The Turing Machine Paradigm in Contemporary Computing*

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

[1] Department of Computer Science, Utrecht University,
Padualaan 14, 3584 CH Utrecht, the Netherlands.
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic.

1 Introduction

The importance of algorithms is now recognized in all mathematical sciences, thanks to the development of computability and computational complexity theory in the 20th century. The basic understanding of computability theory developed in the nineteen thirties with the pioneering work of mathematicians like Gödel, Church, Turing and Post. Their work provided the mathematical basis for the study of algorithms as a formalized concept. The work of Hartmanis, Stearns, Karp, Cook and others in the nineteen sixties and seventies showed that the refinement of the theory to resource-bounded computations gave the means to explain the many intuitions concerning the complexity or 'hardness' of algorithmic problems in a precise and rigorous framework.

The theory has its roots in the older questions of definability, provability and decidability in formal systems. The breakthrough in the nineteen thirties was the formalisation of the intuitive concept of algorithmic computability by Turing. In his famous 1936-paper, Turing [43] presented a model of computation that was both mathematically rigorous and general enough to capture the possible actions that any 'human computer' could carry out. Although the model was presented well before digital computers arrived on the scene, it has the generality of describing computations at the individual bit-level, using very basic control commands. Computability and computational complexity theory are now firmly founded on the Turing machine paradigm and its ramifications in recursion theory. In this paper we will extend the Turing machine paradigm to include several key features of contemporary information processing systems.

1.1 From Machines to Systems that Compute

Turing's model is generally regarded as capturing the intuitive notion of what is algorithmically computable in a very broad sense. The general belief that every algorithm can be expressed in terms of a Turing

* This research was partially supported by GA ČR grant No. 201/98/0717 and by EC Contract IST-1999-14186 (Project ALCOM-FT).

machine, is now known as the *Church-Turing thesis* (cf.[12], [37]). Even though many other formalisations of computability have been proposed and studied through the years, there normally proved to exist effective simulations in terms of standard Turing machines. This has been specifically demonstrated for the many formalisms for defining computable (partial) functions on $\mathbb{N}$ (cf. [8], [15], [26]).

With the origin of Turing machines dating back to the nineteen thirties, the question arises whether the notion of 'computation' as it is now understood is still adequately described by them. The emphasis in modern computer technology is gradually shifting away from individual machines towards the design of systems of computing elements that interact. New insights in the way physical systems and biological organisms work are uncovering new models of computation. Is the Church-Turing thesis as we know it still applicable to the novel ways in which computers are now used in modern information technology? Will it hold for the emerging computing systems of the future?

At least three new ingredients have entered into the world of algorithmic computability that need further examination from this viewpoint: *non-uniformity of programs*, *interaction of machines*, and *infinity of operation*. Whereas these ingredients are well-studied in computer science separately, future computing systems require that they are considered *simultaneously*. It is expected that the synergetic effect among them will lead to new qualities of computing that match the systems that are now emerging. We therefore propose a corresponding extension of the Turing machine paradigm to capture these features of modern-day computing.

The paper is organized as follows. In Section 2 some crucial features of modern computational systems are described and an extended Church-Turing thesis is formulated. In Section 3 we will introduce two basic models for a more formal theory. The first one, the *site machine*, derives from the insights in distributed computing and models personal (network-)computers and the way they are operated. The second model is an extension of a formal model known in the theory of non-uniform computations: the *interactive Turing machine with advice*. In Section 4 we will argue that the models are computationally equivalent. In Section 5 we introduce a formalized model of the Internet and show that the resulting internet machine is of the same computational power as the previous two models again, thus giving further evidence of the proposed thesis. The extended Church-Turing thesis is discussed in Section 6. We assume no specific background from computability theory for the purposes of this paper. For a modern treatment of computability and computational complexity theory, see e.g. [25] or [34].

2 Computational scenarios

In this section we briefly reflect on the study of computability using Turing machines and on some of the new features that are arising in modern information technology. We will motivate the need for a reconsideration of the classical Turing machine paradigm and formulate an extension of the Church-Turing thesis.

2.1 The classical Turing machine paradigm

A Turing machine is an abstract computing device consisting of a finite control, a potentially unbounded memory consisting of one or more tapes, and a finite program. The program is a list of instructions that say what step the machine can take based on the current state of its finite control and the bits in its current 'window' on the information on the tapes. A step normally involves both a change of state in the finite control, a rewrite of the bits in the window, and a move of the window to the left or to the right by one position. After starting, the program is assumed to execute until some halting state is reached.

The computational scenario for Turing machine computations assumes that at the beginning of the computation the entire (finite) input data is available on the input tape. The rest of the input tape and of any other tapes is blank. If the machine terminates in an accepting state, the input is said to be accepted. The result of the computation is given by the contents of the tapes at this time. If a new computation (on new input data) is started, then all previous information is erased. No information can be carried over from the past to the present run of the Turing machine. For a given machine, the *same* fixed program is used *uniformly* for all inputs.

The framework of Turing machines is very suitable for studying the power and efficiency of algorithms, considering the achievements of computability and complexity theory (see e.g. [5], [18], [23], [33]). Various fundamental notions were recognized and added such as *nondeterminism* ('finitely many options per step'), *probability* ('coin flipping') and *parallelism* ('many processors'), with no effect on the Church-Turing thesis but in some cases apparently with severe effects on what can be computed within predefined resource bounds [46]. Even *genetic* [32] and *quantum* features [14] were introduced. The model was also modified to more closely resemble the architecture of digital computers. In the *random access machine* (RAM) the finite control was replaced by an arithmetic-logic unit, the tapes by a potentially unbounded memory of addressable cells, and the instruction set by that of a typical computer. The RAM-model is now often used as an alternative entrance to the study of complexity classes and for the machine-independent analysis of algorithms.

An important extension of the basic Turing machine model which entered into the theory at an early stage is the extension by *oracles* (cf. [5], [33]). Oracles enable one to enter new, external, and possibly non-computable information into a computation. An oracle is given by an *oracle set.* It can be any set of words over an alphabet Σ, in particular it may be a set for which membership cannot be decided by any Turing machine, i.e. there is no algorithm for it. An oracle Turing machine can consult its oracle by submitting membership queries to it, which the oracle then answers 'in one step'. One may also allow oracles that deliver, for any argument supplied by the machine, the value of some (in general, non-computable) function for that argument in one step. Oracles have originally been proposed by Turing [44] and have given rise to the rich theory of *relativised computability* (cf. [33]) and to the modern *relativised complexity theory* [5]. Oracles can have a dramatic effect on the complexity of computations and on the relative power of features like nondeterminism. For our purposes oracles are far too general and powerful. We will use a weaker version of them called 'advice' in Section 3.

2.2 Towards a new paradigm

The experience with present-day computing confronts us with phenomena that are not captured in the scenario of classical Turing machines. Personal computers are running practically uninterruptedly from the day of their installation. Even when they are switched off, the data stored in permanent memory survives ('persists') until the next session. When the computer is changed in favor of a new model, most data is transferred from the old computer to the new one. Very often a computer is 'upgraded' by changing its software or hardware, thus giving it new functionalities in the course of its existence. Its program is thus no longer fixed but evolves over time in a non-programmed, non-uniform way. (In the sequel we will refer to such programs as *non-uniform programs*.)

The idea of having all input data present prior to the start of any computation also no longer applies. Data is streaming into a computer practically continuously via its input ports, often in the very moment of its first (and at the same time perhaps its last) appearance and from various sources: the keyboard, the mouse, the network connection, the microphone, a camera and so on. The results of a computation are likewise streaming out from the computer continuously also, through various channels: texts and pictures appear on the screen or on the printer, sounds are emitted via loudspeakers, messages and data are sent over the network connection, and so on.

Also, computers are not running in isolation anymore but are hooked into dynamic networks of virtually unlimited size, with many computers sending and receiving signals in sheer unpredictable ways. Along this line, one can envision the synthesis of large, world-wide Internet-like systems of communicating processors and networks to form myriads of programmable *global computers* ([9], [16]) offering many different qualities of service.

There even are visions that over the next decades, new technologies will make it possible to assemble systems of huge numbers of computing elements that need not even all work correctly or be arranged in some predefined or even computable way. The development of these *amorphous computing systems* [1] is likely to lead to fundamental changes in the views of computation, possibly with new inspiration from the kinds of interaction that take place in physical systems or among cells in biological organisms. *It potentially leads to complex computational behaviors that cannot be considered to have been explicitly programmed.*

The preceding analysis shows that the classical Turing paradigm may no longer be fully appropriate to capture all features of present-day computing. The contemporary view of computing by means of large and complex distributed systems of computing elements leads to the view of computing as a potentially endless process of interaction among components (or agents) and of components with their environment. Occasionally, and unpredictably, the software or hardware of some components is changed by an external agent ('the system administrator'). In principle this will be allowed to its potential limit of happening infinitely often.

What would the appropriate computational model for the new features be in terms of Turing machines? Can it be described in a tractable mathematical framework at all? Does the change of the basic computational scenario have any effect on the computational power of the respective system? Can the Church-Turing thesis be adjusted to capture the new situation? These questions have been arising recently within several communities, indicating the need for a revision of the classical Turing machine paradigm (see e.g. [9], [49], and [50]). Wegner formulates it as follows ([50], p. 318):

> *"The intuition that computing corresponds to formal computability by Turing machines ... breaks down when the notion of what is computable is broadened to include interaction. Though Church's thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid."*

We note that it is not the Church-Turing thesis that is questioned here but rather the notion of *computation* that it captures. A further discussion of Wegner's views from the viewpoint of computability theory was given in [31] (see also [17] and [51]).

2.3 Non-uniform interactive computations

The aim of the present paper is to consider the following features of modern computing in more detail: *non-uniformity of programs*, *interaction of machines*, and *infinity of operation*. We believe that these features, seen as *simultaneous* augmentations of the basic machine model, will be essential components in the development of computability theory in the years ahead. We will outline an extended Turing machine model and provide evidence that it captures what can be computed in the newer information processing systems. In the approach, the informal meaning of 'what can be computed' will be formalized in terms of *non-uniform interactive computations.* We will show that such computations are exactly those that are realized by *interactive Turing machines with advice*, with 'advice' being a weak but highly appropriate type of oracle that has received only some attention to date. Advice function will liberally model machine upgrades. It motivates the following extension of the Church-Turing thesis:

> *Any (non-uniform interactive) computation can be described in terms of interactive Turing machines with advice.*

We believe that the future developments in computability theory will more and more have to take the new paradigm into consideration. The problems of describing and characterizing the behaviour and computational complexity of dynamic systems of interactive Turing machines with advice will pose many mathematical challenges, akin to those in the emerging area of mobile distributed reactive systems [3] and global computing [9].

3 Two Models of Computation

To model what can be computed, both a computational model and a computational scenario are needed. In this section we design two models of computation which capture some of the new features described above. First we design the *site machine*, a model inspired by considerations from computer networks and distributed computing. The second model is based on Turing machines *with advice* (cf. [5]). We describe an interactive version of it that allows for repeated upgrades and continuous operation. In Section 4 we will show that site machines and 'interactive Turing machines with advice' are computationally equivalent, and that they are more powerful than ordinary Turing machines.

3.1 Site machines

A *site machine* models a personal computer that may, but need not be connected to some computer network. Informally, a site machine consists of a full-fledged computer with a processor and a potentially unbounded, permanent read/write memory. It can be modeled either by a Turing machine, or by a random access machine (RAM) equipped with an external permanent memory (e.g. a disk), or by any other similar model of a universal programmable computer. The machine is equipped with several input and output ports via which it communicates with its environment. Without loss of generality we can restrict its communication ability to sending and receiving messages. Messages are finite sequences of symbols chosen from some finite alphabet Σ. The site machine reads or sends messages symbol by symbol, at each port one symbol at a time. When there are no symbols to be read or sent a special empty symbol $\tau \in \Sigma$ is assumed to appear at the respective ports. Up to this point, site machines resemble the well-formalized *I/O automata* defined by Lynch and Tuttle (cf. [20]) in the context of the study of distributed algorithms.

We assume that a site machine is operated by an *agent*. The agent is considered to be a part of the environment. An agent can communicate with the site machine via its ports, but it can also modify the software or hardware of the machine. In order to modify it, the agent must temporarily switch off the machine. Thanks to the assumption of permanent memory, no data will be lost by this. Below we will describe some reasonable limitations on what an agent can do. After upgrading the machine, the agent switches the machine on again and the machine continues its interaction with the environment, making use of the data residing in the permanent memory. Note that by changing a piece of hardware, new software and new data (residing in a chip in its read-only memory, say) can enter the system. A potentially endless sequence of upgrades may be seen as an evolutionary process acting on the underlying site machine. Formally, this process may be described by a function γ that assigns to each time t the description $\gamma(t)$ of the hardware and/or software upgrade (if any) at that time. In general, this function is non-computable and not known beforehand. Nevertheless it fully determines the computation of a site machine from the time of its upgrade, on input entered after that time and making use of the persistent data.

A *computation* of a site machine is an incremental, symbol by symbol translation of an infinite multiple stream of input symbols to a similar stream of output symbols. If the site has k input and ℓ output ports, with $k, \ell > 0$, then the computed mapping Φ is of the form $(\Sigma^k)^\infty \to (\Sigma^\ell)^\infty$. Thus, an infinite stream of k-tuples is translated to an infinite stream of ℓ-tuples, one tuple after the other.

3.2 Interactive Turing machines with advice

To study the computational power of site machines, we need to compare them to a more basic mathematical model. We will design a suitable analog of the Turing machine that will do for our purposes. The Turing machine will be equipped with three new features: *advice*, *interaction* and *infinity of operation*.

3.2.1 Advice functions

In order to mimic site machines, a Turing machine must have a mechanism that will enable it to model the change of hardware or software by an operating agent. We want this change to be quite independent of the current input read by the machine up to the moment of change. If this wouldn't be the case, one could in principle enter ready-made answers to any pending questions or problems that are being solved or computed into the machine, which is not reasonable. By a similar argument we do not want the change to be too large, as one could otherwise smuggle information into the machine related to all possible present and future problems. In order to fulfill these two conditions, we quite arbitrarily insist that the description of the new hardware (or software or data) will depend only on the *time* t of the change and that the size of the description will be *at most polynomial* in t.

To enter new, external, and possibly non-computable information into the machine we will make use of *oracles* (cf. [5], [33]). We like to view oracles as 'hardware upgrades' and thus need a rather more restricted version of the general notion that only provides its upgrades in a general and largely input-independent way. A good candidate are the *advice functions*. Turing machines with advice were first considered by Karp and Lipton [19] in their fundamental study of non-uniform complexity theory (cf. [5], [36]).

Definition 1. An advice function is a function $f : \mathbb{N} \to \Sigma^\star$. An advice function f is called $S(n)$-bounded if for all n, the length of $f(n)$ is bounded by $S(n)$.

An advice Turing machine with input of size n will be allowed to consult its oracle only for that particular value. More precisely, let $\mathcal{C}$ be a class of languages ('problems') defined by Turing machines and $\mathcal{F}$ be a class of advice functions. Let $\langle , \rangle$ denote some simple encoding.

Definition 2. The class $\mathcal{C}/\mathcal{F}$ consists of the languages L for which there exists a $L_1 \in \mathcal{C}$ and a $f \in \mathcal{F}$ such that the following holds for all n and inputs x of length n: $x \in L$ if and only if $\langle x, f(n) \rangle \in L_1$.

Common choices considered for $\mathcal{C}$ are: P ('deterministic polynomial time'), NP ('nondeterministic polynomial time), $PSPACE$ ('polynomial space'), and $EXPTIME$ ('deterministic exponential time'). Common choice for $\mathcal{F}$ are log, the class of logarithmically bounded advice functions, and *poly*, the class of polynomially bounded advice functions. Classes like $P/poly$ are easily seen to be robust under allowing inputs x of length $\leq n$ in the given definition, or even of length $\leq q(n)$ for some polynomial q ([6]).

The hardware view of advice functions is supported by the following theorem that combines results of Pippenger [27], Yap [53] and Schöning [35]:

Theorem 3. *The following characterisations hold:*
(i) $P/poly$ is precisely the class of languages recognized by (possibly non-uniform) families of polynomial size circuits.
(ii) $NP/poly$ is precisely the class of languages generated by (possibly on-uniform) families of polynomial size circuits.

A classic result by Adleman [2] shows that $P/poly$ contains the class RP consisting of the languages recognized by polynomial time-bounded probabilistic Turing machines with one-sided error. More generally it can be even shown that $BPP \subseteq P/poly$, where BPP is the class of languages accepted by polynomial time-bounded probabilistic Turing machines with two-sided bounded-error. We will return to the power of advice in Section 4.

3.2.2 Interaction and Infinity of Operation

Next the Turing machine must accommodate interaction and infinite computations. Neither of the two features by itself is new in computer science. Notions of interaction and of interactive computing have already received considerable attention, for example, in the theory of *concurrent processes* (cf. Milner [21,22] and Pnueli [28,29]) and in the design of programming systems for parallel processes (cf. [10]). Interaction is also fundamental in the many studies of *communication protocols* and *distributed algorithms* in which the building blocks act as (restricted) interactive Turing machines (cf. [4], [20], and [40]). Interaction is even modelled using concepts from *game theory* and *microeconomics*. Infinite computations have been formalized and studied from the language-theoretic viewpoint in the theory of $\omega - automata$ (cf. [39] and [41,42]).

To accommodate the features in one model of computation, the Turing machine with advice described above must be equipped with finitely many input and output ports like site machines and I/O-automata. Through its input ports the machine will be able to read symbols, one symbol at a time. If at some time there is no interesting symbol from Σ delivered by the machine's environment to an input port, a special *empty* symbol τ is assumed to be input again. Similar situations at output ports are handled analogously. We do not require a fixed or otherwise specified interconnection structure in advance that links the machine to other machines.

The computational scenario of an interactive Turing machine is as follows. The machines starts its computation with empty tapes. It is essentially driven by a standard Turing machine program. At each step the machine reads the symbols appearing at its input ports. At the same time it writes some symbols to its output ports. Based on the current *context*, i.e. on the symbols read on the input ports and in the 'window' on its tapes, and on the current state, the machine prints new symbols under its heads, moves its windows by one cell to the left or to the right or leaves them as they are, and enters a new state. Assuming there is a possible move for every situation (context) encountered by the machine, the machine will operate in this manner forever. Doing so, its memory (i.e. the amount of rewritten tape) can grow beyond any limit. At any time $t > 0$ we will also allow the machine to consult its advice, but only for values of at most t. Also, to prevent the danger of cheating with large advice values, we allow *polynomially bounded advice functions* only.

Turing machines that follow these specifications are called *interactive Turing machines with advice*. For more information concerning the computational power of 'pure' interactive Turing machines (i.e. without any advice) we refer to [47].

4 The Power of Site Machines

In order to support the proposed extension of the Church-Turing thesis, we show that site machines and interactive Turing machines with advice are computationally equivalent. This means that to any machine of the former type there is the machine of the latter type that simulates it, and vice versa. To show that this implies that site machines are essentially more powerful than classical Turing machines, we first digress on the power of ordinary Turing machines with the added features.

4.1 The Power of Advice

Turing machines with advice are widely used in *non-uniform complexity theory* to study the computational power and efficiency of infinite families of devices that are not uniformly programmed. It is easily seen that Turing machines with advice can accept non-recursive languages. By way of illustration we show how to recognize the diagonal language of the halting problem using advice. To define the problem, note that every string $w \in \Sigma^*$ may be seen both as the encoding $\langle M \rangle$ of some Turing machine M (in some agreed-upon syntax and enumeration) and as a standard input string.

Definition 4. K is the set of words that are the encodings of those Turing machines that accept their own encoding.

Turing [43] proved that there does not exist an algorithm, i.e. a Turing machine, that can decide for any Turing machine M given by means of its description $\langle M \rangle$, whether $\langle M \rangle \in K$. In the terms of recursion theory, K is recursively enumerable but not recursive.

Proposition 5. *There exists a Turing machine A with linearly bounded advice that accepts K and always halts.*

Proof. Define an advice function f as follows. For each n it returns the encoding $\langle N \rangle$ of the machine N for which the following holds: $\langle N \rangle$ is of length n, N accepts $\langle N \rangle$, and N halts on input $\langle N \rangle$ after performing the maximum number of steps, where the maximum is taken over all machines with an encoding of length n that accept their own encoding. If no such machine exists for the given n, f returns a fixed default value corresponding to a machine that halts in one step. It is easily seen that f exists and is linearly bounded.

On an arbitrary input w, machine A works as follows. First it checks whether w is the encoding of some Turing machine. If not then A rejects w. Otherwise, if w is the encoding $\langle M \rangle$ of some Turing machine M, then A asks its advice for the value $f(n)$, with $n = |w|$. Let $f(n) = \langle N \rangle$, for some machine N. A now simulates both M and N, on inputs $\langle M \rangle$ and $\langle N \rangle$, respectively, by alternating moves, one after the other. Now two possibilities can occur:

(a) N will stop sooner than M. That is, M has not accepted its input by the time N stops. Since the time of accepting $\langle N \rangle$ by N was maximum among all accepting machines of the given size, we may conclude that M does not accept w. In this case A does not accept w either.

(b) M will stop not later than N. Then A accepts w if and only if M accepts w.

Clearly A stops on every input and accepts K.

By a similar argument it follows that all recursively enumerable languages and their complements can be recognized by Turing machines with linear advice.

It is easily seen that classes like $P/poly$ and most other advice classes contain nonrecursive languages as well [19]. Many problems remain about the relative power of advice. In fact, several classical open problems in uniform complexity theory have their equivalents in the non-uniform world. For example, Karp and Lipton [19] proved, among other results:

Theorem 6. *The following equivalences hold:*
(i) $NP \subseteq P/log$ *if and only if* $P = NP$*.*
(ii) $EXPTIME \subseteq PSPACE/poly$ *if and only if* $EXPTIME$ *equals* $PSPACE$*.*

Molzan [24] and others have explored possible logical frameworks for characterising the languages in non-uniform complexity classes.

4.2 The Power of Site machines

We now consider the relation between site machines and interactive Turing machines with advice. We omit the explicit mention of the polynomial bounds on the upgrades and the advice values that is assumed in the two models.

Theorem 7. *For every site machine* $\mathcal{S} = (\gamma)$ *there exists an interactive Turing machine* $\mathcal{A}$ *with advice such that* $\mathcal{A}$ *realizes the same computation as* $\mathcal{S}$ *does.*

Proof. (Outline.) Assume that the model of a site machine is formalized enough to offer the formal means for describing the complete configuration of the machine at each time $t > 0$. Knowing this so-called *instantaneous description* at time t, which includes the description of the entire memory contents of the machine, its program, its current state and the current input, one can determine the next action of $\mathcal{S}$. This is the central idea used in the simulation of $\mathcal{S}$ by $\mathcal{A}$. At each time t, $\mathcal{A}$ keeps on its tapes the instantaneous description of $\mathcal{S}$. Since it reads the same input as $\mathcal{S}$, it can update this description in much the same way as $\mathcal{S}$ does. In particular, it can produce the same output.

When the agent that is operating machine $\mathcal{S}$ decides to 'upgrade' its machine at some time t, $\mathcal{A}$ calls its advice with argument t. In return it obtains the description $\gamma(t)$ of the new hardware/software by which $\mathcal{S}$ is 'upgraded' at that time. From this and from the fact that the data of $\mathcal{S}$ are permanent, $\mathcal{A}$ can infer the instantaneous description of $\mathcal{S}$ at time $t+1$ and can resume the simulation of $\mathcal{S}$. It is clear that in this way both machines realize the same translations. Note that in order for this simulation to work, the advice at any time t has to contain the record $\gamma(t)$ of the relevant upgrade (if any) of machine $\mathcal{S}$.

Theorem 8. *For every interactive Turing machine $\mathcal{A}$ with advice there exists a site machine $\mathcal{S} = (\gamma)$ such that $\mathcal{S}$ realizes the same computation as $\mathcal{A}$ does.*

Proof. (Outline.) The idea of the simulation is similar to the previous case. At each time, machine $\mathcal{S}$ keeps in its permanent memory the complete instantaneous description of $\mathcal{A}$ at that time. The advice consulting moves of $\mathcal{A}$ are simulated by $\mathcal{S}$ via proper upgrades of its hardware. This is possible via the proper design of a function γ that reflects the changes of $\mathcal{A}$'s instantaneous descriptions caused by advice calls at the respective times.

The given theorems give rise to the following characterisation of the classes of functions computable by site machines and interactive Turing machines with advice, respectively.

Theorem 9. *Let $\phi : (\Sigma^k)^\infty \to (\Sigma^\ell)^\infty$ be a mapping, with $k, \ell > 0$. Then the following are equivalent:*
(i) ϕ is computable by a site machine, and
(ii) ϕ is computable by an interactive Turing machine with advice.

Any mapping ϕ computed by either of the above mentioned machines will be called a *non-uniform interactive mapping*. In the next section we will show that it covers a far greater spectrum of computations.

5 A Computational Model of the Internet and Other Systems

We now turn attention to the mappings computed by *networks of site machines* such as the Internet and to other evolving systems of computing agents. To obtain an adequate model we will introduce so-called *internet machines*. We will argue that even internet machines can be simulated by interactive Turing machines with advice. We implicitly assume again that there are polynomial bounds on the upgrades and the advice values used in the two models.

5.1 Internet Machines

An internet machine will be a finite but time-varying set of sites machines. Each machine in the set will be identified by its address. For each time t, the address function α gives the list $\alpha(t)$ of addresses of those machines that are present at that time in the internet machine. We assume that for all t, the size of the list $\alpha(t)$ is polynomially bounded. If this is the case, the size of the internet machine can grow at most polynomially with time. For this exposition we assume also that the machines work synchronously. Within the internet machine, site machines communicate by means of message exchanges.

In order for a message to be delivered it must contain, in its header, the addresses of both the sending and the receiving machine. When sent from some machine at time t the message will reach its destination in some future time which, however, is unpredictable. Therefore, the message transfer time between any pair (i, j) of site machines with $i, j \in \alpha(t)$, is given by a special function β which for each triple (i, j, t) returns the respective message delivery time. At time t a message can only be sent to a machine that is on the list $\alpha(t)$. If a message is sent to a non-existing machine at that time, an error message will appear at the sending site. The same will happen when the receiving machine will vanish in the mean time (i.e. when at time t of message delivery the respective machine is no longer on the list $\alpha(t)$). When several messages arrive to a site at the same time then they will queue. They will be processed by the receiving machine in order of their sender addresses. All site machines work synchronously and, as before, the operating agents of the machines are allowed to modify the hardware/software. Formally, the respective changes are described similar to the case of site machines by a third function γ which, for each time t and each site machine $i \in \alpha(t)$, returns the formal description $\gamma(t, i)$ of the respective upgrade (if any) at that time.

The three functions α, β and γ together specify the operation of a given internet machine. The functions are in general non-computable. For each t their values are given by finite tables.

The resulting model computes a mapping similar to that computed by a site machine. Messages sent are included among the machine's output and messages received among the machine's inputs. The difference is that the multiplicity of the input and output stream varies along with the number of machines that are present in the internet machine. However, the key observation is that at any time, the description of the internet machine is finite. If the internet machine consists of a polynomial number of sites with a polynomially bounded description, then this allows the encoding of the current 'software' running on all its site machines into an advice, for each time t. In this case the internet machine can again be simulated by an interactive Turing machine with advice.

Of course, the simulation cannot be done in an on-line manner, as the simulating advice machine has only a constant (not time-varying) number of input ports through which it can read the incoming input stream. Therefore, prior to simulation, the stream of data entering the internet machine must be 'sequentialized'. This is done by sequentially reading for each time t (viewed from the perspective of the internet machine) all data that are entered into all sites machine from the list $\alpha(t)$ at that time. Only then the simulating sequential machine can update the instantaneous descriptions of all machines in the set $\alpha(t)$ appropriately and proceed to the simulation of the next step of the internet machine. In addition to this, the simulating machine must keep a list of all messages that are transient among the site machines. Thanks to the knowledge of function β this is possible. For each time t the upgrades of the site machines at that time are recorded in the machine's advice. In fact, the advice contains for each t not only the values of $\gamma(t)$, but also those of $\alpha(t)$ and $\beta(i,j,t)$ for all $i,j \in \alpha(t)$. All in all, for each t the advice value encodes the complete list of all site machines participating at that time in the internet machine, the description of all their software, and all data concerning the current transfer times between each pair of site machines. In [48] one can find a proof of the following theorem:

Theorem 10. *For every internet machine $\mathcal{I} = (\alpha, \beta, \gamma)$ there exists an interactive Turing machine $\mathcal{A}$ with advice that sequentially realizes the same computation as $\mathcal{I}$ does.*

The reverse simulation is trivial since, as stated in theorem 8, a single site machine can simulate any interactive Turing machine with advice. Thus, it follows that internet machines and interactive Turing machines with advice have the same computational power. In [48] several further results are proved concerning time- and space-efficient simulations between resource-bounded internet machines on the one hand and Turing machines with advice on the other. The results place internet machines among the most powerful and efficient computational devices known in complexity theory.

5.2 Other Systems

We briefly mention some further support to the belief that interactive Turing machines with advice capture the intuitive notion of computation by non-uniform interactive systems. Consider evolutionary systems with a social behavior, such as human society. Cipra [11] recently quoted mathematicians of the Los Alamos National Laboratory in New Mexico as saying:

> *"...in a few decades, most complex computer simulations will predict not physical systems, which tend to obey well-known mathematical laws, but social behavior, which doesn't."*

Can there be suitable computational models for this?

Consider the case of a finite system of intelligent autonomous mobile agents that move around in some environment and exchange messages by whatever formalisable means: spoken language, via mobile phones, via the Internet, by ordinary mail, and so on. Occasionally, new agents appear and start to interact with their environment. Some agents may be killed by accident, some will not work properly, but all will die after some time. For some unpredictable reasons some agents may even behave better than any other previous ones. In general all agents are intelligent, which means that they are able to learn. In the course of their education new agents will successively become skilled, start to cooperate with other agents and perhaps invent new knowledge. The whole society will develop.

Under suitable assumptions behaviours like this can be seen as 'computations' of a dynamic system of interactive Turing machines with advice. The assumptions are that all agents behave as interactive algorithms (i.e. can be described as interactive Turing machines) and that their evolutionary upgrades and moving around in the environment as well as their interaction (their encounters or data concerning message delivery times) as well as the times of their birth and death are aptured in the respective advice. The similarity with the internet machine is obvious.

Of course, we are speaking about a possibility in principle. We do not claim that one will ever be able to do such a simulation of the existing world. But, on the other hand, what prevents it, theoretically? In the virtual world modeled in a computer a virtual society could develop as envisioned above. One could monitor the respective development and could make non-computable interventions into the underlying process. No doubt the respective computation would then be equivalent to the one of an interactive Turing machine with advice, in much the same sense as described above.

6 The Extended Church-Turing Thesis

In this paper we have explored the frontiers of the notion of computation as we now know it. Our results do *not* aim to attack the Church-Turing thesis. We have merely tried to identify its proper extension to cover computations that share the following features, which have emerged in modern computing: *non-uniformity of programs*, *interaction of machines*, and *infinity of operation*. We have argued that the proper extension is provided by *interactive Turing machines with advice* and have given evidence to support it. We have shown this primarily with the help of site and internet machine models used in an interactive mode, mimicing the potentially endless life-cycle of real computers and networks, including their unlimited software and hardware upgrades. If the life-span of these machines is taken to infinity, then the computational equivalence of the respective machines with interactive Turing machines with advice is a mathematical consequence. If the life-span of the machines at hand is bounded, any finite portion of the computation will remain within the scope of standard Turing machines and the standard Church-Turing thesis.

Clearly, one cannot expect to prove the extended Church-Turing thesis. It can only be disproved by pointing to some kind of computations that cannot be described as computations of interactive Turing machine with advice. In fact, this has happened to the classical thesis which did not cover the case of computations of advice Turing machines (proposition 5). As long as such computations (of advice Turing machines) are 'not observed' in practice, the thesis is preserved. Once the reality of non-uniform interactive computation is accepted, the original thesis is in question. Consequently, the underlying classical Turing paradigm has to be changed to the paradigm of interactive Turing machines with advice. Examples such as site machines, internet machines, and evolutionary societies of mobile agents indicate that such systems are within reach.

The computations of advice Turing machines, interactive or otherwise, are indeed more powerful than computations of standard Turing machine without any advice. Do the results in this paper mean that now we are able to solve some concrete, a priori given undecidable problems with them? The answer is negative. What we have shown is that some computational evolutionary processes which by their very definition are of an interactive and unpredictable nature, can be modeled *a posteriori* by interactive Turing machines with advice. In principle, observing such a running process in sufficient detail we can infer only a posteriori, after we noted them, all its computational and evolutionary characteristics (such as the time and the nature of their changes).

In recursion theory and in complexity theory, several computational models are known that do not obey the Church-Turing thesis. As examples, oracle Turing machines, non-uniform computational models such as infinite circuit families [5], models computing with real numbers (such as the so-called BSS model [7]), certain physical systems [30], and variants of neural [38] and neuroidal networks [52] can be mentioned. Yet none of these is seen as violating the Church-Turing thesis. This is because none of them fits the concept of a *finitely describable algorithm* that can be mechanically applied to data of arbitrary and potentially unbounded size. For instance, in oracle Turing machines the oracle presents the part of the machine that in general has no finite description. The same holds for the infinite families of (non-uniform) circuits, for the real numbers operated on by the BSS machine or analog neural nets. So far no finite physical device was found that could serve as a source of the respective potentially unbounded (non-uniform) information [13].

Nevertheless, from this point of view, site and internet machines seem to present an interesting case: at each point of time they have a finite description, but when seen in the course of time, they represent infinite non-uniform sequences of computing devices, similar to non-uniform circuit families. What makes them non-fitting under the traditional notion of algorithms is their potentially endless evolution in time. This includes both interaction and non-uniformity aspects. This gives them the necessary infinite non-uniform dimension that boosts their computational power beyond that of standard Turing machines.

The respective results point to the fact that even our personal computers, not speaking about the Internet, offer an example of a real device that can perform computations that no single interactive Turing machine (without an oracle) can. We, as individual agents operating and updating the sites, have only a local influence on what is computed by them. Our activities, spread over time, result in a unique computation that cannot be replicated by any finite computing device given a priori.

References

1. H. Abelson, D. Allen, D. Coore, C. Hansen, G. Homsy, Th.F. Knight Jr, R. Nagpal, E. Rauch, G.J. Sussman, and R. Weiss. Amorphous computing, *Comm. ACM* 43-5 (2000) 74-82.
2. L. Adleman. Two theorems on random polynomial time, in: *Proc. 19th Ann. IEEE Symp. Foundations of Computer Science* (FOCS'78), 1978, pp. 75-83.
3. I. Antoniou *et al.* Future themes in theoretical computer science - Conclusions from a discussion meeting, *Bulletin of the EATCS*, Nr 71, June 2000, pp. 8-14.
4. H. Attiya, J. Welch. *Distributed computing: fundamentals, simulations and advanced topics*, McGraw-Hill Publ. Comp., London, 1998.
5. J.L. Balcázar, J. Díaz, and J. Gábarro. *Structural complexity*, Vol. I, 2nd edition, Springer-Verlag, Berlin, 1995.
6. J.L. Balcázar, M. Hermo. The structure of logarithmic advice complexity classes, *Theor. Comput. Sci.* 207 (1998) 217-244.
7. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity of real computation*, Springer-Verlag, New York, NY, 1998.
8. E. Börger. *Computability, complexity and logic*, Studies in Logic and the Foundation of Mathematics, Vol. 128, North-Holland, Elsevier Science Publ., Amsterdam, 1989.
9. L. Cardelli. Global computation, *ACM Sigplan Notices* 32-1 (1997) 66-68.
10. K.M. Chandy, J. Misra. *Parallel program design: a foundation*, Addison-Wesley Publ. Comp., Reading, MA, 1988.
11. B. Cipra. Revealing uncertainties in computer models, *Science* 287 (2000) 960-961.
12. B.J. Copeland. The Church-Turing thesis, in: E.N. Zalta, *Stanford Encyclopedia of Philosophy*, Center for the Study of Language and Information (CSLI), Stanford University, Stanford, CA, http://plato.stanford.edu/entries/church-turing/, 1997.
13. B.J. Copeland, D. Proudfoot. Alan Turing's Forgotten Ideas in Computer Science, *Scientific American* April 1999, p.77-81.
14. D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer, Proc. Royal Soc. London A 400 (1985) 96-117.
15. H.B. Enderton. Elements of recursion theory, in: J. Barwise (Ed.), *Handbook of Mathematical Logic*, North-Holland, Elsevier Science Publ., Amsterdam, 1977, Chapter C.1, pp. 527-566.
16. I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure*, Morgan-Kaufmann Publ., San Francisco, 1998.
17. D.Q. Goldin. Persistent Turing machines as a model of interactive computation, in: K-D. Schewe and B. Thalheim (Eds.), *Foundations of Information and Knowledge Systems*, Proc. First Int. Symposium (FoIKS

2000), Lecture Notes in Computer Science, vol. 1762, Springer-Verlag, Berlin, 2000, pp. 116-135.
18. J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Company, Reading, Mass., 1979.
19. R.M. Karp, R.J. Lipton. Some connections between nonuniform and uniform complexity classes, in: *Proc. 12th Annual ACM Symposium on the Theory of Computing* (STOC'80), 1980, pp. 302-309.
20. N.A. Lynch. *Distributed algorithms*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1996.
21. R. Milner. *A calculus of communicating systems*, Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin, 1980.
22. R. Milner. Elements of interaction, *C.ACM* 36:1 (1993) 78-89.
23. M.L. Minsky. *Computation: finite and infinite machines*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1969.
24. B. Molzan. Expressibility and nonuniform complexity classes, *SIAM J. Comput.* 19 (1990) 411-423.
25. B.M. Moret. *The theory of computation*, Addison-Wesley, Reading, MA, 1998.
26. P. Odifreddi. *Classical recursion theory – The theory of functions and sets of natural numbers*, North-Holland/Elsevier Science Publishers, Amsterdam, 1989.
27. N. Pippenger. On simultaneous resource bounds, in: *Proc. 20th Ann. IEEE Symp. Foundations of Computer Science* (FOCS'79), 1979, pp. 307-311.
28. A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends, in: J.W. de Bakker, W.-P. de Roever and G. Rozenberg, *Current Trends in Concurrency*, Lecture Notes in Computer Science, Vol. 224, Springer-Verlag, Berlin, 1986, pp. 510-585.
29. A. Pnueli. Specification and development of reactive systems, in: H.-J. Kugler (Ed.), *Information Processing 86*, Proceedings IFIP 10th World Computer Congress, Elsevier Science Publishers (North-Holland), Amsterdam, 1986, pp. 845-858.
30. M.B. Pour-El. The Structure of computability in analysis and physical theory: an extension of Church's thesis, in: E.R. Griffor (Ed.), *Handbook of Computability Theory*, Studies in Logic and the Foundation of Mathematics, Vol. 150, Elsevier, Amsterdam, 1999, Chapter 13, pp. 449-471.
31. M. Prasse, P. Rittgen. Why Church's Thesis still holds. Some notes on Peter Wegner's tracts on interaction and computability, *The Computer Journal* 41 (1998) 357-362.
32. P. Pudlák. Complexity theory and genetics, *Structure in Complexity Theory*, Proc. Ninth Annual Conference, IEEE Computer Society Press, Los Alamitos, Ca., 1994, pp. 383-395.
33. H. Rogers. *Theory of recursive functions and effective computability*, McGraw-Hill, New York, 1967.
34. J.E. Savage. *Models of computation: exploring the power of computing*, Addison-Wesley, Reading, MA, 1998.
35. U. Schöning. On small generators, *Theor. Comput. Sci.* 34 (1984) 337-341.
36. U. Schöning. Complexity theory and interaction, in: R. Herken (Ed.), *The universal Turing machine - A half-century survey*, Oxford University Press, Oxford, 1988, pp. 561-580.

37. J.C. Shepherdson. Mechanisms for computing over arbitrary structures, in: R. Herken (Ed.), *The universal Turing machine - A half-century survey*, Oxford University Press, Oxford, 1988, pp. 581-601.
38. H.T. Siegelmann. Computations beyond the Turing limit, *Science* 268 (1995) 545-548.
39. L. Staiger. ω-Languages, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, Chapter 6, Springer-Verlag, Berlin, 1997, pp. 339-387.
40. G. Tel. *Introduction to distributed algorithms*, Cambridge University Press, Cambridge (UK), 1994.
41. W. Thomas. Automata on infinite objects, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B: *Models and Semantics*, Elsevier Science Publishers, Amsterdam, 1990, pp. 135-191.
42. W. Thomas. Languages, automata, and logic, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, Chapter 7, Springer-Verlag, Berlin, 1997, pp. 389-455.
43. A.M. Turing. On computable numbers, with an application to the *Entscheidungsproblem*, Proc. London Math. Soc., 42-2 (1936) 230-265; A correction, ibid., 43 (1937) 544-546.
44. A.M. Turing. Systems of logic based on ordinals, *Proc. London Math. Soc.* Series 2, 45 (1939) 161-228.
45. A.M. Turing. Computing machinery and intelligence, *Mind* 59 (1950) 433-460.
46. 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.
47. 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. 99-113.
48. J. van Leeuwen, J. Wiedermann. Breaking the Turing barrier: the case of the Internet, Techn. Report, Inst. of Computer Science, Academy of Sciences of the Czech Rep., Prague, 2000.
49. P. Wegner. Why interaction is more powerful than algorithms, *C.ACM* 40 (1997) 80-91.
50. P. Wegner. Interactive foundations of computing, *Theor. Comp. Sci.* 192 (1998) 315-351.
51. P. Wegner, D.Q. Goldin. Interaction, computability, and Church's thesis, *The Computer Journal* 2000 (to appear).
52. 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 Artif. Intelligence, Vol 1720, Springer-Verlag, Berlin, 1999, pp. 63-76.
53. C.K. Yap. Some consequences of non-uniform conditions on uniform classes, *Theor. Comput. Sci.* 26 (1983) 287-300.