Beyond the Turing Limit: Evolving Interactive Systems

Wiedermann, Jiří
2001

Institute of Computer Science
Academy of Sciences of the Czech Republic

Beyond the Turing Limit: Evolving Interactive Systems

Jiří Wiedermann and Jan van Leeuwen

Technical report No. 843

September 2001

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +4202 6605 3520, fax: (+4202) 858 57 89,
e-mail:jiri.wiedermann@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Beyond the Turing Limit: Evolving Interactive Systems[1]

Jiří Wiedermann[2] and Jan van Leeuwen[3]

Technical report No. 843

September 2001

Abstract:

Modern networked computing systems follow scenarios that differ from those modeled by classical Turing machines. For example, their architecture and functionality may change over time as components enter or disappear. Also, as a rule their components interact with each other and with the environment at unpredictable times and in unpredictable manners, and they evolve in ways that are not pre-programmed. Finally, although the life span of the individual components may be finite, the life span of the systems as a whole is practically unlimited. The examples range from families of cognitive automata to (models of) the Internet and to communities of intelligent communicating agents.

We present several models for describing the computational behaviour of evolving interactive systems, in order to characterize their computational power and efficiency. The analysis leads to new models of computation, including 'interactive' Turing machines (ITM's) with advice and new, natural characterizations of non-uniform complexity classes. We will argue that ITM's with advice can serve as an adequate reference model for capturing the essence of computations by evolving interactive systems, showing that 'in theory' the latter are provably more powerful than classical systems.

Keywords:
Church-Turing thesis, interactive computing, interactive Turing machines with advice, web computing

[1]This research was partially supported by GA ČR grant No. 201/00/1489 and by EC Contract IST-1999-14186(Project ALCOM-FT).

[2]Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic

[3]Institute of Information and Computing Sciences, Utrecht University, Padualaan 14, 3584 CH Utrecht, the Netherlands, email: jan@cs.uu.nl

1 Introduction

In the twentieth century, computability theory explored the limits of what can be digitally computed. The prominent claim, known as the Church-Turing thesis, asserts that every *algorithm* can be captured in terms of a standard Turing machine. The classical computing scenario consists of a fixed program, a finite input supplied in either off-line or online mode, and a meaningful result only if the computation halts in finite time. No changes to the program of the machine or its 'architecture' are allowed in the meantime, intermediate results cannot influence the input and no information is carried over to future runs of the machine. This even applies in the case of ω-Turing machines.

Does this correspond to the way modern networked computers operate? Clearly it does not. Today's systems operate practically uninterruptedly since the moment of their installation. They obtain their inputs via many different channels at unpredictable times, deliver corresponding responses continuously at times when they are ready, accumulate information over the course of their entire existence and use it across the boundaries of separate 'runs'. Also, parts of their underlying hardware and software are updated, whenever an 'external agent' that operates some component decides to do so, without loss of vital data.

One can object, as many people do when confronted with this observation, that this use of computers is simply different from the way assumed in the Church-Turing thesis, and that this change is insignificant and can be easily accommodated by adjusting the original model. While the latter is true, the former is questionable: is it really an insignificant change? Answering this question will be a main concern in this paper. It will appear that at least in theory, the traditional notion of algorithmic computation must be extended for it.

Compared to the classical computing scenario, the essence of the changes in modern computing technology we have in mind can be subsumed under three complementary issues: *interactivity*, *non-uniform evolution* (and adaptivity), and *infinity of operation*. Interactivity is often also called 'reactivity' [3]. The systems performing according to these three qualities together constitute what is meant by *evolving interactive computing*.

The most prominent example is the Internet. It can be seen as a wide-area computing infrastructure, a kind of global computer ([4]). As a 'computer', the Internet is hindered by diverse administrative, architectural, and physical constraints. Worse even, it is 'undesigned', evolving unpredictably, with unpredictable computing characteristics. Many people, especially in the software engineering community (cf. [4], [16]), noticed that we are facing a new computing phenomenon that does not fit the classical Turing machine paradigm. For instance, Cardelli [4] writes:

> *'In order to program a global computer we first need to understand its model of computation. For example, does computation on the Web correspond naturally to a traditional model? There are indications that it does not. [For example] when browsing, we actively observe the reliability and bandwidth of certain connections (including zero or time-varying bandwidth), and we take action on these dynamic quality-of-service observables. These observables are not part of traditional models of computation, and are not handled by traditional languages. What models of computation and programming constructs can we develop to automate behavior based on such observables?'*

In this paper we will concentrate only on the first part of Cardelli's question, calling for models of computation that capture the essence of global computing. It will lead us to the concepts of non-uniform computation and to a new approach to several non-uniform complexity classes.

Wegner [16, 17] went even further, by claiming that interactivity alone can lead to computations that are more powerful than computations by Turing machines. In other words, 'interactive computing' would violate the Church-Turing thesis. We will argue that interactivity alone is not sufficient to break the Turing barrier. Interactivity merely extends the objects that one computes on: from finite strings to infinite ones, and the feedback mechanism remains computable. The computing power of a system could go beyond that of classical computing if *non-uniformity* is considered. Non-uniformity enters when one allows 'evolving' changes of the underlying hardware and/or software in the course of uninterrupted computing. This is a standard case with the Internet.

Surprisingly, evolving interactive computing seems to pervade not only the current, highly networked information processing systems but also – and mainly so – the information processing in

(societies of) living organisms. It is a 'technology' that has been invented by nature long ago that has very much the same characteristics as we described. Note that evolution in our setting is fundamentally different from the notion of learning that is often mentioned in connection with interactivity. Learning is usually understood as a software evolution, whereas we will also and especially consider hardware evolution.

The structure of the paper is as follows. In Section 2 we introduce an elementary, and therefore fundamental, tool for dealing with interactive systems: the interactive finite automaton. In Section 3 we introduce sequences of interactive finite automata that share global states, leading us to the model of evolving interactive systems that we have in mind. Next, in Section 4 we describe the basic interactive Turing machine (ITM) that will serve as a platform for the design of its non-uniform variants. In Section 5 we define the ITM with advice, in Section 6 the so-called site machine that models a site in a computer network, and finally in Section 7 we present the web Turing machine – a model of the Internet. Then, in Section 8 we prove the computational equivalence of the non-uniform models. In Section 9 we investigate the efficiency of web Turing machine computations in more detail, and show that these machines belong to the most efficient computational devices known in complexity theory. Finally, in Section 10 we will discuss some interesting issues related to our results.

Most of the results mentioned here can be found in more detail in the original papers [12, 14, 13, 15, 20]. The present paper primarily outlines the overall research framework. The notion of sequences of interactive finite automata with global states and the respective results are new.

2 Interactive finite automata

Under the classical scenario, finite automata are used for recognizing finite strings. Under the interactive scenario, we consider *interactive finite automata* (IFA) which are a generalization of Mealy automata. They process potentially infinite strings (called streams) of input symbols and produce a potentially infinite stream of output symbols, symbol after symbol. To stress the interactiveness, we assume that there is no input tape: the automaton reads the input stream via a single input port. Likewise, it produces the output via a single output port. There is no way to return to symbols once read except when they're stored internally. We assume throughout that the input and output symbols are taken from the alphabet $\Sigma = \{0, 1, \lambda\}$. Symbol λ at a port means that 'presently, there is neither 0 nor 1 appearing at this port'. The steps of an IFA follow a finite, Mealy-type transition function.

Any IFA realizes a translation ϕ that transforms infinite input streams over Σ into similar output streams. The λ's are not suppressed in the translation. Clearly, instead of an IFA one could consider any other device that is capable of entering into only a finite number of different configurations, such as discrete neural (cf. [8]) or neuroidal (cf. [10]) nets, neuromata [9], combinatorial circuits (cf. [2]), and so on. From [2, 19] the next theorem follows:

Theorem 1 *For translations $\phi : \Sigma^\omega \to \Sigma^\omega$ the following are equivalent:*

(a) ϕ is realized by a interactive finite (Mealy) automaton.

(b) ϕ is realized by a neuroid.

(c) ϕ is realized by a discrete neural net.

(d) ϕ is realized by a discrete neuroidal net.

(e) ϕ is realized by a combinatorial circuit.

In the theorem, all respective devices are assumed to work in an interactive mode in processing infinite input streams. Devices such as neural nets or combinatorial circuits that read their input in parallel, process the input stream in blocks that correspond to the number of their input ports.

IFA's embody two features of evolving interactive computing: interactivity, and infinity of operation. The interactivity enables one to describe (albeit *a posteriori*) the interaction between the machine and its environment: inputs succeeding to some outputs may be reactions to these outputs.

Note that we did not yet impose the third desideratum of evolving interactive computing: the evolvability of the underlying computing mechanism. Of course, due to their simplicity, IFA's do not have universal computing power either. Note that the motivation and computational scenario of IFA's differ from those usually considered for ω-automata (cf. [12] or [13]).

3 Sequences of interactive finite automata with global states

In order to achieve universal computing power and support the evolvability property, we consider *sequences* of IFA's. This enables us to realize more complicated translations and will also reveal the dependence of computational efficiency on the size of the underlying devices. The approach is inspired by the similar practice in non-uniform complexity theory where e.g. sequences (or families) of combinatorial circuits are considered (cf. [2, 6]). Our approach differs not only in the use of different 'building blocks' – namely IFA's instead of combinatorial circuits, but also in the use of a communication mechanism between neighboring members in a sequence. Both changes are motivated by the need to accommodate all ingredients of evolving interactive computing.

Definition 1 *Let $\mathcal{A} = \{A_1, A_2, \ldots\}$ be a sequence of IFA's over Σ, and let Q_i be the set of states of A_i. Let $G = \{G_1, G_2, \ldots\}$ be a sequence of nonempty finite sets such that $G_i \subset Q_i$ and $G_i \subseteq G_{i+1}$. Then $\mathcal{A}$ with G is called a* sequence of IFA's with global states*.*

For a sequence $\mathcal{A}$, there need not exist an algorithmic way to compute the description of the A_i, given i. Thus, the only way to describe the sequence may be to enumerate all its members. The set $\bigcup_i G_i$ is called *the set of global states*. From now on we always assume sequences of IFA's to have global states.

On an infinite input stream over Σ, a sequence $\mathcal{A}$ computes as follows. At the start, A_1 is the active automaton. It reads input and produces output for a while, until it passes control to A_2. In general, if A_i is the current active automaton, it performs its computation using the *local states* from the set $Q_i - G_i \neq \emptyset$. If an input symbol causes A_i to enter a global state $g \in G_i$, then A_i stops processing and passes control to A_{i+1}. The input stream is redirected to the input port of A_{i+1}, A_{i+1} enters state $g \in G_{i+1}$ and continues processing the input stream as the new active automaton, starting with the next input symbol.

Thus, in effect the input stream is processed by automata with increasing index. This models the property of evolution. The 'transfer' of control to the next automaton is invoked by the automaton currently processing the input. The next automaton continues from the same state in which the previous automaton stopped. This mechanism enables the transfer of information from the previous stage. In a sequence of IFA's with global states the next automaton can be seen as a 'next generation' machine. Note that in finite time only a finite part of a sequence of IFA's can have become active.

Alternatively, instead of a sequence of automata, one may consider a single automaton that 'evolves' so at any time it acts as A_i iff $A_i \in \mathcal{A}$ is the currently active automaton. That is, the transition function of the automaton at hand is the same as that of A_i as long as A_i is active. Of course, the condition concerning the global states must still be maintained. The resulting automaton may appropriately be called an *evolving interactive finite automaton*.

A sequence of IFA's is called *polynomially bounded* iff there is a polynomial p such that for every $i \geq 1$, the size of A_i is at most $p(i)$. The classes of translations realized by sequences of IFA's with global states and polynomially and exponentially bounded size will be denoted as IFA-POLY and IFA-EXP, respectively. We will also consider the classes NA-LOG (the translations realized by sequences of neuromata [8] of logarithmic size), NN-POLY (the translations realized by sequences of standard recurrent, or cyclic, discrete neural nets of polynomial size reading their inputs in parallel), and CC-POLY (the translations realized by sequences of combinatorial circuits with a polynomial number of gates).

4 Interactive Turing machines

The next tool we consider are interactive Turing machines (ITM's). ITM's differ from standard TM's in one important aspect: they allow for a infinite, never ending exchange of data with their environment.

An ITM reads input from its *input port* and produces output at its *output port*. We assume that in each step the machine reads a symbol from its input port and writes a symbol to its output port. As before, the input and output symbols are taken from the alphabet $\Sigma = \{0, 1, \lambda\}$. We will normally require that an ITM reacts to any non-empty input by producing a non-empty output symbol at its output port after at most some finite time (the *interactiveness* or *finite delay condition*). The finite-delay condition ensures that if an input stream has an infinite number of non-empty symbols, then there must be an infinite number of non-empty symbols in the output stream.

Definition 2 *A mapping $\phi : \Sigma^\omega \to \Sigma^\omega$ is called the* interactive translation *computed by an ITM $\mathcal{I}$ iff for all $\mathbf{x}$ and $\mathbf{y}$, $\phi(\mathbf{x}) = \mathbf{y}$ if and only if $\mathcal{I}$ produces $\mathbf{y}$ on input $\mathbf{x}$.*

As input streams have infinite length, complexity measures for ITM's cannot be defined in terms of the total amount of resources used for processing the entire input, as the resulting values will generally be infinite as well. Therefore we measure the pace of growth of the resource utilizations, as a function of the length of the input stream processed so far.

Definition 3 *We say that for a given input stream the ITM $\mathcal{I}$ is of* space complexity $S(t)$ *iff for any $t > 0$, after processing t input symbols from the given stream no more than $S(t)$ cells on $\mathcal{I}$'s (internal) tapes were ever needed. If this condition holds for every input stream, then we say that the ITM $\mathcal{I}$ is of space complexity $S(t)$.*

The definition of time complexity is more involved. This is so because, as long as empty symbols are counted as legal symbols in input and output streams, any initial segment of a computation by an ITM is of linear time complexity w.r.t. the input read thus far. Yet it is intuitively clear that computing some non-empty output symbols can take more than one step and that in the meantime empty, or other 'ready-made' symbols must have been produced. We will measure the complexity of producing non-empty output symbols from a given input stream, at concrete times, by the *reaction time*. For $t \leq j$, we say that the j-th output *depends* on the input prefix of length t if and only if any change of the $(t + 1)$-st and later input symbols cause no change of the output up to and including the j-th symbol. The value j is a (lower)bound to the reaction time for the prefix.

Definition 4 *We say that for a given input stream the ITM $\mathcal{I}$ is of* reaction time complexity $T(t)$ *iff the reaction time of $\mathcal{I}$ to the input prefix of length t is (upper-)bounded by $T(t)$, for any $t > 0$. If this condition holds for any input stream, then we say that the ITM $\mathcal{I}$ is of reaction time complexity $T(t)$.*

ITM's alone do not lead to a non-recursive computational power. Allowing ITM's to process infinite input streams only extends the operational scope, compared to classical TM's which process merely finite streams. For further details see [12, 13].

5 Interactive Turing machines with advice

Next we introduce *interactive Turing machines with advice* (ITM/A's). An ITM/A is an ITM as described above, enhanced by an advice (cf. [6, 2]). Advice functions allow the insertion of external information into the course of a computation, in this way leading to a non-uniform operation.

Definition 5 *An advice function is a function $f : \mathbf{Z}^+ \to \Sigma^*$. An advice is called $S(n)$-bounded if for all n, the length of $f(n)$ is bounded by $S(n)$.*

A standard TM with advice and input of size n, is allowed to call for the value of its advice function only for this particular n. 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 a distinguished *advice state*. By

writing the value of the argument t_1 on the advice tape and by entering into 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 overwritten.

We will be interested in advice functions whose values are bounded in length by known (computable) functions of t, especially in polynomially or logarithmically bounded functions. Note that the mechanism of advice is very powerful and can provide an ITM/A with highly non-recursive 'assistance'.

The complexity measures for ITM/A's are defined as for ITM's without advice (see Definitions 3 and 4). The length of the rewritten part of the advice tape is counted in the space complexity of the respective machine, not including the actual read-only advice value.

Definition 6 *The class $ITM - \mathcal{C}/\mathcal{F}$ consists of the translations ϕ computed by $ITM - \mathcal{C}$ machines using an advice function from $\mathcal{F}$.*

Common choices for $ITM - \mathcal{C}$ that we shall use are: $ITM - LOGSPACE$ (deterministic logarithmic space), $ITM - PTIME$ (deterministic polynomial time), and $ITM - PSPACE$ (polynomial space). Common choices for $\mathcal{F}$ are *log* (logarithmically bounded advice functions) and *poly* (polynomially bounded advice functions).

For completeness we show that ITM's with advice are indeed more powerful than ITM's without advice. (The result also follows from a countability argument.) Care must be taken that the finite-delay condition is correctly observed.

Consider the translation κ defined as follows. As a special input, we first consider the string consisting of the infinite enumeration of all Turing machine descriptions, in blocks of non-decreasing size. For this input, the translation should assign to each machine description a '1' if and only if the machine at hand accepts its own description, and '0' otherwise. If the input is not of this form and starts to differ at some point from the blocked form described, then κ is assumed to work as described up until the last complete encoding in the sequence, and then copy every (empty or nonempty) symbol that follows after this.

Lemma 1 *Translation κ can be realized by an ITM/A, but there is no ITM (without advice) that can realize κ.*

Proof (sketch): Define the function f that to each n assigns the description of the 'busy beaver machine' of length n. The busy beaver machine is a Turing machine which, among all machines with encodings of length n, performs the maximum number of steps before halting, on an input that is equal to its own description. If no machine description of size n exists, then $f(n)$ is assigned the empty string.

Now design an ITM $\mathcal{A}$ using advice f as follows. $\mathcal{A}$ checks every time whether the input stream contains a 'next' block as expected, i.e. a next machine description. If the next input segment is not a block as expected, $\mathcal{A}$ will know within finite time that this is the case (because the blocks must come ordered by size). If the next input segment is not a valid encoding, $\mathcal{A}$ copies the segment to output and then copies every (empty or nonempty) symbol that follows after this. This is consistent with κ and satisfies the finite-delay condition.

If the input stream presents $\mathcal{A}$ with a next block that is the valid description w of a Turing machine M, $\mathcal{A}$ works as follows. Let $|w| = n$. $\mathcal{A}$ calls its advice for value n and gets the description $\langle B \rangle$ of length n of the respective busy beaver machine B. Now $\mathcal{A}$ alternately simulates one step of M on input w, and one step of B on input $\langle B \rangle$. Under this arrangement, one of the two simulations must halt as the first one. If it is the simulation of M that halts then $\mathcal{A}$ 'accepts' w, i.e. $\mathcal{A}$ outputs 1. Otherwise, $\mathcal{A}$ outputs 0. Thus, $\mathcal{A}$ realizes κ and, as it satisfies the finite-delay condition in all cases, it is an ITM.

The second part of the lemma is proved by using a modification of the standard diagonal argument. For details, see [15]. □

The lemma serves as a means for proving the super-Turing computing power of machines that are computationally equivalent to ITM/A's. As a first result of this kind we prove that sequences of

IFA's with global states are equivalent to ITM/A's, implying that the former also posses super-Turing computing power.

Theorem 2 *For translations $\phi : \Sigma^\omega \to \Sigma^\omega$, the following are equivalent:*

(a) ϕ is computed by a polynomially bounded sequence $\mathcal{A}$ of IFA's with global states.

(b) ϕ is computed by a logarithmically space-bounded ITM/A $\mathcal{M}$ with polynomially bounded advice.

Proof (sketch): $(a) \to (b)$. Let ϕ be computed by $\mathcal{A}$. Simulate the action of $\mathcal{A}$ step by step using an ITM $\mathcal{M}$, with the following advice function. It will be invoked each time when an automaton A_i in $\mathcal{A}$ currently reading the input enters a global state $g \in G_i$. At this time the advice function returns the description of A_{i+1} and $\mathcal{M}$ proceeds with the simulation of $\mathcal{A}$ by simulating A_{i+1} starting from state $g \in G_{i+1}$. To simulate A_{i+1}, logarithmic space is enough for $\mathcal{M}$ since all it has to store is a pointer to the advice tape remembering the current state of A_{i+1}.

$(b) \to (a)$. For the reverse simulation, split the input string into blocks of size 1,2,3, $\ldots, i, \ldots$ and consider first the case when $\mathcal{M}$ does not call its advice. Sequence $\mathcal{A}$ will be designed so the i-th automaton in the sequence can 'continue' the processing of the i-th block, making use of its local states. Each automaton A_i is designed as a simulator of $\mathcal{M}$ on input prefixes of lengths $\ell_i = 1 + 2 + 3 + \ldots + i = O(i^2)$. On these prefixes $\mathcal{M}$ needs $O(\log i)$ space and therefore can enter $O(p(i))$ different states, for some polynomial p. This design warrants not only that A_i has enough states to represent each configuration of $\mathcal{M}$ on an input prefix of length $O(\ell_i)$, but it also enables the 'information transfer', via the corresponding global states, from A_i to A_{i+1} prior to the time when the 'storage capacity' of A_i gets exhausted. Hence, $\mathcal{A}$ can remain polynomially bounded for this case.

Consider now the case when $\mathcal{M}$ can call its advice, which is q-bounded for some polynomial q. In input block i this can happen up to i times. At these moments advices will be of size $O(q(i))$, and at most $O(i^2)$ of them are needed on a length ℓ_i prefix. One can take this into account, by modifying the A_i's so they have all these advices encoded in their states and by simulating $\mathcal{M}$ on the given prefix of the input, this time also including the use of advice by $\mathcal{M}$. The resulting sequence of IFA's with global states is still polynomially bounded.

It is clear that in the given simulations $\mathcal{A}$ satisfies the finite-delay condition iff $\mathcal{M}$ does. □

The theorem implies several analogues for other types of computing devices operating with finite configuration spaces. To circumvent the different input-output conventions in some cases, we call two complexity classes 'equal' only when the devices corresponding to both classes read their inputs sequentially; otherwise, when the devices in one class read their inputs in parallel, we say that they 'correspond'.

Theorem 3 *The following relations hold:*

(a) IFA-POLY equals ITM-LOGSPACE/poly.

(b) NA-LOG equals ITM-LOGSPACE/log.

(c) CC-POLY corresponds to ITM-PTIME/poly.

(d) NN-POLY equals ITM-PSPACE/poly.

(e) IFA-EXP equals ITM-PSPACE/exp.

For later reference we let DSPACE($S_1(t)$)/*advice*($S_2(t)$) denote the complexity class of all $S_1(t)$-space bounded deterministic TM computations making use of $S_2(t)$-space bounded advice.

6 Site machines

Our next aim is to consider *networks* of machines. Under this scenario the individual interacting machines will be called *site machines*, or simply *sites*. A site machine is an ITM enhanced by a mechanism allowing message sending and receiving very much like well-known *I/O-automata* [7], but it also allows the 'instantaneous' external influencing of its computational behaviour by changes of its transition function in the course of the interaction with the environment. More precisely, sites are viewed as follows.

Individual sites in the network are identified by their address, some symbolic number. The addresses of the sites are managed by a special mechanism that exists outside of the site machines (see Section 7). In order to support the efficient communication among the sites, the respective ITM's are equipped with an *internet tape*. This is a tape whose contents can be sent to any other site. To do so, the sending machine must write the address of the receiving site and its own 'return' address, followed by the message, in an agreed-upon syntax, to its internet tape. By entering into a special distinguished state, the message is sent to the site with the given address. Messages sent to sites with non-existing addresses at that time do not leave the sending machine and the sending machine is informed about this by transiting to another special state.

By sending the message successfully, the internet tape of the sending machine becomes empty in a single step. The message arrives at the receiving machine after some finite time. If at that time the receiving machine finds itself in a distinguished 'message expected' state (which can be superimposed onto other states) then the message is written onto its internet tape, in a single step. The receiver is informed about the incoming message by (enforced) entering into a distinguished state called 'message obtained'. Then the receiving machine can read the message, or copy it onto an auxiliary tape. After reading the whole message the machine can enter into a 'message expected' state again. When it does, its internet tape is automatically emptied in one step.

Otherwise, if the receiving machine is not ready to obtain a message (meaning that the machine is engaged in writing onto or reading from its internet tape), the message enters into a queue and its delivery is tried again in the next step. It may happen that two or more messages arrive to a site simultaneously. This 'write conflict' is resolved by giving priority to the machine with the lowest address, and the remaining messages enter the queue at the site.

Each site is operated by an (external) agent. An agent can work in two modes: *network mode*, in which its machine is logged-in and can communicate with other sites, and *stand-alone* mode, in which its machine is not logged-in. Switching between the two modes is done by the agent with a special instruction.

By entering a suitable input sequence via the input port in network mode, an agent can instruct its machine to do various specific things. First, the agent can instruct it to operate its current 'program', sending or receiving messages, and performing any computation making use of all data stored on the machine's working tapes, on its internet tape, and the data read from the input port. However, while working in network mode the agent is not allowed to change the machine's hardware and software, i.e., its transition function. This can only be done in stand-alone mode. A change of a transition function may change the number of the machine's tapes, its working alphabet and the number of states. Such an action is done in finite time, during which time the machine is not considered to be the part of the network. Changing the transition function does not affect the data written on the machine's tapes at that time (except for the case when the number of tapes is decreased, when only data on the remaining tapes persist). After changing the transition function, the agent switches back to network mode. The machine then continues operating following the new transition function. Only the inputs read during network mode are considered to be part of the input stream. The same holds for the output stream.

The *instantaneous description* (ID) of a site machine after performing t steps is given by the description of all its working tapes (including its internet tape), the current symbol at its input port, and the corresponding state of its finite control at time t. The current position of the tape heads is assumed to be marked by special symbols on the respective tape descriptions.

At time t, the 'program' of the site machine M at that time, is described by a binary code denoted as $\langle M \rangle$. It encodes, in an agreed-upon syntax, the transition function of the machine. Note that at

different times t, a machine with the same address can be described by (operationally) different codes, depending upon the activities of its agent.

To formally describe a site machine, we assume that there is a *site encoding function* δ that maps, for each time t, the address i of a machine to its encoding at that time. The *configuration of a site* at time $t > 0$ after processing t inputs consists of its address, followed by its encoding and its ID at that time.

Theorem 4 *For interactive translations $\phi : \Sigma^\omega \to \Sigma^\omega$, the following are equivalent:*

(a) ϕ is computed by a site machine.

(b) ϕ is computed by an ITM $\mathcal{M}$ with advice.

Proof (sketch): If ϕ is computed by a site machine then the value of the site encoding function at time t can serve as an advice to the simulating ITM/A. Vice versa, when the latter machine has to be simulated by a site machine, then each advice reading can be substituted by a site machine update where the advice value is encoded in the description of the update. □

7 The web Turing machine

The ultimate non-uniform model we introduce is the web Turing machine. A web Turing machine (WTM) is a 'time-varying' finite set of interacting sites. The cardinality of this set, the programs of the machines in the set, as well as the message delivery delays can unpredictably vary with time. We only assume that the sites share the same notion of time, i.e., we assume a uniform time-scale within a given WTM.

Let $\mathbb{Z}^+$ denote the set of non-negative integers, and let $\mathbf{N}$ denote the set of natural numbers.

Definition 7 *A web Turing machine $\mathcal{G}$ is a triple $\mathcal{G} = (\alpha, \delta, \mu)$ where:*

- $\alpha : \mathbb{Z}^+ \to 2^{\mathbb{Z}^+}$ *is the so-called* address function *which to each time $t \geq 0$ assigns the finite set of addresses of those sites that at that time are in network mode. Thus, at each time $t \geq 0$, $\mathcal{G}$ consists of the $|\alpha(t)|$ sites from the set $S_t = \{M_i | i \in \alpha(t)\}$ where M_i is the site at the address i.*
- $\delta : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \Sigma^*$ *is the so-called* encoding function *which to each time $t \geq 0$ and address $i \in \alpha(t)$ assigns the encoding $\langle M_i \rangle$ of the respective site M_i at that time at that address.*
- $\mu : \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{N}$ *is the so-called* message transfer function *which to each sending site i and each receiving site j and each time $t \geq 0$ assigns the duration of message transfer from i to j at time when the message is sent, for $i, j \in \alpha(t)$.*

The *description* of a WTM at time $t > 0$ is given by the set of *encodings* of all its sites at that time. The *configuration* of a WTM at time $t > 0$ corresponding to the input read thus far by each site consists of the list of configurations of its sites at that time. The list is ordered according to the addresses of the sites in the list.

A *computation* of a WTM proceeds as follows. At each time, each site which is in network mode and whose address is among the addresses given by the address function for this time, reads a symbol from Σ, possibly the empty symbol, from its input. Depending on this symbol and on its current configuration the machine performs its next move by updating its tapes, state and outputting a symbol (possibly λ) to its output, in accordance with its transition function. Within a move the machine can send a message to or receive a message from an other machine. Also, when the machine is in stand-alone mode, its agent can modify the transition function of the machine or the data represented on the machine's tape. Moreover, at any time an agent can 'log in' ('log out') a site into (from) the WTM by entering the respective mode of operation. This fact is recorded by the values of the functions α and δ that must change accordingly at that time, to reflect the new situation.

Any WTM acts as a *translator* which at each time reads a single input symbol and produces a single output symbol at each site (some of the symbols may be empty). In this way a WTM computes

a mapping from finite or infinite streams of input symbols at the sites to similar streams of output symbols. The number of streams varies along with the number of sites. The incoming messages are not considered to be a part of the input (as they arrive over different ports). Of course, the result of a translation does depend on the messages received at individual sites and on their arrival times at these sites. However, all messages are results of (internal) computations and therefore their sending times are uniquely determined; the arrival times to their destinations are given by μ. Therefore, for a given 'packed' stream of inputs (with each packed symbol unfolding to an input at every site), the result of the translation is uniquely determined.

Definition 8 *Let $\mathcal{G} = (\alpha, \delta, \mu)$ be a WTM. The* web translation *computed by $\mathcal{G}$ is the mapping Γ such that for all packed streams $\mathbf{x}$ and $\mathbf{y}$, $\Gamma(\mathbf{x}) = \mathbf{y}$ iff on input $\mathbf{x}$ to its sites, machine $\mathcal{G}$ produces $\mathbf{y}$ at its sites.*

The *space complexity* $S(t)$ of $\mathcal{G}$ at time t is the maximum space consumed by any site of $\mathcal{G}$, over all input streams of length t. The respective complexity class will be denoted as $WTM-DSPACE(S(n))$. By $WTM - PSPACE$ and $WTM - DLOGSPACE$ we will denote the classes of all polynomially and logarithmically space-bounded web translations, respectively. For a further discussion of the complexity issues, see Section 9.

We conclude this section by a few comments related to the definition of a WTM and its operation. First note that we did not require either α, δ or μ to be recursive functions. Indeed, in general there is neither a known computable relation between the time and the addresses of the site machines, nor between the addresses and the site descriptions or between sender-receiver addresses and message transfer times. Thus, for each $t \geq 0$ the three functions are given by finite tables at best. Second, note that we assumed that at each time $\mathcal{G}$ consists only of a finite number of sites, as implied by the definition of α. Also note that the transfer time of a message depends not only on the address of the sending and receiving sites but also on the message issuing time. This means that even if a same message is sent from i to j at different times, the respective message transfer times can differ. Message delivery time is assumed to be independent of the message length. This assumption may be seen as being too liberal but dependences can be considered to be amortized over the time needed to write the message to the internet tape.

8 The power of web computing

At each moment in time, the architecture and the functionality of a WTM are formally described by its functions α and δ. These two functions model the fact that in practice (as in the case of the Internet) the evolution of the machine depends both on the input to the individual sites and on the decisions of the respective agents from which the changes in network architecture and site functionality may result. The agent decisions may in turn also depend on the results of previous computations and on messages received at individual sites as seen by their respective agents. Under this scenario the description of a WTM may change from time to time in a completely unpredictable manner.

Due to the finiteness of a WTM at each time, its δ at a given time is always finite. Nevertheless, the size of the encoding function over its entire existence, for $t = 1, 2, \ldots$, is in general infinite. Intuitively, this is the reason why a WTM cannot be simulated be a single ITM with a finite encoding. However, for each time t the encoding of a WTM can be provided by an advice function, as shown in the following theorem.

There is one technical problem in the simulation of a WTM by an ITM/A. Namely, by its very definition a WTM computes a mapping from packed input streams to packed output streams, with packed symbols of variable size. The respective symbols are read and produced in parallel, synchronously at all sites. However, a normal TM, working as a translator of infinite input streams into infinite output streams using a single input and a single output tape, cannot read (and produce) packed symbols of variable size in one step, in a parallel manner. What it can is to read and produce packed symbols component-wise, in a sequential manner.

In order to solve this technical problem we assume that the simulating ITM/A has a specific 'architecture' tailored to the problem at hand. First, we assume that it has a single, infinite one-way

read-only input tape at which the original stream of packed inputs $\{(x_{i_1,t}, x_{i_2,t}, \ldots, x_{i_k,t})\}_{t=0}^{\infty}$ to G's sites with $\{i_1, i_2, \ldots, i_k\} = \alpha(t)$, is written as follows: for consecutive $t = 1, 2, \ldots$, it contains a 'block' of inputs $(x_{i_1,t}, x_{i_2,t}, \ldots, x_{i_k,t})$. Blocks and input symbols are separated by suitable marking symbols. Second, we assume that the ITM/A has one infinite one-way write-only output tape to which outputs are written of the form $\{(y_{i_1,t}, y_{i_2,t}, \ldots, y_{i_k,t})\}_{t=0}^{\infty}$, again in a block-wise manner. A pair of two infinite streams of input and output symbols thus obtained is called the *sequential representation* of a web translation Γ.

Theorem 5 *For every WTM $\mathcal{G}$ there exists a single ITM/A $\mathcal{A}$ that acts as a sequential translator of the web translation computed by $\mathcal{G}$, and vice versa.*

Proof (sketch): On its tapes $\mathcal{A}$ keeps the ID's of all sites in $\mathcal{G} = (\alpha, \delta, \mu)$, with their current modes. In the advice, the values of all three functions α, δ and μ are stored. Thanks to this, $\mathcal{A}$ can sequentially update the sites in accordance with the instructions and updates performed by each site.

The idea of the reverse simulation is to show that a single site operated by a suitable agent can simulate a ITM/A. The role of the agent will be to deliver the values of the advice function at times when needed. The machine can do so by switching to stand-alone mode and letting its agent exchange its program for the program that has the value of the advice encoded in its states. Then the interrupted computation will resume. The details are given in [15]. □

9 The efficiency of web computing

The equivalence between WTM- and ITM/A computations was proved with the help of simulations. Because we were primarily interested in characterizing the computing power of WTM's, no attempt was made to make the simulations as efficient as they could be and to relate the complexity classes of the two models. In this section we investigate the computational efficiency of 'web space' and 'web time'.

To get rid of some repeated assumptions in the theorems below we will bound the growth of the 'parameters' of a WTM over time, viz. its number of sites, the sizes of its site descriptions, and the message transfer times. More precisely:

Definition 9 *A WTM $\mathcal{G} = (\alpha, \delta, \mu)$ is called* S(t)-bounded *if it satisfies the following restrictions:*

- *the space complexity of $\mathcal{G}$ is $S(t)$, for all $t \geq 0$,*
- *the address length of any site in $\mathcal{G}$ does not grow faster than the space complexity of $\mathcal{G}$, i.e., for all $t \geq 0$ and any address $i \in \alpha(t)$ we have $|i| = O(S(t))$,*
- *the size of any site encoding does not grow faster than the space complexity of $\mathcal{G}$, i.e., for $t \geq 0$ we have $|\langle M_j \rangle| = O(S(t))$ for all $j \in \alpha(t)$, and*
- *the message transfer times are not greater in order of magnitude than the total size of $\mathcal{G}$, i.e., for all $t \geq 0$ and $i, j \in \alpha(t)$ we have $\mu(i, j, t) = O(|\alpha(t)|)$.*

The first restriction bounds the space complexity of each site. The second one allows at most an exponential growth (in terms of $S(t)$) of the synchronous WTM with time. The third restriction is also quite realistic; it says that the 'program size' at a site should not be greater than the size of the other data permanently stored at that site. The fourth restriction together with the second one guarantees that the size (e.g. in binary) of the values of the message transfer function is also bounded by $O(S(t))$. Note that by restriction one, the message length is also bounded by $O(S(t))$, since in the given space no longer messages can be prepared.

In the complexity calculations that follow we will always consider a $S(t)$-bounded, or 'bounded' WTM. We will first show that any bounded WTM is equivalent to an exponential space-bounded deterministic ITM using an exponential size advice.

Theorem 6 *For all space bounding functions $S(t) \geq 0$,*

$$\bigcup_{c>0} WTM - DSPACE(cS(t)) = \bigcup_{c>0} ITM - DSPACE(c^{S(t)})/advice(c^{S(t)}).$$

In particular,

$$WTM - DLOGSPACE = ITM - PSPACE/poly$$

Proof (sketch): The left-to-right inclusion is proved by keeping a 'mirror image' of the WTM on the tapes of the ITM/A. Since our WTM is $S(t)$-space bounded it can have up to $O(c^{S(t)})$ sites of size $S(t)$, for some c. Hence the mirror image of the WTM can be maintained in exponential space as claimed. For each t the advice size is also bounded by the same expression and the ITM/A uses it to simulate the WTM updates. For proving the opposite inclusion we simulate the i-th cell of the ITM/A by a special site that keeps the contents of this cell plus the information whether the machine's head is scanning this cell. The advice tape is represented in a similar manner. For the full proof see [15]. □

This result for space-bounded WTM computations is analoguous to similar results known for so-called synchronized computations in uniform models (see for example [5], [18]).

Next we study 'time' as a computational resource for WTM's, viz. the potential of a WTM to perform parallel computations. In order to make use of this potential one has to ensure e.g. when sending requests to two sites to run some computations, that these requests will be accomplished with only a small delay. Similarly, after finishing the computation, one has to ensure that both results will be returned again with only a small delay. This cannot be guaranteed under the original mild assumption that each message will be delivered in an finite, albeit unpredictable time.

In order to enable a genuinely parallel realization of computations we therefore strengthen the restriction on the duration of message deliveries within a bounded WTM further. We introduce the *unit cost* WTM in which each message is assumed to be delivered to its destination within unit time.

Definition 10 *A unit-cost WTM $\mathcal{G}$ is a bounded WTM $\mathcal{G} = (\alpha, \delta, \mu)$ in which $\mu(i, j, t) = 1$ for all $i, j \in \alpha(t)$ and $t \geq 0$.*

Let $WTM_U - PTIME$ denote the class of all translations that can be realized by a unit-cost WTM within polynomial reaction time.

It turns out that a unit-cost WTM can simulate an ITM/A very fast, by involving an exponential number of sites in the simulation. Vice versa, a fast WTM with 'many processors' can be simulated by an ITM/A in 'small' space. The simulation is sketched in the proof of the following theorem. When speaking about the respective models we shall use similar input/output conventions as those in Section 8.

Theorem 7

$$WTM_U - PTIME = ITM - PSPACE/poly$$

Proof (sketch): When attempting to simulate a polynomial time-bounded WTM in polynomial space on a ITM/A, we run into the problem that a WTM can activate an exponential number of processors whose representations cannot all be kept on a tape simultaneously. Thus, a strategy must be designed for re-using the space and recomputing the contents of each site when needed. The non-uniformity of WTM updates is simulated, as expected, by advice calls.

To simulate a ITM/A of polynomial space complexity $S(t)$ on a WTM of polynomial time complexity, imagine the infinite computational tree T of the ITM/A computations. For a given input, consider the subtree of T of depth $c^{S(t)}$ with the same root as T and an exponential number of nodes. The simulated ITM/A processes the first t inputs successfully iff the path in T that starts in an initial ID, ends in an ID that produces the 'further' output which is a reaction to the t-th input. The existence of such an accepting path is found by making use of the parallel version of algorithm that computes the transitive closure of T in polynomial space w.r.t. $S(t)$. This simulation must be run for each $t = 1, 2, \ldots$. This is achieved by starting the simulation at each time t for that particular value of t at a suitable site on the WTM. The involved details of both parts of the sketched proof can be found in [15]. □

Corollary 1 *For any* $S(t) \geq 0$

$$\bigcup_{c>e0} WTM - DSPACE(cS(t)) = \bigcup_{c>0} WTM_U - TIME(c^{S(t)}).$$

In particular,

$$WTM - DLOGSPACE = WTM_U - PTIME$$

The last result says that (bounded!) WTM's make use of their space in an optimal way: in the given space one cannot perform more time-bounded computations than a unit-cost WTM does.

The result on time-bounded unit-cost WTM's and its proof, mirrors the similar result for uniform, idealized computational models from the 'second machine class' (cf. [11]). Its members fulfill the so-called *Parallel Computation Thesis* which states that sequential polynomial space is equivalent to parallel polynomial time on devices from this class. In the non-uniform setting similar results are known for infinite families of neural networks of various kinds (for a recent overview of known results see [8]).

The results on time and space efficiency of WTM computations rank WTMs among the most powerful and efficient computational devices known in complexity theory.

10 Afterthoughts

Theorem 1 points to a rich world of interactive devices that can serve as a basis for the investigation of evolving interactive computing systems. In fact, in [20] these devices have been interpreted as *cognitive automata*. Each of them can serve as a model of a 'living organism' and can be used for further studies of the computational aspects of complex systems created from these elementary computing units.

Theorem 3 reveals that not all cognitive automata are equally efficient from the viewpoint of their descriptional economy: systems representing their configuration space in unary representation (this is the case of finite automata) suffer from space inefficiency. Other systems that make use of more efficient state representations and can reuse their space, such as neural nets, are much more effective from this point of view.

Theorems 2, 4, and 5 point to the central equivalence of the various models, summarized in the following Theorem. It points to the fact that the notion of evolving interactive computing is a robust and fundamental one.

Theorem 8 *For translations* $\phi : \Sigma^\omega \to \Sigma^\omega$*, the following are equivalent:*

(a) ϕ *is (sequentially) computed by a sequence of IFA's with global states.*

(b) ϕ *is (sequentially) computable by an ITM/A.*

(c) ϕ *is (sequentially) computable by a site machine.*

(d) ϕ *is computable by a WTM.*

Lemma 1 shows the super-Turing computing power of the ITM/A's and separates the model from ITM's. It also implies that the WTM, which can be seen as a quite realistic model of the Internet as far as its computing power is concerned, can perform computations that cannot be replicated by any standard interactive TM. This answers Wegner's claim concerning the power of interactive computing. Interaction can lead to computations that no Turing machine can mimic, providing that we allow updates of the underlying machinery and consider unbounded, potentially infinite computations. The latter condition is a crucial one since otherwise one would have but a finite number of updates that could be built-in beforehand into the architecture of the ITM.

Theorem 8 also points to the various ways in which non-uniform features may enter into a computing system. First, non-uniformity can be hidden in the 'architecture' of a computing system. This is the case in sequences of finite automata with global states where the description of the system as a whole is given by an infinite, in general non-computable string. Second, non-computable information

may enter into a computing system from an 'external' source. E.g. in the case of ITM/A's, this is done by advice functions, and in the case of site machines or WTM's there are agents that can change the machine architecture in an unpredictable manner.

The results from theorem 8 have interesting interpretations in the world of cognitive automata and computational cognition. The basic idea is as follows: in a 'robotic' setting, any interactive finite automaton (viewed as a *cognitive automaton*) can be seen as a simple model of a living organism. A (finite) set of cognitive automata can communicate basically in two different ways. First, the automata can communicate using a fixed 'pre-wired' communication pattern, so to speak holding hands with their physically immediate neighbors. By this we get systems equivalent to sequences of interactive finite automata. If the automata communicate in arbitrary patterns or do not have global states, the corresponding system of cognitive automata resembles certain types of *amorphous computing systems* [1]. In principle it is no problem to define cognitive automata in such a way that they will also possess a replication ability. Then one can consider systems of cognitive automata that grow while computing. As a result one gets various *morphogenetic computational systems*.

The second possibility for cognitive automata to communicate, is the case when the automata are equipped with sensors and effectuators by which they scan and change their environment. In the case of ordinary TM's the 'living environment' of a single cognitive automaton working under such conditions is modelled by TM tapes and read/write heads. Following this analogy further, a WTM can be seen as a set of cognitive automata. They share the same living environment and communicate via message exchange. They can even move and exchange messages, either by encountering each other, or leaving a message elsewhere (probably in a distinguished place) in the environment or by sending a message via a chain of neighbors. A specific view of a human society as that of a community of agents communicating by whatever reasonable means (language, e-mail, letters, messengers, etc.) also leads to a model of WTM with specific parameters. What is important and interesting from the point of view of cognitive sciences is the fact that irrespectively which possibility is taken, we always get a system equivalent to a WTM and hence in general possessing a super-Turing computing power.

A challenging question still remains unanswered: could one indeed make use of the super-Turing potential of the underlying machines to one's advantage? Can one solve certain concrete undecidable problems by such machines? The answer is, (un)fortunately, no. From a practical point of view our results mean that the corresponding devices cannot be simulated by standard TM's working under a classical scenario. This is because the evolving interactive machinery develops in an unpredictable manner, by a concurrent unpredictable activity of all agents operating the sites.

Nevertheless, the above results point to quite realistic instances where the classical paradigm of a standard Turing machine as the generic model which captures all computations by digital systems, is clearly insufficient. It appears that the time has come to reconsider this paradigm and replace it by its extended version, viz. by ITM's with advice. For a more extended discussion of the related issues, see [14].

Bibliography

[1] H. Abelson, D. Allen, D. Coore, Ch. Hanson, G. Homsy, T.F. Knight, R. Nagpal, E. Rauch, G.J. Sussman, R. Weiss: Amorphous computing, *Comm. ACM*, Vol. 42, No. 5, May 2000, pp. 74-82.

[2] J.L. Balcázar, J. Díaz, J. Gabarró : *Structural Complexity* I, Second Edition, Springer-Verlag, Berlin, 1995.

[3] G. Berry: The foundations of Esterel, in: G. Plotkin, C. Stirling and M. Tofte (Eds), *Proof, Language, and Interaction - Essays in Honour of Robin Milner*, The MIT Press, Cambridge MA, 2000, pp 425-454.

[4] L. Cardelli: Global computation, *ACM Sigplan Notices*, Vol. 32, No. 1, 1997, pp. 66-68.

[5] J. Dassow, J. Hromkovič, J. Karhumäki, B. Rovan, A. Slobodová: On the power of synchronization in parallel computing, Computing, in: A. Kreczmar and G. Mirkowska (Eds), *Mathematical Foundations of Computer Science 1989*, Proceedings, Lecture Notes in Computer Science, Vol 379, Springer-Verlag, Berlin, 1989, pp. 196-206.

[6] R.M. Karp, R.J. Lipton: Some connections between non-uniform and uniform complexity classes, in: *Proc. 12th Annual ACM Symposium on the Theory of Computing* (STOC'80), 1980, pp. 302-309, revised as: Turing machines that take advice, *L'Enseignement Mathématique*, II^{e} Série, Tome XXVIII, 1982, pp. 191-209.

[7] N.A. Lynch: *Distributed algorithms*, Morgan Kaufmann Publishers Inc., San Francisco CA, 1996.

[8] P. Orponen: An overview of the computational power of recurrent neural networks, in: Proc. Finnish AI Conference (Espoo, Finland, August 2000), Vol. 3: *AI of Tomorrow*, Finnish AI Society, Vaasa, 2000, pp. 89-96.

[9] J. Šíma, J. Wiedermann: Theory of Neuromata. *Journal of the ACM,* Vol. 45, No. 1, 1998, pp. 155-178.

[10] L.G. Valiant: *Circuits of the Mind*, Oxford University Press, New York, 1994.

[11] P. van Emde-Boas: Machine models and simulations, in: J. van Leeuwen (ed.). *Handbook of Theoretical Computer Science*, Vol. A: Algorithms and Complexity, Elsevier Science Publ, Amsterdam, 1990, pp. 3-66.

[12] J. van Leeuwen, J. Wiedermann: On algorithms and interaction, in: M. Nielsen and B. Rovan (Eds), *Mathematical Foundations of Computer Science 2000*, 25th Int. Symposium (MFCS'2000), Lecture Notes in Computer Science, Vol. 1893, Springer-Verlag, Berlin, 2000, pp. 99-112.

[13] J. van Leeuwen, J. Wiedermann: A computational model of interaction in embedded systems, Technical Report UU-CS-02-2001, Dept. of Computer Science, Utrecht University, 2001.

[14] J. van Leeuwen, J. Wiedermann: The Turing machine paradigm in contemporary computing, in: B. Enquist and W. Schmidt (Eds), *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, Berlin, 2001, pp. 1139-1155.

[15] J. van Leeuwen, J. Wiedermann: Breaking the Turing barrier: the case of the Internet, manuscript in preparation, February 2001.

[16] P. Wegner: Why interaction is more powerful than algorithms, *C. ACM* 40, (1997) 315-351.

[17] P. Wegner, D.Q. Goldin: Interaction, computability, and Church's thesis, *The Computer Journal* 2000 (to appear).

[18] J. Wiedermann: On the power of synchronization, *J. Inf. Process. Cybern.* (EIK), Vol. 25, No. 10, 1989, pp. 499-506.

[19] J. Wiedermann: The computational limits to the cognitive power of neuroidal tabula rasa, in: O. Watanabe and T. Yokomori (Eds), *Algorithmic Learning Theory*, Proc. 10th International Conference (ALT'99), Lecture Notes in Artific. Intelligence, Vol. 1720, Springer-Verlag, Berlin, 1999, pp. 63-76.

[20] J. Wiedermann, J. van Leeuwen: Emergence of super-Turing computing power in artificial living systems, in: J. Kelemen (Ed.), *Artificial Life 2001*, Proceedings 6-th European Conference (ECAL 2001), Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 2001 (to appear).