Knowledge, Representation and the Dynamics of Computation

Jan van Leeuwen and Jiří Wiedermann

Abstract Cognitive processes are often modelled in computational terms. Can this still be done if only minimal assumptions are made about any sort of representation of reality? Is there a purely knowledge-based theory of computation that explains the key phenomena which are deemed to be computational in both living and artificial systems as understood today? We argue that this can be done by means of techniques inspired by the modelling of dynamical systems. In this setting, computations are defined as curves in suitable metaspaces and knowledge is generated by virtue of the operation of the underlying mechanism, whatever it is. Desirable properties such as compositionality will be shown to fit naturally. The framework also enables one to formally characterize the computational behaviour of both knowledge generation and knowledge recognition. The approach may be used in identifying when processes or systems can be viewed as being computational in general. Several further questions pertaining to the philosophy of computing are considered.

1 Introduction

Can cognitive processes be simulated by machines? Can cognitive processes be understood in computational terms at all? Can this be done without making severe assumptions about any sort of representation of the subjective environment and on the nature of the underlying computational mechanisms? These questions not only

J. van Leeuwen (✉)
Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands
e-mail: J.vanLeeuwen1@uu.nl

J. Wiedermann
Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou v. 2, 182 07 Prague 8, Czech Republic

J. Wiedermann
Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Zikova Street 1903/4, 166 36 Prague 6, Czech Republic
e-mail: jiri.wiedermann@cs.cas.cz

G. Dodig-Crnkovic and R. Giovagnoli (eds.), *Representation and Reality in Humans, Other Living Organisms and Intelligent Machines*, Studies in Applied Philosophy, Epistemology and Rational Ethics 28, DOI 10.1007/978-3-319-43784-2_5

challenge our deepest understanding of cognition and its computational modelling [1], but also that of computation in itself, being commonly tied to algorithms and discrete-state systems only since the ground-breaking insights of Turing. What are the basic properties needed of a process in the human or animal brain in order for it to be regarded as being computational?

The term "computation" is increasingly being used to describe aspects of natural processes outside the well-formalized domain, including many occurring in cognition but also e.g. in living cells. In all these cases, computation is more than just a metaphor. It appears as a much more general notion than what is implied by current theories. It has already given rise to many alternative views, including many in which computation is seen as a process for transforming information in some way. Is there a notion of computation that is more fundamental? In particular, should one not focus more on *what* computations do rather than on *how* they do what they do?

In previous studies [2, 3] we presented computation as some process that generates *knowledge* instead of one that merely manipulates symbols or transforms information. The question of what constitutes knowledge and how it may be generated in any subjective context clearly depends on the views or theories of the observer. However, if we adopt the Aristotelian view that knowledge, old or new, should be demonstrable from basic premises, it makes sense to assume an underlying process, mental or otherwise, that can acquire, deduce, combine, transform, adapt and create knowledge, using some kind of causality as an ordering principle. A knowledge-generating process is likely to interact with an environment and even evolve over time, using new premises as new knowledge is generated, dispensing with old knowledge that is no longer viable. We take the view that computation is what these knowledge-generating processes do (and vice versa).

Viewing computation as a knowledge-generating process presupposes certain manipulative possibilities and rules for knowledge. For our purposes we assume that knowledge can be specified as *items* and that distinct knowledge items can be recognized or observed. We assume that items can be processed, combined and composed (fused) in some way but we will not be more specific than this. In the interest of generality, we make no further assumptions about the concrete representation of knowledge (items) or about any deductive or generative framework for the knowledge domain (theory) that is considered. In Sect. 2 we introduce *metaspaces* to capture the sets of items we need.

Viewing computation as knowledge generation entails that computations are 'observed' in suitable metaspaces, while the generating mechanisms 'live' in other suitable metaspaces. After defining the types of metaspaces involved, we give a general definition of *computation* in Sect. 3. We will define computations as *curves*, in a suitable topological setting. Clearly, the question arises of whether this approach can explain known or new computational phenomena in cognition or otherwise, in a more satisfying and general way than earlier approaches. We give several examples that aim to show that it does. The approach enables us to formally characterize knowledge generation and knowledge recognition as computational processes. We do so in Sects. 4 and 5.

Having knowledge and knowledge generation as a starting point is not only interesting for understanding computation. By linking the worlds of computation and knowledge generation we bring two domains together which have been remarkably converging to each other in the present information age. By concentrating on *what* computation is and does, we may be losing the finer details of computation and the strength of the mathematical theory as we know it since Turing, but this may be required to achieve the abstraction we need today. This paper explores a theoretical framework that implements some of the ideas and viewpoints of our philosophy. The new focus brings further insight into the essence of computation and its intimate relation to cognitive processes and knowledge generation.

2 Metaspaces

The classical approaches to computation rely on machine models and algorithms, but this severely limits the general interpretation of the notion. In order to explain computation in the broadest possible way, we need a new way to abstract from the underlying mechanisms that effectuate it.

In this section we introduce *metaspaces* as generic sets of items that can arise in capturing mechanisms in some way. We subsequently introduce two types of metaspaces that play a role here: *action spaces* and *knowledge spaces*. In the next section we show how these spaces enable us to give an elegant and general definition of computation.

2.1 General

Given our premise that knowledge is generated by a process of some kind, it is implicit that there is some underlying mechanism producing it. We refrain from making any further technical assumptions about such mechanisms, and merely posit that their features can be *captured* at any desired moment in time. The joint features at any time will form the *meta-item* for the mechanism at that time. The collective set of all meta-items corresponding to a mechanism that can arise is called a *metaspace*.

Metaspaces occur in any context where systems or processes are observed. For example, the *configuration spaces* obtained when physical systems are modelled using vectors of parameters are metaspaces. Metaspaces typically have some structure, derived from the way the underlying mechanism is observed or explained. Hence, meta-items will adhere to some descriptive framework or theory for the space we are interested in. A consequence is that meta-items are presentable and distinguishable, even though we do not care how. We note that meta-items do not necessarily characterize a given mechanism completely. Meta-items need not be unique over the lifetime of a process and may repeat even when the observed system is not cyclic.

Metaspaces are *intensionally* defined. Even if meta-items are observable, this does not mean that we know all of them before we observe them nor that we will actually ever observe them in reality. In particular, we do not assume that all items that 'look' like valid meta-items according to some perception of the descriptive framework actually are members of the given metaspace. For example, if meta-items are like theorems of a non-trivial theory, it is clear that we can at best hope to discover some of them in a gradual way. This is also seen e.g. in metaspaces arising in cognition and in Nature.

For every metaspace $\mathbb{M}$ we assume that there is a *core set* $\mathbb{M}_0$ which is 'known' and that there is a process of some sort to *discover* the remaining elements of $\mathbb{M}$, especially when meta-items that contain valuable information ('knowledge') are believed to exist. If no such process is available, we may wish to design it. *Metaspace discovery* will become crucial later on. We will not worry about questions like: are metaspaces sets (we assume they are), are metaspaces enumerable (they probably are) and are meta-items representable (we will discuss this later).

2.2 Action Spaces

Consider any mechanism (process) that is regarded as being computational. The meta-items of the mechanism in action form a metaspace which we will call the *action space* of the mechanism. The notion of action space is dependent on the way meta-items are viewed and thus on the framework used to model the mechanism. Hence, different frameworks could lead to different action spaces for the same mechanism. This happens, for example, when a refined framework or a different theory altogether is used to capture a mechanism. It is similar to the way different 'spectators' may have different views of reality, as in [4].

Action spaces are not arbitrary metaspaces. Observing meta-items while a given mechanism is acting implies a notion of *proximity* among the meta-items as they are occurring in sequence. This is an aspect of action spaces which is intuitively associated with the idea of being computational. Action spaces may be 'continuous' or 'discrete' in this respect, or a combination of both. In order to delineate the action spaces that we need, we resort to mathematical topology and postulate the following:

Action spaces are topological spaces, with a topology consistent with the proximity relation between action items.

The postulate expresses that the topology of an action space 'derives' from the proximity relation observed during the action of the mechanism (i.e. over time or as induced by some other measure). Given the postulate, one can make use of common topological notions and e.g. define continuous mappings over action spaces. The core set of an action space is the collection of meta-items that correspond to valid 'initializations'.

We do not make any further assumptions about action spaces now; in particular, we make no assumptions on how the mechanism that corresponds to it actually *works*. A mechanism may follow any mode of operation, consist of any number of cooperating components, and interact with any environment. This gives action spaces the full generality we wish to preserve.

Example 1 The observable descriptions of a living *cell* form an action space. The meta-items give information about its development over time, a level of abstraction away from the concrete content of the cell. We may also be interested in some special knowledge, e.g. the fluctuation of a chemical compound or a property of the cell, all to be gleaned from the meta-items. (Note that meta-items may be real-valued.) We may also be interested in the metaspace of a family of cells, as in an experiment.

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 in bits. 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 may correspond to any mode of execution (sequential, parallel or distributed) and to any level of abstraction at which we want to observe the mechanism, i.e. the computer system.

2.3 Knowledge Spaces

In viewing computation as knowledge generation, it is implicit that computations generate knowledge that is meaningful in a suitable knowledge domain. In philosophy one distinguishes between many different types of knowledge. We will be mostly concerned with knowledge in a broad Aristotelian sense, as this is most naturally quantized. Knowledge is then basically the collection of 'actualities justified by an understanding', in a sense that may vary over time.

The strong assumption we make is that the knowledge over a given domain can be qualified and described, and bound to a definite 'point of view'. This may be expressed as in some formal theory, but even the use of natural language is not excluded here. The collection of potential 'knowledge items' for a domain will be called the *knowledge space* of the domain. We assume that there is always a descriptive framework or a deductive theory for defining or generating the items of the knowledge spaces that we consider. Hence we postulate that *knowledge spaces are metaspaces.* The core set of a knowledge space consists of the facts that are known by observation or experience, or just by assumption.

Many ways are known by which knowledge can be generated. Knowledge generation has been studied in philosophy ever since the times of Plato and Aristotle. It has given rise to principles such as formal inference, informal reasoning, analogy, knowledge acquisition by communication, cognition, causality and so on. In [2, 3] we argued that computation is a general mechanism for generating knowledge as well. By definition, knowledge generation is merely an instance of the, more general, metaspace generation problem.

Example 3 The theory of a first-order structure $\mathbb{A}$ as known in mathematical logic forms a knowledge space. The knowledge items are sentences that hold in $\mathbb{A}$. The core set of $\mathbb{A}$ consists of the postulates of $\mathbb{A}$. The mechanism underlying the metaspace is a combination of first-order inference and the evaluation ('invention') of new sentences. Knowledge here follows the standard pattern of a formalized theory.

Example 4 The structures (worlds) that are possible instantiations of a given first-order language $\mathbb{L}$ over a fixed base set form a knowledge space. The knowledge items represent the way the 'world' could be shaped, using the functions and relations as they are defined in it. The core set of the space consists of the 'initial worlds' one wishes to observe. The mechanisms underlying the metaspace are 'programs' that modify the assigned values of the functions and relations in a stepwise way, with external influences possibly taking place as well. Worlds correspond to 'states' and the mechanisms to *abstract state machines* as defined by Gurevich [5, 6], provided that certain additional restrictions are imposed (notably, the so-called bounded exploration condition).

Knowledge spaces are special because knowledge is. One may well have mechanisms that act on the items of a knowledge space, turning it into an action space itself. Thus, action spaces and knowledge spaces may be viewed as dual structures, even giving rise to formal equivalences between them if the corresponding actions match, in analogy to similar correspondences between formal structures studied in computer science. Alternatively, a knowledge space may serve as the action space for another, higher-level knowledge space, potentially leading to a hierarchy of levels of abstraction [7].

It is an intriguing thought that the (dispositions of the) *brain* may be viewed as a knowledge space. The knowledge items are our possible mindsets (possibly restricted to a certain topic), and the underlying mechanisms are provided by the facilities of thought. The eternal question of whether the brain is a computer or not (cf. [8, 9]) amounts to the very question of whether, and if so how, the corresponding knowledge spaces can be explored by computation.

3 Computation

Our premise is that, in principle, every computation effectuates some knowledge. We need to have a good model in order to design, explain, prove or understand this and highlight the nature of computation. In this section we give a definition of computation from this viewpoint. The definition will be fully machine- and algorithm-free, and uses minimal assumptions on representation.

We start out by assuming that computation is performed by some process and for some purpose, but how do these things connect? The duality we continually noted between underlying mechanism and knowledge generation is similar to the duality

between agency and goal in the *philosophy of action*. We will give a possible formalisation of this intuitive setting, while staying as general as possible. The formalization implicitly leads to a possible criterion for the computationality of (cognitive) processes as well.

The approach presented here uses ingredients from the modelling of dynamical systems. It does not necessarily implement all aspects of our philosophy of computation as knowledge generation [2, 3]. For example, we will make some concrete assumptions in cases where normally more options would have to remain open. However, the framework as presented is an excellent testbed for the ideas.

We first introduce the metaspaces we need, and then define the notion of computation in our present setting. In Sect. 4 we will show that the framework allows one to manipulate, viz. to compose computations in a natural way and reflect on the various further aspects of the framework.

3.1 Relevant Spaces

In our view, a computational process will always involve two metaspaces: an action space $\mathbb{A}$, and a knowledge space $\mathbb{E}$. The two spaces reflect the 'two sides' of the process. We use this to explain computation, but one may use it to explain knowledge generation quite generally as well. Let $\mathbb{A}_0$ and $\mathbb{E}_0$ denote the core sets of $\mathbb{A}$ and $\mathbb{E}$, respectively.

The spaces $\mathbb{A}$ and $\mathbb{E}$ are coupled. In particular, (some) action items x with $x \in \mathbb{A}$ will carry information that maps to (some) knowledge items in $\mathbb{E}$. We do not require uniqueness. Thus, a knowledge item may be obtainable from several different action items. We assume that the mapping is achieved by way of a simple readout functionality called a *semantic map* which aims to bring out the knowledge that is contained in an action item, in the terms of the knowledge domain.

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 $\delta(\mathbb{A}_0) \subseteq \mathbb{E}_0$.

Given δ and $x \in \mathbb{A}$, we assume that $\delta(x)$ is obtained by only a simple 'extension' of the observational means that produced x to begin with. In other words, no substantial extra effort should be involved that has not already been expended by the underlying mechanism. Note that $\delta(x)$ may be undefined for some items x, reflecting the fact that an action item may not always contain knowledge that is ripe for 'display'. The condition that $\delta(\mathbb{A}_0) \subseteq \mathbb{E}_0$ is required for *consistency*: the knowledge embedded in the (initial) core set of the action space should be part of the core knowledge known at the outset. In particular, it is assumed that $\delta(x)$ is defined for all $x \in \mathbb{A}_0$.

Example 5 Consider any programming language, implemented on a (universal) machine $\mathscr{M}$. Let $\mathbb{A}$ consist of all possible items $\langle \pi, x, J, y \rangle$, where π is a single-input single-output program, x an input value, J the 'full information vector' with the register contents of $\mathscr{M}$ at any moment during π's execution, and y the output or

$\perp$ (undefined) as implied by J. Clearly $\mathbb{A}$ contains the action items of $\mathscr{M}$, seen as a mechanism (cf. Example 2). Let $\mathbb{E}$ consist of the items $\langle f, a, b\rangle$ with $f : \mathbb{N} \to \mathbb{N}$ a partial function, $a \in \mathbb{N}$, $b \in \mathbb{N} \cup \{\perp\}$, and $f(a) = b$. $\mathbb{E}$ is the knowledge space of all single-parameter partial functions. The two spaces can be linked by the semantic map $\delta : \mathbb{A} \to \mathbb{E}$ defined as follows:

$$\delta(\langle \pi, x, J, y\rangle) = \begin{cases} \text{if } J \text{ indicates that the computation is ongoing:} \\ \quad \textit{undefined} \\ \text{if } J \text{ indicates that the computation has terminated:} \\ \quad \langle f_\pi, x, y\rangle \end{cases}$$

where f_π denotes the function determined by program π and $f_\pi(x) = y$. The subspace $\delta(\mathbb{A})$ of $\mathbb{E}$ corresponds to the knowledge of the computable functions only. We have $\mathbb{A}_0 = \{\langle \pi, x, J_{\text{init}}, \perp\rangle \mid \pi \text{ a program}, x \in \mathbb{N}, J_{\text{init}} \text{ the initial information vector}\}$ and $\mathbb{E}_0 = \emptyset$.

Instead of a single knowledge space $\mathbb{E}$ it may be desired to use several spaces and have several semantic maps, to capture different facets of the knowledge that may be generated. This is easily reduced to the case of a single knowledge space only.

3.2 Defining Computation

We can now give a definition of computation, in the present setting. We first define single computations, and then focus on so-called bundles.

Let $\mathbb{E}$ be a knowledge space we are interested in, and let $\mathbb{A}$ be the action space of an underlying mechanism. Let $\delta : \mathbb{A} \to \mathbb{E}$ be a semantic map as above. By assumption, $\mathbb{A}$ is a topological space, and thus we can have topological objects in $\mathbb{A}$ such as *curves*. We posit that curves are precisely the sort of 'trajectories' that are traced by computations.

A *curve* is any continuous function $c : S \to \mathbb{A}$, where S is any segment on the real or integer line that is possibly half-open to the right. (The lines are topological spaces by virtue of the standard metric.) We usually identify c and the image $c(S)$ in $\mathbb{A}$. Given a curve c, we let c^{init} be its starting point and, if it is defined, c^{end} its ending point.

Definition 2 A *computation* is any curve $c \subseteq \mathbb{A}$ with the following properties:

- $\delta(c^{\text{init}})$ is defined, and
- if c^{end} is defined, then $\delta(c^{\text{end}})$ is defined as well.

We require that any computation must start with 'some knowledge'. We do *not* insist a priori that $\delta(c^{\text{init}}) \in \mathbb{E}_0$. If we would be perfectly general, a computation might request 'input knowledge' at later points on the curve as well, but we will not elaborate on this in the present setting. Along the curve, δ need not be defined in every

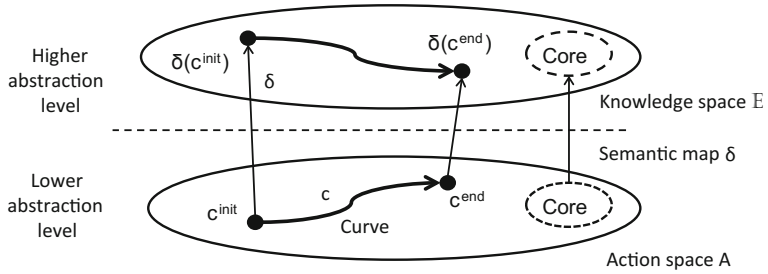

Fig. 1 A schematic diagram of a computation. (Abstraction levels can be iterated)

intermediate action item. However, if the curve ends, δ must be defined in the ending point. A schematic view of a computation is depicted in Fig. 1.

The definition of computation by means of curves is a natural one, fitting the intuition that a computation moves through consecutive action items while respecting the proximity relation in the space. Even 'real-continuous' curves may be needed, for modeling certain natural mechanisms [10].

All information about *how* the computation is effectuated is presumed to be hidden in the details of the action items, which is of no specific concern to us. We may even hide the interactions with other computations in it, checking that their curves 'match' separately. This enables us to concentrate on *what* the computation does. The semantic map will help us read out the 'knowledge' that is generated over the curve (cf. Fig. 1).

A computation c in $\mathbb{A}$ inherits a natural orientation as a curve, progressing from c^{init} towards c^{end} (if it exists). The orientation reflects the (broadly viewed) *serializability* of computations, a notion that is found in many conceptions of computational systems as they operate in any context, regardless of any causal effects whatsoever (but normally consistent with it). A curve can be self-intersecting, without necessarily implying any looping behaviour of the underlying mechanism.

Definition 3 A computation (curve) c is called *convergent* if c^{end} is defined (i.e. as a definite element of $\mathbb{E}$). It is called *divergent* otherwise.

Finally, the term 'computation' is often used to denote not just a single computation but a whole *family of computations* that can be effectuated by the same mechanism or by some conglomerate of mechanisms. We use the term *bundle* here. Let $\mathbb{A}$ be an action space.

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

Whereas the computations are defined by the underlying mechanism of the action space, it is rather more difficult to define what keeps them together as a bundle. An example of a bundle we are most likely interested in is: $\mathscr{B} = \{c \in F \mid c_{\text{init}} \in \mathbb{A}_0\}$, the bundle of all computations in some feasible set F that begin computing in the

core set. We do not insist that bundles are defined in some finitistic way, for example by a program or some other artefact.

Definition 5 A bundle $\mathscr{B} = \{c_i\}_{i \in I}$ is called *(always) convergent* if, for every $i \in I$, c_i is convergent. We call $\mathscr{B}$ *potentially divergent* otherwise.

One may argue that all notions of computation reviewed in [2], classical or otherwise, can be made to conform to our definition. For example, computation seen as *information processing* is obtained by taking appropriate knowledge spaces which just contain 'information' about their domain.

Example 6 Let $\mathbb{I}$ be the internet, M a computer connected to the internet, and π the client program of a search engine running on M. Answering queries using π is seen to be computational as follows: Let $\mathbb{E}$ be the collection of all 'facts' that can be known, and $\mathbb{E}_0$ the subset of currently known facts. Let $\mathbb{A}$ be the set of tuples $\langle q, a, r\rangle$, where q is a query, a a knowledge item or $\perp$ (undefined), and r any possible instantaneous description of π as running on M and accessing $\mathbb{I}$. Let semantic map δ be defined by $\delta(\langle q, a, r\rangle) = a$. Clearly $\mathbb{A}$ can be a very large set, but *we do not need to know all its elements* as long as the search mechanism can generate the ones we need. Define the core $\mathbb{A}_0$ as the set of all tuples $\langle q, \perp, r\rangle$ where q is a query and r is an initial instantaneous description of π. The chain of consecutive action items that result from initializing a search, moving through items as given by the instantaneous descriptions, up to and including an item which has an answer to the query (if any) is a curve in $\mathbb{A}$ and thus a computation. This is easily modified to the case of many answers. It follows that internet searching, viewed as the collection of all searches on initial queries, is a computation bundle.

4 Dynamics

We have seen how computation as a knowledge-generation process can be defined using action and knowledge spaces. However, computations do not stand alone and their result (knowledge) is often used, and needed, in other computations. We will show that this feature can be expressed naturally in the given framework. We will sacrifice some generality in order to show how this can be done. Next we discuss various further aspects of the framework, from a philosophical viewpoint.

4.1 Composing Computations

The question is: What do computations as defined actually entail. Can a given computation c be effectuated even when $\delta(c^{\text{init}}) \notin \mathbb{E}_0$? If not then, supposing c is part of a bundle $\mathscr{B}$, may c be effectuated based on knowledge that is generated by another computation in $\mathscr{B}$? Realistically, many computations will depend on knowledge that

is yet to become available. These computations have to wait for their 'turn' until other computations have produced the lacking knowledge.

Definition 6 A computation c is called *input-enabled* if c can be effectuated fully as soon as c^{init} is available.

There is no reason beforehand to require that computations are input-enabled. It might happen e.g. that a computation needs extra knowledge that is not contained in the core set $\mathbb{E}_0$ and that it cannot compute itself. We will not model all modalities here, and simply assume that each computation is self-contained and 'runnable' whenever c^{init} is 'known', delegating any interactions to the definition of the curve. This will be sufficient to illustrate the principles. Thus:

The computations in all bundles $\mathscr{B} = \{c_i\}_{i\in I}$ we consider are assumed to be input-enabled.

By the assumption, all computations $c \in \mathscr{B}$ with $c^{\text{init}} \in \mathbb{A}_0$ can be effectuated immediately. However, it also makes sense now to define an important further property that is often desired, namely that of *compositionality*.

Definition 7 Let c, d be computations (curves). Let c be convergent and let $c^{\text{end}} = d^{\text{init}}$. Then the curve $c' = c \circ d$ obtained by glueing c and d together at c^{end}, is called the *direct composition* of c and d.

An immediate consequence of this definition is that, if c is enabled and convergent and $c^{\text{end}} = d^{\text{init}}$, then $c \circ d$ is well-defined and enabled as well. The following associativity property is easily verified.

Proposition 1 *Let c, d and e be computations, let c and d be convergent, and let $c^{\text{end}} = d^{\text{init}}$ and $d^{\text{end}} = e^{\text{init}}$. Then $(c \circ d) \circ e = c \circ (d \circ e)$.*

From a computational point of view, direct composition alone is not satisfactory. As a computation c proceeds, we may want to *pre-empt* it at any point that is somehow reasonable, viz. at any point x for which $\delta(x)$ is defined. Any such point might be a valid starting point of a new computation.

Definition 8 Let c, d be computations (curves). Let x be any point on c for which $\delta(x)$ is defined. Let c_x be the curve of c from c^{init} to x, and let $x = d^{\text{init}}$. Then the curve $c' = c_x \circ d$ obtained by glueing c_x and d together at $x = c_x^{\text{end}}$, is called a *grafted composition* of c and d. The set of all possible grafted compositions of c and d is denoted by $c \bigtriangleup d$.

Note that the definition does not require either c or d to be convergent. Also note that a point x may occur more than once on a curve. In general $c \bigtriangleup d$ may consist of many curves (computations). One easily verifies the following:

Proposition 2 *If c, d and e are computations (curves), then:*
- *If $c \circ d$ is defined, then $(c \circ d) \in c \bigtriangleup d$.*
- $(c \bigtriangleup d) \bigtriangleup e = c \bigtriangleup (d \bigtriangleup e)$.

Definition 9 A bundle $\mathscr{B} = \{c_i\}_{i \in I}$ is said to be *closed under composition* if, for all $i, j \in I$: if $c_i \bigtriangleup c_j$ is defined, then $c_i \bigtriangleup c_j \subseteq \mathscr{B}$.

If a bundle $\mathscr{B}$ is closed under (grafted) composition, we simply call it *compositional*. In the next section we will see what role compositionality plays in the analysis of computations for knowledge generation.

4.2 Reflections

The question of identifying the nature and character of computation has been the subject of many studies and discussions [2]. The idea of viewing computation as a process of some kind seems well accepted [11], but the opinions on what makes processes computational differ considerably. For example, concrete processes may be viewed as being computational if a representative abstraction of them is [12, 13]. The definition of computation we gave here implements our philosophy that computation is a process of *knowledge generation* and that this is its driving characteristic.

4.2.1 Cross Connections

Various connections to other areas should be observed. In particular, the way we view computations here is reminiscent of the way systems are viewed in *control theory*. Any dynamical system that evolves over time may be viewed as computational, provided it is generating knowledge from some perspective in the first place. Connections between computer science and control theory were observed before, e.g. by Arbib [14] in the 1960s. Like [14], we believe that the philosophy of computation 'can gain tremendously' from the ideas in general systems theory.

Another connection can be found in the theory of *concurrent systems*. In particular, Mazurkiewicz [15] already showed in the 1970 s that the behavioural aspects of these systems can be adequately studied using *traces* that represent the possible serializations of the interactions that can take place. Sets of traces are an analogue of what we called bundles. This is where the analogies diverge, as the theory of traces has been elaborated entirely at the symbolic level.

Last but not least, we note that topology has been used extensively in the construction of semantic models of programming systems, notably of the λ-calculus [16]. This has led to powerful approaches to the computability of functions and type theory [17]. In general, *computable topology* has focussed on the computability of 'topological objects', rather than on computational processes as we do here. Again, we believe that much can be gained from the ideas in this domain.

4.2.2 Evaluation

Our definition of computation is sufficiently different from extant notions that a critical evaluation is warranted. For example, there is still considerable flexibility in the

underlying notions of knowledge and action spaces. This could give the impression that the definition will allow one to declare just about anything as being computational. We reject this idea, as knowledge and action are sufficiently refined notions to exclude abuse.

Nevertheless, we stress again that the notions are relative to the frameworks and theories in which they are understood, as is the resulting notion of computation. As an example, consider a light switch. The operation of a light switch is not seen as computational, as no notion of knowledge is involved. However, if one declares the signals going around in the switch as being 'knowledge' of its components at suitable times, then one may say that what goes on in the light switch *is* computational. We recall the actor-spectator phenomenon again (cf. Sect. 2.2).

An interesting issue is whether the definition of computation as we gave it is 'free' of representation. It has been claimed that the intensionality of computation requires some form of grounding in a symbolic domain, a view which seems to have been inspired heavily by the classical notion of computing by 'computers'. Fodor [18] has expressed this very eloquently as follows:

> ...it is natural to think of the computer as a mechanism that manipulates symbols. A computation is a causal chain of computer states and the links in the chain are operations on semantically interpreted formulas in a machine code. To think of a system (such as the nervous system) as a computer is to raise questions about the nature of the code in which it computes and the semantic properties of the symbols in the code. In fact, the analogy between minds and computers actually implies the postulation of mental symbols. There is no computation without representation.

In the definition of computation we gave, however, 'symbolic representation' plays no role. Representation is left entirely implicit. This conforms to the view of Piccinini [19], who argues that functional properties of computation may be specified without a need for any semantic properties. It is a great advantage to separate the two notions.

Finally, note that we concentrated on 'what' a computation does and not on 'how' it is effectuated by an underlying mechanism, following [2, 3] and in keeping with the broader views of computation today. Nevertheless, one may still argue that some intuitive machine concept is embodied in the notion of action space. We do not object in principle, as long as the notion of machine is kept as general and open as e.g. in the following definition by Beck [4]:

> A machine [...] is an arrangement of matter devised so that a dependable correspondence is secured between controlled input and usable output.

However, our definition of action spaces does not involve any constraints in terms of input or output or any determined correspondence between them beforehand, leaving room for arbitrary influences from an 'uncontrolled' environment. We reject the idea that computation as a notion requires an analogy to artefacts such as machines, viz. computers. Cases that make use of it are easily subsumed by our definition.

Example 7 It can be argued that the nervous system is computational, using the analogy to complex computing systems. For example, Piccinini and Bahar [20] reason

that the nervous system may be seen as "an information-processing, feedback control, functionally organized, input-output system", although they also point out that this may not explain all of the neural processes involved. In particular they argue that the neural processes are neither analog nor digital. Restricting to the computational part, it is easily seen that this follows from our definition without resort to any functional properties of a computing system.

5 Exploring Knowledge Spaces by Computation

The general problem of discovery in metaspaces was introduced in Sect. 2.1. Can one characterize the knowledge that can be discovered by means of some underlying computational mechanism? And, can one turn the question around and 'recognize' the knowledge items that can be computed?

Let $\mathbb{E}$ be a knowledge space we are interested in, and let $\mathbb{E}_0$ be the core set we have for it. In this section we will explore the following key problems in knowledge space exploration:

- *Knowledge generation*: generate all knowledge items of $\mathbb{E}$.
- *Knowledge recognition*: given a knowledge item e, is it an element of $\mathbb{E}$?

Both knowledge generation and knowledge recognition, when viewed as processes embedded in the human or animal brain, are likely to be constrained further in many ways. For example, knowledge is likely to be aggregated in a coded rather than enumerative way only. Also, recognition may be restricted to the knowledge in a 'known' subset of the items that are potentially knowable. We will not address these constraints but aim to characterize the full extent of the knowledge space that has to be mastered.

We immediately note that knowledge generation and knowledge recognition, when viewed as processes without further constraints, may be indefinite, i.e. without any finite bound on their duration or effect. In the case of knowledge generation this is evident, e.g. when items can be generated multiple times or when the knowledge space itself is infinite and only finitely many knowledge items can be discovered at a time. However, the same can be said of knowledge recognition, e.g. when it relies on some kind of searching without a criterion for when to stop, especially for items that are not valid knowledge and thus cannot be found in the knowledge space at all. This is a well-known phenomenon that occurs when these processes are simulated by artefacts such as Turing machines [21].

This leads to the question of how knowledge generation and recognition can be characterized in our framework. We will first show how to characterize the knowledge in $\mathbb{E}$ that can be generated by computation from $\mathbb{E}_0$, from our present perspective. Next we show how knowledge recognition can be characterized as a computational process, in the defined framework.

5.1 Knowledge Generation

Let $\mathbb{E}$ and $\mathbb{E}_0$ be as above. Assume that we have some mechanism for exploring $\mathbb{E}$ that is in essence computational. Let $\mathbb{A}$ be its action space, and let $\delta : \mathbb{A} \to \mathbb{E}$ be the relevant semantic map. Let $\mathscr{B}$ be the bundle of computations in $\mathbb{A}$ that we potentially have at our disposal. How do we go from here?

A crucial question is how 'knowledge' is actually extracted from enabled computations $c \in \mathscr{B}$. If $c^{\text{init}} \in \mathbb{A}_0$ and c^{end} is defined, then we may tacitly assume that $\delta(c^{\text{end}})$ 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 regarded as available too (assuming it is accessible). Thus, if c is enabled, the entire set $\delta(c) \subseteq \mathbb{E}$ may be seen as generated knowledge.

Making this more precise, we first define when a computation is regarded as being enabled (runnable), cf. Definition 6. We do this recursively as follows:

Definition 10 A computation $c \in \mathscr{B}$ is called *enabled* when either $c^{\text{init}} \in \mathbb{A}_0$ or some enabled computation $d \in \mathscr{B}$ contains c^{init}.

The knowledge-generation process in $\mathbb{E}$ now proceeds as follows: We begin with $\mathbb{E}_0$ and all computations $c \in \mathscr{B}$ with $c^{\text{init}} \in \mathbb{A}_0$, and see what we get. Whenever any new computation gets enabled in the process, we allow it to perform as well. Iterating this ad infinitum, we obtain all knowledge in $\mathbb{E}$ that can possibly become 'known' or, at least, generated (i.e. by this mechanism).

Definition 11 Let $e \in \mathbb{E}$ be a knowledge item. We say that e is *producible* and that computations $c_1, \ldots, c_n$ with $c_i \in \mathscr{B}$ $(1 \leq i \leq n)$ constitute a *production* for e (denoted by $c_1, \ldots, c_n \vDash e$) if and only if the following properties hold:

- $c_1^{\text{init}} \in \mathbb{A}_0$, and
- $e \in \delta(c)$ for some $c \in c_1 \bigtriangleup \cdots \bigtriangleup c_n$.

Recalling that $c_1^{\text{init}} \in \mathbb{A}_0$ expresses that c_1 is enabled as a first 'step' in the computational argument, the definition captures precisely what it means for an item to be knowable (by computation).

Let $K_{\mathscr{B}} \subseteq \mathbb{E}$ be the set of all producible knowledge items. We will show that $K_{\mathscr{B}}$ is indeed a well-defined set. To this end, we first define the function $g : \mathbb{E} \to 2^{\mathbb{E}}$ as follows, for all $e \in \mathbb{E}$:

$$g(e) = \{e' \mid \text{there are computations } c_1, \ldots, c_n, c_{n+1} \in \mathscr{B}\ (n \geq 0) \text{ such that } c_1, \ldots, c_n \vDash e \text{ and } c_1, \ldots, c_n, c_{n+1} \vDash e'\}.$$

Notice that $g(e) = \emptyset$ for any e that is not producible or in case it is and $c_1, \ldots, c_n \vDash e$, if no computation exists in $\mathscr{B}$ that can still be grafted onto c_n. The effect of g is that it extends the knowledge obtainable through productions of some length n to all knowledge producible by one computation more. Now consider the following set-theoretic operator $G : 2^{\mathbb{E}} \to 2^{\mathbb{E}}$:

$$G(K) = K \cup \bigcup_{e \in K} g(e).$$

One observes that the iterative procedure for generating all knowledge in $\mathbb{E}$ that can possibly be produced implies that $K_{\mathscr{B}} = \mathbb{E}_0 \cup G(\mathbb{E}_0) \cup G^2(\mathbb{E}_0) \cup \cdots$.

Theorem 1 *$K_{\mathscr{B}}$ is the least fixpoint of G that includes the core set $\mathbb{E}_0$. In particular, $K_{\mathscr{B}}$ is well defined.*

Proof Clearly $2^{\mathbb{E}}$ is a complete partially ordered set (cpo) under inclusion. By its very definition G is a monotone operator, but a stronger property can be proved:

Claim G is *chain-continuous*, i.e. if $K_1 \subseteq K_2 \subseteq \cdots$ and $\bigcup_{i \geq 1} K_i = K$, then $G(K_1) \subseteq G(K_2) \subseteq \cdots$ and $\bigcup_{i \geq 1} G(K_i) = G(K)$.

Proof By monotonicity one has $G(K_1) \subseteq G(K_2) \subseteq \cdots$ and $\bigcup_{i \geq 1} G(K_i) \subseteq G(K)$. To prove that $G(K) \subseteq \bigcup_{i \geq 1} G(K_i)$ as well, consider any e with $e \in G(K)$. As $G(K) = K \cup \bigcup_{e \in K} g(e)$, we can distinguish the following cases:

- $e \in K$: then there is an $i \geq 1$ such that $e \in K_i$. By monotonicity we obtain that $e \in G(K_i)$ and thus that $e \in \bigcup_{i \geq 1} G(K_i)$.
- $e = g(e')$ *for some* $e' \in K$: then there is an $i \geq 1$ such that $e' \in K_i$. It follows by definition that $e \in G(K_i)$ and again that $e \in \bigcup_{i \geq 1} G(K_i)$.

We conclude that $\bigcup_{i \geq 1} G(K_i) = G(K)$. □

It now follows from the Tarski–Kantorovich fixed point theorem[1] that $K_{\mathscr{B}}$ is indeed the least fixpoint of G in the collection of all sets K with $K \supseteq \mathbb{E}_0$. □

If a bundle is closed under (grafted) composition, then the characterization of $K_{\mathscr{B}}$ reduces to a much simpler form.

Corollary 1 *Let $\mathscr{B}$ be compositional. Then $K_{\mathscr{B}} = G(\mathbb{E}_0)$.*

Proof Let G be the operator as defined above. One easily verifies that the compositionality of $\mathscr{B}$ implies that $G^2(\mathbb{E}') = G(\mathbb{E}')$, for any $\mathbb{E}' \subseteq \mathbb{E}$. Hence, we obtain that $K_{\mathscr{B}} = \mathbb{E}_0 \cup G(\mathbb{E}_0) \cup G^2(\mathbb{E}_0) \cup \cdots = \mathbb{E}_0 \cup G(\mathbb{E}_0) = G(\mathbb{E}_0)$. □

Corollary 1 shows that, if bundles are compositional, all knowledge that can be generated from $\mathbb{E}_0$ can be generated using at most *one* computation from the bundle. This may also be seen from the definition of compositionality directly. Compositionality is a strong property, but it can be expected to hold for all knowledge spaces that are based on a deductive theory.

Finally, the characterization of $K_{\mathscr{B}}$ allows us to define another important notion for knowledge generation by computation, namely *universality*. The concept is of key

[1] The Tarski–Kantorovich fixed point theorem states the following: Let $\langle X, \leq \rangle$ be a cpo and let $H : X \to X$ be chain-continuous. If there is an $x \in X$ such that $x \leq H(x)$, then $x' = \sup_n H^n(x)$ is a fixpoint and in fact the least fixpoint of H among all y with $y \geq x$. For a proof see e.g. [22]. Chain-continuity is also known as Scott-continuity.

importance in many branches of science and philosophy. In our approach here, we may use it to signify that the underlying mechanism is powerful enough to generate the entire knowledge space.

Definition 12 A bundle $\mathscr{B}$ is *universal* for $\mathbb{E}$ if and only if $K_{\mathscr{B}} = \mathbb{E}$.

In classical computability theory, universality refers to the property that all Turing machine programs can be simulated on one single (universal) Turing machine. However, in the approach here, the notion of simulation is completely avoided. This may lead to a possible answer to the quest for a *clear-cut notion of universality* as expressed by Abramsky [23].

5.2 Knowledge Recognition

Now consider the recognition problem, i.e. the problem of determining whether a given knowledge item $e \in \mathbb{E}$ is 'obtainable' from the core knowledge. Our aim will be to define recognition as a process, and show that this process is computational.

Before we get into this question, it should be noted that 'recognition' of knowledge 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. One may argue that in recognition there is as much generation of knowledge going on as there is in any computation, except that the *usage scenario* differs. Let us make this more precise.

We want to think of recognition as a concrete computational process, working on an input datum e from some 'interesting' subdomain $D \subseteq \mathbb{E}$ and 'flagging' it as soon as the process finds that e is recognized. Typically, D will consist of items that have the right form but have to be tested for being valid knowledge, i.e. for being in $\mathbb{E}$. Clearly, when a recognition process is brought to bear on an item e with $e \notin K_{\mathscr{B}}$, one should allow for the indefinite behaviour alluded to before, notably when computational criteria are lacking for items not in $K_{\mathscr{B}}$.

It is well-known from classical automata and formal language theory [21] that the processes of recognition and generation are closely related. We show that this phenomenon emerges at the present, very general level as well. In order to make this concrete, we will show how to define recognition as a computational process in our framework, in a way that it is dual to generation.

The following definition expresses exactly what we expect from the recognition process, hiding all specificities of how the computations in a bundle work. For every $d \in D$, let d^+ be a (new) knowledge item expressing its positive recognition. Let $D^+ = \{d^+ \mid d \in D\}$.

Definition 13 A *recognizer* $\mathscr{R}$ 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 $\{\mu(x) \mid x \in \mathbb{B}_0\} \subseteq \mathbb{F}_0 = D \cup D^+$
- A computation bundle $\mathscr{S}$

$\mathscr{R}$ is said to *recognize* item $e \in D$ if there are computations $s_1, \ldots, s_n \in \mathscr{S}$ with $\delta(s_1^{\text{init}}) \in \{e, e^+\}$ and $s_1, \ldots, s_n \vDash e^+$. The set of all knowledge items from D recognized by $\mathscr{R}$ is denoted by $D_{\mathscr{R}}$.

We now show how a recognizer can be constructed from the computational, generative process that underlies $\mathbb{E}$. We assume that the items in $\mathbb{E}$ have a known form so they can be identified as reasonable inputs. Let $\mathbb{A}$ and $\delta : \mathbb{A} \to \mathbb{E}$ correspond to the computational mechanism for $\mathbb{E}$. Let $\mathbb{B}$ be the bundle we have available for it. Assume that $\{\delta(x) \mid x \in \mathbb{A}_0\} = \mathbb{E}_0$.

Theorem 2 *With the given conventions, a recognizer for the full set $K_{\mathscr{B}}$ can be constructed from the computational mechanism underlying $\mathbb{E}$.*

Proof We define the components of a recognizer $\mathscr{R}$ with $D \equiv \mathbb{E}$, using the generative process as follows:

- Let $\mathbb{B} = \mathbb{A} \times \mathbb{E}$, and let $\mathbb{F} \supseteq D \cup D^+$. Note that, if we supply $\mathbb{E}$ with the discrete (pointwise) topology and take the product with the topology of $\mathbb{A}$, then $\mathbb{B}$ is a topological space again (as required).
- Define the semantic map $\mu : \mathbb{B} \to \mathbb{F}$ in terms of δ as follows:

  $\mu([x, d]) =$ ‘if $\delta(x) = d$ then d^+ else d’.

  The map reflects the intention that, whenever an action item contains evidence that an item d is recognized, it is flagged.
- Let $\mathbb{B}_0 = \mathbb{A}_0 \times \mathbb{E}_0$ and $\mathbb{F}_0 = D \cup D^+$.
- In order to define $\mathscr{S}$ we do the following: For each $c \in \mathscr{B}$ and $d \in D$, let c_d be the curve $c \times \{d\}$, which is a curve in the product topology on $\mathbb{B}$. Let $\mathscr{S} = \{c_d \mid c \in \mathscr{B} \text{ and } d \in D\}$.

The construction specifies a recognizer for $D \equiv \mathbb{E}$, as desired. Moreover, by the assumption that $\{\delta(x) \mid x \in \mathbb{A}_0\} = \mathbb{E}_0$, it follows that the items d that can be recognized ‘at the start’, are precisely those of $\mathbb{E}_0$. Definition 13 implies that the further items that can be recognized are precisely those that can be generated. It follows that $D_{\mathscr{R}} = K_{\mathscr{B}}$. □

By a similar argument one can show that a recognizer for a domain $D \subseteq \mathbb{E}$ which satisfies the specifications of Definition 13 can be ‘moulded’ into a generator for D. This would prove the *functional equivalence* of knowledge generation and recognition, now resulting from the philosophy of computation that we followed.

6 Conclusion

The question of how to characterize computation as an intrinsic notion is a complex one. While analogies to classical models of computation have proved quite satisfactory in the past, the spreading of the computing metaphor to natural systems has made those analogies far less convincing and productive. The question of defining computation adequately therefore remains an intriguing one. Can one capture computation in such a way that the forms of *computationality* as understood today are covered. Can new, so far unfathomed forms of computationality be identified?

In this paper we have followed up on the philosophy developed in [2, 3], in which computation is viewed as a process of generating knowledge. We have presented a theoretical framework in which computations are viewed as processes operating against the backdrop of suitable spaces of knowledge and actions. The framework is widely applicable and allows for a theory of computation which covers the wide variety of processes that are all regarded as computational, without any assumptions on how they work but focussing solely on *what* they do.

Computation is, in our theory, the generation of knowledge in action, with the help of a suitable underlying mechanism. The framework we developed does not require any concrete assumptions on representation, except that there is a 'natural' topology in the relevant action space so computations can be characterized as being *continuous* over the course of their existence. In the resulting framework, knowledge generation can be shown to be a well-defined process. Also knowledge recognition can be captured computationally, from a logico-epistemic point of view.

While the notion of computation has wide reach as intended, it will be of interest to test it on more cases than the current ones we used from conventional and unconventional computing. Philosophically intriguing boundary cases are plenty and can be found e.g. in cognition [24], the more general *computational theory of mind* [25] and in the even more general realm of *pancomputationalism* [26]. As an example one might consider the presumed computationality of the *Universe*. It could be seen as a system which evolves dynamically, producing (implicit) knowledge in the form of life, and life eventually produces explicit knowledge. See also [27] for an expansion on this theme. Hence one may view the meaning of life as being to compute, to produce knowledge and, eventually, wisdom.

Whether a phenomenon can be meaningfully viewed as computational depends on the frameworks and theories through which it is viewed. We posit that, if a process or system is to be viewed as computational, one should be able to characterize it as a knowledge-generating process in some perspective. Then, of course, the decision whether a process is computational becomes observer dependent. Nevertheless, in this way we have provided a 'test' for computationality with wider applicability than the previous tests based on analogies to classical computing systems.

Acknowledgements The work of the second author was partially supported by ICS AS CR fund RVO 67985807 and the Czech National Foundation Grant No. 15-04960S.

References

1. Farkaš, I.: Indispensability of computational modeling in cognitive science. J. Cognit. Sci. **13**, 401–435 (2012)
2. Wiedermann, J., van Leeuwen, J.: Rethinking computations. In: 6th AISB Symp. on Computing and Philosophy: The Scandal of Computation—What is Computation? AISB Convention 2013 Proceedings, pp. 6–10. AISB, Exeter, UK (2013)
3. Wiedermann, J., van Leeuwen, J.: Computation as knowledge generation, with application to the observer-relativity problem. In: 7th AISB Symp. on Computing and Philosophy: Is Computation Observer-Relative? AISB Convention 2014 Proceedings, AISB, Goldsmiths, University of London (2014)
4. Beck, L.W.: The actor and the spectator—foundations of the theory of human action. Yale University Press (1975) (Reprinted: Key Texts, Thoemmes Press, 1998)
5. Blass, A., Gurevich, Y.: Algorithms: a quest for absolute definitions. Bulletin EATCS **81**, 195–225 (2003)
6. Gurevich, Y.: Foundational analyses of computation. In: Cooper, S.B., Dawar, A., Löwe, B. (eds.), How the World Computes, Proc. CiE 2012. Lecture Notes in Computer Science, vol. 7318, pp. 264–275. Springer (2012)
7. Floridi, L.: The Philosophy of Information. Oxford University Press, Oxford (2011)
8. Searle, J.R.: Minds, brains, and programs. Behavioral Brain Sci. **3**, 417–457 (1980)
9. Searle, J.R.: Is the brain a digital computer? Proceedings and Addresses of the American Philosophical Association **64**(3), 21–37 (1990)
10. Tong, D.: The unquantum quantum. Sci. Am. **307**, 46–49 (2012)
11. Frailey, D.J.: Computation is process. In: Ubiquity Symposium 'What is Computation?', ACM Magazine Ubiquity, November issue, Article No 5 (2010)
12. Horsman, C., Stepney, S., Wagner, R.C., Kendon, V.: When does a physical system compute? Proc. Royal Soc. A **470**(2169), 20140182 (2014)
13. Horsman, C., Kendon, V., Stepney, S., Young, J.P.W.: Abstraction and representation in living organisms: when does a biological system compute? In: Representation and Reality: Humans, Animals and Machines. Springer, Heidelberg (2017)
14. Arbib, M.A.: Automata theory and control theory—a rapprochement. Automatica **3**, 161–189 (1966)
15. Mazurkiewicz, A.: Concurrent program schemes and their interpretation. Technical Report No. PB-17, DAIMI, Datalogisk Afdeling, Aarhus University, Aarhus (1977)
16. Scott, D.S.: Continuous lattices. In: Lawvere, F. (ed.), Toposes, Algebraic Geometry and Logic. Lecture Notes in Mathematics, vol. 274, pp. 97–136. Springer (1972)
17. Longo, G.: Some topologies for computations, invited lecture. In: Géométrie au XX siècle, 1930–2000, Paris. http://www.di.ens.fr/users/longo/files/topol-comp.pdf (2001)
18. Fodor, J.A.: The mind-body problem. Sci. Am. **244**, 124–132 (1981)
19. Piccinini, G.: Computation without representation. Philos. Stud. **137**(2), 205–241 (2008)
20. Piccinini, G., Bahar, S.: Neural computation and the computational theory of cognition. Cognit. Sci. **37**(3), 453–488 (2013)
21. Hopcroft, J.E., Ullman, J.D.: Formal Languages and their Relation to Automata. Addison-Wesley, Reading, MA (1968)
22. Ok, E.A.: Elements of order theory. Ch 6: Order-theoretic fixed point theory. https://sites.google.com/a/nyu.edu/efeok/books
23. Abramsky, S.: Two puzzles about computation. In: Cooper, S.B., van Leeuwen, J. (eds.) Alan Turing—His Work and Impact, pp. 53–57. Elsevier, Amsterdam (2013)
24. Pylyshyn, Z.W.: Computation and cognition: toward a foundation for cognitive science. MIT Press, Cambridge MA (1984)
25. Putnam, H.: Brains and behavior. Presented to the American Association for the Advancement of Science, section L (History and Philosophy of Science), 27 Dec 1961

26. Piccinini, G.: Computational modelling vs computational explanation: is everything a Turing machine, and does it matter to the philosophy of mind? Aust. J. Philoso. **85**(1), 93–115 (2007)
27. Dodig-Crnkovic, G.: Modeling life as cognitive info-computation. In: Beckman, A., Csuhaj-Varjú, E., Meer, K. (eds.), Language, Life, Limits, Proc. CiE 2014. Lecture Notes in Computer Science, vol. 8493, pp. 153–162. Springer (2014)