is that we normally think of it as a bunch of bits that are evolving over time plus an engine – the computer.* Deutsch [10] holds an even tougher view: there must also be a physical realization of a computational model. Finally, Zenil [32]

has put programmability at the center of the discussion and of the definition of computation. Many opinions exist but many fall short in capturing the full notion as intuitively understood nowadays.

The previous efforts in defining computation have several things in common. First, all seem to agree that computation is a process. Secondly, all definitions tend to express computation as *'what the underlying hardware is doing'* or, in other words, *HOW the process of computation is realized.* This does not give much insight because 'what the hardware does' is performing operations on data; this forces us to see meaningless operations with data as computation. However, we are primarily interested in *WHAT the computation does*, i.e. what it does for the designers, users, observers. What a computation does, is only expressed by the design of the implementing system. Knowing how a computation does what it does is less interesting.

The intriguing question remains: what is it a computation does? Our answer [29] is simple: *computation generates knowledge.* It generates knowledge over some domain of interest for which the underlying computational system was designed or developed, or in which the system itself has evolved. Of course, the notion of knowledge itself is as hard to define as is computation. It has been debated ever since the Greek philosophers captured its many forms. For our purposes a general definition as given in Wikipedia will be good enough [31]. It stresses that knowledge is 'a familiarity with something or someone' and 'the theoretical or practical understanding of a subject'. Skills and behaviour are also considered to be knowledge.

Following this definition, knowledge essentially is an *observer-dependent* entity. Thus, the notion of computation as we defined it here is essentially observer dependent as well. This is a clear contrast to viewing computation as information processing, which we rejected above. It is far more fitting to our intuition.

The arguments to support our understanding of computation by means of examples from the history of computing will be given in Section 2. In Section 3 we concentrate on the internal structure of knowledge from the epistemic viewpoint. Section 4 presents a formal definition of computation as knowledge generation. In Section 5 we discuss the potential benefits of the epistemic view of computation. Section 6 contains conclusions.

This paper excerpts the presentation of our ideas in [29] and [30]. The readers interested in more details of the topics outlined here are kindly referred to our original works ([29,30]). In a forthcoming paper we digress on the possibility of a new theory of computation based on our philosophy [25].

2 Computation as Knowledge Generation

In Section 1 we asserted our main thesis: *computation is the process of knowledge generation.* In this section we review some examples of computational systems and how they can be seen as processes utilizing and producing knowledge.

In Table 1 (from [29]) we give an overview of a number of computational systems, together with the respective knowledge domains and types of knowledge

they produce. The items in the first part of the table (*contemporary computing systems*) clearly show an increasing 'growth' in knowledge generation: the further down one gets in the table, the more general and less formal the underlying knowledge domains are and the bigger part of reality are captured. From this point of view, classical computational systems as we know them are only very primitive systems for generating knowledge. Newer systems are vastly more versatile.

The items in the second part of the table capture *natural computing systems*. These systems are not designed by people but exist in the natural world. They obviously belong to the class of systems (processes) producing knowledge according to our definition, i.e. they are computational. Note that we would not

Table 1. Computation as knowledge generation (cf. [29])

Computational system	**Underlying knowledge domain**	**What knowledge is produced**
Contemporary computing systems		
Acceptors	Formal languages	Language membership
Recognizers	Formal languages	Membership function
Translators	Functions, relations	Function value
Scientific computing	Mathematics	Solutions
Theorem provers	Logic	Proofs
Operating systems	Computer's devices and peripheries	Management of computer's own activities
Word processors and graphical editors	Graphical layout, spelling, grammar	Editing skills
Database and information systems	Relations over structured finite domains	Answers to formalized queries
Control systems	Selected domains of human activity	Monitoring, control
Search engines	Relations over unstructured potentially unbounded domains	Answers to queries in natural language
Artificial cognitive systems	Real world, science	Conjectures, explanations
Natural computing systems		
Living systems, cells	Real world	Life, behavior, intelligence
Brain, mind, social networks	Knowable world	Knowledge of the world
The Universe	Science	Living systems
Non-Turing computing systems		
Compass and ruler	Euclidean geometry	Euclidean constructions
BSS machine [6]	Theory of real numbers	Values of real functions
Oracles [22]	A set $A \subseteq \Sigma^*$	Characteristic function of A
Super-Turing computations	Formal languages in Σ_2	Language membership

have been able to include these examples under most of the classical definitions of computation, as the underlying 'computational mechanisms' are not known.

The items in the last part of the table seek the limits of our definition, resulting from attempts to falsify our thesis with the help of somewhat exotic examples which cannot be realized by Turing machines but are, nonetheless, still regarded as computations. It is clear that the examples still fit. They point to the fact that it is generally not a good idea to require that there must be a physical realization of the computational model at hand. This is in a good agreement with the practices in geometry, mathematics and even in (higher order) computability theory where processes similar to the ones we considered are routinely considered as computational processes.

3 The Structure of Knowledge

It goes without saying that once we concentrate on *WHAT* computations do instead of on *HOW* they do what they do, we lose the opportunity to investigate possible finer details of the effectuating processes as they are studied in e.g. computability theory or in computational complexity. On the other hand, our approach opens the way towards the investigation of other, so far mostly neglected aspects of computations: the insight into the character of the knowledge generating mechanisms used by computations.

In order to achieve its goals, a computation requires familiarity with the knowledge domain for which it is designed. In the sequel we will assume that this domain is given in the form of a *theory*. We do not restrict ourselves to theories in the strict formal sense of mathematics only. In this section we outline the broader analysis from [30] on the theories and structures that computations may exploit.

We will view 'theories' as an analytical tool for expressing knowledge. This may include e.g. describing, understanding, explaining and answering queries, providing solutions and predictions in areas of science or life, or the generation or control of behavior. A theory will normally consist of a collection of facts, sentences, statements, patterns of behavior, or linguistic descriptions and principles needed for deriving other statements using formal or informal inference rules. However, a theory could also have a form of a semantic network, or of a set of conditions and restrictions holding for a computation. A theory could also be a map, a scheme, and so on. Knowledge produced within the scope of a theory will have to fit the 'language' of the underlying domain. New knowledge that is generated may be kept in a *knowledge base* and become an integral part of the theory.

In general there is no a priori need for a theory to be correct or truthful. A theory, in the general sense we are using, can even be based on erroneous, unproven or non-verified beliefs and facts. Nevertheless, whatever 'knowledge' is generated within a flawed theory is formally considered to be knowledge, and thus truthful, within that theory.

A possible way of viewing the highly generalized notion of a theory that we use here, is to see it as a model of the world in which a computation is rooted

Table 2. The structure of knowledge (cf. [30])

	Mathematics, logic, & computer science	**Philosophy and natural sciences**	**Mind and humanoid cognitive systems**
Domain of discourse	Abstract entities	Ideas, empirical data	Perception, cognition
Elements of knowledge	Axioms, definitions	Facts, observations	Stimuli, multimodal concepts, episodic memories
Inference rules	Deductive system, programming languages	Rational thoughts, logics	Rules and associations formed by statistical learning
Final form of knowledge	Predicates, theorems, proofs, solutions	Statements, theorems, hypotheses, explanations, predictions, theories	Conceptualization, behavior, communication, natural language (NL), thinking, knowledge of the world formed mostly in a NL and in form of scientific theories

(cf. [26,27]). An important characteristic is that knowledge according to a theory can be generated time and again from the same base facts and principles, e.g. by computations that do so. In evolving domains, the theory corresponding to the underlying domain will have to evolve along with it.

Table 2 illustrates the structure of the theories in various knowledge domains. In the table, from left to right, the domains range from *theory-full domains* with formal theories to *theory-less domains* that admit no formal description for what they capture (cf. [23]). The examples shows the different levels of formalization, completeness and truthfulness of the theories that may underlie respective domains.

In cases where heterogeneous knowledge is used, natural language is an important mediator among theories. Semantics is of crucial importance here. Semantics is a form of knowledge and thus it is to be represented by a theory again. From this viewpoint all computations, including computations generating knowledge based on natural language understanding, bear a homogeneous structure. The knowledge framework behind the latter computations will normally be based on *cooperating theories.*

In general, theories depend not only on the knowledge at hand, but also on the context in which this knowledge is used and even on the history of past uses of this knowledge. In the case of embodied cognitive systems the context does not only refer to the grammatical context, but to the entire perceptual situation. All this leads to a complex intertwining of the respective theories. In general we do not know much about cooperating theories [26]. But here one can see the benefit

of viewing computations as knowledge generating processes again: whatever deep and detailed (classical) view of the mechanisms realizing a computation can in no way contribute to the elucidation of the semantics of the computation.

The just presented view of the structure of computations contains one more principal observation. It illustrates that *computations generate knowledge from the knowledge in which computations are rooted* (see also Section 4). One might say that computation is *knowledge in action.*

4 An Epistemic Definition of Computation

Given the extensional definition given in Section 1, knowledge is an observer-dependent notion. After all, the decision of knowledge being based on a 'familiarity with something or someone' clearly is in the eye of the beholder, especially when it concerns knowledge that is not generally accepted. Therefore, computation as a process generating knowledge must be observer-relative, too ([30]).

As designers/observers of a computation, we must be aware of how a particular computation is related to the specific knowledge it generates. In response to its input a computation is not allowed to generate completely arbitrary knowledge. Each computation is required to generate knowledge over the domain for which the underlying system was designed or into which it has evolved. Similar to how intelligent behavior of an embodied robot arises from the interaction between brain, body and world, so is knowledge generated by computation in its interaction with the underlying knowledge domain.

More formally, there must be a way to *verify* the correspondence between a given computation and the domain over which the computation generates its output in the form of knowledge. For this, every computation must exploit some cognisance of the underlying knowledge domain. It is obliged to only use the facts, statements, rules and laws that describe the knowledge domain and that hold in this domain. This is what is meant when we said above that a *computation is rooted in its knowledge domain.* As illustrated in Table 2, the required attributes of computations can take different forms, depending on our knowledge of the underlying knowledge domain and on our ability to formally describe it, including the rules and laws holding in this domain.

We will now argue how the verification obligation might be expressed, i.e. that a given computational process generates knowledge that is expressed by means of the theory of the underlying domain. Of course, there must be an *explanation* (e.g., a proof) that the computational process works as expected. The explanation should express that the process generates knowledge that can be inferred from the underlying theory. The latter is also the key to a more formal definition of computation.

Before given the definition, we need one more notion. By a *'piece of knowledge'* we will denote any constant, term or expression which belongs to the theory or which can be derived using the respective rules and laws of the given theory. In [25] the collection of all items of knowledge possibly pertaining to a given computation, will be termed the *meta space* of the underlying theory. Although

we will make use of the terminology used in mathematical logic below, one has to bear in mind that our notions of 'theory' and 'inference rules' are much broader than in logic and also include informal theories and informal rules of rational thinking.

Definition ([30]): *Let T be a theory, let ω be a piece of knowledge serving as the input to a computation, and let $\kappa \in T$ be a piece of knowledge from T denoting the output of a computation. Let Π be a computational process and let E be an explanation. Then we say that process Π acting on input ω generates the piece of knowledge κ if and only if the following two conditions hold:*

- *$(T, \omega) \vdash \kappa$, i.e., κ is provable within T from ω, and*
- *E is the (causal) explanation that Π generates κ on input ω.*

We say that the 5-tuple $C = (T, \omega, \kappa, \Pi, E)$ is a computation rooted in theory *T which on input ω generates knowledge κ using the computational process Π with explanation E. The device or mechanism realizing process Π is called a* computer.

Under suitable conditions, computations may be *composed* to obtain new computations that fit the definition. Compositionality is an important property for the controlled behaviour of classes of computations (cf. [25]).

In the definition above, ω may be a set of numbers, a query in a formal or natural language, or a statement whose validity we are looking for, and so on. The computational process Π is a parameter of the computation. This captures that the same knowledge may be generated within the same theory by different computational processes. A change of computational process will most likely result in a different explanation. Whatever Π has to know about T must either be encoded in the design of Π and in ω or Π must have access to T. The condition $(T, \omega) \vdash \kappa$ implies that T is closed with respect to the inferences in T. The means that, once κ has been computed, it can be added as an explicit piece of knowledge to T, thus extending the knowledge base of T. In [30] we have argued by means of concrete examples that both conditions in the definition above are necessary.

The proposed definition of computation corresponds very well to the contemporary theory (and hopefully also the practice) of programming. The designer of a program must be aware of theory T, of the required result κ and of the fact that $(T, \omega) \vdash \kappa$. Then there is a computational model for which one has to design a computational process Π generating the required knowledge κ. One has to deliver also the evidence E, since otherwise one cannot be sure that the program does what was assumed.

Notice that the closure of T can be used e.g. to model interactive computations where after each interaction, the knowledge base is updated by the recently computed piece of knowledge. When a computation can modify the underlying theory we speak of an *evolving computation.* In this way one can model potentially infinite, interactive and evolutionary computations (cf. [28]). The formalism also enables one to define *universal computations* for some domain D, i.e. computations where the same computation process Π is used for generating the corresponding pieces of knowledge for all $\omega \in D$. (For more details see also [25].)

Finally, observe the natural way in which our approach accommodates previous efforts to define computation, by giving a common procedural platform for all kinds of computation. Using the previous notation, in the majority of the classical approaches to computation, a computation would look like this: $C = (\Pi)$. No other conditions are required from Π. In our approach we have found a different common denominator of all computations: this is the respective knowledge generation aspect.

Example. In [30] we give several examples of how our definition can be used to explain cases of 'computation' which have proved to be hard using classical definitions. Here we only discuss the example of the computing rock ([7,19]). Searle [19] describes the example as follows:

> *'Consider the example ... of a rock falling off a cliff. The rock satisfies the law* $s = \frac{1}{2}gt^2$*, and that fact is observer independent. But notice, we can treat the rock as a computer if we like. Suppose we want to compute the height of the cliff. We know the rule and we know the gravitational constant. All we need is a stop watch. And we can then use the rock as simple analog computer to compute the height of the cliff.'*

How does this conform to our definition of computation? First of all, for computing the height of a cliff, a rock alone is not enough. In addition to it we need both a stop watch and a person to observe the falling rock, operate the stop watch and know how to compute the distance traveled by the falling rock during the fall. Thus, the 'computer' consists of a rock *and* of a person endowed with the abilities just described. The theory behind the computation and the explanation are quite complex if all details are to be mentioned, from Newtonian physics to the visual observation ability of the observer and his capability to perform arithmetic operations, and so on. But, in principle, all these details can be delivered with sufficient plausibility. We conclude that the whole system as described indeed performs a computation according to our definition. (Actually, considering the complexity of the components of this analog computer, one could hardly call it 'a simple analog computer' as Searle does.)

It is important to realize that the conclusion above was possible only due to our insight into the entire process. An observer who has no understanding of stop watches or of the physical laws obeyed by falling bodies, can never come to such a conclusion. The example also is a clear instance of a computation that is observer-relative. In [30] several other examples are given, including e.g. the analysis of Searle's Chinese Room problem ([17]), which relies on a careful understanding of computation.

5 Discussion

There is no doubt that the classical view of computation, essentially based on the notion of Turing machines, has proved to be a very potent paradigm. It has lead to computability and computational complexity theory as we know them today.

It has learned us what can and what cannot be computed by the underlying models of computation and how efficiently this can or cannot be done. All this is carried out in the observer independent framework. The broadest framework is probably provided by Gurevich' approach using *Abstract State Machines* [5,14].

However, we seem to be reaching the limit of this approach. Namely, when it comes to solving the omnipresent problems e.g. related to artificial intelligence and especially cognition, we do not know at all what the actual potential of our computers is and what undiscovered mechanisms may be available one time to compute. For instance, we have no good clue of how to program computers in order to (learn them to) think, to be conscious, to acquire, understand and use natural languages or to create new knowledge. All these abilities are considered to be observer-relative qualities.

Our approach opens the road towards approaching the latter problems. As seen from our definition of computation, this is because it concentrates on the meaning of computation – what they do from the viewpoint of an observer or designer, and how they achieve their goals. There are many advantages of defining computation as a knowledge generating process:

(i) It gives a better way to distinguish objects which perform computation from those that do not. For example, according to our definition one can assign a computing ability to a rock (cf. [7]) only when it provably generates knowledge. In [30] we have shown that a rock can be seen as an analog computer in a scenario in which it is heated in order to 'compute' its own melting point. Under a different scenario a rock might be able to compute different knowledge – e.g., its momentum, weight, volume, and so on. More examples of our approach to unconventional computations are given in [30].

(ii) It offers a framework for modeling/discussing the question of observer dependency. Namely, an observer can be modeled by the same means as a computation. In this approach an observer is seen as a computation that 'observes' an other computation. In this way, an observer has some information about the observed computation as input (as required by our definition) and his/her task is to decide, whether it is a computation or not. This, of course, depends on the observer's own knowledge. A case analysis of the related computational scenarios reveals a number of new non-trivial insights that until now were not accessible to such a treatment. For instance, one can prove that a so-called *universal observer* whose decisions always agree with those of every other observer, for any computation, does not exist. (For details, see [30].)

(iii) It offers independence of the notion of computation from the underlying mechanisms. The definition covers not only all known instances of computation but also many hitherto unknown instances. It seems that for understanding computation one should investigate 'natural' rather than 'artificial' computations.

(iv) It supports the thinking about computation at a high level of abstraction, which is important for the design of artificial systems and for understanding other natural systems developed by some evolutionary process.

(v) It answers certain problems from cognitive science whose 'intractability' was due, as it seems now, to the use of the classical definition of computation.

For example, there is a widely discussed question in cognitive science what cognition is, if not computation [24]. As long as cognition is seen as an ability to gain, collect, produce and exploit knowledge, then it corresponds to our definition of computation in its most crystal form. Under such a view the original problem dissolves.

(vi) It puts the semantics of computations in the foreground which, classically, has so far been viewed as something secondary by which a computation can somehow be endowed *a posteriori* in the programming process. In fact, we have identified a new intermediate stage between computations (in the classical sense) and intelligence, viz. the ability to produce knowledge. Intuitively, the ability to produce knowledge is a prerequisite of intelligence. What ingredients make intelligence stronger than computation, in our sense? For a further discussion of this question, see our analysis of Searle's Chinese Room problem in [30].

(vii) It indicates that the formal framework that works for theory-full domains can be extended to theory-less domains. In doing so, natural language plays the central role: it enables dealing with knowledge domains and epistemic theories for which no formalization is available. Moreover, natural languages offer semantic means for bridging the gaps between seemingly unrelated knowledge domains, enabling one to draw analogies between such domains to be used as knowledge creation mechanisms (cf. [27]).

(viii) It offers a novel way of analysing prevailing trends in the history of information technologies when viewing them via their ability and potential to generate knowledge. It shows a steady shift towards interactivity, communication in natural languages, and to knowledge production.

(ix) Last but not least, our definition has great philosophical and methodological merit since it concentrates on the sense of computation, viz. knowledge generation, promoting computation to the key notion which is behind all progress.

6 Conclusion

In [15], David Deutsch is quoted as saying:

> *. . . the creation of knowledge [. . .] now has to be understood as one of the fundamental processes in nature; that is, [. . .] fundamental in the sense that one needs to understand them in order to understand the universe in a fundamental way.*

In this paper we have argued that computation is the fundamental process underlying this, which explains why computation is the far-reaching process as claimed in computer science already for many years.

We believe that the time that computation was seen as an intrinsically physical process only has passed and that it is necessary to consider computation as an observer-relative process as well. This is because we are increasingly facing problems where, due to their nature, such a framework is required. This

is especially the case of computations related to AGI (*artificial general intelligence*) which are all firmly rooted in theory-less, observer-dependent domains. We expect that changing the philosophy of computation towards that of viewing them as knowledge generating processes will help in the understanding and the creation of new intelligent information technologies.

References

1. Abramsky, S.: Two puzzles about computation. In: Barry Cooper, S., van Leeuwen, J. (eds.) Alan Turing: His Work and Impact, pp. 53–56. Elsevier (2013)
2. Aho, A.V.: Computation and computational thinking. Ubiquity 2011, Article No. 1 (2011)
3. Almond, P.: Machines like us, an interview by Paul Almond with John Searle. Machines Like Us (March 2009), `http://machineslikeus.com/interviews/machines-us-interviews-john-searle-0`
4. Bajcsy, R.: Computation and information. Comput. J. 55(7), 825 (2012)
5. Blass, A., Gurevich, Y.: Algorithms: a quest for absolute definitions. Bull. EATCS (81), 195–225 (2003)
6. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bulletin of the American Mathematical Society 21(1), 1–46 (1989)
7. Chalmers, D.J.: Does a rock implement every finite-state automaton? Synthese 108, 309–333 (1996)
8. Conery, J.S.: Computation is symbol manipulation. Comput. J. 55(7), 814–816 (2012)
9. Denning, P. J.: What is computation? (opening statement). Ubiquity 2010, Article No. 1 (October 2010)
10. Deutsch, D.: What is computation (How) does nature compute? In: Zenil, H. (ed.) A Computable Universe: Understanding and Exploring Nature as Computation, pp. 551–566. World Scientific Publishing Company (2012)
11. Fortnow, L.: The enduring legacy of the Turing Machine. Comput. J. 55(7), 830–831 (2012)
12. Frailey, D.J.: Computation is process. Ubiquity 2010, Article No. 5 (November 2010)
13. Fredkin, E.: What is Computation? (How) Does Nature Compute (Transcription of a live panel discussion, with participants Calude, C.S., Chaitin, G.J., Fredkin, E., Leggett, T.J., de Ruyter, R., Toffoli, T., Wolfram, S.). In: Zenil, H. (ed.) A Computable Universe: Understanding and Exploring Nature as Computation, pp. 673–726. World Scientific Publishing Company (2012)
14. Gurevich, Y.: Foundational analyses of computation. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) CiE 2012. LNCS, vol. 7318, pp. 264–275. Springer, Heidelberg (2012)
15. Peach, F.: Interview with David Deutsch. Philosophy Now (30), (December 2000/January 2001)
16. Rosenbloom, P.S.: Computing and computation. Comput. J. 55(7), 820–824 (2012)
17. Searle, J.: Minds, Brains and Programs. Behavioral and Brain Sciences 3, 417–457 (1980)

18. Searle, J.: Is the brain a digital computer? Proceedings and Addresses of the American Philosophical Association 64, 21–37 (1990)
19. Searle, J.: The explanation of cognition. Royal Institute of Philosophy Supplement 42, 103 (1997)
20. Searle, J.: The Rediscovery of the Mind. MIT Press, Cambridge (1992)
21. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. Series 2 42, 230–265 (1936)
22. Turing, A.M.: Systems of logic based on ordinals. Proc. London Math. Soc. Series 2 45, 161–228 (1939)
23. Valiant, L.: Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World. Basic Books, New York (2013)
24. van Gelder, T.: What might cognition be, if not computation? The Journal of Philosophy 92(7), 345–381 (1995)
25. van Leeuwen, J., Wiedermann, J.: Knowledge, representation and the dynamics of computation. In: Dodig-Crnkovic, G., Giovagnoli, R. (eds.) Representation and Reality: Humans, Animals and Machines. Springer (to appear, 2015)
26. Wiedermann, J.: On the road to thinking machines: Insights and ideas. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.) CiE 2012. LNCS, vol. 7318, pp. 733–744. Springer, Heidelberg (2012)
27. Wiedermann, J.: The creativity mechanisms in embodied agents: An explanatory model. In: 2013 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 41–45. IEEE (2013)
28. Wiedermann, J., van Leeuwen, J.: How we think of computing today. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) CiE 2008. LNCS, vol. 5028, pp. 579–593. Springer, Heidelberg (2008)
29. Wiedermann, J., van Leeuwen, J.: Rethinking computation. In: Proc. 6th AISB Symp. on Computing and Philosophy: The Scandal of Computation - What is Computation?, AISB Convention 2013 (Exeter, UK), AISB, pp. 6–10 (2013)
30. Wiedermann, J., van Leeuwen, J.: Computation as knowledge generation, with application to the observer-relativity problem. In: Proc. 7th AISB Symposium on Computing and Philosophy: Is Computation Observer-Relative?, AISB Convention 2014 (Goldsmiths, University of London), AISB (2014)
31. Wikipedia (2013), `http://en.wikipedia.org/wiki/Knowledge`
32. Zenil, H.: What is nature-like computation? A behavioural approach and a notion of programmability. In: Philosophy & Technology (Special Issue on History and Philosophy of Computing). Springer (2013)