The Philosophy of Computation

Jan VAN LEEUWEN

Department of Information and Computing Sciences, Utrecht University
Princetonplein 5, 3584 CC Utrecht, The Netherlands
J.vanLeeuwen1@uu.nl

> *"The underlying phenomena is the generative ability of computational systems [. . .]. Knowledge is the posited extensive form of all that can be obtained potentially from this process."*
>
> A. Newell [16]

Abstract. Computation used to be synonymous to calculation. Now computation is what computers do, and it has become the engine of science. Increasingly more powerful machines push the limit of what is known in many fields. Intelligent systems compete with humans and win. Even natural systems like cells or the brain are occasionally modeled in computational terms. Is this use of computation still consistent with Turing's ground-breaking insights from the 1930s that shaped our understanding of computation till the present? Is there a notion of computation that is more fundamental? This challenging question is one of many in the *philosophy of computation*. We consider some of the relevant issues. What does computation actually do for us? Why do we compute? Is it plausible that cognition is computational as a process? Is there some systematic theory that explains it all? We consider possible answers.

1 Introduction

Computation used to be synonymous to calculation. Now it is one the pillars of modern IT and tends to be identified with everything computers do. Ever more powerful machines push the limits of what models can tell us, in any branch of science. Intelligent systems such as *DeepBlue* [15] and *Watson* [7] compete with humans and win. Robots such as *CHAPPiE* [3] make us believe that feelings, emotions, and even consciousness are computational. Can this really be?

Computation is traditionally about calculating functions and solutions to equations, with or without the aid of a calculating machine. The mathematical interest for computation changed character as the result of Hilbert's famous question [13] whether there exists a finite decision procedure for first-order logic. While solving this problem, Turing [22] developed his ground-breaking model of computation that has shaped our thinking of computation since.

One may question the apparent implication that this has defined computation forever. In fact, even Turing himself [24] conceived of various other types of machines that e.g. used real or random numbers, had modifiable programs, or ran forever. He invented, what we now call, 'artificial neural nets' and speculated about how machines might think, learn and be intelligent. In 1951, Turing [25] contended that *'machines can be constructed which will simulate the behaviour of the human mind*

IIT.SRC 2015, Bratislava, April 23, 2015, pp. 1–12.

very closely', suggesting his belief that mental processes could eventually be understood in terms of, or even *as* computation.

It appears that computation is a much more varied notion than its present technological incarnations suggest. In recent years, the term computation is indeed increasingly being used to describe many other phenomena. Even natural systems like cells or the brain are occasionally viewed as substrates that perform some kind of computation. In his recent book, Valiant [26] even went so far as to explain evolutionary processes like adaptation by computation.

Is Turing's model of computation still the most appropriate for them? Is there a notion of computation that is more fundamental? Is there some overall theory, applicable to whatever sort of computational system, that explains it all? Would it enable us to give satisfactory answers to Abramsky's seminal questions [1]: *why* do we compute, and *what* do we compute?

These challenging question are only some of the many questions in modern *philosophy of computation*. Here we outline some of the issues in the understanding of computation, emphasizing a recent approach by J. Wiedermann and the author [29, 30] and some newer work in [27]. Is it conceivable that eventually a new theory of computation can be devised?

2 Philosophy

Computation has evolved into a highly pluriform notion. This has led to many descriptions of what computation 'is'. These descriptions invariably aim at some characterisation of *how* computation must work on some underlying model of sufficient generality. We give a brief overview of the issues.

2.1 General

The philosophy of computation began, in principle, with the understanding of calculation and arithmetic in ancient times. It now explores the potential of discrete and scientific computation as we know it today. Other origins are found in information processing and in the development of agent-based systems. Also, the computational modeling of mental and natural processes gave rise to many new perspectives on computation.

The philosophy of computation concerns itself with all questions related to the concrete understanding of computation, in all contexts in which it is recognized. Unlike notions like 'information' and 'communication', computation does not have a long tradition in philosophy. Much of what has been studied arose from questions in logic, mathematics, and the sciences from the late nineteenth century onward.

Some of the major issues include: identifying the objects of study within science and technology, understanding the development of computation and computational modeling in context, understanding computation from e.g. an information- or complexity-oriented viewpoint, the role of representations and theories, and the impact of computation on our thinking and our future. Many questions overlap and have aspects that are not listed here.

2.2 Current Views

Computation is now seen as a much broader notion than captured by Turing's model. Even if there are ways to make e,g, Turing's models fit, this exercise may not give a truthful model [28]. Current approaches either aim to enrich existing models like Turing machines with new mechanisms (such as oracles), devise new models inspired by natural systems (like DNA- and quantum computing), or change the perspective altogether (as in abstract state machines [12]).

Many of the current views hold that computation is some kind of *process* that can be seen as manipulating symbols and transforming information, iteratively or otherwise. The refinements most often added, aim to capture the kind of operational mode by which the transformations are effectuated. Here are some of the typical definitions:

– Computation is the execution of step-by-step procedures for processing information (Valiant [26]).

– The most general definition of computation is as information processing. Computation is the process (or collection of processes) of acquiring information, transforming it, and providing the outcome to the outside world (Akl [2]).

– A computation is a physical process in which physical objects like computers, or slide rules or brains, are used to discover or to demonstrate or to harness properties of abstract objects - like numbers and equations (Deutsch [6]).

– Computation in the broadest sense is anything that happens (as opposed to things being static). If so, then the principles of computation are, in fact, the principles of processes (Frailey [11]).

Even though some scientists prefer to stay with Turing's concept of computation from the 1930s ([10]), the different viewpoints reflect the broader scope of computation as perceived now. But, where do the definitions point? Have we really gained much so far?

The viewpoints above have in common that they all (try to) describe computation in some absolute, i.e. observer-*independent* way. This may be possible for computation by machines but becomes problematic in other cases e.g. in computational views of natural systems. Shouldn't one focus on the question *what computation does* rather than on *how it does it*? We will outline the viewpoint developed recently by J. Wiedermann and the author [29, 30] that does exactly this.

3 Computation and Knowledge

Can a change of viewpoint help and make a difference? In this section we will argue that it does. Given the limitations of the machine-oriented views, we will look at computation at another level of abstraction. We will argue that computation, above all, is a process of *knowledge generation*.

3.1 A View of Computation

The views of computation have evolved considerably over time. Computation has been looked at as *calculation*, *symbol manipulation*, *information processing* and as a *process*, often in connection to a model of computation that is considered as being realistic now or in the future.

Also the targets of computation have evolved considerably. Computation used to be reserved for evaluating (mathematical) functions and solving models. Nowadays it is seen as the driving force behind any reactive (input-output) behaviour of whatever system or agent, or their components, for which this makes sense. In medicine, the decision schemes for diagnostics or treatments are being used in a similar sense. How can this be read, philosophically?

Following [29], we look at computation as a phenomenon *above* the systems level, at - what Newell [16] has called - the *knowledge level*. The notion of 'knowledge' is purposely left open and intuitive. What do we gain by taking this perspective?

The connection between computation and knowledge is not a new one. Philosophers like Lull (1230-1315) and Hobbes (1588-1679) already believed ideas and thought were brought about by computation. Leibniz (1646-1716) contended that all 'reasoning' can be reduced to computation, an ideal that is slowly becoming reality. Now we increasingly see processes becoming knowledge-enriched, learning, and *smart*. Computation is increasingly seen as producing knowledge, i.e. information of some kind that is *meaningful* to some substrate or party.

This points to the idea that computation has an epistemical connotation, leading to the following thesis:

– *Computation is a (any) process of knowledge generation, in the context of some suitable knowledge domain* [29].

This viewpoint is attractive because it emphasizes the 'goal' rather than the 'means' of computation. It does require that one knows what is meant by 'knowledge' and how it is taken up, but this is usually understood (and a good test). Some examples of computational systems from different areas are shown in Table 1 (from [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 a 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 [4]	Theory of real numbers	Values of real functions
Oracles [23]	A set $A \subseteq \Sigma^*$	Characteristic function of A
Super-Turing computations	Formal languages in Σ_2	Language membership

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

The table is mostly self-explaining. Note that the *natural computing systems* in the table satisfy our criterion of computationality, but do not do fit most of the classical definitions of computation. The items in the last part of the table seek the limits of our definition, illustrating that computation may be meaningful as a concept even when there is no immediate physical realization of the computational mechanism at hand.

We do not digress on the further details of the definition here. It would require us to express what it means for a *process* Π with input ω to generate some *knowledge item* κ in the context of a *theory* T and the available knowledge for the knowledge domain at hand. To keep the desired flexibility, we have to allow both *formal* and *informal* theories here, as seen in Table 1. Further details are given in [29, 30].

A major advantage of the given definition is that it is no longer fixed to a specific algorithmic mode of implementation. Abramsky's questions [1] about the nature of computation have a direct answer in this framework as well: we compute in order to generate the knowledge we want or need.

3.2 Observing Computations

The question what constitutes knowledge in a certain context clearly depends on the views or theories of the *observer*. What is knowledge to some, may not be for others. The question is whether observer-dependence is *avoidable* at all when deciding whether some process is computational. We outline a provocative argument from [30] that shows that it may not be.

Consider observers, or agents, that are designed to decide the computationality of any process C that they 'see', using the definition of computation above. How might an observer do this? Is there a universal test for computations? Let us specify the question a little further.

Assume, first of all, that observers are computational processes themselves. This is a reasonable assumption if we believe in the testability of computations. A consequence is that it leads to the situation in which one computational process (the observer) has to observe the properties of another, possibly computational process and decide something about it. In the extreme case, an observer may be observing another observer, possible even itself. Before we argue that this is bound to lead to problems, we discuss how an observer could realistically 'do its best'.

When observing a process C, an observer must verify all requirements of the definition we gave, in order to decide whether C is computational. Thus, the observer must do a systematic check whether the knowledge item κ is indeed derivable from the input ω according to theory T, and whether there is an adequate explanation that the underlying process indeed generates it. In [30] various options are described how an observer might do this, even in the context of 'informal' theories. Clearly some observers may be more expert at this than others.

Can there be observers that are competent enough to always make the right decision? This is a difficult question, because we have no absolute notion of decisions being right or wrong when the issues involved are observer-relative like we have. In fact, it might well be that observers inherently disagree many times. The following, informal, result shows that this is indeed unavoidable, under the assumptions we made. The proof is similar to that of Rice's theorem in computability theory.

Theorem 1 ([30]) *There exists no universal observer whose verdict always agrees with the verdict of every other observer.*

The theorem is almost a 'proof' of the observer-relativity of computation. Nevertheless, the claim may be countered by attacking some of the assumptions on which the argument is based. Is the assumption that (universal) observers are computational correct? Can one actually specify and observe processes so a decision of their computationality can always be made? Are observers only useful if they are restricted to the knowledge domains they can handle?

4 A New Model of Computation?

If we accept the view that computation is a knowledge generating process, could it lead to a new kind of theory of computation? It is too early to tell, but we sketch a possible direction in which this may be found. We outline a theoretical model, recently suggested by the author and J. Wiedermann [27].

4.1 Defining Computation

We have distinguished between the 'system' level, where the underlying mechanism of a computation unfolds, and the 'knowledge' level, where the generated outcomes are observed (as in Newell [16]). Collecting the observables at the respective levels, leads us to consider two separate entities for a computation first: the *action space* of a computation, and the *knowledge space* of a computation.

4.1.1 Action spaces

An action space $\mathbb{A}$ contains the meta-items that capture a given computational mechanism in action. The notion is essential in our understanding of computation and is of course observer-dependent.

Observing the meta-items of a computation in progress suggests that there must be a sense of *proximity* among the meta-items as they occur in sequel. Thus action spaces must have some local structure such that computations can be defined in them in a meaningful way at all. We capture this by postulating the following: *Action spaces have an induced topology.*

The topology of an action space derives from the proximity relation that is obeyed by the action of the mechanism. With this postulate, we can use topological notions and define e.g. continuous mappings over action spaces.

We assume that any action space $\mathbb{A}$ contains a *core set* $\mathbb{A}_0$, consisting of the meta-items that correspond to valid 'initialisations' of the underlying mechanism. We make no assumptions on how the mechanism that underlies the computation actually *works*. The mechanism may follow any mode of operation, consist of any number of cooperating components, interact with any environment and more. This gives action spaces the generality we want. We give some examples from [27].

Example 1 *The observable descriptions of a living* cell *form an action space. The meta-items give information about its development, a level of abstraction away from the concrete cell. We may be interested in some special knowledge, e.g. a chemical compound or a property of the cell, which is to be gleaned from the meta-items. (Note that meta-items may be real-valued.) We may also consider the metaspace of a family of cells, as in an experiment. In this case the metaspace is defined by the joint behaviour of the cells over time, with or without taking environmental influences into account.*

Example 2 *The possible 'full information descriptions' of a computer executing a (known or unknown) chain of instructions form an action space. Meta-items display the possible instances of registers and memory filled with bits. By 'observation' we may read out or interpret any meta-item as knowledge, if indeed it fits the sort of knowledge we are interested in. The meta-items can correspond to any mode of execution (sequential, parallel or distributed).*

4.1.2 Knowledge Spaces

At the knowledge level of a computation, knowledge presumably can be qualified in terms of 'knowledge items' (in some domain). We assume that there is some way of delineating the potentially occurring knowledge items in the space (which does not mean that these will all be generable). We assume that every knowledge space $\mathbb{E}$ contains a *core set* $\mathbb{E}_0$ of facts that are initially known, by observation or experience, or just by assumption. The following example is taken again from [27].

Example 3 *The theory of a first-order structure $\mathbb{S}$ forms a knowledge space. The knowledge items are sentences that hold in $\mathbb{S}$. The core set of $\mathbb{S}$ consists of the postulates of $\mathbb{S}$, and the mechanism underlying the space is a combination of first-order inference and the evaluation ('invention') of new sentences. Knowledge in this case follows the standard pattern of a formalized theory.*

One could view the 'dispositions' of the *brain* as a knowledge space as well. In this case, the knowledge items would be our possible mind sets, possibly restricted to a certain topic, and the underlying mechanisms would be provided by our thought. The question whether the brain is a 'computer' or not (cf. [20, 21]) then reduces to the question how the knowledge space is actually explored.

4.1.3 Computation

We now aim for a definition of computation. The definition will be fully machine- and algorithm-free.

Suppose that we have an action space $\mathbb{A}$ and a knowledge space $\mathbb{E}$. Let $\mathbb{A}_0$ and $\mathbb{E}_0$ denote the core sets of these space, respectively. We first express that (some) action-items x with $x \in \mathbb{A}$ may contain information that maps to knowledge in $\mathbb{E}$. We do this by means of a simple readout function called a *semantic map*.

Definition 1 *A* semantic map *from $\mathbb{A}$ to $\mathbb{E}$ is any partial mapping $\delta : \mathbb{A} \to \mathbb{E}$ with the property that $\delta(\mathbb{A}_0) \subseteq \mathbb{E}_0$.*

We require that $\delta(x)$ is obtained by only a simple 'extension' of the observational means that produce x. No substantial extra effort should be involved. The condition that $\delta(\mathbb{A}_0) \subseteq \mathbb{E}_0$ expresses that any knowledge in the items of $\mathbb{A}_0$ should be part of the initial knowledge in $\mathbb{E}_0$.

We now define computations, in this model. Recall that $\mathbb{A}$ is assumed to be a topological space. Thus it makes sense to define *curves* in $\mathbb{A}$. We posit that curves are precisely the sort of trajectories that are traced by computations. Given a curve c, let c^{init} be its starting point and, if it is defined, let c^{end} be its ending point.

Definition 2 *A* computation *is any curve $c \subseteq \mathbb{A}$ with the following properties:*
(i) $\delta(c^{init})$ is defined, and
(ii) if c^{end} is defined, then $\delta(c^{end})$ is defined as well (i.e. $\in \mathbb{E}$).

We require that any computation must start with 'some knowledge' but do *not* insist a priori that $\delta(c^{init}) \in \mathbb{E}_0$. At intermediate points, δ need not always be defined. However, if the curve ends, δ must be defined in its ending point.

Definition 3 *A computation c is said to be* enabled *whenever $\delta(c^{init})$ is known.*

The definition of computation by means of curves seems natural. All information about *how* the computation works is hidden, yet one can formulate all usual phenomena like *convergence* (termination) and *composition* with great ease. We refer to [27] for details.

Finally, the term 'computation' is often used for a whole *family of computations* that are realized by a same mechanism or some conglomerate of mechanisms. We use the term *bundle* here, indicative of a 'programmed' set of computations.

Definition 4 *A* computation bundle *is any collection of computations $\mathcal{B} = \{c_i\}_{i \in I}$ where I is an index set and for every $i \in I$, c_i is a computation (curve) in $\mathbb{A}$.*

One may argue that all computational systems reviewed in Table 1 can be made to fit the model here, using suitable knowledge- and action spaces. We note that there are various other areas as well in which computations are viewed as 'objects' in suitable spaces. We mention control theory, trace theory (for concurrent systems), and computable topology (in type theory). See [27] for more information.

4.2 Exploring Knowledge Spaces by Computation

Given a knowledge space, how can one discover ('reach') the knowledge items in the space? The problem of knowledge generation is well-studied in philosophy and has given rise to principles like formal inference, informal reasoning, analogy and so on. According to the views in [29, 30], the overriding mechanism for knowledge generation is computation.

In trying to make this more tangible, we consider the definition of computation as given above. What knowledge does a computation generate, in this setting? Can one compute more knowledge items as more are found? Can one characterise the set of all knowledge items that can be generated this way? And, can one 'recognize' the knowledge items that can be generated, computationally, in this framework?

4.2.1 Generation

Let $\mathbb{E}$ and let $\mathbb{E}_0$ be as above. Assume that we have a computational mechanism at our disposal for exploring $\mathbb{E}$. Let $\mathbb{A}$ be the underlying action space, $\delta : \mathbb{A} \to \mathbb{E}$ the semantic map that reads out the items, and $\mathcal{B}$ the bundle of computations that we can use.

A crucial question is how 'knowledge' is actually extracted from a computation $c \in \mathcal{B}$. If $c^{init} \in \mathbb{A}_0$ and c^{end} is defined, then we may assume that $\delta(c^{end})$ is a 'logical consequence' of $\mathbb{E}_0$ and thus knowledge of the sort we are after. However, any knowledge computed 'on the way' may be considered as being generated as well. Thus, if c is enabled, the entire set $\delta(c) \subseteq \mathbb{E}$ may be seen generated knowledge. As more and more knowledge is generated, an increasing number of computations from $\mathcal{B}$ get enabled as well. This leads to more knowledge that can be generated, and so on, indefinitely.

Let $K_{\mathcal{B}} \subseteq \mathbb{E}$ be the set of all knowledge items that can be produced and thus become known in this way, using computations from $\mathcal{B}$. In [27] it is shown how to define $K_{\mathcal{B}}$ as a set. More precisely, a set-theoretic operator $G : 2^{\mathbb{E}} \to 2^{\mathbb{E}}$ can be defined such that the following holds.

Theorem 2 ([27]) *$K_{\mathcal{B}}$ is the least fixed point of G that includes the core set $\mathbb{E}_0$. In particular, $K_{\mathcal{B}}$ is well-defined.*

The proof uses the monotonicity of the knowledge generation process, and relies on the Tarski-Kantorovitch Theorem to prove the existence of the fixed point.

4.2.2 Knowledge Recognition

Finally we consider the *recognition* problem for $\mathbb{E}$. This is the problem of determining, for any given knowledge item $e \in \mathbb{E}$, whether e can be obtained by computation from the core set. Recognition can be of greater concern than generation. For example, recognition processes take place in natural systems such as found on the surfaces of cells and in cognition. Is recognition a computational process?

One may think of recognition as a (new) process that generates all knowledge items that are possible and that 'flags' an input item e as soon as e is encountered. It would work precisely for all e with $e \in K_{\mathcal{B}}$. This type of connection between recognition and generation is well-known in classical automata theory [14]. However, is this process computational again, by our definition?

The following definition shows how recognition may be defined as a computational process. For every $d \in D$, let d^+ be a corresponding knowledge item that expresses that d is recognised. Let $D^+ = \{d^+ \mid d \in D\}$.

Definition 5 *A* recognizer *$\mathcal{R}$ for some domain $D \subseteq \mathbb{E}$ consists of the following components:*

- *an action space $\mathbb{B}$ and a knowledge space $\mathbb{F} \supseteq D \cup D^+$,*
- *a semantic mapping $\mu : \mathbb{B} \to \mathbb{F}$,*
- *core sets $\mathbb{B}_0$ and $\mathbb{F}_0$ such that $\{\mu(x) \mid x \in \mathbb{B}_0\} \subseteq \mathbb{F}_0 = D \cup D^+$, and*
- *a bundle of computations $\mathcal{S}$.*

$\mathcal{R}$ is said to recognize *item $d \in D$ if, for some $n \geq 1$, there are computations $s_1, \cdots, s_n \in \mathcal{S}$ with $\delta(s_1^{init}) \in \{d, d^+\}$ such that the composition of $s_1, \cdots, s_n$ leads to knowledge item d^+.*

One can now argue that a recognizer can be construed from the computational process that underlies $\mathbb{E}$. Let $\mathbb{A}$ and $\delta : \mathbb{A} \to \mathbb{E}$ be given for the computational mechanism, and let $\mathbb{B}$ be the bundle we have at our disposal. Assume that $\{\delta(x) \mid x \in \mathbb{A}_0\} = \mathbb{E}_0$.

Theorem 3 ([27]) *With the given conventions, a recognizer for precisely the items in $K_{\mathcal{B}}$ can be constructed, based on the computational mechanism underlying $\mathbb{E}$.*

The proof shows that one can modify the action space $\mathbb{A}$ and the (view of the) computations, such that a suitably designed semantic map that checks equality between knowledge items does the rest.

The approach using curves enables one to model various properties of computation that seem to be inherent to the notion. The given overview represents 'work in progress' that should eventually learn us how far the model can be extended. We refer to [27] for a more complete description of the ideas.

5 Reflections

We have explored several aspects of the philosophy of computation but focused mainly on the core question, namely understanding the nature of computation itself. We have argued that the rise of the computational paradigm in all sciences calls for a broader concept of computation than is expressed in models like Turing's [22]. Where do we stand?

5.1 When is a Process Computational

In many views of computation, one aims to capture how a process must 'compute', in some absolute sense. In order to arrive at a more broadly applicable notion, we have pursued the belief that computations should be understood at - what Newell [16] calls - the knowledge level of the underlying processes. To concretise this, we described the ideas of J. Wiedermann and the author which assert that, in order for a process to be computational, one should be able to view it as a process of knowledge generation. We followed up on this philosophy by designing a topological model of computation.

The details of the model provide a kind of 'test' for the computationality of arbitrary processes. The test should be more widely applicable than previous tests, which all use some kind of analogy to information processing by (networks of) computers. Here are the steps needed for testing the broader notion, given some observed or artificial process or set of related processes.

- Specify the action- and knowledge spaces, and their core sets, that play a role. This step involves crucial decisions on the part of the observer on how the processes are to be observed, sensed, measured, and so on.
- In connection to this, a semantic map should be defined that links the observables of the action space, when applicable, to items in the knowledge space.
- Next, the allowable progressions of the underlying mechanism should be described as a bundle of 'motions' in action space. Again, this step involves crucial decisions on the part of the observer on what the process is effectuating.
- Finally, there should be a justification that knowledge is generated correctly by the laws of the knowledge space, and that it can be 'explained' that this is achievable by the process in action space (regardless of how the process actually does it).

The test has various informal elements. If there is no sense of knowledge being produced, then the test will fail. However, if the test succeeds, the processes at hand can validly be seen as computations, by the philosophy we described. No further assumptions are needed, in particular we do not need any finitary symbolic representations up-front as is often stipulated in the classical case (cf. Fodor [9]). Clearly, some other form of representation may come in when specifying an action space.

5.2 Computation as Knowledge Generation

The understanding of computation as a knowledge generation process brings a variety of advantages that are occasionally lacking in older, mechanistic views. A major advantage is that less conventional cases of computation like processes in cells are covered, in this understanding of computationality. The advantages of viewing computation as knowledge generation can be summarized as follows.

- It gives a criterion for separating computational processes from non-computational ones. We argued that the implicit observer-dependence of this test seems hard to avoid, but also that this seems necessary for applying the notion of computation in modern system contexts.

- It focuses computation on its intrinsic meaning. This gives computation a sound position in philosophy and a potential impact on understanding processes in other areas like cognition and artificial intelligence. For example, in [31] the philosophy is applied in an attempt to understand the nature of *epistemic creativity*. There also may be an impact on epistemology itself, e.g. by the renewed attention for the question of identifying 'knowledge generation'.

- It does not depend on any operative model of the underlying mechanisms. This also recognizes the computational models which emphasize the role of the knowledge level.

- It resolves some of the notorious 'boundary cases' that are difficult with the classical definitions of computation. For example, according to our definition, ecorithms (Valiant [26]) and cognition are computation. For other cases, see [29].

- It allows us to consider computations at a (very) high level of abstraction. This opens the way to new formalizations of computation as a phenomenon, hopefully leading to a new theory of computation that is suited for the applications in modern systems.

- It gives new meaning to the 'computation-centric' perspective on computer science. It explains the omnipresence of computer science as a key discipline, as a consequence of the omnipresence of knowledge generation.

- And, finally, it enables one to answer Abramsky's questions [1] about the essential understanding of computation: what do we compute, and why? The answer presents itself at the moment one steps away from the system level to the knowledge level, as explained above.

The philosophy of [29, 30] clearly needs further scrutiny. What are its limits? Boundary cases to investigate are plenty and can be found e.g. in the computational views of cognition [18, 19] and in pancomputationalism [17].

6 Conclusion

The philosophy of computation has become a broad subject, as computational modeling is now recognized (and used) in the study of almost all scientific phenomena. In this overview, we focused on only one question, albeit a crucial one for the entire field namely: what is computation?

We have especially focused on the recent ideas from [27, 29, 30] which relate computation to the generation of *knowledge* in some suitable knowledge domain. What represents knowledge in this context is observer-dependent. The philosophy differs from previous approaches which have tried to find unified models at the systems levels, aiming to explain how computation is performed. In stead, we aim to understand *what* computation does for us, or for that matter, for any agent that values or even relies on the outcome for its future goals. This makes computation to a philosophically sound and relevant notion.

By bringing the worlds of computation and knowledge generation together, the philosophy reflects the fact that the domains have been remarkably converging to each other in the present time. The approach may well lead to a different type of theory than is known for computation in Turing's sense. This may be unavoidable if one wants to capture the broad range of notions that fit under the term computation today. Understanding computation thus continues to be very much on the agenda. Can new, so far undiscovered forms of computation be identified?

Acknowledgement: This paper is based on recent, joint work with Jiří Wiedermann, Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic.

References

[1] S. Abramsky, Two puzzles about computation, in: S.B. Cooper, J. van Leeuwen, *Alan Turing - His Work and impact*, Elsevier, 2013, pp. 53-57.

[2] S.G. Akl, What is computation?, *Technical Report* 2013-608, School of Computing, Queens's University, Kingston (Ont.), Canada, 2013.

[3] N. Blomkamp, *CHAPPiE*, Columbia Pictures, 2015.

[4] L. Blum, M. Shub, S. Smale, 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).

[5] S.B. Cooper, J. van Leeuwen, *Alan Turing - His Work and Impact*, Elsevier, 2013.

[6] D. Deutsch, What is computation? (How) does nature compute? In: H. Zenil (Editor), *A Computable Universe: Understanding and Exploring Nature as Computation*, World Scientific Publishing Company, 2012, pp. 551-566.

[7] D. Ferrucci *et al.*, Building Watson: An overview of the DeepQA project. *AI Magazine* 31:3 (2010) 59-79.

[8] L. Floridi, *The Philosophy of Information*, Oxford University Press, Oxford, 2011.

[9] J.A. Fodor, The mind-body problem, *Scientific American* 244 (1981) 124-132.

[10] L. Fortnow, The enduring legacy of the Turing Machine, *Comput. J.* 55:7 (2012) 830-831.

[11] D.J. Frailey, Computation is process, *Comput. J.* 55:7 (2012) 817-819.

[12] Y. Gurevich, Foundational analyses of computation, in: S.B. Cooper, A. Dawar, B. Löwe (Eds.), *How the World Computes*, Proc. CiE 2012, Lecture Notes in Computer Science 7318, Springer, 2012, pp. 264-275.

[13] D. Hilbert and W. Ackermann (1928). *Grundzge der theoretischen Logik*,a Springer-Verlag, Berlin, 1928.

[14] J.E. Hopcroft, J.D. Ullman, *Formal languages and their relation to automata*, Addison-Wesley Publishing Company, Reading, MA, 1968.

[15] M. Newborn, *Kasparov versus Deep Blue: computer chess comes of age*, Springer, 1997.

[16] A. Newell, The knowledge level, Presidential address, AAAI'80, 19 August 1980, in: *AI Magazine* 2:2 (1981) 1-20.

[17] G. Piccinini, Computational modelling vs computational explanation: Is everything a Turing machine, and does it matter to the philosophy of mind?, *Australasian Journal of Philosophy* 85:1 (2007) 93–115.

[18] G. Piccinini, S. Bahar, Neural computation and the computational theory of cognition, *Cognitive Science* 37:3 (2013) 453–488.

[19] Z.W. Pylyshyn, *Computation and cognition: Toward a foundation for cognitive science*, MIT Press, Cambridge (MA), 1984.

[20] J.R. Searle, Minds, brains, and programs, *Behavioral and Brain Sciences* 3 (1980) 417-457.

[21] J.R. Searle, Is the brain a digital computer?, *Proceedings and Addresses of the American Philosophical Association* 64:3 (1990) 21-37.

[22] A.M. Turing, On computable numbers, with an application to the *Entscheidungsproblem*, *Proc. London Math. Soc. Series 2*, 42 (1936) 230-265.

[23] A.M. Turing, Systems of logic based on ordinals, *Proc. London Math. Soc. Series 2*, 45 (1939) 161-228.

[24] A.M. Turing, Intelligent machinery, *Report*, National Physical Laboratory, 1948.

[25] A.M. Turing, Intelligent machinery: a heretical theory, in: B.J. Copeland, A lecture and two radio broadcasts on machine intelligence by Alan Turing, in: *Machine Intelligence* Vol 15, Oxford University Press, 1999, pp 445-476.

[26] L. Valiant, *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*, Basic Books, New York, 2013.

[27] J. van Leeuwen, J. Wiedermann, Knowledge, representation and the dynamics of computation, in: G. Dodig-Crnkovic and R. Giovagnoli (Eds), *Representation and Reality: Humans, Animals and Machines*, Springer-Verlag, 2015 (to appear)

[28] J. Wiedermann, J. van Leeuwen, How we think of computing today, In: *Computability in Europe*, Proc. CiE 2008, LNCS 5028, Springer, 2008, pp. 579-593.

[29] J. Wiedermann, J. van Leeuwen, rethinking computations. in: *6th AISB Symp. on Computing and Philosophy: the Scandal of Computation*, AISB 2013 Convention, Proceedings, AISB, Exeter, 2013.

[30] J. Wiedermann, J. van Leeuwen, Computation as knowledge generation, with application to the observer-relativity problem. In: *7th AISB Symp. on Computing and Philosophy*, Proceedings, AISB 2014 Convention, London, 2014.

[31] J. Wiedermann, J. van Leeuwen, Towards a computational theory of epistemic creativity, in: *Proc. AISB 2015 Symposium on Computational Creativity*, AISB 2015 Convention, Canterbury, 2015 (to appear).