How We Think of Computing Today*

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

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz
[2] Department of Information and Computing Sciences, Utrecht University,
Padualaan 14, 3584 CH Utrecht, The Netherlands
j.vanleeuwen@cs.uu.nl

Abstract. Classical models of computation no longer fully correspond to the current notions of computing in modern systems. Even in the sciences, many natural systems are now viewed as systems that compute. Can one devise models of computation that capture the notion of computing as seen today and that could play the same role as Turing machines did for the classical case? We propose two models inspired from key mechanisms of current systems in both artificial and natural environments: evolving automata and interactive Turing machines with advice. The two models represent relevant adjustments in our apprehension of computing: the shift to potentially non-terminating interactive computations, the shift towards systems whose hardware and/or software can change over time, and the shift to computing systems that evolve in an unpredictable, non-uniform way. The two models are shown to be equivalent and both are provably computationally more powerful than the models covered by the old computing paradigm. The models also motivate the extension of classical complexity theory by non-uniform classes, using the computational resources that are natural to these models. Of course, the additional computational power of the models cannot in general be meaningfully exploited in concrete goal-oriented computations.

Keywords: Turing machines, evolving automata, interactive computation, non-uniform complexity.

1 Introduction

Can the Internet be simulated, at least in principle, by a Turing machine? Can the living cell, the brain, and any other natural information processing system be simulated likewise? The answer is not at all clear and depends very much on one's viewpoint. While the Turing machine paradigm is well suited for modeling stepwise computational processes, it may be less suited for modeling the behaviour of the computational systems as we know them today. What model

* This research was partially supported by project BRICKS in the Netherlands, and by Institutional Research Plan AV0Z10300504 and grants No. 1ET100300419 and 1ET100300517 within the Czech National Research Program 'Information Society'.

A. Beckmann, C. Dimitracopoulos, and B. Löwe (Eds.): CiE 2008, LNCS 5028, pp. 579–593, 2008.

of computation could replace the classical Turing machine and serve as the new paradigm?

The shift from Turing machines to a new computational model should correspond to the shift in thinking about computing in the systems of today. It is no longer the case that only isolated computers compute. We have no problem admitting that sensor nets, embedded control systems in all kinds of interacting devices and robots, 'always operating' information services, and in fact the Internet, as a whole, perform computations, albeit in some non-standard way. Moreover, it is no longer the case that only artificial gadgets compute. Biologists frequently speak of living cells or entire organisms as complex information processing systems, as do psychologists in the case of the human mind and sociologists in the case of animal or human societies. Some physicists even believe that the entire Universe can be viewed in this way [3]. Can there be a single model of computation covering all these cases, like the Turing machine did for early computing, or are we deemed to have many different models, tailored to each case at hand?

In this expository paper we describe a number of computational paradigms that have emerged in recent years and that lead to ingredients for new models of computation. We give the background for these paradigms and of some models that have been based on them. We show that some of these models are indeed, at least in principle, more powerful than classical computing in the sense that they are provably computationally more powerful models than those fitting the old paradigm. The models also enable extensions of classical complexity theory (cf. [18,19]), showing that our modern notions of computing can lead naturally into the domain of non-uniform complexity.

2 From Isolated to Interactive Computation

> *New technologies, from the telegraph to the World Wide Web, have expanded our abilities to communicate widely, flexibly, and efficiently. This urge to communicate will continue to drive the expanding technology with the advent of widespread two-way video, wireless connectivity, and high-bandwidth audio, video, 3-D imaging, and more yet to be imagined.*
> T. Winograd ([23], 1997).

The way we think of computing is closely related to what we consider to be a computation.

2.1 Classical Computing

Historically, when arithmetic was invented in the early days of mankind, computing seemed an ability by which only people are endowed. Later, when abaci and mechanical calculators appeared, it was taken for granted that 'computing devices' are artifacts designed by people. Consequently, computing was perceived as an activity intrinsic to people or to devices invented by people for that purpose. Computing was seen as an 'invented' process, in contrast to the 'natural'

processes which were driven by natural laws and which worked 'by itself'. Computing was something artificial or non-natural: it had to be planned ahead, streamlined, powered and monitored. Computing devices had a rigorous, regular and highly organized structure and therefore, had to be engineered with great ingeniousness. Mainframe-, midi- and minicomputers and the many types of PCs confirm this view of computing. In theory, their functioning is suitably modeled by the Turing machine paradigm or by easily simulated models like the random access machine.

Ever since Turing's formulation of the model [12] in 1936, classical Turing machines have dominated the thinking of computing. However, the paradigm of Turing machines does not only suggest that any process which deserves to be called algorithmic can be modeled by a Turing machine. There is more to it: the paradigm also assumes a certain *computational scenario* which determines how the machine is used. In classical Turing machines, this scenario requires that a finite amount of input data is present prior to the start of a computation; during the computation no new data can be added. The result is to be extracted after a finite number of steps and only after, and when, the computation has terminated. This allows one to view a computation as a process that maps finite input data to finite output data and hence to view a computer as a device realizing standard mathematical functions, calculating their value given an input value. Computability theory has been based on this view.

2.2 'Always On' Computing

A different view of computing systems arose when the first automated control systems emerged. Here the computer was not used to compute function values. Instead it was used to monitor, to serve, or to process potentially infinite streams of data. Normally, as in the first computer operating systems and modern 'always on' systems, infinite input streams are presented as un-ending streams of finite chunks of data. Each chunk is processed according to the Turing machine paradigm. Aside from mathematical reasons, it inspired computability theory to study infinite computations, by using Turing machines (or restricted variants like finite automata) under the *generalized* scenario of infinite input strings and infinite output strings. The results were seen as a natural generalization of the finitary case, not as a revision of the Turing machine paradigm. After all, the machine model had remained the same, merely the computational scenario had changed. There is now a refined theory of ω-automata [10], with many applications in e.g. process theory. In relativistic computing, infinite computations are seen from yet a different angle [4,21].

2.3 Interactive Computing

From computations over infinite streams of data it is only a small step to *interactive* computations, where a machine interacts continuously with its environment. The computational view of interaction was propagated by Wegner [20]. In interactive computing we have continuous on-line entry of input data and delivery of

output data. Interactive machines do not have input and output tapes but input and output ports. Interactive computing principally differs from computing over infinite streams in two ways. First, in interactive computing we only consider *potentially infinite streams*, i.e., streams that are always finite but can be prolonged without limit, with unpredictable next inputs at any time. We include port symbols denoting 'no input' and 'no output', as valid inputs or outputs in streams, respectively. Second, a *finite delay condition* [17] may be required during computation, asserting that after any non-empty input symbol a non-empty output symbol must be produced sometime. Any infinite input string that can be fed to the interactive machine properly in this way, is called a 'valid' input.

The previous conditions mean e.g. that internal and external phases of computing alternate, depending on whether an interactive device needs to do some finite computation before outputting a non-empty response symbol (or taking in a new non-empty input) or not. Unlike the scenario of classical machines performing infinite computations over infinite streams [10], interactive machines cannot answer questions requiring the processing of infinite streams 'in the limit' (such as 'is there a finite number of 1's in the given infinite stream?'). The interactive use of standard stand-alone PCs corresponds well to this view of interactive machines. The corresponding change in the Turing machine model leads to so-called *interactive Turing machines* (ITMs) introduced in [16]. The processing of infinite streams of symbols by ITMs leads to so-called 'interactively realizable translations' on valid (infinite) input strings, see [18] for formal details.

From the viewpoint of computability theory, interactive computing e.g. with ITMs does not lead to super-Turing computing power. Interactive computing merely extends our view of classically computable functions over finite domains to computable functions (translations) defined over infinite domains. Interactive computers simply compute something different from non-interactive ones because they follow a different scenario. Remembering the respective inputs over time, a finite computation of an interactive machine can always be replayed *a posteriori* by a non-interactive machine giving the same outputs as the interactive machine [17].

3 From Interactive to Evolving Computation

> *Every physical system registers information, and just by evolving in time, by doing its thing, it changes that information, transforms that information, or, if you like, processes that information.*
> S. Lloyd ([3], 2002).

Data interaction with computers differs from classical computing, as reflected in the change of computational scenario. Historically it allowed the use of computers in many more applications than before and thus it extends our apprehension of computing even though, from a computability point, interactive Turing machines cannot compute more, i.e. other mappings than non-interactive machines. A change in computational power can only come from a change of view on the

functioning of an isolated PC itself. In particular, does it make sense to change a computing device, or to let it change, *during* its computation?

Obviously, this feature could be especially useful in the case of interactive computing which potentially prolongs indefinitely: programs may be upgraded over time, and so can the hardware. For instance, the user could add more internal memory, upgrade the disk (while maintaining the original data), or exchange the processor for a newer one. One could couple it to several other computers, or connect it to a network like the *Internet*. Can changes like this be accommodated within the Turing machine paradigm? What happens, from a computational viewpoint?

This brings us back to a question posed in the beginning: can the Internet be simulated by a Turing machine? This is a difficult question, since it asks for comparing a piece of high-level computing and communication technology that exists in the real world with a highly simplified model of computation that exists in the abstract world. Therefore we will answer the question in two steps. In the first step, we propose an abstract model capturing the important features of the Internet (and as we will see later, those of many other computational gadgets in the sciences). In the second step, we compare this model with the classical Turing machine.

3.1 Modeling the Internet

What could an appropriate abstract model of the Internet be that has the same abstract simplicity as a Turing machine? The Internet has one important feature that we want to capture: *its structure evolves over time*. New sites are added to or deleted from the network all the time, possibly even connecting to it by means of wireless technologies. A 'site' can be anything: a workstation connected to the net via a cable, a notebook in an airplane, or a mobile phone. In order to capture all this variety in a simple model one must choose a fairly abstract viewpoint.

Consider the evolution of the Internet over time from its very beginning till now. Concentrate on the moments when it underwent some 'hardware changes' as mentioned above: a computer joined or left the network, or a computer on the net was upgraded. (We ignore cabling issues.) Between these moments of change, the structure of the Internet can be seen as stable. In these periods, one can view the entire Internet as a huge finite automaton, with finitely many input and output ports corresponding to all data entry and exit points (like keyboards, cameras, monitors, terminals, printers, and so on). In this automaton, the contents of the Internet is modeled by the (huge number of) states. Transitions between states correspond to operations taking place over the Internet, like changes in its contents by new inputs.

The automaton works in a 'parallel' interactive mode, receiving inputs through all its inputs ports and producing outputs over all its output ports. Of course, this abstraction neglects many other issues, like variable message transfer times. Allowing this simplification we go even farther: we merge all input streams into a single input stream while remembering the identity of the individual elements

(i.e., we can always say which element belongs to which individual stream) and do the same with the outputs. What we get is an equivalent finite automaton with a single input and a single output port which, in principle, computes the same transformation of input to output as the Internet did *in a period in which it had a stable structure*. Note that in the same way we can model any single computer over its lifetime with consecutive upgrades. In-between two consecutive periods, the structure of the net, and hence of the modeling finite automaton, is said to *evolve*.

3.2 Evolving Automata

In order to model the evolution of a computational system like the Internet over time, we consider the (ordered) *sequence* of finite automata corresponding to the successive stable periods. The notion of sequence has been used in computational complexity theory before in different contexts e.g. to capture the computational power of non-uniform families of circuits (cf. [1]). In a sequence of automata, the i-th automaton corresponds to the Internet contents and computations during the i-th stable period of the Internet. In the course of this time, only the i-th automaton receives input and produces output.

We have arrived at the following computational model called the *evolving automaton*, introduced in [16], [19]. (In [18] the model is called a 'lineage' of automata but we give a simplified formulation for expository reasons.)

Definition 1. *For $i = 1, 2, \ldots$, let A_i be a finite automaton with a single input and a single output port, let its alphabet be Σ_i, let S_i be the set of states of A_i, and let $\emptyset \neq Q_i \subseteq S_i$, be a set of 'preserving' states in A_i. Let $T_{\mathcal{A}} = t_1, t_2, \ldots$, with $t_i \in \mathbf{N}$, $t_1 = 1$ and $t_i < t_{i+1}$ be the sequence of switching times. The infinite sequence of finite automata $\mathcal{A} = A_1, A_2, \ldots$ is called an* evolving automaton *with schedule $T_{\mathcal{A}}$, or just an evolving automaton if $T_{\mathcal{A}}$ is understood, if $Q_i \subsetneq Q_{i+1}$, for $i = 1, 2, \ldots$ and the switching in processing from one automaton to the next takes place at the times given by $T_{\mathcal{A}}$.*

In the model, the condition $Q_i \subsetneq Q_{i+1}$ captures the persistence of the relevant data over time (cf. [6]). In the language of finite automata the condition ensures that some information available to A_i and represented in the states in Q_i, is available also to A_{i+1} after the 'change moment'. We require here that the transferred information can only grow, but this can easily be avoided by slightly modifying the definition, see [18]. The schedule $T_{\mathcal{A}}$ of an evolving automaton $\mathcal{A}$ determines the 'switching times' $t_i \in T_{\mathcal{A}}$ when the input stream to A_i must be redirected to A_{i+1}, in a state in Q_i.

An evolving automaton $\mathcal{A}$ clearly is an infinite object, given by an explicit enumeration of its elements. However, at each time the computation is performed by only one element of $\mathcal{A}$, which is a finite object. In general, there need not exist an algorithm for computing A_i given the previous elements in the sequence, the input and the schedule. A similar remark holds for the switching schedule; in general, its elements are non-computable from knowing $\mathcal{A}$ and the input sequence. Evolving automata are *non-uniform* systems just like families of circuits:

their development over time cannot be described by an algorithm. The Internet is a case in point: the decision to upgrade a computer or connect it to the Internet, depends entirely on the person owning the computer and has nothing to do with the computability.

3.3 Complexity of Evolving Automata

Given an evolving automaton $\mathcal{A} = A_1, A_2, \ldots$, it is natural to consider the number of states of the individual automata A_i as a measure for the complexity of $\mathcal{A}$. Define the *size complexity* of an evolving automaton $\mathcal{A}$ as the function g such that for every i, $g(i)$ is the number of states of A_i. Given this complexity measure, one can now try to categorize the translations realized by evolving automata $\mathcal{A}$ with any possible time schedule $T_{\mathcal{A}}$.

Definition 2. *A translation ϕ of infinite streams to infinite streams is said to be of complexity g if there is an evolving automaton of complexity g that realizes ϕ. For any function $g : \mathbf{N} \to \mathbf{N}$, let $SIZE(g)$ be the class of all translations ϕ that can be realized by an evolving automaton of complexity g.*

By the non-uniformity of the model, the classes $SIZE(g)$ will in general contain an abundance of non-computable translations, even though all of them will be 'non-uniformly realizable' by evolving automata within 'growth bound' g. A precise characterization of the translations that are non-uniformly realizable by evolving automata was given in [18,19]. The complexity measure is a realistic one, as indicated by the following result from [18,19].

Theorem 1. *Let $g, h : \mathbf{N} \to \mathbf{N}$ be positive non-decreasing functions such that $g(i) \leq h(i)$ for all i and $g(i) < h(i)$ for at least one i. Then $SIZE(g)$ is properly contained in $SIZE(h)$.*

In fact, in [18,19] it is shown that $SIZE(g)$ and $SIZE(h)$ differ whenever g and h do. Thus evolving automata have a fitting complexity theory, directly derived from the nature of the model.

3.4 Modeling the Internet 2

Finally, if one would want to build an evolving automaton simulating the existing Internet, then we could construct such an automaton only *a posteriori,* after watching the Internet's evolution and taking snapshots of it at the times of its changes, plus a recording of all input streams. The snapshots would then be used for constructing the sequence of automata which, on the recorded input streams, would produce the same translation as the Internet did. Of course, we are not seriously proposing to do it, it is only a *Gedankenexperiment,* serving as proof of principle.

Conversely, can some network simulate an evolving automaton $\mathcal{A} = A_1, A_2, \ldots$ with switching schedule $T_{\mathcal{A}} = t_1, t_2, \ldots$? Of course it can. To show it, we begin with a computer simulating A_1. At time t_1 we replace it by (or upgrade it to) a computer simulating A_2 and continue processing the inputs till time t_2, etc.

4 Two New Models of Computation

Evolving automata and Turing machines are both defined using the same formal language. This allows us to compare the computational power of both models.

Proposition 1. *Every classical Turing machine, or even an ITM, $\mathcal{T}$ can be simulated by an evolving automaton.*

Proof. (Sketch) Observe the computation of $\mathcal{T}$ on an input stream σ and note the times t_i when any of the Turing machine's heads moves past the 'next' rightmost symbol on its tape. These times define the switching schedule. Between times t_i and t_{i+1}, the computation of $\mathcal{T}$ can be modeled by a finite automaton A_i. This leads to a sequence of automata $\mathcal{A}$ and a schedule $T_{\mathcal{A}}$ computing the same translation as $\mathcal{T}$. □

The proposition establishes that computationally, evolving automata are at least as powerful as (interactive) Turing machines. Observe that, in order to simulate $\mathcal{T}$, the construction of $\mathcal{A}$ and that of the switching schedule depended, in a computable way, solely on $\mathcal{T}$ and on the input σ. Thus, $\mathcal{A}$ and $T_{\mathcal{A}}$ were computable from knowing $\mathcal{T}$ and σ. Note that in general, the definition of an evolving automaton does not require the latter to be the case. Thus, there seems to be some 'room' in the computational performance of evolving automata. Could they even simulate devices that are computationally more powerful than those modeled by ITMs?

As we shall see below, this is indeed the case: evolving automata are provably more powerful than Turing machines. Does it mean that the Turing machine is out of the game when looking for a new paradigm that captures the ideas of contemporary computing? Not entirely.

4.1 Computing with Advice

Rather than attempting a reverse simulation of evolving automata, let us try to simulate a yet more powerful model of a Turing machine by evolving automata: the so-called *interactive Turing machine with advice* (ITM/A). The model extends the well-known and well-studied model of (ordinary) Turing machines with advice in computational complexity theory (cf. [8]).

Definition 3. *An* interactive Turing machine with advice *(ITM/A) is an interactive Turing machine as described before, enhanced by an advice function $f : \mathbf{N} \to \Sigma^*$. Advice allows the insertion of external information $f(t)$ into the course of a computation at suitable times.*

A standard Turing machine with advice, with input of size n, is allowed to 'ask' for the value of its advice function only for that particular value of n. Similarly, an ITM/A can call its advice at time t only for values $t_1 \leq t$. To realize such

a call an ITM/A is equipped with a separate *advice tape* and among its states it has a distinguished *advice state*. By writing t_1 on the advice tape and by entering the advice state at time $t \geq t_1$ the value of $f(t_1)$ will appear on the advice tape (in a single step). By this action the original contents of the advice tape is completely rewritten. Note that the value of $f(t_1)$ does not depend on the input read before or after time t: the advice called at time t with argument $t_1 \leq t$ is the same for all input streams. This makes advice different from oracles also considered in the computability theory: oracle values can depend on the current input (cf. [13]).

The mechanism of advise functions is very powerful and can provide an ITM/A with any non-computable 'assistance'. For theoretical and practical reasons it is useful to restrict the size of advice growth in ITM/As to polynomial functions. With advice functions that grow exponentially one could encode arbitrary oracles in advice.

Proposition 2. *Evolving automata can simulate interactive Turing machines with advice and vice versa.*

Proof. (Sketch) First we sketch how an evolving automaton, $\mathcal{A}$, can simulate an ITM/A $\mathcal{O}$. Follow the given simulation of an ITM without advice, but now also consider the actions of $\mathcal{O}$ with its advice: include the times of calling $\mathcal{O}$'s advice in the schedule of switching times as well. At each switching moment, the respective automaton will also encode the corresponding advice in its states. Note that now the members of $\mathcal{A}$ cannot be computed solely from knowing σ and $\mathcal{O}$ as before. This time, we also have to know the advice at each calling time. Note that the automaton sizes in $\mathcal{A}$ grow proportionally with the space complexity and the advice size of the $\mathcal{O}$ in the simulated time segment.

The reverse simulation by means of an ITM/A is easy. An ITM/A $\mathcal{O}$ is supplied with the description of $\mathcal{A}$'s members and the times of $T_{\mathcal{A}}$ 'on demand', via its advice tape. The computation of $\mathcal{O}$ on input σ starts by calling the advice. $\mathcal{O}$ gets the description of A_1 followed by the value of t_1. All $\mathcal{O}$ has to do is to simulate A_1 on the next t_1 input symbols. Then $\mathcal{O}$ calls its advice again, obtaining description of A_2 and the value of t_2, and $\mathcal{O}$ simulates A_2 for the next $t_2 - t_1$ steps. Then the process of calling advice repeats again, etc. Now the space complexity of $\mathcal{O}$ grows as fast as the automata size in $\mathcal{A}$. □

As a corollary we obtain that *the Internet, modeled by an evolving automaton, can be simulated by an interactive Turing machine with advice.* By this, we have finished the second step of our plan: we have identified a model which is an extension of Turing machines and whose computational power matches exactly that of a highly simplified model of the (unrestricted) Internet.

4.2 Complexity of ITM/As

As for evolving automata we consider the question whether ITM/As admit an 'own' complexity theory.

Proposition 3. *ITM/As are more powerful than ITMs (without advice).*

Proof. (Sketch) We begin by exhibiting a translation κ that can be realized by an ITM/A, but not by any ITM without an advice. The computation will ask for solving the halting problem (known to be undecidable) for all classical Turing machines.

As input stream, we consider a computable enumeration of all Turing machines. Given this enumeration, κ should output with each valid machine description a 1 if and only if this machine accepts its own description, and 0 otherwise. We construct an ITM/A $\mathcal{I}$ that does this. In-between producing 0s or 1s, $\mathcal{I}$ will output only empty symbols.

$\mathcal{I}$ enumerates all TMs in the same order as they occur in the input stream. Then $\mathcal{I}$ can recognize whether a segment of the input stream is indeed a valid encoding of a TM. On segments that are not encodings of a TM, $\mathcal{I}$ produces 0. On a segment w that is an encoding of length n of some TM, $\mathcal{I}$ calls its advice with value n (note that the advice is called for a value which does not depend on the particular input read thus far). The advice gives the encoding $\langle M \rangle$ of a TM whose running time is the longest from among the running time of all TMs of size n that terminate on their own description. Running this machine on input $\langle M \rangle$ in parallel with the simulation of the w, $\mathcal{I}$ has an upper bound on the running time on w within which the machine must halt on its own description when it does. In this way $\mathcal{I}$ can correctly answer the halting problem for w in finite time, and proceed with the next segment of the input.

Now we sketch that no ITM without advice can solve the halting problem. Suppose there was an ITM $\mathcal{H}$ computing κ. Obviously, due to the properties of interactive machines, $\mathcal{H}$ should produce the answer to any particular halting problem in finite time. Thus, if we were interested in solving only a particular halting problem it would be enough to run a classical TM simulating $\mathcal{H}$ until it produces, in finite time, the solution of our decision problem. This contradicts the undecidability of the halting problem by classical TMs. □

Given an ITM/A with advice function f, it is natural to consider the size of the advice $f(t)$ for each individual value of t as a measure for the 'complexity' of the ITM/A (in addition to the usual measures of time and space for the Turing machine part). Define the *advice complexity* of an ITM/A with advice function f as the function $\alpha : \mathbf{N} \to \mathbf{N}$ such that for every t, $\alpha(t) = |f(t)|$ (the length of the string $f(t)$). Given this measure one can try to distinguish between the computational power of different ITM/As. For example, Verbaan [18] proved the following interesting result, extending a similar result known for ordinary Turing machines with advice.

Theorem 2. *Consider ITM/As over input and advice alphabets with a fixed size bound b. Let α and β be integer-valued functions such that $\alpha = o(\beta)$ and $\beta(t) \leq \frac{b^t}{\log b}$ for all t. Then there is a translation ϕ of infinite streams to infinite streams that can be realized by an ITM/A of advice complexity β, but not by any ITM/A of advice complexity α', for any function α' with $\alpha'(t) \leq \alpha(t)$ for all but finitely many t.*

In fact, as soon as α is strictly 'below' β for all but finitely many values of t, ITM/As of advice complexity β are more powerful than ITM/As of advice complexity α [18].

The computational equivalence between evolving automata and ITM/As also opens the question whether their complexity theories can be linked. An example of a result in this direction is the following.

Theorem 3. *Let ϕ be a translation of infinite streams to infinite streams. Let ϕ be realizable by an evolving automaton of size complexity g. Then ϕ can be realized by an ITM/A of advice complexity $O(g \log g)$ and space complexity $O(\log g)$.*

It follows e.g. that evolving automata of polynomially bounded size complexity can be simulated by an ITM/A of polynomially bounded advice complexity and logarithmic space. The converse result can be shown as well [18].

5 Extending the Turing Machine Paradigm

> *[· · ·] a comprehensive theory of computation must reflect in a stylized way aspects of the underlying physical world.*
> T. Toffoli ([11], 1982).

In our search for a new computational model, we have presented three important insights:

(i) we have devised a model of evolving automata capturing interactive and non-uniformly evolving computing,
(ii) we have shown the computational equivalence of evolving automata and interactive Turing machines with advice and, last but not least,
(iii) we have shown that these two models are computationally more powerful than interactive Turing machines (without advice) which only capture interaction.

This leads to the new computational paradigm that we have in mind:

> **Extended Turing Machine paradigm:** *A computational process is any process whose evolution over time can be captured by evolving automata or, equivalently, by interactive Turing machines with advice.*

The new paradigm represents a new understanding of computing, motivated by developments like the Internet and even by the computational views of living systems. It innovates the classical view of computing in three ways: a shift from finite computations to potentially infinite interactive ones, a shift from rigid computing systems towards systems whose architecture and functionality evolve over time and, last but not least, an understanding that in general the latter process of evolution happens in an unpredictable, non-uniform, non-computable way.

In our view, both models mentioned in the extended paradigm, the ITM/A and evolving automata, have their use. Together they illustrate the dual view of

a non-computable evolution. The metaphor of an ITM/A corresponds better to our intuition and experience in which computers are perceived as well engineered devices with a fixed architecture driven solely by input data, now with their 'evolution' driven by data as well (namely by those from the advice). In this way, an TM/A models non-uniform software evolution. On the other hand, the metaphor of an evolving automaton models a hardware evolution. This makes this model more suitable for modeling systems where a non-uniform hardware evolution is readily visible (as was the case of the Internet). Of course, both models are only different sides of the same coin.

5.1 Non-computability Issues in the Extended Paradigm

The new paradigm indirectly asserts that the 'new computing' is computationally more powerful than classical computing, since the models of computing serving in the new paradigm are provably computationally more powerful than those in the old paradigm. Does it mean that the new paradigm encompasses some form of super-Turing computing capability, and if so, can the extra power be used for 'solving' undecidable problems?

Of course the answer to both questions is negative, as seen from the proof of Proposition 3. In order to solve the halting problem, the constructed ITM/A had to be provided with non-computable information. In general, an ITM/A can solve classically undecidable problems if and only if its advice contains the respective non-computable information. In the proof we were not interested in how this information could be obtained, we just made use of the fact that such information in principle exists. Thus, there is nothing miraculous in our result: if a device has non-computable information at its disposal, it can solve non-computable tasks. This has been known since Turing's times (cf. [13], [2]).

In 'real' computational environments (such as in the Internet), the non-computability manifests itself, e.g., as the non-predictability of their evolution or in the unpredictable variance in message transfer times among the systems part. We do not know of any computational exploitation of these phenomena (except, perhaps, as a source of random numbers). In fact, in most of our computing activities we strive for being shielded from these phenomena. Hence, the hyper-Turing power implied by the extended paradigm is needed for the purposes of theoretical modeling, but, unfortunately, cannot be purposefully harnessed for any goal-oriented computational purposes.

5.2 The Scope of the Extended Paradigm

What remains is to see whether the other systems, man-made or natural, mentioned in the introduction as examples of systems 'that process information and compute' are covered by the extended paradigm. No doubt that, once we agree that the models in the paradigm capture the Internet, then they also capture all variations of this theme: wireless ad hoc networks, sensor nets, etcetera. Generalizing, one can say that to the extent to which finite automata mirror the data-processing capability of some entity (such as that of a biological cell or of

a biological neuron), the extended paradigm also mirrors the data-processing capabilities and computations of ensembles (such as organisms or brains) and communities of such entities (such as swarms of ants or bees, or also communities of humans). Cf. [22] for a more in-depth, complexity-oriented study of such an approach.

The case of physical systems in general, and especially of the Universe itself, is interesting. Apparently, there are 'no external inputs' to the Universe. A current state of the universe completely determines its next state (albeit not in the deterministic way, as in the case of deterministic finite automata). Or, to quote Toffoli [11]: *In a sense, nature has been continually computing the 'next state' of the universe for billions of years; all we have to do - and actually, all we* can *do - is 'hitch a ride' on this huge ongoing computation.* What we 'observe' is the potentially infinite sequence of instances of the Universe. Could this be modeled by a kind of a gigantic, 'natural' evolving automaton (perhaps a quantum automaton?) whose evolution is governed by the laws of the Nature? According to Lloyd [3], the Universe computes its own evolution. This seems to be close to the spirit of our paradigm.

6 Conclusions

The contemporary perception of computing sees it as any act of information processing and transfer, occurring in both the local and global behavior of systems. In this view, computation encompasses communication, interaction, reaction, receiving, sending, storing, retrieving and transformation of information. The Extended Turing Machine paradigm captures it in an abstract manner.

The extended paradigm keeps the central position of Turing machines in our apprehension of computing, continuing in this way the tribute to A.M. Turing. In fact, the new paradigm also makes use of the language of classical Turing machines, upgraded this time, by the notions of interaction and advice. The new paradigm also encompasses non-uniform computing, which seems to be far more ubiquitous and less artificial than believed before. The complementary view of interactive Turing machines as that of evolving automata stresses the dual sides of both software and hardware evolution. In addition to the known cases of computing artifacts, the extended paradigm also covers the information processing occurring in the Nature. For informal use, there is no need to formally revise the good old Turing machine paradigm. What is needed is to be more liberal in understanding different variants of Turing machines and their scenarios. This was also concluded in a debate on Lance Fortnow's weblog: [5].

> *Call me a rationalist then as I continue to hold the belief that no matter how complicated the computational model, we can still use the simple Turing machine to capture its power.*
> L. Fortnow ([5], 2006).

There is some advantage in having the paradigms of science formulated in not very precise terms. Namely, in such a case, their rejection requires a real

revolution to happen in the field. Otherwise, let our paradigms evolve along with the evolution of the notions they deal with. But it is good to know that when it comes to the details, we are able to make our paradigms more precise.

References

1. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity I. 2nd edn. Springer, Berlin (1995)
2. Davis, M.: The myth of hypercomputation. In: Teuscher, C. (ed.) Alan Turing: Life and Legacy of a Great Thinker, pp. 195–212. Springer, Heidelberg (2004)
3. Edge, The Computational Universe: Seth Lloyd [10.24.02] (October 24, 2002), http://www.edge.org/3rd_culture/lloyd2/lloyd2_index.html
4. Etesi, G., Németi, I.: Turing computability and Malament-Hogarth spacetimes. International Journal of Theoretical Physics 41(2), 342–370 (2002), http://arxiv.org/abs/gr-qc/0104023
5. Fortnow, L.: Principles of problem solving: A TCS Response, weblog Computational Complexity, Friday (July 14, 2006), http://weblog.fortnow.com/2006/07/principles-of-problem-solving-tcs.html
6. Goldin, D.Q., Smolka, S.A., Attie, P.C., Sonderegger, E.: Turing machines, transition systems, and interaction. Information and Computation 194(2), 101–128 (2004)
7. Goldin, D.Q., Smolka, S., Wegner, P. (eds.): Interactive Computing: The New Paradigm. Springer, Berlin (2006)
8. Karp, R.M., Lipton, R.: Turing machines that take advice, L'Enseignement Mathématique, II^e Série, Tome XXVIII, pp. 191–209 (1982)
9. Lloyd, S.: The Computational Universe. Originally published on Edge, (October 24, 2002), http://www.edge.org/3rd_culture/lloyd2/lloyd2_p2.html
10. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Formal Models and Semantics, vol.B, ch. 4, pp. 133–192. Elsevier Science Publishers, Amsterdam (1990)
11. Toffoli, T.: Physics and computation. Int. Journal of Theor. Physics 21, 165–175 (1982)
12. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. Series 2 42, 230–265 (1936)
13. Turing, A.M.: Systems of logic based on ordinals. Proc. London Math. Soc. Series 2 45, 161–228 (1939)
14. van Leeuwen, J., Wiedermann, J.: The Turing machine paradigm in contemporary computing. In: Enquist, B., Schmidt, W. (eds.) Mathematics Unlimited - 2001 and Beyond, pp. 1139–1155. Springer, Berlin (2001)
15. van Leeuwen, J., Wiedermann, J.: A computational model of interaction in embedded systems, Technical Report UU-CS-2001-02, Dept.of Information and Computing Sciences, Utrecht University (2001)
16. van Leeuwen, J., Wiedermann, J.: Beyond the Turing limit: Evolving interactive systems. In: Pacholski, L., Ružička, P. (eds.) SOFSEM 2001: Theory and Practice of Informatics. LNCS, vol. 2234, pp. 90–109. Springer, Heidelberg (2001)
17. van Leeuwen, J., Wiedermann, J.: A Theory of Interactive Computation. In: Goldin, D., Smolka, S., Wegner, P. (eds.) Interactive Computing: The New Paradigm, ch. 6, pp. 119–142. Springer, Berlin (2006)

18. Verbaan, P.R.A.: The Computational Complexity of Evolving Systems, Ph.D.Thesis, Dept.of Information and Computing Sciences, Utrecht University (2006)
19. Verbaan, P.R.A., van Leeuwen, J., Wiedermann, J.: Complexity of evolving interactive systems. In: Karhumäki, J., et al. (eds.) Theory Is Forever. LNCS, vol. 3113, pp. 268–281. Springer, Berlin (2004)
20. Wegner, P.: Why interaction is more powerful than algorithms. C. ACM 40, 315–351 (1997)
21. Wiedermann, J., van Leeuwen, J.: Relativistic computers and non-uniform complexity theory. In: Calude, C., et al. (eds.) UMC 2002. LNCS, vol. 2509, pp. 287–299. Springer, Heidelberg (2002)
22. Wiedermann, J., van Leeuwen, J.: The emergent computational potential of evolving artificial living systems. AI Communications 15(4), 205–215 (2002)
23. Winograd, T.: From computing machinery to interaction design. In: Denning, P., Metcalfe, R. (eds.) Beyond Calculation: The Next Fifty Years of Computing, pp. 149–162. Springer, Berlin (1997), `http://hci.stanford.edu/~winograd/acm97.html`