Computation as knowledge generation, with application to the observer-relativity problem

Jiří Wiedermann
Institute of Computer Science of AS CR, Prague, Czech Republic
email: jiri.wiedermann@cs.cas.cz

Jan van Leeuwen
Center for Philosophy of Computer Science, Utrecht University, the Netherlands
email: J.vanLeeuwen1@uu.nl

"Creation of knowledge ... now has to be understood as one of the fundamental processes in nature; fundamental in the sense that one needs to understand them in order to understand the universe in a fundamental way."

D. Deutsch [6]

Abstract. We elaborate our recent thesis [17] stating that computation is a process of knowledge generation. We give two conditions for a process to be computational, i.e. to be a knowledge generating process. First, the epistemic domain in which the computation is carried out must be known, and second, there must be evidence that the generated knowledge is indeed derivable within the underlying domain of discourse by the rules governing the domain and the underlying computational mechanism. The fulfillment of these conditions may be decided by an observer which, again, is modeled as a computational process according to our definition. As a consequence, our definition of computation is observer-relative. The viability of our definition is scrutinized by several examples of computations considered widely in the literature. Among them, we consider the question whether a rock can compute as well as some aspects of Searle's Chinese room thought experiment. The examples illustrate that the epistemic approach to computation brings valuable new insight into the nature of computation and helps to resolve some classical problems related to these examples.

1 INTRODUCTION

When one wants to investigate whether computation is observer-relative, one has to specify the meaning of three key notions of this paper: what is *computation*, what is an *observer*, and how do these notions *link up*.

At first sight it seems surprising that, after sixty years of living with computers, we still ask what the essence of computation is. For people working with computers and computations all their life, the answer seems clear. All agree that computation is a process. Then, depending on their point of view, they usually require that a computational process satisfies some additional properties: it must conform to a formal model of computation [2][7], it must be physically realizable [6], or must be driven by a program [5], etc. Under such definitions of computation, the question boils down to the problem of identifying or recognizing whether a given physical gadget possesses the required qualities. We call this the classical view of computation. This view clearly leads to the conclusion that computational properties are physical, i.e. are intrinsic to physics, and that as such computation cannot depend on the observer, no matter what or who the observer is.

The classical view of computation has proved to be a very potent paradigm. It has lead to the development of the theory of computability and computational complexity as we know it today, showing what can and what cannot be computed by the underlying models of computation and how efficiently this can or cannot be done. All this is carried out in the observer independent framework. However, we seem to be reaching the limit of this approach. Namely, when it comes to solving the omnipresent problems e.g. related to artificial intelligence, and especially to cognition, we do not know at all what the actual potential of our computers is. For instance, we have no good clue of how to program computers in order to (learn them to) think, to be conscious, to acquire, understand and use natural languages or to create new knowledge. All these qualities are considered to be observer-relative qualities.

As a further argument, consider the questions recently posed by Abramsky in [1]: "Why do we compute? What do we compute?" Note that he does not ask "how do we compute?", leaving the question of the implementation of computational processes aside. Obviously, Abramsky aims at an answer from some systematic theory holding for "all computations", i.e. holding for all known and so-far unknown computational processes. From this point of view Abramsky's questions are not general enough. We are increasingly aware that computing is a far more common phenomenon than we believed a few decades ago. For example, many processes in so-called natural computing (or "computing by nature") can be viewed meaningfully as computational. Here computation exists aside of human activities. Thus, more general questions arise: "Why is there computation?", and "What is there computed?"

Our answer to these questions is: computation is there in order to create knowledge, and knowledge is what is computed there. Thus, computation is a far more profound notion than it seemed at first. Our answer implies that computation is not an observer-independent process. It seems that within computer science, little attention was given so far to the possibility that the identification of processes as being computational can depend on an observer. One of the reasons is the relatively restrictive view of computation in computer science. Namely, in this field, computations are designed and analysed against the backdrop of a concrete computational model. There is no room for questions of the sort "does this model indeed compute"? For a computer scientist, a computation is what any of the accepted computational models does (cf. [7]). He takes for granted that whatever mechanism he considers, a computational process ensues, by assumption. Interestingly, when solving a concrete problem, computer scientists inevitably become interested in the goal of such a process. This has led to broader views of computation, to include the desired computational features and results into the definition of computation. Questions about the meaning of computation have appeared only recently [1][4][17].

In this paper we will study the question what computation is and give support to the view that computation is an observer relative phenomenon. We will do so e.g. by specifying and formalizing, to some extent, the properties that seem to be required for an observer so he can give a qualified judgment about the observed process. In this context we will view an observer as a decision process that, in order to decide, must obey the same rules as the processes he observes. This scenario allows a more qualified discussion of conditions under which an observing process can issue its decision.

This paper is a continuation of our earlier paper [17] in which the idea of computation as knowledge generation was presented for the first time. Here we further elaborate our thesis by determining the conditions for a process to be declared to produce knowledge. To this end, a process is considered within the context in which it is used. As a result, a definition of computation is obtained which allows the identification of computational processes. This definition is then applied to the notion of observer itself, modelling him as a computational process as well. We specify what an observer must know in order to reach its decisions. We argue that the paradigm imposes certain limits on the capabilities of observers, forcing them to stay within certain frames of reference. We test the viability of our definitions on various examples of classical and non-classical computation. This includes the current programming practice using the standards of software engineering, the widely considered question whether a rock can compute, the example of an analog computer for measuring the height of a cliff, the Chinese room thought experiment, and the nature of cognitive computations.

Our paper aims to contribute to the further understanding of computation in several ways. First, it leads to a further shift in appreciating the essence of computational processes, by stressing their epistemological aspect. Conditions are stated by which a process can be qualified as being computational, i.e., as a process generating knowledge. A new intermediate stage between computations (in the classical sense) and intelligence - viz. ability to produce knowledge – is identified. Second, we propose a scenario in which an observer is modelled by the same means as a computation. The use of our more detailed definition of a computation allows us to analyse several cases in which an observer has to reach his judgement whether an observed process is computational. Third, the potential of our approach is demonstrated by applying it to widely known "problematic" cases of computation from the philosophical literature. Finally, our approach has great methodological potential, in allowing us to overcome the narrow view of computations as merely being physical processes. The view allows us to concentrate on the main meaning of computations – viz. knowledge gleaning, accumulation and creation. We believe that, as a paradigm, the new view may be equally potent for the field of cognitive computation as the classical view was for classical computability and complexity theory.

The paper is organized as follows. In Section 2 we elaborate the idea of computation as a knowledge generation process and introduce the necessary terminology. The main result of Section 2 is the definition of computation, seen as a process that must fulfil certain conditions in order to be considered as computational. In Section 3 we sketch the scenario of a computational agent "observing" another agent at work and discuss the possibilities of the observer concerning the ways in which he can gain knowledge about the observed process. In Section 4 we examine the power of our approach on various examples of computations generally investigated in the literature. Section 5 contains the conclusions.

2 WHAT IS COMPUTATION

2.1 Computation as knowledge generation When it comes to considering the possibility that computation might not be an observer-independent phenomenon, one has to abandon the stance that computation is a process intrinsic to physics. Instead of the "white box" approach to computational processes through the underlying mechanisms that realize them, one has to concentrate on the properties that make a process computational rather than anything else. This is the approach coined by the authors in [17]. According to this approach, the property that distinguishes computational processes from any other processes is the fact that the former are recognized to explicitly generate knowledge. This being said, one must of course state what is meant by "knowledge". The definition of knowledge is an elusive matter attempted by generations of philosophers, scientists, lawyers, etc. Nevertheless, for our purposes the following "enumerative" definition will do (cf. [18])

"Knowledge is a familiarity with someone or something, which can include facts, information, descriptions, skills or behaviour acquired through experience or education. It can refer to the theoretical or practical understanding of a subject. It can be implicit (as with practical skill or expertise) or explicit (as with the theoretical understanding of a subject); it can be more or less formal or systematic."

In [17] numerous examples are given. These range from the computations of finite and infinite automata, through the past and recent uses of information technologies (scientific computing, transaction systems, data bases, search engines, etc.) up to computations by nature, cognitive computation, and non-Turing computations (e.g., computations with real numbers and compass-and-ruler constructions). These computations can all be seen as knowledge producing processes.

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

If we accept the given extensional definition, knowledge is not observer-independent. After all, the decision on "familiarity with someone or something" is in the eye of the beholder, especially when this concerns knowledge that is not generally accepted. Therefore computation as a process generating knowledge in this sense must be observer-relative. Obviously this statement is not yet fully satisfactory: we must state how a particular computation is related to the specific knowledge it generates. A computational process is not allowed to generate completely arbitrary knowledge.

Each computation is required to generate knowledge over the domain for which the underlying system was designed or to which they both evolved. Similar to how intelligent behaviour of an embodied robot arises from the interaction between brain, body and world, so is knowledge generated by computation in its interaction with the underlying knowledge domain. More formally, there must be a way to verify the correspondence between a given computation and a certain domain over which the computation generates its output in the form of knowledge. For this, every computation will exploit some cognisance of the underlying knowledge domain. A computation is obliged to only use the facts, statements, rules and laws that describe the given knowledge domain and that hold in this domain. We say that a *computation is rooted in its knowledge domain.*

The required attributes of computations can take different forms, depending on our knowledge of the underlying knowledge domain and on our ability to formally describe it, including the rules and laws holding in this domain. The above *table* gives several examples of knowledge domains and their different degrees of formalism. The examples show what aspects must be taken into account when we want to recognize a process as being computational, i.e. as being a knowledge generating process.

2.2 Structure of knowledge In what follows we assume that the knowledge domain that underlies a computation is given in the form of a *theory*. We will not consider theories in the narrow formal sense as in e.g., logic or mathematics. Rather, we apprehend theories as an analytical tool for describing, understanding, explaining and answering queries, for providing solutions and predictions in various areas of science or life, or for the generation or control of behavior. Usually, a theory bears the form of facts, sentences, statements, principles or linguistic descriptions needed for deriving other statements. Nevertheless, other forms of theories are possible as well. For instance, a theory can have the form of a semantic network, a set of restrictions holding for a computation, it can be a map, a scheme, etc. Knowledge produced within the scope of such a theory will have to be of a form that fits the "language" of the underlying domain. The "new" knowledge may be kept in a knowledge base that becomes a part of the theory under consideration. Note that there is no need for such a theory to be correct or truthful w.r.t. the "real world". A theory can even be based on erroneous, unproven or non-verified beliefs and facts. For example, consider some theory in which various myths produce "knowledge" in the form of explanations of various phenomena (such as the weather) due to the intervention of divine beings. Within such a theory, whatever (knowledge) is derived need not involve truth "w.r.t. the correct theory". Yet whatever is derived within a flawed theory is formally considered to be knowledge in that theory (and thus truthful within such a theory).

A possible way of viewing such a highly generalized notion of a theory is to see it as a model of the "world" in which a computation is rooted (cf. [16]). An important characteristic of the notion of theory is that knowledge according to it can be generated time and again from the same base facts and principles, e.g. by computations that do so. In evolving domains, the appropriate theory for the domain will need to evolve as well.

From the table above one can see that, from left to right, the domains range from *theory-full* domains with formal, abstract theories to *theory-less* domains that admit no formal descriptions for capturing e.g. behavior in common life situations (cf. [13]). At the same time, the table characterizes the different levels of formalization, completeness and truthfulness of known theories.

Heterogeneous knowledge characterizes more complex cases. In this case, natural language is an important mediator among theories. Semantics mainly, rather than syntax, is of crucial importance here. Semantics assigns meaning to the individual words and this meaning has the form of knowledge (cf. the previous enumerative definition of knowledge also includes behavior, thus even encompassing embodied semantics). Semantics is knowledge and therefore it is to be represented by a theory again. From this viewpoint all computations, including the computations that generate knowledge based on understanding natural language, bear a homogeneous structure. The knowledge framework behind a computation will normally be based on *cooperating* theories. This is extremely complex since in principle to each word a theory (in our general sense) is attached, controlling the proper use of this word. In general, such a theory depends not only on the word at hand, but also on the context in which the word is being used. In the case of embodied cognitive systems the context does not only refer to the grammatical context, but also to the entire perceptual situation. All this leads to a complex intertwining of the respective theories. In general we do not know much about such cooperating theories.

2.3 Defining computation Based on the considerations as given, we will assume that computations are rooted in knowledge domains and that there exists a theory behind each computation. Within this theory the respective computational process generates knowledge expressed by means of this theory. Of course, in order for this statement to hold there must be evidence (e.g. a proof) that explains that the computational process works as expected. The evidence should ascertain that the process generates the specified knowledge and that this knowledge can be inferred from the underlying theory. The latter is the key to the following definition. In this definition we assume that the input to a computation is part of both the underlying domain (and thus of the theory) and the initial data of the computational process. The notion *"piece of knowledge"* will denote any constant, term or expression which belongs to the theory or can be derived using the respective inference rules of the given theory. In the following definition we will make use of the terminology used in logic, but bear in mind that our notion of "theory" is much broader than in logic and also includes "informal" theories.

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

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

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

Note that, under suitable conditions computations can be composed to obtain new computations, as required by the definition. The property of compositionality is an important one to have.

In the above definition, ω may take any form, e.g. it may be a set of numbers, a query in a formal or natural language, or a statement whose validity we are looking for, etc. The computational process Π acts as a parameter of a computation. Thus, the same knowledge may be generated within the same theory by different computational processes. A change of computational process will result in a different explanation. E.g., the realization of a computational process on a Turing machine requires a different explanation than a realization of this process on a neural net. Whatever Π has to know about T must either be encoded in the design of Π and in ω or Π must have access to T. The condition $(T, \omega) \vdash \kappa$ implies that T is closed with respect to the inference rules of T. This means that, once κ is computed, it can be added to T to extend the knowledge base of T. This can be used, e.g., in an interactive computation where after each interaction, the knowledge base is updated by the recently computed piece of knowledge. When a computation can modify the underlying theory we speak of evolutionary computation. In this way we can model potentially infinite, interactive, evolutionary computations (cf. [15]). Moreover, the formalism also enables us to define universal computations for some domain D which is a subset of T, i.e., a computation where the same computation process Π is used for generating corresponding pieces of knowledge for all ω in D. However, we will not pursue this in this paper.

The two conditions in the previous definition can be shown to be needed. We will demonstrate this by means of some examples from formal language theory, since computations in this domain are well understood.

Note that without the second condition it could happen that the computational mechanism Π is "weaker" than theory T. E.g., Π could be some computational process generated by a finite automaton, whereas T might be a theory of recursive functions. Then it could happen that $(T,\omega) \vdash \kappa$ but there cannot be any E proving that Π generates κ. If Π is "stronger" than T, the computation can still work. For instance, Π could be a process generated by a pushdown automaton whereas T can be theory of regular expressions. However, it may happen that Π is "not compatible" with T. E.g., think of Π being a "process" generated by a logarithmic ruler and T the theory of addition of natural numbers. Then Π cannot compute the sum of two numbers, because the first condition is violated.

A proof that T proves κ on input ω is derived entirely within T and this proof can even be included in κ. This need not be the case with the explanation that Π generates κ on ω. For instance, Π might involve relativistic computations which assume the laws of relativistic physics. These laws need not be a part of T. On the other hand, it can happen that Π makes a direct and exclusive use of the means of T (that is, it "implements" the operations from T). For instance, this is the case when T is the λ-calculus. Then E is equivalent to $(T,\omega) \vdash \kappa$. Interestingly, when considering knowledge generating processes in the brain we accept the fact that thinking occurs in a theory-less domain T best described in a natural language. When attempting to prove that $(T,\omega) \vdash \kappa$, we can reason only informally and rely on the fact that the brain (or the mind, for that matter) "implements" our mental procedures. This means that the "linguistic proof" of $(T,\omega) \vdash \kappa$ serves at the same time as evidence that the brain indeed generates the required knowledge.

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

3 COMPUTATION AS AN OBSERVER-RELATIVE PROCESS

The given definition of computation requires a process to satisfy several conditions in order for it to be a computational process. Who or what can decide whether the conditions from the above definition are satisfied? We shall call any entity designed to do so an *"observer."* How can an observer decide whether a process is a computation (i.e., a computational process)? How can an observer do it? Does it need a kind of `Turing test' for computational processes?

For the observer to be a realistic party, we have no alternative but to view him as a computational process also. This brings us to model the situation in which a computational process (the observer) "observes" another process with the goal to decide whether the observed process is computational one or not. Can an observer achieve this, in a universal sense? (As an ultimate case, we will have to face the situation in which an observer is asked to test, and presumably affirm, another observer for being computational.)

Following our thesis, we are obliged to use the same definition of computation for both the observer and the process to be observed. That is, for both parties the previous definition must be used. Under such a scenario the decision whether an observed process is computational clearly depends on the knowledgeability and computational abilities of the observer. The observer is assumed to have all information required by our definition of computation at its disposal, i.e., the entire 5-tuple $C = (T, \omega, \kappa, \Pi, E)$ of the observed process. This 5-tuple serves as the input to the process run by the observer. Without it, the observer has a totally different task which we will not consider here.

The task of the observer is to verify that C is indeed a computation. In case C is a computation according to our definition, the output (or *verdict*) of the observer, i.e, the knowledge produced by him, consists of a single bit with value 1. Otherwise, the verdict is 0. Thus, what an observer does is:

- checking whether $(T, \omega) \vdash \kappa$, and
- checking whether E is the explanation that Π generates κ on input ω.

Checking the first condition requires the ability to derive a chain of derivations within T starting with ω and ending with κ. In order to check the second condition, the observer must be an "expert" which is able to "judge" the sufficiency of explanation E. We assume that the expertise of the observer is given by a theory T'. The 5-tuple characterizing the computation of the observer is then $C' = (T \cup T', C, v, \Pi, E')$, with $v \in \{0,1\}$ representing the verdict and E' an explanation validating the generation of the respective verdict.

In general, there are several possibilities how C can become known to the observer. A straightforward situation occurs when the observer himself acts as a designer of the computation C, since in this case he will be aware of the information needed to design a computation according to his own intentions. This is the most frequently encountered situation in programming. A similar situation occurs when the information is provided to the observer by a "third", trustworthy person. Next, in case the observer is not the designer of a computation, he alone is forced to reconstruct, or discover the missing information about this computation. This is the situation we find ourselves in when trying to decide whether a certain natural process (perhaps arising by evolution) is a computational process. In theory-full domains it may be possible to "mechanically" verify the correctness of a computation, e.g. with the help of theorem provers.

In all cases, the scenarios make clear that our definition of computation is *observer-relative.* It may well happen (and in practice it does) that an observer sees an observed process as a computational process whereas other observers don't. For instance, until the nineteen eighties or so we did not speak about "computations by nature" simply because we, in the role of observer, were not aware of the computational mechanisms that hide in many processes occurring in nature. Vice versa, it may also happen that an observer wrongly decides that an observed process is computational because his decision may be based on incorrect assumptions concerning the underlying process. This may especially happen in cases where there is no sufficiently formalized and verified epistemic theory in which the computation is rooted. In theory-less domains, observer relativity follows mainly due to the vagueness of the involved reasoning. In most cases the reasoning is sketchy and relies on the ability (or the willingness) of the observer to complete the missing parts of an argumentation and to fill in the "holes" in the respective reasoning. For instance, for a theologian an explanation of God's existence may be fully acceptable but it will not be so for an atheist.

We argued that observer relativity is often explained by insufficient precision of argumentation or low competency of the observer. These reasons may not be sufficient to explain observer relativity in all cases, but they are always present in some form. In fact we will argue that there are deeper reasons for the phenomenon that make it inevitable, reasons rooted in the very nature of computation.

The *obiter dictum* here is that any computational observer is bound to make `personal choices' that are not supported by other observers. Namely, we prove that there can be no *universal* observer O which for each process rooted in theory T decides whether it is a computational process or not and do so in agreement with the verdict of all other observers. In order to see this we will construct a scenario mimicking the argument underlying Rice's theorem in computability theory. In what follows we assume that all processes are rooted in the same theory.

Proposition. There exists no universal observer whose verdict always agrees with the verdict of every other observer, for each 5-tuple $C= (T, \omega, \kappa, \Pi, E)$.

Proof (Sketch). Suppose we had a universal observer O of the claimed quality. We may assume that O is non-trivial, i.e. that processes Q and R exist such that O decides that process Q is computational but that R is non-computational. (If Q or R does not exist, then O decides that every process is or is not computational, which is not what we expect but maybe this is his view of the world. We assume otherwise.) Now design a process P as follows. P uses a copy of the computational observer process O and works as follows, acting on any observed piece of knowledge that represents some process as input:

"P observing process X does the following:

- *if O decides that the process of `X observing process X' is computational, then P continues to behave like R, else*
- *if O decides that the process of `X observing process X' is non-computational, then P continues to behave like Q;"*

Because O is assumed to be universal, this defines a valid process P. But what happens if P observes (a copy of) itself? Then we have P observing process P and

- if O thinks `P observing process P' is computational, then by definition the process of P observing process P actually continues to behave like R (which is non-computational), else
- if O thinks `P observing process P' is *not* computational, then the process of P observing process P continues to behave like Q (which is computational like the steps that led up to this).

This leads to a contradiction. Whatever O's abilities are, O cannot issue a correct verdict from the viewpoint of P. Hence O cannot be universal in the claimed sense. □

The given proposition shows that computational observers are bound to err on some processes, if we merely require them to be general observers. It is almost a 'proof' of the observer-relativity of computation. Nevertheless, the proposition may be countered, by attacking some of the assumptions on which the argument is based. Short of abandoning the assumption that observers can be computational, the only alternative is to accept their limitation or restrict their range of inputs. Are observers only useful if they are restricted to being domain-specific with theories which they can handle?

4 APPLICATIONS

Our definition of computation (cf. Section 2) requires that for a computational process the following questions are answered: what is the underlying knowledge domain, what knowledge is being computed, and how is it computed? Thus, in order for an observer ("agent") to qualify a process as computational, he must decide whether these questions can be answered adequately for the observed process.

First of all, note that the proposed definition of computation corresponds very well to the contemporary theory (and hopefully, also to the practice) of programming. The designer of a program must be aware of a theory T and of the required result κ, and he or she must be convinced that $(T,\omega) \vdash \kappa$. Then there is a computational model in which the designer has to 'program' a computational process Π generating the required knowledge κ. The designer has to deliver also the evidence E that validates the computation, since otherwise one cannot be sure that the program does what was assumed. Obviously, our definition will work for any reasonably formalized model of computation.

Let us verify the strength of the definition, by applying it to a number of types of processes for which it is not obvious, at the first sight, whether they can be seen as computational processes.

First, let us investigate a notorious example of a "computing rock" (cf. [3]). If somebody claims that a rock can compute (or that it implements any finite automaton, for that matter) then he or she must be able to provide arguments or answers for the following. First, what is the underlying knowledge domain (described by T) in which a computation of a rock is rooted? Second, what is the input ω to rock's computation and what is the output κ? Third, what is the proof that $(T,\omega) \vdash \kappa$? And, last but not least, what is the causal evidence E that the rock will produce the expected output κ on a given input ω? The claim that a rock possibly computes depends, of course, on the imagination of the observer, but stating the scenario of such a computation is the hard part of the proof that a rock computes. For instance, one can see a rock as an analog gadget that "computes" its own melting point. Namely, it is known [8] that igneous rocks form through the crystallization of magma. There is a considerable range of melting temperatures for different compositions of magma. All the silicates are molten at about 1200°C and all are solid when cooled to about 600°C. Often the silicates are grouped as high, medium and low-melting point solids. Thus, a set of different silicates can serve as an analog computer for approximately determining the temperature. This is an example where the underlying knowledge domain would include at least geology (or volcanology) and (material) physics. It illustrates a domain of discourse which cannot be described formally; yet for the experts, the usual way of dealing with knowledge in the aforementioned sciences is enough for providing qualified arguments supporting the view that a rock can indeed realize certain computations. This makes the computation performed by our rock computer observer-relative.

The previous example was an example of a measurement. In general, a measurement is a process that for the elements of a theory decides, whether an element at hand satisfies a certain property (temperature in the previous case), or possibly, to what degree the property is satisfied. Thus, a piece of knowledge is assigned to such elements and therefore the underlying process is a computational process. More generally, the question arises whether experimentation can be seen as computation.

Experiments are carried out in order to verify or refute the validity of a conjecture or a hypothesis. Experiments provide insight into cause-and-effect by demonstrating what outcome occurs when a particular factor is manipulated. Experiments vary greatly in their goal and scale, but always rely on repeatable procedure and logical analysis of the results. This description fits perfectly into our framework of computation. The goal of an experiment is to gain knowledge in the domain of a scientific theory. An experiment is typically carried out in a controlled environment consisting of a physical device, or a sample of population of living organisms, with the help of observations, etc. In fact, the setup of an experiment resembles that of an analog computer. The explanation is an important part of any experiment and may even represent the conclusion of the experiment. Special kinds of experiment are thought experiments (Gedankenexperiments). These experiments concern some hypothesis or theory and are designed solely for the purpose of thinking through their consequences. Thus, the goal of a thought experiment is to produce knowledge, and its framework can easily be recast into the context of our definition of computation.

Next, let us inspect another example of a computation involving a rock, described in [11]: *"Consider the example ... of a rock falling off a cliff. The rock satisfies the law $s=1/2gt^2$,* *and that fact is observer independent. But notice, we can treat the rock as a computer if we like. Suppose we want to compute the height of the cliff. We know the rule and we know the gravitational constant. All we need is a stop watch. And we can then use the rock as simple analog computer to compute the height of the cliff."* How does this idea of a computer conform to our definition of a computation? First of all, it appears that for computing the height of a cliff, a rock alone is not enough. In addition to it we need both a stop watch and a person who would observe the falling rock, operate the stop watch and know how to compute the distance travelled by the falling rock given the duration of the fall. Thus, in this case the "computer" consists of a rock and of a person endowed with the abilities just described possessing a stop watch. The cliff serves as an input. The theory behind the computation and the explanation why it works is quite complex if all its details should be mentioned. It will include Newtonian physics, knowledge of the purpose of a stop watch and its use, cognitive theory (for explaining the visual observation ability of the observer and his reaction times, his capability to perform arithmetic operations), etc. But in principle, all these details can be delivered with sufficient plausibility. We conclude that, indeed, the whole system as described does perform a computation according to our definition. (By the way, considering the complexity of the components of this analog computer, one could hardly call it "a simple analog computer" as Searle does.)

Note that arriving at this conclusion has only been possible due to our insight into the entire process. An observer having no idea about the purpose of stop watches and about the laws satisfied by falling bodies, can never come to such a conclusion. Therefore, this instance of computation is clearly observer-relative.

As a third example we consider Searle's Chinese Room Argument [9]. In this thought experiment Searle considers a person speaking only English, located alone in a room. This person follows English instructions for manipulating strings of Chinese characters in such a way that to a Chinese outside the room it appears as if someone in the room understands Chinese. Searle claims that the person in the room cannot understand what the computation is about: the person cannot make the link between the syntactic manipulation of symbols and their semantics. We consider a far simpler question: can the operator learn that the process he is participating in is a computational process (i.e., a process generating knowledge)? To decide this we will use our definition of computation.

First of all we must ask what knowledge domain is behind the computation performed by the room. Obviously, it is the "theory of being Chinese". This is an epistemic theory in which not only the syntax of the language, but also its semantics must be described. As explained in the previous section, semantics of a word depends on the context in which it is being used. Learning this context might need consultation of the "model" of the (Chinese) world (which must be available in the form of another theory or theories) and also inspection of the past contexts in which the word at hand had been uttered in order to recover the history and past information. Using similar information resources an answer can be generated. Having a theory of being Chinese the next step is to decide how the derivations (computations) within such a theory will be implemented on the available computer. This is an easier task and indeed, it can be implemented (albeit in an extremely inefficient way) with the help of a human operator manipulating a set of boxes filled with various cards inscribed by Chinese characters, in accordance with the list of English instructions. Having all this information available a non-Chinese observer can verify that the conditions from the definition of computation are met by the process in the room. By the way, this observer cannot certify that the room "speaks and understands Chinese" – the observer can only certify that what the room does agrees with the information about the process given to him. Whether the behavior produced by the room corresponds to the behaviour of real Chinese can only be certified by "real Chinese".[1] Only then one can claim that the room behaves as if it understood Chinese.

In general, no computation can understand what it is computing unless it is designed so. That means that in the computational mechanisms it must be incorporated how the desired understanding is provided. In the case of the Chinese room, this has been ensured by rooting the computation in the "theory of being Chinese" and the respective world model. This is a necessary condition for the room, as a whole, to be able to explain its activities. From our considerations it is obvious that no part of the room following blindly the instructions can be endowed by understanding since no part of the room on a Chinese input produces Chinese output corresponding to knowledge rooted in the "theory of being Chinese". Hence, no uninitiated non-Chinese operator, being a part of the room, manipulating the cards inside the room can understand what the conversation is about. Searle is right. But behold: what if the observer, with all his knowledge required by our definition about the underlying computation, would take the role of the operator inside the room? Would he "understand" then? Well, no. This is because the observer-operator is not rooted in the same theory (namely in the theory of being Chinese) as the underlying computation is. What he can only do is to verify that the process he is participating in proceeds correctly, as described in his background papers. That is, he can certify that the room generates knowledge – it performs a computation, but he cannot state anything more. Stated otherwise, the observer cannot tell whether the object inside the room makes the link between the "syntax" of the generated knowledge and the "semantics" of it, which seems exactly what Searle seems to claim, but he does it without ascribing another quality to the process inside the room.

The previous example has identified a new intermediate stage between computations (in the classical sense) and intelligence, viz. the ability to produce knowledge. Intuitively, ability to produce knowledge is a prerequisite of intelligence. What ingredients make intelligence stronger than computations, in our sense?

Our final example deals with cognition, namely, with the question mused upon by several authors (cf. [14]): what is cognition if not computation? We believe that part of the problem, if not its essence, in answering such a question lies

[1] This situation reminds of the recent episode with President Obama's fake sign language interpreter at the occasion of N. Mandela's funeral, where the non-deaf people believed that the interpreter was correctly translating Obama's speech (December 2013).

in the definition of computation considered by the respective authors. Namely, if one sees a computation “classically”, as some information process modeled by some machine (e.g. by a Watt governor rather than a Turing machine, as Van Gelder in [14] is proposing), i.e. as a process “intrinsic to physics”, then one immediately loses the main ingredient of cognition, namely its observer dependency (or rather, its “self-dependency”, since in many cases a cognitive system can be seen as an observer viewing itself). It seems beyond any doubt that the main purpose of cognition is to gain knowledge, and that the implementation of this process is immaterial. This is in full agreement with our thesis that computation is a knowledge generation process. Under this definition of computation, cognition is a computational process, or computation, indeed.

5. CONCLUSION

Computation has to be understood as one of the fundamental processes in nature, fundamental in the sense that one needs to understand it in order to understand the universe in a fundamental way. This is the motto of this paper in which “computation” is substituted by the “creation of knowledge”. This is perfectly in order since according to our thesis defended in this paper, “computation is the generation of knowledge”.

Computation is an observer-relative phenomenon in many cases. It is inevitably so, unless one severely restricts the reach of the observers and the domains over which observations are made, which would exclude many types of process which we would want to call computational. It has consequences also for a possible `Turing test’ for computations. The original question behind the Turing test was whether a human observer can distinguish a man from a machine in a conversation, challenging whether the ‘machine’ can be intelligent. Famously, up to now the answer has been positive, i.e. no computer appears to be at par with humans in the Turing test. Let us consider a simpler question: can we distinguish a man from a machine producing knowledge (i.e., according to our definition, a computation)? It appears that there is no straightforward answer favouring humans. For example, search engines can produce knowledge in many domains in a way with which no human can compete. Also, IBM’s Watson offers a compelling example of computational ability that triumphs over the ability of humans to demonstrate knowledge in a large theory-less domain.

Computation is a core notion in computer science. Up to now the view that computation is a process intrinsic to physics has prevailed. We believe that the time has come in which it may be useful – if not necessary - to consider computation as an observer-relative process. This is because we are increasingly facing problems where, due to the nature of the problems to be solved, such a framework is required. This is especially the case of computations related to AGI (artificial general intelligence) which are all rooted in theory-less observer-dependent domains. Changing our thinking of computations towards the view as knowledge generating processes will help in the further development of intelligent information technologies.

ACKNOWLEDGEMENT The work of the first author was partially supported by RVO 67985807 and the GA CR grant No. P202/10/1333.

REFERENCES

[1] Abramsky, S.: Two puzzles about computation. In: S. Barry Cooper, J. van Leeuwen, Eds., Alan Turing: His Work and Impact, Elsevier, 2013, p. 53-56

[2] Aho, A.V.: Computation and computational thinking. Magazine Ubiquity, Volume 2011. Issue January, January 2011, Article no. 1

[3] Chalmers, D.J.: Does a rock implement every finite-stete uutomaton? *Synthese*, 1996, Vol 108, pp. 309-333, Kluwer

[4] Cooper, S.B.: Turing’s Titanic Machine? Comm. of the ACM, March 2012, Vol. 55, No. 3, pp.74-83

[5] Denning, P. J.: What is computation? (opening statement), Magazine Ubiquity, Volume 2010, Issue October, October 2010, Article No.1

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

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

[8] Melting points of rocks: http://hyperphysics.phy-astr.gsu.edu/hbase/geophys/meltrock.html

[9] Peach, F.: Interview with David Deutsch. Philosophy Now. Issue 30 December 2000 / January 2001

[10]Searle, J., 1980, Minds, Brains and Programs, *Behavioral and Brain Sciences*, 3: 417–57

[11] Searle, John R. (1997). The Explanation of Cognition. Royal Institute of Philosophy Supplement 42:103

[12] Searle, J.: The Rediscovery of the Mind. MIT, 270 pp., 1992

[13] Valiant, L.: Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World, Basic Books, (2013)

[14] van Gelder, T.: What might cognition be, if not computation? *The Journal of Philosophy*, Vol. 92, No. 7. (Jul., 1995), 345-381.

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

[16] Wiedermann, J.: On the road to thinking machines: Insights and ideas. Proc. CiE 2012, LNCS 7318, Springer, 2012, pp. 733-744

[17] Wiedermann, J. van Leeuwen , J: Rethinking computation. Proc. 6th AISB Symp. on Computing and Philosophy: The Scandal of Computation - What is Computation?, AISB Convention 2013 (Exeter, UK), AISB, 2013, pp. 6-10

[18] Wikipedia, http://en.wikipedia.org/wiki/Knowledge, 2013