Technical Report No. 2005-492
THE MYTH OF UNIVERSAL COMPUTATION*

Selim G. Akl
School of Computing
Queen's University
Kingston, Ontario, Canada K7L 3N6

January 31, 2005

Abstract

It is shown that the concept of a Universal Computer cannot be realized. Specifically, instances of a computable function $\mathcal{F}$ are exhibited that cannot be computed on any machine $\mathcal{U}$ that is capable of only a finite and fixed number of operations per step. This remains true even if the machine $\mathcal{U}$ is endowed with an infinite memory and the ability to communicate with the outside world while it is attempting to compute $\mathcal{F}$. It also remains true if, in addition, $\mathcal{U}$ is given an indefinite amount of time to compute $\mathcal{F}$. This result applies not only to idealized models of computation, such as the Turing Machine and the like, but also to all known general-purpose computers, including existing conventional computers (both sequential and parallel), as well as contemplated unconventional ones such as biological and quantum computers. Even accelerating machines (that is, machines that increase their speed at every step) are not universal.

Keywords: Universal computer, Turing machine, simulation, parallel computation, superlinear speedup, real-time computation, uncertainty, geometric transformation.

1 Introduction

Universal computation, which rests on the principle of simulation, is one of the foundational concepts in computer science. Thus, it is one of the main tenets of the field that any computation that can be carried out by one general-purpose computer can also be carried out on any other general-purpose computer. At times, the imitated computation, running on the second computer, may be faster or slower depending on the computers involved. In order to avoid having to refer to different computers when conducting theoretical analyses, it is a generally accepted approach to define a model of computation that can simulate all computations by other computers. This model would be known as a Universal Computer $\mathcal{U}$. Without doubt, the most well-known contender for the title of $\mathcal{U}$ is the Turing Machine.

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

It is widely believed that if a computation can be carried out on any model of computation, then it can be carried out on the Turing Machine. Conversely, it is also believed that if a computation cannot be performed by a Turing Machine then it cannot be computed at all (regardless of what model one uses). In recent years, however, it has become obvious to a growing number of computer scientists that the Turing Machine cannot in fact capture the entire spectrum of new applications of computers. In order to remedy this situation several models were proposed to replace the Turing Machine as a universal model of computation. These include models that communicate with the outside world during their computations, models that appeal to the laws of physics to perform a huge numbers of operations every time unit, models that can manipulate real numbers, parallel computers, computers that defy the limits of what is possible (such as accelerating machines, which are those computers that double their speed at every step), and so on.

Our premise here is quite simple. Suppose there exists a Universal Computer capable of n elementary operations per step, where n is a finite and fixed integer. This computer will fail to perform a computation *requiring* n' operations per step, for any $n' > n$, and consequently lose its claim of universality. Naturally, for each $n' > n$, another computer capable of n' operations per step will succeed in performing the aforementioned computation. However, this new computer will in turn be defeated by a problem requiring $n'' > n'$ operations per step.

The contribution of this paper is twofold. First we show that all known finite computational models, including the Turing Machine, are inadequate as universal models of computation. Then we prove that, in fact, the notion of a finite Universal Computer is itself absurd. As outlined above, this is accomplished by exhibiting problems that can be solved by a machine capable of executing n operations at every step, but these problems are not solvable by any machine capable of at most $n - 1$ operations per step, for any integer $n > 1$. Of course, n is a variable, rendering the notion of a finite Universal Computer meaningless. In other words, if the purported finite Universal Computer has been defined as capable of a (finite and fixed) number n of operations per step, then that computer will fail to compute a function requiring $n + 1$ operations per step. This result applies not only to idealized models of computation, such as the Turing Machine and the like, but also to all known general-purpose computers, including existing conventional computers (both sequential and parallel), as well as contemplated unconventional ones such as biological and quantum computers. Even accelerating machines are not universal.

This paper is intended to present an existence proof. We provide three examples of computations that cannot be performed by any finite machine, regardless of how much time it is allowed to operate. This remains true even if the finite machine is allowed to *interact* with the outside world. In fact, our examples allow the machine to receive input from, and deliver output to, its environment *during* the computation, *as well as* to compute without deadline; it still fails to perform the computations. Of course, one counterexample to the notion of a Universal Computer would have sufficed. However, we felt that by offering three distinct examples, with actual computations to support the theoretical set-up, we would demonstrate that our result is not an isolated curiosity.

The remainder of this paper is organized as follows. Section 2 introduces the idea of universal computation by first defining the Universal Computer $\mathcal{U}$, as a general model, then presenting the Turing Machine as the most widely accepted embodiment of a Universal Computer. Models previously proposed as capable of performing computations that cannot be carried out on a Turing Machine are reviewed briefly in Section 3. Three computational problems are described in Section 4. None of these problems can be solved on a computer that obeys the *finiteness condition*, that is, a computer capable of only a finite and fixed number of operations per step. Computers obeying the finiteness condition include all 'reasonable' models of computation, both theoretical and practical, such as the Turing Machine and all of today's existing and contemplated general-purpose computers. The first of these computational problems involves a set of independent time-varying variables. In the second problem a function is to be evaluated over a set of physical variables that affect one another. The third problem requires a transformation to be performed on a geometric object while obeying a certain mathematical condition throughout the computation. The failure to solve these problems by any computer that obeys the finiteness condition (regardless of whether it has an infinite memory, is capable of communicating with the outside world while computing, or is allowed to compute for as long as it needs), leads us to conclude in Section 5 that the concept of the Universal Computer cannot be realized (neither in theory nor in practice) so long as the finiteness condition holds. Some final remarks are offered in Section 6.

2 Universal Computation

We begin by providing a high-level description of $\mathcal{U}$, a Universal Computer. This description is intended to be sufficiently general to encompass any particular manifestation of such a computer. This is followed by a formal presentation of the most famous of all candidates for a Universal Computer, namely, the Turing Machine.

2.1 The universal computer

A Universal Computer $\mathcal{U}$ is defined as a computing system with the following capabilities:

1. A means of communicating with the outside world with the purpose of receiving input and producing output, at any time during a computation.
2. The ability to perform all elementary arithmetic and logical operations (such as addition, subtraction, logical AND, and so on).
3. A program made up of basic input, output, arithmetic, and logical operations.
4. An unlimited memory in which the program, the input, intermediate results, and the output are stored and can be retrieved.

Furthermore, $\mathcal{U}$ obeys the ***finiteness condition***: In one *step*, requiring one *unit of time*, $\mathcal{U}$ can execute a finite and fixed number of basic operations. Specifically, it can:

1. Read a finite and fixed number of finite and fixed-sized inputs;

2. Perform a finite and fixed number of elementary arithmetic and logical operations on a finite and fixed number of finite and fixed-sized data;

3. Return a finite and fixed number of finite and fixed-sized outputs.

We consider this to be a 'reasonable' model of computation. What makes computer $\mathcal{U}$ 'universal' is its supposed ability to simulate any computation performed on any other model of computation: Anything that can be computed on some model, can be computed on $\mathcal{U}$. There is no bound on the number of steps that $\mathcal{U}$ can perform to solve a problem; a simulation can run for as long as required.

2.2 The Turing Machine as Universal Computer

> "As far as we know, no device built in the physical universe can have any more computational power than a Turing machine. To put it more precisely, any computation that can be performed by any physical computing device can be performed by any universal computer, as long as the latter has sufficient time and memory." [47]

The Turing Machine was proposed by Alan Turing in 1936 in his famous paper [81]. The following description follows from the treatment of Turing machines in [56].

2.2.1 Formal definition

A Turing Machine consists of three physical components:

1. A **control unit** that can be in any one of a finite set of states $K = \{q_0, q_1, \ldots, q_{v-1}\}$. There are two special states: the *initial state* $s \in K$, and the *halt state* $h \notin K$.

2. An **input/output tape** that extends, from one fixed end at the left, indefinitely to the right. The tape is divided into squares, each of which capable of holding one symbol (of finite and fixed-size) from a finite alphabet $\Sigma = \{a_0, a_1, \ldots, a_{w-1}\}$. One of the elements of Σ is the blank symbol #.

3. A **read/write head** allows the control unit and the input/output tape to communicate. The head can read/write a symbol from/to the tape and move itself either one tape square to the left (L) or one tape square to the right (R).

A computation by the Turing Machine proceeds in discrete steps. During each step, a finite number of operations (precisely two!) are executed. Depending on the current state q_i of the control unit and on the symbol a_j written on the tape square at which the head is presently positioned,

1. The control unit enters a new state (possibly staying in q_i),

2. The read/write head

   (a) either writes a new symbol in the tape square at which it is positioned (possibly writing a_j again),

   (b) or executes one of the L or R moves.

Formally, a Turing Machine is a quadruple

$$(K \cup \{h\}, \Sigma \cup \{L, R\}, \delta, s),$$

where δ is a function from $K \times \Sigma$ to $(K \cup \{h\}) \times (\Sigma \cup \{L, R\})$. The function δ describes the operation of the machine at each step; thus $\delta(q_i, a_j)$ is equal to one of (q_k, a_ℓ), (q_k, L), or (q_k, R), for some $q_k \in K \cup \{h\}$ and $a_\ell \in \Sigma$.

The Turing Machine receives its input on tape squares beginning at the left end of the tape. It produces its output also on the tape. When the control unit enters state h, the computation terminates.

Suppose we want to compute the product 3×5. We would put the two strings 111 and 11111 on the tape and set up the δ function so that the Turing Machine reads the two inputs and writes on the tape three adjacent copies of 11111, producing as output 111111111111111.

It is important to note the following crucial properties that make the Turing Machine a 'reasonable' model of computation:

1. The cardinalities of the sets K and Σ, the size of each symbol in Σ, and the number of operations performed during each step, are all *finite and fixed*. (The size of a symbol is finite and fixed in order to prevent an entire–possibly infinite–computation from being encoded in a single symbol. The size of a symbol may be measured, for example, by the number of 0s and 1s used in a binary representation of that symbol.)

2. The only part of the model that is not finite is the input/output tape. This is understandable, since all intermediate calculations between reading the input and producing the output are written on the tape. Owing to the fact that for many computations it is impossible to predict the number of steps required to produce the output (in fact, some computations may never terminate), the number of tape squares is assumed to be infinite.

The reader will have noticed that, unlike the Universal Computer defined in Section 2.1, the Turing Machine does not have the means to communicate with the outside world, particularly during a computation. Furthermore, there is no explicit mechanism for placing the input on the tape initially, nor is there any cost (for example, running time) incurred for this operation. The input is miraculously 'there' when the computation begins.

2.2.2 Universality thesis

While fairly simple conceptually, the Turing Machine is a truly powerful model of computation. So powerful in fact, that it was believed until recently that no model more powerful than the Turing Machine can possibly exist (in other words, a model that would be able to perform computations that the Turing Machine cannot perform). This belief is captured in the following statement, known as the

> **Church-Turing Thesis**: Any computable function can be computed on a Turing Machine [89, 70].

The statement may sound like a tautology, so let's reword it: If a function f can be computed 'somehow' on 'some model of computation', then it is always possible to compute f on a Turing Machine. Originally stated by Church [29] and Turing [81] independently, the thesis appears to have evolved and there are many ways of formulating it [30, 40, 41, 55, 59]. (Originally, the Turing Machine was meant to be 'universal' in the sense that it could imitate any computation performed by a *human* mathematician who was following an algorithm. The evolution of the 'thesis' from its modest origins to its current elevated status is traced in [30].)

The statement is a 'thesis' and not a theorem because of the difficulty of formally defining what it means "to compute". However, it was generally felt that the statement could have as well been called a theorem based on the following overwhelming evidence:

1. Every conceivable computation was shown to be executable by a Turing Machine.
2. Every attempt to extend the Turing Machine (by adding more features such as, for example, multiple tapes, multiple heads, a doubly infinite tape, even the ability to behave nondeterministically, and so on) resulted in a model equivalent to the Turing Machine.

But how does one show that all known computations can be carried out by a Turing Machine? This is usually done quite simply by appealing to a very important idea that permeates the science of computing, namely, the

Principle of Simulation: Any computation by some model A can be simulated on a Turing Machine.

The idea is that the Turing Machine can (painstakingly and excruciatingly slowly, for sure, but successfully nevertheless) replicate all the steps performed by model A. In other words, *given enough time*, the Turing Machine can compute *anything that can be computed*. As a model of computation, the Turing Machine may not be *efficient*, but it is certainly *effective*. As put by Lewis and Papadimitriou [56]:

" ... as primitive as Turing machines seem to be, attempts to strengthen them seem not to have any effect. ... Thus any computation that can be carried out on the fancier type of machine can actually be carried out on a Turing machine of the standard variety. ... *any*

way of formalizing the idea of a 'computational procedure' or an 'algorithm' is equivalent to the idea of a Turing Machine."

In some sense, using the Simulation Principle, the Church-Turing thesis implies a definition of computability. If the Church-Turing thesis is correct, then a function is computable if and only if it is computable by a Turing Machine.

For all the reasons presented, the Turing Machine has been called a *universal model of computation*. (Technically, a *Universal Turing Machine* is a general Turing Machine that can simulate any special-purpose Turing Machine.) Formal grammars, μ-recursive functions, and a variety of different models were shown to be equivalent to the Turing Machine [56, 70, 89].

2.2.3 Limitations

Careful examination of the definition of the Turing Machine reveals that the model is actually capable only of evaluating a restricted set of functions. In fact, the first person to recognize the limitations of the Turing Machine appears to have been Turing himself. Indeed, Turing proposed a number of variants to the *automatic machine* (or *a*-machine, the name he gave to what is now known as the Turing Machine), in an attempt to extend the model. These variants included the *choice machine* (or *c*-machine) [81] and the *unorganized machine* (or *u*-machine) [83], neither of which seems to have survived later scrutiny. Most intriguing, however, is the *o*-machine that he described in his Ph.D. thesis of 1938 (supervised by Church), and later published as [82] (see also [34]). An *o*-machine is a Turing Machine augmented with an oracle; the latter is capable of returning the value of a function that cannot be computed on the Turing Machine (such as a function which determines whether a given arbitrary computation terminates, also known as the *halting function*). This is clearly a machine more powerful than the Turing Machine; its fate, nevertheless, has been no better than other variants proposed by Turing.

The Church-Turing thesis (as interpreted today) implies that there does not exist a computable function that *cannot* be computed by a Turing Machine. This thesis has had its fair share of critics (by far outnumbered, however, by staunch defenders of the thesis) [79]. It has been argued, often successfully, that the Turing Machine is not a Universal Computer. Several examples of computations have been exhibited that cannot be performed by the Turing Machine. Sometimes these computations are referred to as *super-Turing computations* [31, 72, 78], a restricted subset of which are called *hypercomputations* [21, 32, 35, 49, 80]. We review this work in Section 3. (It is interesting to note that the word 'myth' appearing in the title of this paper, has been used equally by both sides of the debate; see, for example, [39, 43, 45] for differing views.)

3 Previous Work

> "It is theoretically possible, however, that Church's Thesis could be overthrown at some future date, if someone were to propose an alternative model of computation that was publicly acceptable as fulfilling the requirement of 'finite labor at each

step' and yet was provably capable of carrying out computations that cannot be carried out by any Turing machine. No one considers this likely. " [56]

Since the early 1960s, some researchers began to question the adequacy of the Turing Machine as a universal model of computation. Over the years, two types of results were obtained:

1. Unconventional computations were exhibited that can be computed on other models of computation, but not on the Turing Machine.

2. Conventional problems believed to be *undecidable* (that is, unsolvable, or uncomputable), because they had been proven not to be computable on the Turing Machine, were shown to be decidable on other models of computation.

One distinguishing characteristic of these results, is the willingness of the researchers, whose work we briefly describe in this section, to address the thorny question referred to in Section 2.2.2: What does it mean "to compute"? In answer, they offer a broad perspective. Specifically, *computation* is viewed as a process whereby information is manipulated by, for example, acquiring it (input), transforming it (calculation), and transferring it (output). They describe instances of processes, each of which is a *computation*, including,

1. Measuring a physical quantity,

2. Performing a basic arithmetic operation on a pair of numbers, and

3. Setting the value of a physical quantity,

to cite but a few. Of these, only the second represents what we could call a conventional computation. The other two are referred to as *unconventional computations*, and arise in such diverse applications as data acquisition in signal processing [66], and the control of nuclear power plants [19]. Furthermore, each of these computational processes may itself be carried out by a variety of means, including, of course, conventional (electronic) computers, but also through physical phenomena in the quantum [22] and optical [57] realms, through chemical reactions [74], and through transformations in living biological tissue [1].

3.1 Interacting computing agents

One of the shortcomings of the Turing Machine pointed out early on, is its inability to simulate several communicating computing agents. A model, seemingly able to do just that, namely, the Actors Model, was proposed as an alternative [46]. A related criticism of the Turing Machine is its inability to interact with the outside world (see, for example, [45, 84, 85, 86] and the references therein). As stated in [85] (where TM stands for Turing Machine): "*Turing machines* are finite computing agents that noninteractively transform input into output strings by sequences of state transitions. TM computations for a given input are history-independent and reproducible, since TMs always start in the same initial state and

shut out the world during computation." It is then argued in [85] that "interactive finite computing agents" are more powerful than Turing Machines, as they are capable of receiving input from, and producing output to, the outside world throughout the computation. Similar ideas are articulated in [36, 58, 61]. These approaches present a strong and convincing case against the Turing Machine as a Universal Computer. It may be said in defense of the Turing Machine that both the idea of interaction among agents, as well as interaction with the outside world, can be simulated using a number of independent Turing Machines operating in sequence. Specifically, the computations occurring between two interactions are carried out each time by a completely new Turing Machine. In other words, whenever an input is received, the previous Turing Machine halts, and a new Turing Machine takes over from that point on. At no point are two machines operating simultaneously. It is far from clear, however, how the transition from one machine to the next would be executed.

3.2 Physical realizations

Certain physical realizations of computers have also been proposed as possible demonstrations of models more powerful than the Turing Machine. Some of the most exciting ideas come from Physics and Biology. Thus, in theory at least, *quantum computation* promises to offer great advantages when it comes to solving certain computational problems [18, 26, 25, 62]. Because of their ability to perform an enormous number of operations simultaneously, while requiring very little in terms of hardware, quantum computers can solve extremely quickly problems that are infeasible to solve on conventional electronic computers. Two computations, in particular, namely, factoring large numbers and searching large databases, can be readily solved on a quantum computer while requiring a prohibitive time on a classic computer. This has led some to argue that the quantum computer is 'more powerful' than the Turing Machine (see, for example, [22, 50] and the references therein). This claim, however, must be qualified. Strictly speaking, a Turing Machine will, given enough time, solve any problem solvable on a quantum computer. The quantum algorithms involve exhaustive enumeration of cases. There is nothing, except our patience (or lack of it!), that impedes the Turing Machine *in principle*, preventing it from arriving at the same answer as the quantum computer (much later in time). However, while the claim of the superiority of the quantum computer does not hold in theory, one could say, in fairness, that the claim does makes sense in practice (the Turing Machine may take longer than the age of the Universe).

Another claim, in the same vein, was made in [68]: It is impossible to use models of computation, in general, and the Turing Machine, in particular, in order to arrive at a full understanding of the functioning of the human brain; instead, our conscious mentality can be explained (only?) by quantum physics. It is an interesting open question whether this claim could be verified in the near future.

In [28] it is argued that a quantum computer, using a probabilistic approach, can evaluate a function that a Turing machine eminently cannot, the notorious *halting function*. The quantum computer, through sampling, can predict with a probability approaching 1, whether or not a program running on its input will halt (see also [87, 88]). Another computation

quantum computers might be capable of carrying out, if ever built, is to generate *true random numbers*, as opposed to *pseudo-random numbers* generated by functions. This is a feat that a Turing machine has no way of accomplishing. (We note in passing that obtaining true random numbers is something conventional computers of today can do routinely by reading values of a variety of physical parameters from their environment; see [69] for a discussion.) These results should represent sufficient proof that the Turing machine is not a Universal Computer.

Finally, current research may show some day that certain unconventional models, such as *biological* [27, 38] and *optical* [67] computers, are more powerful than the Turing Machine.

3.3 Pushing the limits of what's possible

In the same spirit that seemed to have motivated those who extended the Church-Turing thesis from its modest original claim to its current overarching status in computing, it is generally believed today that the *halting function* is uncomputable on any model of computation. (In fact, Turing showed only that the *halting function* is uncomputable on the Turing Machine; but, of course, if one assumes that the Turing Machine is 'universal', it is easy to see how the uncomputability of the *halting function*, on all models, must follow.) As it turns out, a very simple and quite ingenious proof shows how a Turing Machine, extended with the ability to perform each step in half the time taken by the previous step, can solve the halting problem in *two time units*! This proof holds even if the computation being checked for termination does *not* terminate, indeed, the sum of the terms 2^{-i}, for $i = 0, 1, \ldots, \infty$, is 2 [31]. The origins of this approach are traced in [26, 27, 33, 76, 77] along with a discussion of its validity. It is debatable whether the assumption of an accelerating machine is a reasonable one; however, the argument is clearly sound from a logical point of view. In fact, a proposal is made in [27] for implementing this idea using living cells.

Another way of solving the halting problem, that appeals to so-called X-machines [42], is proposed in [75]. Unlike *discrete* models of computation (such as the Turing Machine and all *digital* computers), X-machines are *analog* systems capable of behaving like *continuous* functions. For example, they can explore the entire *infinite* interval $[0, 1]$. Thus, in order to determine whether or not a certain program halts when running on its input, the (discrete) steps of the computation are mapped to the (continuous) interval $[0, 1]$. The X-machine then scans that interval in the same way as a ball traverses a gently sloped plane.

Somewhat related, is a chaotic dynamical system presented in [72, 73], that models the behavior of neural networks and analog machines. The model is shown to be more powerful than the Turing Machine. This power depends primarily on the system's assumed ability to represent and manipulate *real* numbers of infinite precision. Though no one knows for sure, perhaps this is the way computations occur in the natural world (that is, using infinite precision reals). See also [20] for an extension of the Turing Machine that allows computations with infinite real numbers.

A model of computation is described in [51] whose size is allowed to increase as a polynomial in the size of the input. Specifically, the solution to a problem of size n is computed using an acyclic interconnection of $p(n)$ digital components. This model, reminiscent of the-

oretical models of parallel computation (see Section 3.4), is shown to be more powerful than the Turing Machine.

Finally, and along the same lines, classical general relativity theory is invoked in [44] to conjure up machines capable of "an infinitude of calculations in finite time", and thus able to perform computations that are uncomputable on the Turing Machine.

Note, however, that *none* of the machines in this section, capable of solving the Turing Machine halting problem (collectively referred to as *hypercomputers*), can compute its own halting function [65]. The same is true for quantum hypercomputers [52], and adiabatic quantum hypercomputers [53, 54, 48]. It is important to point out that this does *not* show that these hypercomputers are not universal (where universality means the ability to simulate any computation possible on another machine). What it merely proves is the existence of problems that are *uncomputable*, even on hypercomputers. By contrast, we demonstrate in this paper that hypercomputers are not universal: When the finiteness condition holds, there indeed exist problems that impose a hierarchy among hypercomputers. For every problem size, these problems cannot be computed on a given machine, but are computable on a machine higher in the hierarchy.

3.4 Parallel computers

Some of the earliest examples of the inability of the Turing Machine to execute all possible computations used an entirely different approach. Beginning in [15], and continuing with [2, 23, 24] and [7, 8, 9, 10], *parallel computation* was used to exhibit two classes of computational problems that expose the limitations of the conventional sequential (that is, one-processor) models of computation in general (including the Turing Machine):

1. Problems for which an algorithm running on a parallel computer achieves a significant improvement in performance over an algorithm running on a sequential computer (for example, the Turing Machine). In each case, the improvement in the *speed of execution* or in the *quality of the solution*, obtained by a parallel computer with n processors operating simultaneously, is *superlinear in* n. Typically, if T_1 and T_n are the sequential and parallel running times, respectively, then $T_1/T_n = 2^n$. The same is true of the quality of the solution returned, where quality is defined according to the problem being solved (for example, quality might be *closeness to the optimal solution* in a combinatorial optimization problem [5, 11]). It is important to note that all the problems in this class are in fact solvable on a sequential computer. They include, for example, problems in cryptography [4, 13], numerical computation [12], and the evaluation of nonlinear feedback functions [6]. The point, however, is that superlinear performance is achieved in each case, either because the sequential computer is *unable to simulate the parallel computation* (in $n \times T_n$ time units), or simply because *the simulation does not make sense*.

2. Problems that can readily be solved in parallel but cannot be solved by a sequential computer (such as the Turing Machine). Here, the inability of the sequential computer to carry out the computation successfully is due to the circumstances in which

the computation is being performed. For example, it may be that there are certain human-imposed deadlines for processing the input and producing the output in a real-time computation, and these deadlines cannot be met by a sequential computer [8]. Alternatively, the computation may have to be performed in an environment that is under the control of the laws of nature, and no sequential computer can cope with the transformations undergone by the data during the computation [7]. Finally, the computation could be subject to certain mathematical constraints that dictate the result of every step, and no sequential computer can satisfy these conditions throughout the entire computation [9].

These two classes of problems showed that there are models more powerful than the Turing Machine and that, consequently, the latter is not a universal model of computation. Furthermore, all these computations share one important characteristic that has particular relevance to our subsequent discussion: If a problem in any of these two classes requires n processors in order to be solved as described, but at most $n - 1$ processors are available instead, then no gains whatsoever are obtained through parallel computation. Thus, with fewer than n processors, no speedup at all (let alone superlinear speedup) is achieved for the problems in the first class, and all problems in the second class are no longer solvable. This work culminated in the main result described in this paper.

4 Three Computational Problems

We begin by making two important points regarding our assumptions in the remainder of this paper. The first point concerns the definition of computation used in what follows. Traditionally, *to compute* has been defined as *to evaluate a function*: Given some initial input, the computer evaluates a function of this input and produces a final output. This definition, of course, has been challenged, as pointed out in Section 3 (see, for example, [2] and [86]). The argument has been presented that computation today (unlike the case perhaps only three decades ago) involves computing agents that have many properties not possessed by their ancestors. They can receive input from their environment, as well as other agents, during the computation, they can move freely and autonomously exploring a landscape virtually without bounds, and they can affect their environment by producing the outputs of their calculations to the outside world. Despite all this, and in order to make our case as simply yet as strongly as possible, we shall restrict ourselves here to the conventional (albeit narrow) interpretation of the nature of computation. This will avoid confusing the issue with additional assumptions (often valid, but not necessary in the present analysis). Thus, all of our examples will be restricted to the evaluation of functions. In addition, all the input is available at the outset and all the output need only be produced at the end of the evaluation.

The second point concerns the capabilities of the Turing Machine. In the remainder of this paper we will assume that the standard Turing Machine defined in Section 2.2 is extended in the following way. If external input is to be read onto the machine's tape at any

time during the computation, then this will be allowed, provided that: In one step (requiring one time unit) the Turing Machine can read *exactly one* finite and fixed-sized datum x_i onto its tape, possibly from among n data x_0, x_1, ..., x_{n-1}. (An n-tape Turing Machine can read the n data x_0, x_1, ..., x_{n-1}, in one step, requiring one time unit, one datum per tape.)

Now, consider the following three computational problems. For a positive integer n larger than 1, all three problems have in common the fact that they involve n functions, each of one variable, namely, F_0, F_1, ..., F_{n-1}, operating on the n variables x_0, x_1, ..., x_{n-1}, respectively. Specifically, all three problems call for the computation of $F_i(x_i)$, for $i = 0, 1, \ldots, n-1$. For example, $F_i(x_i)$ may be equal to x_i^2. The problems differ from one another with respect to the conditions imposed on the x_i, $0 \leq i \leq n-1$. Finally, once the $F_i(x_i)$ have been obtained, it is required to compute a simple function $\mathcal{F}$ of the form

$$\mathcal{F}(F_1(x_1), F_2(x_2), \ldots, F_{n-1}(x_{n-1})).$$

For example, $\mathcal{F}$ could be the sum of the $F_i(x_i)$, or their minimum, and so on.

4.1 Computation 1: Time-varying variables

In this computation, the x_i are themselves functions that vary with time. It is therefore appropriate to write the n variables as $x_0(t)$, $x_1(t)$, ..., $x_{n-1}(t)$, that is, as functions of the time variable t. It is important to note here that, while it is known that the x_i change with time, the actual functions that effect these changes are not known (for example, x_i may be a true random variable).

All the variables are available on an external input medium at the beginning of the computation but not stored in the memory of the computing device on which the computation is to be performed. Each variable $x_i(t)$ must therefore be read from the external input before $F_i(x_i(t))$ can be computed.

Time is divided into intervals, each of duration one time unit. It takes one time unit to read $x_i(t)$. It also takes one time unit to compute $F_i(x_i(t))$.

The problem calls for computing $F_i(x_i(t))$, $0 \leq i \leq n-1$. In other words, once all the variables have assumed their respective values at time t, and are available for reading at the beginning of the computation, the functions F_i are to be evaluated for all values of i.

4.1.1 Example 1: Space exploration in real time

On the surface of Mars n robots, R_0, R_1, ..., R_{n-1}, are roaming the landscape. The itinerary of each robot is unpredictable; it depends on the prevailing conditions in the robot's environment, such as wind, temperature, visibility, terrain, obstacles, and so on. At regular intervals, each robot R_i relays its current coordinates $x_i(t) = (a_i(t), b_i(t))$ to mission control on Earth. Given the coordinates of the n robots at time t, mission control determines the distance of R_i, $0 \leq i \leq n-1$, to a selected landmark $L(t)$ using a function F_i. (Similar examples were previously described in [2, 3, 8] and the references therein.)

4.1.2 Conventional solution

A Turing Machine fails to compute all the F_i as desired. Indeed, suppose that $x_0(t)$ is read initially and placed onto the tape. It follows that $F_0(x_0(t))$ can then be computed correctly (perhaps at a later time). However, when the next variable, x_1, for example, is to be read, the time variable would have changed from t to $t+1$, and we obtain $x_1(t+1)$, not $x_1(t)$. Continuing in this fashion, $x_2(t+2)$, $x_3(t+3)$, ..., $x_{n-1}(t+n-1)$ are read from the input. In Example 1, by the time $x_0(t)$ is read, robots R_1, R_2, ..., R_{n-1} would have moved away from $x_1(t)$, $x_2(t)$, ..., $x_{n-1}(t)$.

Since the function according to which each x_i changes with time is not known, it is impossible to recover $x_i(t)$ from $x_i(t+i)$, for $i = 1, 2, \ldots, n-1$. Consequently, this approach cannot produce $F_1(x_1(t))$, $F_2(x_2(t))$, ..., $F_{n-1}(x_{n-1}(t))$, as required.

4.2 Computation 2: Interacting variables

In this case, x_0, x_1, ..., x_{n-1}, are the variables of a physical system. They need to be measured in order to compute $F_i(x_i)$, $0 \leq i \leq n-1$.

The physical system has the property that all variables are related to, and depend on, one another. Furthermore, measuring one variable disturbs any number of the remaining variables unpredictably (meaning that we cannot tell which variables have changed value, and by how much). Typically, the system evolves until it reaches a state of equilibrium and, in the absence of external perturbations, it can remain in a stable state indefinitely.

Formally, the interdependence among the n variables can be modeled using n functions, $f_0, f_1, \ldots, f_{n-1}$, as follows:

$$\begin{aligned} x_0(t+1) &= f_0(x_0(t), x_1(t), \ldots, x_{n-1}(t)) \\ x_1(t+1) &= f_1(x_0(t), x_1(t), \ldots, x_{n-1}(t)) \\ &\vdots \\ x_{n-1}(t+1) &= f_{n-1}(x_0(t), x_1(t), \ldots, x_{n-1}(t)) \end{aligned} \tag{1}$$

These equations describe the evolution of the system from state $(x_0(t), x_1(t), \ldots, x_{n-1}(t))$ at time t to state $(x_0(t+1), x_1(t+1), \ldots, x_{n-1}(t+1))$, one time unit later. While each variable is written as a function of time, there is a crucial difference between the present situation and that in Section 4.1: When the system is in a state of equilibrium, its variables do not change over time. It is also important to emphasize that, in most cases, the dynamics of the system are very complex, so the mathematical descriptions of functions $f_0, f_1, \ldots, f_{n-1}$ are either not known to us or we only have rough approximations for them.

Assuming the system is in an equilibrium state, our task is to measure its variables (possibly to evaluate a global property of the system at equilibrium). In other words, we

need the values of $x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)$ at moment τ, when the system is in a stable state, in order to compute $F_i(x_i(\tau))$, $0 \leq i \leq n-1$.
We can obtain the value of $x_0(\tau)$, for instance, by measuring that variable at time τ (noting that the choice of x_0 here is arbitrary; the argument remains the same regardless of which of the n variables we choose to measure first). Although we can acquire the value of $x_0(\tau)$ easily in this way, the consequences for the entire system can be dramatic. Unfortunately, any measurement is an external perturbation for the system, and in the process, the variable subjected to measurement will be affected unpredictably.

Thus, the measurement operation will change the state of the system from $(x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$ to $(x'_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$, where $x'_0(\tau)$ denotes the value of variable x_0 after measurement. Since the measurement process has a non-deterministic effect upon the variable being measured, we cannot estimate $x'_0(\tau)$ in any way. Note also that the transition from $(x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$ (that is, the state before measurement) to $(x'_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$ (that is, the state after measurement) does not correspond to the normal evolution of the system according to its dynamics described by functions f_i, $0 \leq i < n$.

However, because the equilibrium state was perturbed by the measurement operation, the system will react with a series of state transformations, governed by equations (1). Thus, at each time step after τ, the parameters of the system will evolve either towards a new equilibrium state or maybe fall into a chaotic behavior. In any case, at time $\tau + 1$, all n variables have acquired new values, according to the expressions of functions f_i:

$$\begin{aligned} x_0(\tau+1) &= f_0(x'_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)) \\ x_1(\tau+1) &= f_1(x'_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)) \\ &\vdots \\ x_{n-1}(\tau+1) &= f_{n-1}(x'_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)) \end{aligned} \tag{2}$$

Consequently, unless we are able to measure all n variables, in parallel, at time τ, some of the values composing the equilibrium state

$$(x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$$

will be lost without any possibility of recovery.

4.2.1 Example 2: Biological laws of nature

In a laboratory, n living organisms are under observation in a closed environment that they share. The organisms depend on each other for survival. It is required to perform a test

on each of these organisms. Suppose that, in preparation for a test, the parameters of *one* organism are measured in exclusion of the others. This has the effect of disturbing the equilibrium sufficiently to provoke a serious adverse effect on several of the remaining organisms. (This example is a variant of [37]. The role played by the model of computation when the laws of nature interfere with the computation was first studied in [7]. Other examples, based on the laws of physics and dealing with dynamical systems, are presented in [14, 16, 17]. In [63, 64] the implications of parallelism in quantum information theory are studied and it is shown that a parallel approach can make the difference between success and failure when trying to distinguish between (entangled) quantum states.)

4.2.2 Conventional solution

Here, as in Section 4.1, the Turing Machine fails to compute the required F_i. Suppose that x_0 is measured first, this choice being arbitrary, and placed onto the tape. While this allows a correct evaluation of $F_0(x_0)$, this initial measurement affects any number of the remaining variables x_1, x_2, ..., x_{n-1} irremediably. Since we cannot recover the original values of x_1, x_2, ..., x_{n-1}, the computation of $F_i(x_i)$, for $i = 1, 2, \ldots, n-1$, is impossible. In Example 2, as the parameters of one organism are being measured, the remaining organisms may be altered irreparably, or may even die.

4.3 Computation 3: Variables obeying a global condition

Here the x_i, $0 \leq i \leq n-1$, are all available (they even *already* reside in memory). The present computation has three distinguishing properties:

1. All the x_i obey a certain global condition (a mathematical property). This condition must hold throughout the computation. If the condition happens to be violated at some point during the computation, the latter is considered to have failed.

2. Applying F_i to x_i produces a new value for x_i; thus,

$$x_i^{\text{new}} = F_i(x_i), \quad 0 \leq i \leq n-1.$$

3. Computing $F_i(x_i)$ for any *one* of the variables causes the global condition to no longer be satisfied.

4.3.1 Example 3: Computing subject to a geometric constraint

The object shown in in Fig. 1 is called a *convex subdivision*, as each of its faces is a convex polygon. This convex subdivision is to be transformed into that in Fig. 2. The transformation can be effected by removing edges and replacing them with other edges. The condition for a successful transformation is that each intermediate figure (resulting from a replacement) be a convex subdivision as well. There are n edges in Fig. 1 that can be removed and replaced with another n edges to produce Fig. 2 (where $n = 12$ for illustration). These

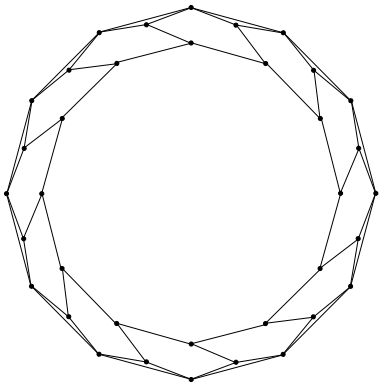

Figure 1: Original convex subdivision.

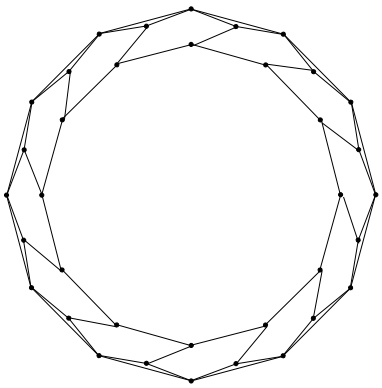

Figure 2: Destination convex subdivision.

are the 'spokes' that connect the outside 'wheel' to the inside one. However, as Fig. 3 illustrates, removing any *one* of these edges and replacing it with another creates a concavity, thus violating the condition [9, 60].

4.3.2 Conventional solution

With all the x_i on its tape, a Turing machine evaluates

$$x_0^{\text{new}} = F_0(x_0),$$

for example. This causes the computation to fail, as the set of variables x_0^{new}, x_1, x_2, ..., x_{n-1} does not obey the global condition. In Example 3, only one edge of the subdivision in Fig. 1 can be replaced at a time. Once any one of the n candidate edges is replaced, the global condition of convexity no longer holds.

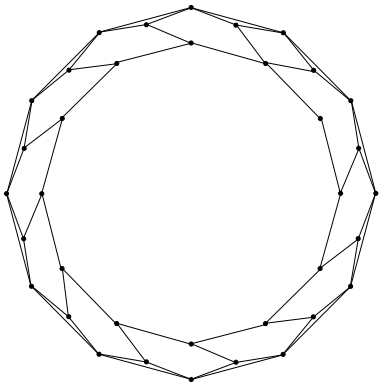

Figure 3: A subdivision with a concavity.

5 The Universal Computer Does Not Exist

In all three computations examined in Section 4, once an attempt was made to obtain an input or to evaluate one of the functions F_i, the input to each of the remaining functions became invalid. Because the Turing Machine (as defined quite reasonably by Turing) can only perform a finite and fixed number of operations at each step, it failed to carry out these computations. This remains true as long as the size of the input exceeds (even by one) the number of operations per step that the machine can perform, regardless of the size of the machine's memory, its ability to receive input and deliver output to the outside world during the computation, and the length of time it is allowed to run.

We now ask two questions in connection with the computations in Section 4.

5.1 Question 1: Are they computable?

The answer here is clearly *Yes*. For a given n, *any* computer capable of performing n operations per step, can easily:

1. Read all the $x_i(t)$ simultaneously (in computation 1)
2. Measure all the x_i simultaneously (in computation 2)
3. Compute all the $F_i(x_i)$ simultaneously (in computation 3).

For example, n independent processors may perform all the computations in parallel, one processor per variable, leading to a successful computation in each case. In Example 1, the distances of all robots to $L(t)$ can be found concurrently before any of them strays away. In Example 2, measurements on all n living organisms are performed at the same time, thus avoiding harming any of them. In Example 3, n edges are removed from Fig. 1 and n new edges replace them to obtain Fig. 2, all in one step. In each case, once $F_i(x_i)$ has been obtained for $0 \leq i \leq n-1$, the function $\mathcal{F}$ can be easily evaluated (whether it is equal to the sum of the $F_i(x_i)$, or their minimum, and so on). It is important to note here that:

1. The computer capable of n operations in one step may belong to any one of several possible models of computation, such as the n-tape Turing Machine [56], the Parallel Random Access Machine, a network of processors, or even a combinational circuit (a memoryless model) [2].

2. Any model capable of fewer than n operations in one step fails to perform the computation successfully. Even if $n-1$ operations are possible in one step, but not n, then the computation of $\mathcal{F}$ fails.

3. The above observations hold regardless of the technology used to build the computer. In other words, the computer can be mechanical, electronic, optical, quantum, chemical, or biological, and the same limitations will be true.

5.2 Question 2: Are they computable when the finiteness condition holds?

As the preceding analysis shows, the answer here is an emphatic *No*. In fact, the analysis suggests that simulating the algorithms given in response to Question 1 on *any* computer capable of fewer than n operations per step is impossible, regardless of how much time is available to perform the simulation. Even if given *infinite time*, the computer still fails to compute $\mathcal{F}$. This remains true regardless of the size of the memory on the computer attempting the simulation (it can be infinite), and whether or not the computer is allowed to receive input and produce output *during the computation*.

While various computers in Section 5.1 succeeded to evaluate $\mathcal{F}$, *none* qualifies as a Universal Computer. The reason is simple: All obey the finiteness condition. Once n, the number of operations a computer is capable of performing in one step is finite and fixed, that computer loses its ability to be a Universal Computer, thanks to the computations described in Section 4. In each of these computations, n is a variable; if it exceeds the pre-established number of operations per step of which a computer is capable, *that* computer cannot be universal. For the same reason, none of the computers mentioned in Section 3, despite being more powerful than the Turing Machine, can be a Universal Computer, if it obeys the finiteness condition.

Perhaps this point is worth repeating. For *every* $n > 1$, there exists an n-tape Turing Machine (or any equivalent n-operations-per-step computer) that can perform all three computations of size n described in Section 4. A single-tape Turing Machine can perform *none* of these computations. However, *even* an n-tape Turing Machine (or any equivalent n-operations-per-step computer) cannot claim to be *universal* as it fails to perform any of the three computations in Section 4 when the input size is $n+1$.

Therefore, the Universal Computer $\mathcal{U}$ as defined in Section 2.1 is clearly a myth. Another consequence is that the Church-Turing thesis (in the current stage of its evolution) does not hold. It may either have to be revised, or better still forgotten.

6 Conclusion

It has been demonstrated by many researchers that the Turing Machine is not an appropriate model for describing the increasingly complex and diversified computations that need to be performed on today's computers. In response, some have argued that the Turing Machine is a 'closed system' never meant to interact with the outside world; that the Turing Machine is meant to operate neither on external input that may be changing, nor under evolving computational conditions; that the input tape should contain at the start of the computation all that is required to carry out that computation. But this is precisely the main shortcoming of the Turing Machine as illustrated here and elsewhere.

In fact, we have shown in this paper that any model of computation that aspires to replace the Turing Machine as a Universal Computer, misses the mark as long as it is restricted to a finite and fixed number of operations per step. This holds even if the contender is allowed to have extraordinary powers that the Turing machine possesses (but not today's computers, such as being endowed with an infinite memory and an unlimited amount of time to perform a required computation), as well as the routine capabilities that today's computers take for granted (but not the Turing Machine, such as the ability to interact with the outside world).

A good way to think about the result of this paper is to imagine a contest between the two perennial characters Alice and Bob. The latter claims to have a Universal Computer, that is, a computer capable of performing *any* computation (as long as that computation can be carried out successfully on some other machine, regardless of that machine's type). Alice wants to prove him wrong, that is, she wants to exhibit a problem that cannot be solved on the so-called Universal Computer (but is otherwise solvable on some other machine). Bob's computer can perform n operations per step, where n is known, finite, and fixed. Alice creates a problem of size $n + 1$ (it can be any one of the three problems defined in Section 4), and presents it to Bob's computer. The latter, of course, fails since Alice's problem requires the ability to perform $n + 1$ operations per step. Now Bob designs a new 'Universal Computer' capable of $n+1$ operations per step and solves Alice's problem. Whereupon Alice creates a new problem of size $n + 2$, and so on. Bob continually needs to switch to a new machine. Eventually, he gets the point and concedes to Alice: There is no such a thing as a Universal Computer.

The finiteness condition, the key to our claim, is a reasonable one. It is the hallmark of a *feasible* computer. Turing formulated it as an essential requirement of a realistic computer [81] (see also [71] for a discussion). However, it is interesting to note that the result of this paper indeed applies to certain computational models that do not, on the surface, obey the finiteness condition. Consider, for example, an accelerating machine (AM), that is, one capable of performing the first step of a computation in 1 time unit, the second in 1/2 time unit, the third in 1/4 time unit, and so on. It is clear that AM fails to compute any of the functions in Section 4. In fact, it turns out that even if AM were to be endowed with superior abilities (beyond doubling its speed), computations could be found to render it powerless. Thus, for example, suppose that AM can *square* the number of operations it can do at each successive step (the number of operations thus being 2, 4, 16, 256, and so on). In this case it suffices to specify that for $n > 2$ and $0 \leq i \leq n - 1$, the evaluation of the function $F(x_i)$

requires 2^{2^t} operations, if computed at time t, for $t \geq 0$, and insist that all of $F(x_0)$, $F(x_1)$, $\ldots$, $F(x_{n-1})$ be available at $t = 2$. This computation is impossible for AM but trivial for an n-processor computer.

Will there ever be a Universal Computer? The result of this paper suggests that the only way to achieve universal computation is to do away with the finiteness condition. In other words, a Universal Computer must be able to perform an infinite number of operations per step (for example, by having an *infinite* number of processors). This allows any computation, requiring n simultaneous operations per step, to be performed, for any arbitrary value of n. In that case, however, it is likely that the only Universal Computer possible would be the Universe itself!

References

[1] L.A. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, Vol. 266, 1994, pp. 1021–1024.

[2] S.G. Akl, *Parallel Computation: Models And Methods*, Prentice Hall, Upper Saddle River, New Jersey, 1997.

[3] S.G. Akl, Discrete steepest descent in real time, *Parallel and Distributed Computing Practices*, Vol. 4, No. 3, 2001, pp. 301–317.

[4] S.G. Akl, Secure file transfer: A computational analog to the furniture moving paradigm, *Parallel and Distributed Computing Practices*, Vol. 5, No. 2, 2002, pp. 193–203.

[5] S.G. Akl, Parallel real-time computation: Sometimes quality means quantity, *Computing and Informatics*, Vol. 21, No. 5, 2002, pp. 455–487.

[6] S.G. Akl, Parallel real-time computation of nonlinear feedback functions, *Parallel Processing Letters*, Vol. 13, No. 1, 2003, pp. 65–75.

[7] S.G. Akl, Computing in the presence of uncertainty: Disturbing the peace, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, June 2003, Vol. I, pp. 442–448.

[8] S.G. Akl, Superlinear performance in real-time parallel computation, *The Journal of Supercomputing*, Vol. 29, No. 1, 2004, pp. 89–111.

[9] S.G. Akl, Inherently parallel geometric computations, *Parallel Processing Letters*, Vol. 16, No. 1, March 2006.

[10] S.G. Akl, Coping with uncertainty and stress: A parallel computation approach, *International Journal of High Performance Computing and Networking*, Vol. 4, No. 2, February 2006.

[11] S.G. Akl and S.D. Bruda, Parallel real-time optimization: Beyond speedup, *Parallel Processing Letters*, Vol. 9, No. 4, 1999, pp. 499–509.

[12] S.G. Akl and S.D. Bruda, Parallel real-time numerical computation: Beyond speedup III, *International Journal of Computers and their Applications*, Special Issue on High Performance Computing Systems, Vol. 7, No. 1, 2000, pp. 31–38.

[13] S.G. Akl and S.D. Bruda, Improving a solution's quality through parallel processing, *The Journal of Supercomputing*, Vol. 19, No. 2, 2001, pp. 219–231.

[14] S.G. Akl, B.J. Cordy, and W. Yao, An analysis of the effect of parallelism in the control of dynamical systems, *International Journal of Parallel, Emergent and Distributed Systems*, Vol. 20, No. 2, June 2005, pp. 147–168.

[15] S.G. Akl and L. Fava Lindon, Paradigms admitting superunitary behaviour in parallel computation, *Proceedings of the Joint Conference on Vector and Parallel Processing* (CONPAR 94, VAPP VI), Linz, Austria, September 1994, pp. 301–312.

[16] S.G. Akl and W. Yao, Parallel computation and measurement uncertainty in nonlinear dynamical systems, *Journal of Mathematical Modelling and Algorithms*, *Special Issue on Parallel and Scientific Computations with Applications*, Vol. 4, 2005, pp. 5–15.

[17] S.G. Akl and W. Yao, A parallel approach eliminates measurement perturbations in RLC circuits, *Journal of Supercomputing*, Vol. 35, No. 2, February 2006, pp. 155–164.

[18] I. Antoniou, C.S. Calude, and M.J. Dinneen, Eds., *Unconventional Models of Computation*, Springer-Verlag, London, 2001.

[19] B. Barutçu, S. Şeka, E. Ayaz, and E. Türkcan, Real-time reactor noise diagnostics for the Borsele (PWR) nuclear power plant, *Progress in Nuclear Energy*, Vol. 43, Nos. 1 - 4, 2003, pp. 137–143.

[20] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*, Springer-Verlag, New York, 1998.

[21] M. Burgin and A. Klinger, Eds., *Theoretical Computer Science*, Special Issue on: Super-recursive algorithms and hypercomputation, Vol. 317, Issues 1–3, June 2004.

[22] J. Brown, *The Quest for the Quantum Computer*, Simon & Schuster, New York, 2000.

[23] S.D. Bruda and S.G. Akl, Pursuit and evasion on a ring: An infinite hierarchy for parallel real-time systems, *Theory of Computing Systems*, Vol. 34, No. 6, 2001, pp. 565–576.

[24] S.D. Bruda and S.G. Akl, A case study in real-time parallel computation: Correcting algorithms, *Journal of Parallel and Distributed Computing*, Vol. 61, No. 5, 2001, pp. 688–708.

[25] C.S. Calude, M.J. Dinneen, and F. Peper, Eds., *Unconventional Models of Computation*, Springer-Verlag, Berlin, 2002.

[26] C.S. Calude, J. Casti, and M.J. Dinneen, Eds. *Unconventional Models of Computation*, Springer-Verlag, Singapore, 1998.

[27] C.S. Calude and G. Păun, Bio-steps beyond Turing, *BioSystems*, Vol. 77, 2004, pp. 175–194.

[28] C.S. Calude and B. Pavlov, Coins, quantum, measurements, and Turing's barrier, *Quantum Information Processing*, Vol. 1, Nos. 1–2, 2002, pp. 107–127.

[29] A. Church, An unsolvable problem of elementary number theory, *American Journal of Mathematics*, Vol. 58, 1936, pp. 345–363.

[30] B.J. Copeland, The Church-Turing thesis, http://plato.stanford.edu/entries/church-turing/

[31] B.J. Copeland, Super Turing-machines, *Complexity*, Vol. 4, 1998, pp. 30–32.

[32] B.J. Copeland, Hypercomputation, *Minds and Machines*, Vol. 12, No. 4, 2002, pp. 461–502.

[33] B.J. Copeland, Accelerating Turing machines, *Mind and Machines*, Vol. 12, No. 2, 2002, pp. 281–301.

[34] B.J. Copeland and D. Proudfoot, Alan Turing's forgotten ideas in computer science, *Scientific American*, Vol. 280, No. 4, 1999, pp. 77–81.

[35] B.J. Copeland and D. Proudfoot, Introduction to hypercomputation: Computing the uncomputable, http://www.calculemus.org/x/Copeland-etc/copeland1.html

[36] B.J. Copeland and R. Sylvan, Beyond the universal Turing machine, *Australasian Journal of Philosophy*, Vol. 77, No. 1, 1999, pp. 46–66.

[37] B.J. Cordy, private communication.

[38] M. Daley and L. Kari, DNA computing: Models and implementations, *Comments on Theoretical Biology*, Vol. 7, No. 3, 2002, pp. 177–198.

[39] M. Davis, The myth of hypercomputation, in [79].

[40] D. Deutsch, Quantum theory, the Church-Turing principle of the Universal Quantum Computer, *Proceedings of the Royal Society of London*, 1985, pp. 97–117.

[41] D. Deutsch, *The Fabric of Reality*, Penguin, New York, 1997.

[42] S. Eilenberg, *Automata, Languages and Machines*, Vol. A, Academic Press, New York, 1974.

[43] C. Eliasmith, The myth of the Turing machine: The failings of functionalism and related theses, *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 14, 2002, pp. 1–8.

[44] G. Etesi and I. Németi, Non-Turing computations via Malament-Hogarth space-times, *International Journal of Theoretical Physics*, Vol. 41, No. 2, February 2002, pp. 341–370.

[45] D. Goldin and P. Wegner, The origins of the Turing thesis myth, manuscript, University of Connecticut, June 2004.

[46] C.E. Hewitt, P. Bishop, and R. Steiger, A universal modular actor formalism for artificial intelligence, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, California, August 1973, pp. 235–245.

[47] D. Hillis, *The Pattern on the Stone*, Basic Books, New York, 1998.

[48] Hypercomputation, http://encyclopedia.laborlawtalk.com/Hypercomputation

[49] Hypercomputation, http://en.wikipedia.org/wiki/Hypercomputation

[50] G. Johnson, *A Shortcut Through Time: The Path to the Quantum Computer*, Random House, New York, 2003.

[51] R.M. Karp and R. Lipton, Turing machines that take advice, *L'Enseignement Mathématique*, Vol. 28, 1982, pp. 191–209.

[52] T.D. Kieu, Quantum hypercomputation, *Minds and Machines*, Vol. 12, No. 4, November 2002, pp. 541–561.

[53] T.D. Kieu, Computing the noncomputable, *Contemporary Physics*, Vol. 44, No. 1, January-February 2003, pp. 51–71.

[54] T.D. Kieu, Hypercomputation with quantum adiabatic processes, *Theoretical Computer Science*, Vol. 317, Nos. 1–3, June 2004, pp. 93–104.

[55] S.C. Kleene, *Mathematical Logic*, Wiley, New York, 1967.

[56] H.R. Lewis and C.H. Papadimitriou, *Elements of the Theory of Computation*, Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[57] K. Li, Y. Pan, and S.-Q. Zheng, Eds., *Parallel Computing Using Optical Interconnections*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[58] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, New York, 1992.

[59] W. Marciszewski, Text commented: B. Jack Copeland's "The Church-Turing Thesis", http://www.calculemus.org/forum/4/CT-thesis.html

[60] H. Meijer and D. Rappaport Simultaneous Edge Flips for Convex Subdivisions, *16th Canadian Conference on Computational Geometry*, Montreal, August 2004.

[61] R. Milner, Elements of interaction, *Communications of the ACM*, Vol. 31, No. 1, January 1993, pp. 78–89.

[62] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, England, 2000.

[63] M. Nagy and S.G. Akl, On the importance of parallelism for quantum computation and the concept of a universal computer, *Proceedings of the Fourth International Conference on Unconventional Computation*, Sevilla, Spain, October 2005, pp. 176–190.

[64] Nagy, M. and Akl, S.G., Quantum measurements and universal computation, *International Journal of Unconventional Computing*, Vol. 2, No. 1, 2006, pp. 73–88.

[65] T. Ord and T.D. Kieu, The diagonal method and hypercomputation, *The British Journal for the Philosophy of Science*, Vol. 56, No. 1, March 2005, pp. 147–156.

[66] G. Okša, M. Bečka, and M. Vajteršic, Parallel computation with structured matrices in linear modeling of multidimensional signals, *Parallel and Distributed Computing Practices*, Vol. 5, No. 3, 2004, pp. 289–299.

[67] S. Pavel and S.G. Akl, Area-time trade-offs in arrays with optical pipelined buses, *Applied Optics*, Vol. 35, 1996, pp. 1827–1835.

[68] R. Penrose, *The Emperor's New Mind*, Oxford University Press, Oxford, England, 1989.

[69] J. Raskin, Computers are not Turing machines, http://humane.sourceforge.net/unpublished/turing_machines.html

[70] J.E. Savage, *Models of Computation*, Addison-Wesley, Reading, Massachusetts, 1998.

[71] O. Shagrir, Effective computation by humans and machines, *Minds and Machines*, Vol. 12, 2002, pp. 221–240.

[72] H.T. Siegelmann, Computation beyond the Turing limit, *Science*, Vol. 268, No. 5210, 1995, pp. 545–548.

[73] H.T. Siegelmann, *Neural Networks and Analog Computation: Beyond the Turing limit*, Birkhäuser, Boston, 1999.

[74] C. Siehs and B. Mayer, Dynamical hierarchies of structure and control in chemical reaction networks, *Nanotechnology*, Vol. 10, 1999, pp. 464–471.

[75] M. Stannett, X-machines and the halting problem: Building a super-Turing machine, *Formal Aspects of Computing*, Vol. 2, No. 4, 1990, pp. 331–341.

[76] I. Stewart, Deciding the undecidable, *Nature*, Vol. 352, August 1991, pp. 664–665.

[77] I. Stewart, The dynamics of impossible devices, *Nonlinear Science Today*, Vol. 1, 1991, pp. 8–9.

[78] Super-Turing computation, http://en.wikipedia.org/wiki/Super-Turing_computation

[79] C. Teuscher, Ed., *Alan Turing: Life and Legacy of a Great Thinker*, Springer-Verlag, New York, 2004.

[80] C. Teuscher and M. Sipper, Hypercomputation: Hype or computation?, *Communications of the ACM*, Vol. 45, No. 8, 2002, pp., 23–24.

[81] A.M. Turing, On computable numbers with an application to the Entscheidungsproblem, *Proceedings of the London mathematical Society*, Ser. 2, Vol. 42, 1936, pp. 230–265; Vol. 43, 1937, pp. 544–546.

[82] A.M. Turing, Systems of logic based on ordinals, *Proceedings of the London Mathematical Society*, Ser. 2, Vol. 45, 1939, pp. 161–228.

[83] A.M. Turing, Intelligent machinery, Report, National Physics Laboratory, 1948. Reprinted in: B. Meltzer and D. Michie, Eds., *Machine Intelligence* 5, Edinburgh University Press, Edinburgh, Scotland, 1969, pp. 3–23.

[84] P. Wegner, Why interaction is more powerful than algorithms, *Communications of the ACM*, Vol. 40, No. 5, May 1997, pp. 80–91.

[85] P. Wegner and D. Goldin, Interaction, computability, and Church's thesis, manuscript, Brown University, 1999.

[86] P. Wegner and D. Goldin, Computation beyond Turing Machines, *Communications of the ACM*, Vol. 46, No. 4, May 1997, pp. 100–102.

[87] C.P Williams and S.H. Clearwater, *Explorations in Quantum Computing*, Springer-Verlag, Heidelberg, 1998.

[88] C.P Williams and S.H. Clearwater, *Ultimate Zero and One: Computing at the Quantum Frontier*, Springer-Verlag, Heidelberg, 2000.

[89] D. Wood, *Theory of Computation*, Harper & Row, New York, 1987.