IS COMPUTATION OBSERVER-RELATIVE?

Peter Leupold[A]

[A]Institute of Informatics,
University of Leipzig,
Leipzig, Germany
Peter.Leupold@web.de

Abstract
John Searle raised the question whether all computation is observer-relative. We sketch and illustrate his arguments, which lack a clear definition of what exactly an observer is and what it is able to do. In order to be able to explore the arguments in a more formal way, we propose required properties for the observer, for example non-interference.

Then we argue that the observers employed in the paradigm of Computing by Observing formalize these requirements quite adequately. Under this assumption we provide strong evidence for multiple and even universal realizability in the realm of computation.

0 What is a Computation?

The starting point for our considerations is the following very fundamental question: What is a Computation? For a long time, maybe most scientists' would have replied in similar ways when asked this, especially researchers working in Theoretical Computer Science. These answers would have included a process (physical or abstract), and a function that this process implements and which models some type of information processing. And implicitly, most would probably have assumed some intention that was put into the design of this process in order to implement just this function.

During the last decades, however, many people have come to consider computation in a much wider sense. This is especially true for advocates of a strong version of natural computing, who often see computations already in processes that occur in nature. As an example, take the following quote from Laura Landweber and Lila Kari:

> " . . . ciliated protozoans of genus Oxytricha and Stylonychia
> had solved a potentially harder problem using DNA
> several million years earlier." [12]

Later they add that "in principle, these unicellular organisms may have the capacity to perform at least any computation carried out by an electronic computer." We can assume that several million years ago there was no being with anything that we would call an understanding of

computation, at least not on planet Earth. So there was certainly no intention to compute behind the design of the processes that these ciliates were executing then and which they still execute today.

Today however, there are human beings with two abilities. On the one hand, they can compute and think about computation in an abstract manner. On the other hand, they have the phantasy to map very difficult computations onto the activities of this kind of simple unicellular organisms. Now the question is: are we discovering computations that have been there all the time? Or is computation maybe not an intrinsic property of a process at all? Then there would be no objective way of judging what a computation is.

This set of questions was investigated by John Searle, the inventor of the famous Chinese Room argument [17], in his influential book *The Rediscovery of the Mind* [18]. And his main conclusion is that computation must be *observer-relative*; it is not an intrinsic property that some process either has or has not. Moreover, one and the same process can be interpreted as different computations by different observers. This is what he calls *multiple realizability*; the extreme case that one process actually instantiates all possible computations is called *universal realizability*. These are just adaptions of the general concept of multiple realizability [3] to the realm of computability.

Hillary Putnam showed what follows if this point of view is taken to the extreme. Without getting into technical details of his argumentation we cite his statement that "every ordinary open system realizes every abstract finite automaton" [16]. What this implies was pointed out by David Chalmers in his essay with the somewhat polemic title "Does a Rock Implement Every Finite-State Automaton?" that quite efficiently sums up the contents of the article [9]. So is just about anything doing every imaginable computation all the time? One central concept used in this discourse that is used without a clear definition

Without getting deeper into this mainly philosophical debate, we try to explore the possible role of an observer in a computation in a purely formal manner. For this we use the paradigm called *Computing by Observing*. It already includes an explicit observer in its architecture. Unlike the observer that one might think about when pondering about Searle's argument, our observer seems to be quite integrated into the system that is actually computing. Further, it has some computational power itself although this power is quite limited. Thus one could doubt that any conclusions for the philosophical debate can be drawn from results obtained for Computing by Observing.

However, we argue that the features of our observers are reasonable for any entity that is supposed to observe and understand a computation and its results. To his end we start with an example for the observation of a computation. Then we introduce the computational model of Computing by Observing in its technical details and construct an example that translates the simple introductory example into this domain.

1 Observing a Computation

We first try to get a feeling for the role that an observer might have in a computation. To this end we take a look at an example of how a computation is done. As human beings we maybe have the best intuition when we consider a human computer. So for the format in which the computation is done we choose the same one that Alan Turing had in mind, when he developed the famous Turing Machine: a human being doing a calculation with a pencil and a piece of paper [19]. The observer is somebody who can only read the notes that are written on the piece of paper. He does not know anything about the processes going on in the computer's brain, for example. Our choice for the calculation that is done is very basic: we consider addition in the basic form that many of us were taught in school.

Before we start, let us reflect for a moment on what we expect from an observer in order to classify him as such. One seemingly obvious assumption is that he does not interfere with the observed process; otherwise it would form part of the system rather than observe it. This assumption might be problematic when we get down to an atomic level. Much discussion has revolved around this in Quantum Theory. On the other hand, it is reasonable to assume that macromolecular processes can be observed without changing their course at this level. Watching the computer in our example should not change the way in which he computes. So we will assume there is no interference.

Secondly, the observer must be able to do what its very name promises: observe, that is, gather information. So it should have a way of perceiving the state or activity of the computing process. For example, the observer of a common digital computer should know what is in the (relevant part of the) memory or what instructions are being executed. Note that this type of information goes beyond the mere output of the system that typically is delivered to the user in some format. We assume it safe to suppose that something more than a finite number of input/output pairs are necessary to judge whether a given process is computing or not. Such pairs could at best indicate with a certain probability that more than a random output is produced.

After these preliminaries, let us now finally get to the computation. Suppose that a person is observed while he is writing down the following sequence of symbols from left to right and top to bottom:

$$\begin{array}{cccc} & & \mathrm{X} & \mathrm{Y} \\ + & & & \mathrm{Z} \\ \hline & & \mathrm{Z} & \mathrm{X} \end{array}$$

Assuming the role of the observer ourselves for a moment, we probably do not see immediately that this is a computation, because the symbols used are not the numbers we normally use. On the other hand, already the symbol + and the spacial arrangement of the symbols indicate that we are dealing with an addition. Then the fact that X and Z appear twice leave only a limited number of choices. There are some, however, that work out.

So the most probable consequence is that we will believe that we have observed the computer doing an addition like $29 + 3 = 32$ or $59 + 6 = 65$ with the letters representing the respective digits. These solutions work in the decimal system that we are used to. At the same time there

are solutions with different digits, if we employ number systems of a lower base. For example, an observer, who is more accustomed to numbers in base 8, would probably first arrive at one of the interpretations $27 + 3 = 32$ or $57 + 6 = 65$, because these are closer to his experiences and expectancies. The first point to note here is that different observers can interpret one and the same process as different calculations.

Secondly we note that in order to interpret the writing as a computation, the observer must be able to interpret not only the final result but the entire process. If someone simply spits out a number, there is no way of telling whether it has been computed in some way, or whether it is just produced randomly. So if the observer wants to judge, whether something is a computation at all, he must answer the more specific question what is computed. Without the meaning, computation and random symbol manipulations cannot be distinguished. We only consider the example from above a computation, if we are able to find digits that substitute the letters in a sensible way, i.e. if we manage to assign an interpretation to the steps of the process we witness.

This makes a point similar to the one of the Chinese Room argument, which in Searle's own words showed that '... semantics is not intrinsic to syntax' [18]. In our example different semantics can be assigned to the syntax. We can conclude that there is not one specific semantics that is intrinsic to the sequence of symbols, i.e. to the syntax. As a consequence, meaning is given to such a process only by some agent from outside.

However, when Searle speaks about observer-relativeness, he locates this between the levels of syntax and the physical implementation of this syntax. Thus our example does not yet bring us closer to his conclusion that

> "There is no way you could discover that something is intrinsically a digital computer because the characterization of it as a digital computer is always relative to an observer who assigns a syntactical interpretation to the purely physical features of the system."

Because in our example syntax and the physics beneath it basically coincide, it does not serve to distinguish between these two levels. Nonetheless we want to exploit it a little bit more.

Of course, one might argue that the observer here is more than a mere observer. Even translating the string 32 into a number requires the computation $3 \cdot 10 + 2$ in the decimal system; similarly 65 in base eight is translated into a number via $6 \cdot 8 + 5 = 53$. For the decimal system we usually do not need to think about this kind of calculation, since we are so used to this system as long as the numbers do not become very large. But even for a two digit octal number most humans will need to compute seriously, which illustrates more clearly that even reading and understanding a number requires some computation. It is worth noting that addition and multiplication, which are necessary here, are not mere regular transductions, i.e. they cannot be done by standard transducers. More than regular computational power is necessary to execute them.

From our example we deduce the following: most people would say that the main part of the computation was done by the agent writing down the symbols, not by the observer. So it makes

sense to allow some simple calculations to be done by an observer, if we want to capture the intuitive meaning of computation. Now the role of the observer is a crucial factor, when we speak about observer-relativeness. What is an observer allowed to do? What does it mean to observe? Observing the digit 5 lets us think of the corresponding number in a more direct way than observing 5534; and for octal numbers the way from reading to understanding the number is still less direct. Where do we draw the border between observation and computation?

We will now try to give some formal answers to these questions. Before we start, we should note that our example is still a crude simplification of the situation for Observer Systems, which we will now introduce. In all of the above interpretations, $XY + Z = ZX$ is just one instantiation of the function called addition. But an instance of a model of computation normally specifies an entire function with its infinitely many possible inputs.

2 Computing by Observing

First off, let us again point out the following about the example from Section 1. In order to recognize that something has been computed, we need to look at all the notes that have been taken. Since they are relatively complete and we have some experience with this type of computation, in this case we can reconstruct the rules that the computer followed.

If we do not understand the rules behind what is happening, it is hard to distinguish a computation from just random symbol manipulations. One might even say that it is impossible to decide if something has been computed without knowing what has been computed. It is hard to imagine a way of answering the first question Without being able to answer the second one. Thus we need to see the process in a rather complete way. Suppose the computer in the introductory example had done a bigger part of the calculation only in his head. We could only give meaning to the notes that we observe by doing the calculation again ourselves, filling in the gaps. Only the systematic sequence of steps convinces us.

A somewhat similar situation is the standard setup of an experiment in the natural sciences. Take, for example, the relation between the population sizes of hunter and prey. From observing the population numbers over a long period of time, one can infer the rules that this dynamic equilibrium follows. This role of an observer that logs certain values produced by a process is essential to most experiments. Without this extraction of data there would be no extraction of knowledge.

With this we come to the field of Natural Computing. Its goal is the use of mechanisms present in nature for computing. Biochemical reactions changing DNA strands are an example for a candidate mechanism for building a whole new kind of computer. As we have seen in the quote from Landweber and Kari, sometimes it is even claimed that there is already some computation present in nature that we only need to discover. Advocates of a narrower interpretation of Natural Computing would attribute computations probably only to processes that use mechanisms and materials present in nature, but do so out of their original context and in ways designed by humans.

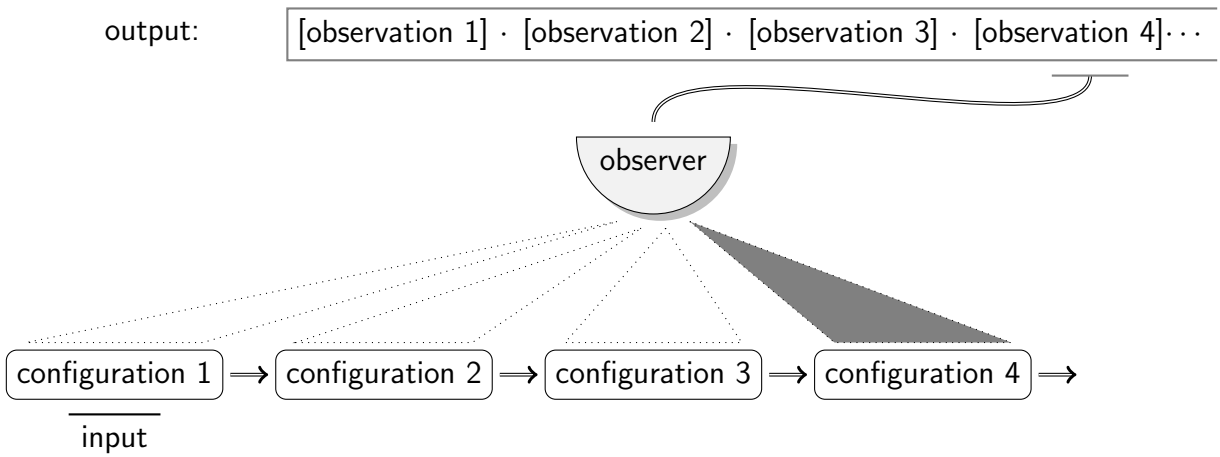

Figure 1: Schematic representation of a transducing observer system.

Normally, the new models for computation that are based on biochemical phenomena present in nature follow the standard paradigm of Computer Science: an input is transformed into an output by some kind of process(or) that follows a certain programme or a set of rules. The output constitutes the result. This is the common approach from Adleman's seminal experiment [1] to the many theoretical models that have been designed since then [15, 10].

So there is a big difference between the ways in which computer scientists and researchers from experimental sciences "use" biochemical systems or abstractions thereof. Computer Science employs biochemical reactions, but does so in a different way from those scientists that have dealt with these phenomena for a much longer time. In the light of this, Matteo Cavaliere and the present author asked themselves, how one could formalize the role of the observer in an experiment that is so important, when scientists want to gain information about processes in nature. Or in other words: how could one compute with biochemical systems using the methods that are used by those people who have dealt with these kinds of systems for a much longer time than the ones who are aspiring to build biocomputers or to discover computation in nature?

The result was the paradigm called *Computing by Observing* that is inspired by the setup of experiments in the natural sciences. Figure 1 depicts the role of a separate observer in this architecture. As in most models of computation –be they standard ones or bio-inspired ones– there is a system that evolves from one configuration to another in discrete steps. This system starts working on the input. However, the output is not produced directly by this system. Rather, each configuration of the system is read by an observer that maps it to a letter. The concatenation of all these letters until the underlying system stops is then the result of the computation. Just like a sequence of (pairs of) numbers is the result of observing hunter and prey populations.

Thus the computation of the underlying system can work on data in any kind of format as long as the observer can read this format. For example, in the initial work on the topic Membrane Systems formed the underlying systems [6]. Their configurations are represented by multisets.

Just as well, there could be a Turing Machine in the role of the underlying system. Then the observer would look at its state and its tape after every step that the machine takes.

In some sense the result that is produced by the observer is an abstraction of the entire computation. We assign one (or none) of finitely many letters to each one of the infinitely possible configurations of the underlying system. Thus we put them into one of finitely many equivalence classes. The final result is then isomorphic to a sequence of classes of configurations rather than to the sequence of configurations. Thus we lose some information, which in the best case is like losing the intermediate notes of a calculation and retaining only the final result.

Another important point is that we usually give the observer the possibility to stop a computation. That is, after reading a configuration it cannot only produce an output letter, but also a special output $\perp$ that immediately invalidates the entire result. In this way, the observer can separate good or meaningful computations from bad ones.

The first central question addressed in work on Computing by Observing has been the following: Is it possible to go beyond the computational power of the components by combining them in this way? Only then the additional effort would be justified. For example, if we already had Turing Machines as underlying systems, adding the observer could obviously not lead to an increase in computational power. The further the underlying system is from being computationally complete, the bigger the increase can be. So the question was if the architecture can lead to a gain in computational power and how big this gain can be?

Many different bio-inspired models and also plain string-rewriting systems were used to explore possible increases in power. Without entering into detail on this topic, the most common pattern was the following one: regular observers with underlying systems of context-free power sufficed to attain computational completeness, for example in the cases of grammars and string-rewriting systems [7, 8]. There were two exceptions to this. For sticker systems [2] and insertion systems [11] already models of regular power were sufficient to compute everything that is Turing-computable. And, of course, sometimes there is no increase in power.

These differences raised another question: can we identify key features in the underlying systems that are crucial for a big increase in computational power? There seem to be two very important features. The first one is the ability to expand without limit the space that is used. Just like any class of Turing Machines with a space bound cannot be computationally complete, an observer system needs the ability to use an unlimited amount of space. For example, If this space is fixed or linearly bounded for a string-rewriting system, then this system and any regular observer can easily be simulated by a linear bounded automaton.

The second feature that is always present in the underlying system, when we observe a big increase in computational power is unlimited re-usability of the working space. For instance, for a string-rewriting system this means that the contents of a position can change arbitrarily often. In such a system, the only processing is the rewriting done by the rules. So information that is not rewritten any more cannot have any influence on the further evolution of the computation.

Because there are only finitely many letters, frequent rewriting will, of course, eventually lead to a repetition of letters in that position. But the relevant information is not only in the symbol

itself, but also in the sequence it runs through. So unlimited access to the information that has been produced is essential. Further, as pointed out by Morse [14] with only three symbols one can obtain a sequence of symbols that is not repetitive in the sense that no subsequence is directly followed by itself again.

Looking again at a linear bounded automaton, if we put a constant bound on the number of times that it can rewrite its cells, this further limits its computational power [20]. So the key features for attaining computational completeness via Computing by Observing can also be recognized in classical models like the Turing Machine.

3 Accepting Observer Systems

Before defining the details of accepting observer systems, we want to point out a weakness of the introductory example: The ambiguity works only for specific combinations of digits. By looking at several different additions we could eventually see that there is only one number system where all of them work. Already the number of distinct digits that appear would tell us with high probability, in which base the calculation is done. Further, there are just some additions where this ambiguity arises. But in Theoretical Computer Science a function consists of infinitely many input/output pairs. So after a few examples for additions, one would see which function is actually computed.

For making our point, the example was good enough. But now we want to say that a system computes different functions depending on how it is observed. This means that these distinct observers must map every input to the corresponding output of the function that is computed. We give an example for such a case. To this end we need to enter into a little more detail about the definition of the instantiation of the Computing by Observing architecture that we will use. So we introduce the formal definitions for the concepts that we have presented in an informal manner in the preceding section. Then we can proceed to explore specific examples. As the observed systems we use string-rewriting systems; for details on these the reader is referred to the monograph by Book and Otto [4].

3.1. The Observer

Central to all of our discussion is the observer. In our context this will always be a monadic transducer. These are deterministic finite automata with the following change: there are no final states; instead there is an output function that maps every state to an output letter.

Definition 1 A *monadic transducer* is a tuple $[Q, \Sigma, \Gamma, \delta, q_0, \phi]$ where Q, Σ, δ, and q_0 are the same as for deterministic finite automata. Γ is the *output alphabet*, and ϕ is the output function, a mapping $Q \mapsto \Gamma \cup \{\lambda\}$ which assigns an output letter or the empty word to each state.

The mode of operation is that the monadic transducer reads the input word, and then the image under ϕ of the state it stops in is the output. We introduce a few notations that will be convenient in describing the interactions between monadic transducers and string-rewriting systems. For a set of string-rewriting rules R, we will use the notation $R(w)$ to denote all sequences of words $(w_1, w_2, \dots, w_k)$ that form terminating derivations $w_1 \Rightarrow_R w_2 \Rightarrow_R \cdots \Rightarrow_R w_k$ of R. For such a sequence σ and a monadic transducer $\mathcal{O}$ we will denote by $\mathcal{O}(\sigma)$ the result of concatenating all the images of the words in the sequence, that is $\mathcal{O}(\sigma) = \mathcal{O}(w_1) \cdot \mathcal{O}(w_2) \cdots \mathcal{O}(w_k)$.

So monadic transducers do not see what actually happens in the derivation steps of the underlying system. They only read the resulting strings that in general do not reveal from which other string they have resulted by the application of which rule. This is much like the observer in the example from Section 1 that can see what is written, but does not know anything about why (by which rule) the writing is done. However, monadic transducers with their regular power cannot do the calculations that were necessary there to understand the computation. Thus what they can do does not seem to exceed an intuitive idea of an observer.

3.2. Computing with the Observer

Now we define how exactly the observation of the string-rewriting system leads to a result of the computation.

Definition 2 An *accepting observer system* is a quadruple $\Omega = [\Delta, R, \mathcal{O}, D]$, where $\mathcal{O}$ is a monadic transducer, R is a string-rewriting system over the input alphabet Σ of $\mathcal{O}$, the system's input alphabet Δ is a subset of Σ, and D is a regular language over the output alphabet of $\mathcal{O}$.

The language accepted by such a system is the set of all words $w \in \Delta^*$ such that there exists a terminating derivation sequence $s \in R(w)$, whose observation is accepted by the decider, i.e. $D(\mathcal{O}(s)) =$ `accept` if D is given in the form of a finite automaton; formally

$$L(\Omega) := \{w : \exists s[s \in R(w) \wedge D(\mathcal{O}(s)) = \texttt{accept}]\}.$$

Note that $\mathcal{O}(s)$ is what is called output in Figure 1 that depicts the general architecture, which can also be used to do transductions or to generate languages. Here the observer can discard computations by outputting a symbol $\perp$ that does not appear in the language D. For more details and results on the computational power of accepting observer systems the reader may consult the article of the present author with Matteo Cavaliere, where these systems were introduced [8].

To illustrate the operation of an accepting observer system we now look at an example. Here the underlying system is the string-rewriting system with the four rules $a \to A$, $A \to b$, $b \to B$, and $B \to C$. The input words consist only of the letters a and b. Configurations of such a system are simply strings. First we construct an observer, with which this system can recognize words of the form $a^n b^n$.

The interesting computations in this case mark the left-most a by rewriting it to A; then they do the same to the left-most b by rewriting it to B. If both types of markings can be done the same number of times, obviously the original number of a and b was the same. The role of the observer consists mainly in filtering out and discarding those computations, where the string-rewriting rules are not applied in exactly the manner just described. This is why all the rewritings are done in two steps, for example $b \to B \to C$ instead of directly rewriting $b \to C$. In this way, every rewriting leaves a kind of trace (in this case the letter B) in a configuration and can thus be detected by the observer.

In detail, the observer realizes the following mapping:

$$\mathcal{O}(w) = \begin{cases} I & \text{if } w \in a^+b^+ \\ 1 & \text{if } w \in b^*Aa^*c^*b^* \\ 2 & \text{if } w \in b^*Aa^*c^*Bb^* \\ 3 & \text{if } w \in b^*a^*c^*Bb^* \\ 4 & \text{if } w \in b^*a^*c^*b^* \\ \oplus & \text{if } w \in b^*c^+ \\ \lambda & \text{if } w \in b^*Bc^+ \\ \perp & \text{else} \end{cases}$$

Here we use regular expressions to specify the languages and λ denotes the empty word. The words that are mapped to I are input words that have the correct order of symbols, i.e. only a followed by only b. The words mapped to numbers 1 to 4 correspond to the four phases of marking and rewriting one a and one b as described above. If we arrive at a string mapped to $\oplus$, the numbers of a and b were the same. Then it remains to rewrite all b to c, because the system has to stop in order to accept. Since there is no rule that rewrites c, a string of only c brings the system to a halt.

So the observations that lead to acceptance of the input string have the form $I(1234)^*\oplus^+$ and thus form a regular language. The language accepted by the system is $\{a^nb^n : n > 0\}$. It is important that the clauses of the observer mapping are disjoint; otherwise the corresponding monadic transducer could not be deterministic. Further, the capital letters play a key role. In accepting computations there are never more than two of them in the string at any given time. Further, the sequence

$$\cdots \Rightarrow \cdots A \cdots \Rightarrow \cdots A \cdots B \cdots \Rightarrow \cdots B \cdots \Rightarrow \cdots$$

is the only one in which they can appear and disappear producing the sequences 1234 in the output. In this way the observation indirectly contains information about the sequence of rules that has been applied.

Now we want to use the same string-rewriting system with a different observer to accept the following language: all the strings a^p where p is a prime. We use the facts that multiplication is repeated addition and that addition is just concatenation for unary numbers. Thus for every number k the string a^k can be factored into $a^i \cdot a^i \cdots a^i$ for every divisor of k. Therefore k is a prime if and only if such a factorization with $1 < i < k$ can be found.

The string-rewriting system realizes the following algorithm: guess a divisor d of the length of the input word. Rewrite the first d letters a to A and then to b. Now rewrite the left-most a to A, the left-most b to B and so on until all b are gone. Then all A are rewritten to b and all B to c. In this way we again arrive at a suffix of d letters b followed only by a. Iterating this, we rewrite the entire word to c with a final suffix of b only if the length is a multiple of d.

In this case the observer mapping is as follows:

$$\mathcal{O}_2(w) = \begin{cases} I & \text{if } w \in A^*a^+ \cup b^*A^+a^+ \\ S & \text{if } w \in bb^+a^+ \\ 1 & \text{if } w \in c^*b^*Bc^*a^* \\ 2 & \text{if } w \in c^*b^*Bc^*Aa^* \\ 3 & \text{if } w \in c^*b^*c^*Aa^* \\ 4 & \text{if } w \in c^*b^+c^+b^*a^* \\ F & \text{if } w \in c^+b^+a^+ \\ \oplus & \text{if } w \in c^+b^+ \\ \lambda & \text{if } w \in c^+Bb^+ \\ \bot & \text{else} \end{cases}$$

While the divisor is guessed, strings are mapped to I. When a string of class S is reached, the check for divisibility starts. The part bb^+ guarantees that we have not guessed the trivial divisor one, the presence of more a guarantees that we have not guessed the number itself. The rewriting of the letters b is started at the right end of the block. Otherwise the border between the two blocks would not remain clear after the first a is rewritten to b. We run through the phases 1 to 4 until the last iteration where the final configuration is mapped to F. If there are no more a left, which means we have found a divisor, the resulting observation is $\oplus$. Then we only need to replace the remaining b by c so the systems stops, just like in the example above.

So a word is accepted if the observation belongs to the language

$$I^+S[(1234)^+(123F)]^*[(1234)^+(123\oplus)]\oplus^* .$$

If the input is not of prime length, then at some point there cannot be a 2 after a 1. So only the desired words are accepted.

What we have seen is that we can accept very different languages with the same string-rewriting system and different observers. So we have generalized the example from Section 1 from the calculation of one specific addition to the general notion of function as it is used in computability theory.

It is worth noting that the string-rewriting system we have used is extremely simple. Its rules only replace one single letter by another. The context cannot play any role, and the length of the string remains constant, when such a rule is applied. This type of string-rewriting systems is called a *painter* system. Taken by itself, such a system cannot compute much. To be more precise, a language is accepted by a painter system, iff it can be represented as A^*BA^* where A and B are arbitrary subsets of the alphabet. In the Computing by Observing architecture,

however, all context-sensitive languages can be computed with these systems [8]. It is also clear that nothing more can be computed, because the working space is fixed and cannot be expanded.

After these two examples, it is clear that the string-rewriting system used above could also be used to accept many other languages. Straight-forward examples are $\{a^n b^n a^n : n > 0\}$ or the language of all words that contain the same number of a and b. Languages like $\{a^k :$ k is a multiple of $\ell\}$ for any ℓ are just special cases of the language of prime length that we have treated above. In general, there is no limit to the number of languages that can be accepted with the same underlying system. So we definitely have an example of multiple realizability here. But we can do even better than that,

The strongest result that generalizes our example is Theorem 3 in the work of Cavaliere, Frisco, and Hoogebom [5]. They construct a single rewriting system S that is universal in the following sense: for every Turing-computable language L there exists an observer, such that L is computed by S in combination with the specific observer. They show this for the instantiation of Computing by Observing that generates languages. But it is straightforward to adapt this for accepting observer systems.

Theorem 3 *There exists a context-free string-rewriting system R such that for every recursively enumerable language L there exists a monadic transducer $\mathcal{O}_L$ such that the accepting observer system composed of R and $\mathcal{O}_L$ accepts L.*

A context-free string-rewriting system is one whose rules have left sides of length only one. It is worth noting that the rewriting system used here is even simpler than that. Besides painter rules as we have used them in the examples above, only one additional rule of the form $a \to ab$ is necessary. Here the letter a is preserved, which in many cases is a severe restriction for such a rewriting system. Thus without the observer the system would have less than context-free power and it is really the combination of the two components that yields the computational completeness here.

Theorem 3 tells us that one and the same process can be used for doing the main part of any computation that is possible at all in the sense of Turing. In other words: more observer-relativeness is not possible in this context, and we have found an example for universal realizability.

The forms of the rules $a \to b$ and $a \to bc$ are two very simple patterns that –with a little phantasy– can be found in almost any place in nature. For instance, we can take biochemical reactions. $a \to bc$ could be the splitting up of a larger molecule, $a \to b$ could be a change in the folding of a protein. Looking only at the molecules and disregarding matters like energy that is set free or bound, this would be quite direct implementation of the rules. If we monitor just one organic molecule in a solution of many molecules, the rule $a \to ab$ mentioned above could simply be the appending of another molecule at a certain point, for example the addition of one more base to a DNA strand.

Of course, we have to abstract away from many details to see certain phenomena as instantiations of our rules. However, the same kind of abstraction is necessary when we look at human or electronic computers and interpret their syntactical manipulations as computations. Thus

Theorem 3 gives strong support to Searle's claim that just about anything can be viewed as a digital computer.

4 Further Questions

The reader who basically accepts our arguments up to this point might now be troubled when he thinks about the digital computer he is probably using. These devices are widely accepted as machinery that computes. However, normally the interaction with the user is only via input/output. Humans do not monitor what is going on inside the memory and the processor. And even though this would still be technically possible, barely anyone would be able to check whether something, let alone what exactly is being computed. Other observers are not in sight either? So are we mistaken when we assume that computers compute?

We can resolve this seeming contradiction if we allow some of the observation to be replaced by trust. Let us imagine a user that tries to decide whether a digital computer about which he has no knowledge computes some function or not. He has no idea about the operating system and the programs that are running and just receives output corresponding to his inputs. It is safe to assume that he cannot say much about the device's activity. But normally we are in a different situation.

There are instruction manuals for the programs we use. There messages on the screen that explain what the output is, and thus implicitly what has been computed. Only if we trust in the correctness of their claims the results become useful to us. in other words, we are convinced that there could in principle be an observer that interprets the process that has run as just the computation that is claimed to have been executed. Of course, this trust might no be justified. Indeed, often enough computer programs compute things different from the ones they claim to compute. But without this trust it would be hard to recognize and use computations that are as complex as the ones executed by our digital computers.

A second question that arises is of a more technical manner. Let us accept that there is no computation without an observer, and that monadic transducers adequately formalize such an observer that does not form part of the computation. Then Theorem 3 basically tells us that a context-free mechanism is sufficient to compute all the computable functions. Why then do we use much more complex mechanisms like Turing Machines to characterize computations? And where does the observer disappear on the way to a Turing Machine? Of course, we could assume Turing Machines to implicitly be underlying systems in our architecture. With them, computational completeness would be easy to attain. And on the other hand, they could simulate a monadic transducer observing their own computation.

Acknowledgments

The present author wishes to stress the important contributions of Matteo Cavaliere in the developing the paradigm of Computing by Observing and in establishing some of the technical

result that constitute the basis for the observations laid out here.

Figure 1 is a slight adaption of one that was drawn by Norbert Hundeshagen.

This work expands on ideas that were first laid out by the present author at the 7th Symposium on Computation and Philosophy [13].

References