NONUNIVERSALITY EXPLAINED [*]

Selim G. Akl
School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
akl@cs.queensu.ca

July 30, 2015

Abstract

The Nonuniversality in Computation Theorem (NCT) states that no computer capable of a finite and fixed number of basic operations per time unit can be universal. This result, obtained in 2005, disproves major prevailing dogmas in computer science, on the strength of several counterexamples that differ significantly from one another. Thus, the NCT shows that the Church-Turing Thesis (CTT) is false: It is not true that the Turing Machine can execute any computation possible on any other computer. It also disproves the inflated CTT, whereby there exists a universal computer, that is, a physical device capable of carrying out any computation conceivable. At the heart of the NCT is a refutation of the simulation principle, which states that any computation possible on a general-purpose computer A can be simulated, more or less efficiently, on any other general-purpose computer B.

While more than ten years have now elapsed since the NCT was established, the result is still widely misunderstood. This state of affairs is due to a number of misconceptions about the nature of the counterexamples and the significance of the NCT. The purpose of this paper is to dispel these misconceptions and convey the simple, yet important idea that universality in computation cannot be achieved, except by a computer capable, each time unit, of an infinite number of basic operations executed in parallel.

Keywords and phrases: nonuniversality in computation; simulation; models of computation; inherently parallel computations; Church-Turing Thesis; Gödel's incompleteness theorem; quantum computing;

1 Introduction

For a long time people believed that the Earth is flat. We now know better. And so it is with universality in computation. Consider the following statement:

"It can also be shown that any computation that can be performed on a modern-day digital computer can be described by means of a Turing Machine. Thus if one ever found a procedure that fitted the intuitive notions, but could not be described by means of a Turing Machine, it would indeed be of an unusual nature since it could not possibly be programmed for any existing computer [33], p. 80."

The first sentence in the preceding quote is clearly false, and well known counterexamples abound in the literature (see, for example, [34, 60], and for further references [6, 11]). The second sentence, on the other hand, is only half true. Indeed, it is perfectly justified to say that a computation that cannot be performed on a Turing Machine must be of an *unusual nature*. However, this does not mean that such a computation cannot be programmed on any existing computer. Rather, the existence of such a computation would instead mean that the Turing Machine is simply not universal and that the "Church-Turing Thesis" (whose validity is assumed implicitly in the opening quote of this paper) is in fact invalid.

In 2005 it was shown that the concept of a Universal Computer cannot be realized [6]. Specifically, instances of a *computable* function $\mathcal{F}$ were exhibited that cannot be computed on any machine $\mathcal{U}$ that is capable of only a finite and fixed number of operations per step (or time unit). 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 holds if, in addition, $\mathcal{U}$ is given an indefinite amount of time to compute $\mathcal{F}$.

[*]This paper is an abbreviated and revised version of: Akl, S.G.: Nonuniversality in Computation: Fifteen misconceptions rectified. In: Adamatzky, A. (ed.) Advances in Unconventional Computing. Springer, Switzerland (2015).

1.1 The Main Theorem

Formally, the result is stated as follows:

Nonuniversality in Computation Theorem (NCT): No computer is universal if it is capable of exactly $T(i)$ operations during time unit i of a computation, where i is a positive integer, and $T(i)$ is finite and fixed once and for all.
Proof: The computer cannot be universal for it will be faced with a computation requiring $V(i)$ operations during time unit i, where $V(i) > T(i)$ for at least one i. ■

A constructive proof of NCT seeks to define a function $F^n(X_1, X_2, \ldots, X_n)$ of n spatially and temporally connected physical variables, $X_1, X_2, \ldots, X_n$, where n is a positive integer, that obeys the following property: F^n is readily computable by a machine M_n capable of exactly n operations per time unit; however, this machine cannot compute F^{n+1} when the number of variables is $n+1$. While a second machine M_{n+1} capable of $n+1$ operations per time unit can now compute the function F^{n+1} of $n+1$ variables, M_{n+1} is in turn defeated by a function F^{n+2} of $n+2$ variables. In principle, the escalation continues without end.

This point deserves emphasis. While the function $F^{n+1}(X_1, X_2, \ldots, X_{n+1})$ is easily computed by M_{n+1}, it cannot be computed by M_n. Even if given infinite amounts of time and space, machine M_n is incapable of simulating the actions of M_{n+1}. Furthermore, machine M_{n+1} is in turn thwarted by $F^{n+2}(X_1, X_2, \ldots, X_{n+2})$, a function computable by a third machine M_{n+2}. The process repeats indefinitely. Thus, no computer can evaluate F^n for any arbitrary n, unless it is capable of an infinite number of operations per time unit.

Therefore, in order to actually prove NCT constructively, it suffices to exhibit (at least) one concrete instance of the function F^n satisfying the property described in the previous two paragraphs. Indeed there exist several examples of such a function occurring in:

1. *Computations with time-varying variables:* The variables, over which the function is to be computed, are themselves changing with time.

2. *Computations with time-varying computational complexity:* The computational complexity of the function to be computed is itself changing with time.

3. *Computations with rank-varying computational complexity:* Given several functions to be computed, and a schedule for computing them, the computational complexity of a function depends on its position in the schedule.

4. *Computations with interacting variables:* The variables of the function to be computed are parameters of a physical system that interact unpredictably when the system is disturbed.

5. *Computations with uncertain time constraints:* There is uncertainty with regards to the input (when and for how long are the input data available), the calculation (what to do and when to do it), and the output (the deadlines are undefined at the outset); furthermore, the function that resolves each of these uncertainties, *itself* has demanding time requirements.

6. *Computations with global mathematical constraints:* The function to be computed is over a system whose variables must collectively obey a mathematical condition at all times.

It should be clear at this point that all computable functions used in this work are in fact generalizations of the definition of a mathematical function in that they may enjoy one or more of the following properties: They have real (that is, physical) time as a variable, their variables interact with one another, there exists a condition that must be satisfied while a function is being evaluated, and so on.

1.2 Consequences

The implication of this result to the theory of computation is akin to that of Gödel's incompleteness theorem to mathematics. In the same way as no finite set of axioms A_i can be complete, no computer C_i is universal that can perform a finite and fixed number of operations per time unit. For every set of axioms A_i there exists a statement G_{i+1} not provable in A_i, but provable in A_{i+1}; similarly, for every machine C_i there is a problem P_{i+1} not solvable on C_i, but solvable on C_{i+1}.

This result applies to any computer that obeys the *finiteness condition*, that is, a computer capable of only a finite and fixed number of operations per step (that is, per time unit). It should be noted that computers obeying the finiteness condition include all "reasonable" models of computation, both theoretical and practical, such as the Turing Machine, the Random Access Machine, and other idealized models [50], as well as all of today's general-purpose computers, including both existing conventional computers (both sequential and parallel), and contemplated unconventional ones such as biological and quantum computers [8]. It is true for computers that interact with the outside world in order to read input and return output (unlike the Turing Machine, but like every realistic general-purpose computer). It is also valid for computers that are given unlimited amounts of time and space in order to perform their computations (like the Turing Machine, but unlike realistic computers). Even accelerating machines [11] that increase their speed at every step (such as doubling it, or squaring it, or any such fixed acceleration) at a rate that is set in advance, cannot be universal! (Divine intervention, in the form of the Oracle Machine [58], for example, is clearly beyond the scope of the present discussion.) The only constraint that we have placed on the computer (or model of computation) that claims to be universal is that the number of operations of which it is capable per time unit be finite and fixed once and for all. In this regard, it is important to note that:

1. The requirement that the number of operations per time unit, or step, be *finite* is necessary for any "reasonable" model of computation; see, for example, [53], p. 141.

2. The requirement that this number be *fixed* once and for all is necessary for any model of computation that purports to be "universal"; see, for example, [24], p. 210.

Without these two requirements, the theory of computation in general, and the theory of algorithms, in particular, would be totally irrelevant.

The consequences to theoretical and practical computing are significant. Thus the conjectured "Church-Turing Thesis" is false. It is no longer true that, given enough time and space, any single general-purpose computer, defined a priori, can perform all computations that are possible on all other computers. Not the Turing Machine, not a personal computer, not the most powerful of supercomputers. In view of the computational problems mentioned in Section 1.1 and presented more fully in Section 3 (for details, see [4] – [18], [27], and [42] – [47]), the only possible universal computer would be one capable of an infinite number of operations per time unit. In fact, this work has led to the discovery of computations that can be performed on a quantum computer but not on any classical machine (even one with infinite resources), thus showing for the first time that the class of problems solvable by classical means is a true subset of the class of problems solvable by quantum means [44]. Consequently, the only possible universal computer would have to be quantum (as well as being capable of an infinite number of operations per time unit).

1.3 Motivation

The results described in the previous paragraphs were obtained and published between 2005 and 2015. Yet, ten years after they were first announced, they remain generally misunderstood, and consequently controversial. The purpose of this paper is to explain nonuniversality by clarifying all the misconceptions surrounding it.

The remainder of the paper is organized as follows. Section 2 provides definitions for many of the terms used in the discussion to follow (some of the more common of these terms were already introduced in this section without definition, in order not to interrupt the flow of the opening paragraphs). Section 3 summarizes previous work, culminating in the fundamental result that the 'Universal Computer' is a myth. Section 4 is a dialog with an imaginary debater who expresses all the misconceptions about the nonuniversality result. Section 5 offers a computational challenge to the readers.

2 Preliminaries

A *time unit* is the length of time required by a processor to perform a *step* of its computation, consisting of three elementary operations: A *read* operation in which it receives a constant number of fixed-size data as input, a *calculate* operation in which it performs a fixed number of constant-time *arithmetic* and *logical* calculations (such as adding two numbers, comparing two numbers, and so on), and a *write* operation in which it returns a constant number of fixed-size data as output.

A *sequential computer*, consists of a single processor. A *parallel computer* has n processors, numbered 1 to n, where $n \geq 2$. Both computers use the same type of processor and that processor is the fastest possible [2]. The assumption that the computers on hand, whether sequential or parallel, use the fastest conceivable processor is an important one. This is because the speed of the processor used is what specifies the duration of a time unit, as defined in the previous paragraph; the faster the processor, the smaller the time unit. In the time-varying variables computation (see Section 3.1), for example, the value of an input variable $x_i(t)$ changes at the same speed as the processor in charge of evaluating the function $f_i(x_i(t))$.

The concept of *computational universality* is one of the dogmas in Computer Science, stated as follows: Given enough time and space, any general-purpose computer can, through simulation, perform any computation that is possible on any other general-purpose computer. Statements such as this are commonplace in the computer science literature, and are served as standard fare in undergraduate and graduate courses alike. Sometimes the statement is restricted to the Turing Machine (TM), and is referred to as the "Church-Turing Thesis" (CTT), which stipulates that there exists a Universal Computer, namely, the TM; thus, the following quote is typical:

"A Turing Machine can do everything that a real computer can do [53], p. 125."

Other times, the statement is made more generally about a *Universal Computer*, that is, a computer capable of executing any computation that can be performed by any other computer; thus, we are told:

"It is possible to build a universal computer: a machine that can be programmed to perform any computation that any other physical object can perform [24], p. 134."

As its name indicates, the CTT is a *conjecture*, whose proof has remained elusive due to the difficulty in defining what it means *to compute*. A few points are worth making in this regard:

1. The *Universal Turing Machine* (UTM) defined by Alan Turing, is 'universal' in the sense of being able to simulate any computation performed by any other special-purpose TM. In other words, the UTM can be programmed to execute any TM computation. This is a provable property, and is not in question here or elsewhere.

2. However, the UTM is not known to be 'universal' in the more general sense of being able to simulate by its own means any computation performed by means of any other computational device (not necessarily a TM). It is this universality of the UTM, in the more general sense, that is expressed by the CTT.

3. The NCT shows the CTT, and more generally computational universality, to be false. Specifically, the NCT is a stronger result, namely, that no 'Universal Computer' is possible; this eliminates, not only the UTM, but also all other models of computation as contenders for the title.

3 Inherently Parallel Computations

The term *inherently parallel computation* refers to a computation that can be performed successfully only on a parallel computer with an appropriate number of processors. Examples are computations that involve: Time-varying variables, time-varying computational complexity, rank-varying computational complexity, interacting variables, uncertain time constraints, and variables obeying mathematical constraints. These computations also provide counterexamples to the existence of a universal computer [4] – [18], [27], [42] – [47].

3.1 Time-Varying Variables

For a positive integer n larger than 1, we are given n functions, each of one variable, namely, f_1, f_2, $\ldots$, f_n, operating on the n physical variables x_1, x_2, $\ldots$, x_n, respectively. Specifically, it is required to compute $f_i(x_i)$, for i = 1, 2, $\ldots$, n. For example, $f_i(x_i)$ may be equal to x_i^2. What is unconventional about this computation, is the fact that the x_i are themselves (unknown) functions $x_1(t), x_2(t), \ldots, x_n(t)$, of the time variable t. It takes one time unit to evaluate $f_i(x_i(t))$. The problem calls for computing $f_i(x_i(t))$, $1 \leq i \leq n$, at time $t = t_0$. Because the function $x_i(t)$ is unknown, it cannot be inverted, and for $k > 0$, $x_i(t_0)$ cannot be recovered from $x_i(t_0 + k)$.

A sequential computer fails to compute all the f_i as desired. Indeed, suppose that $x_1(t_0)$ is initially operated upon. By the time $f_1(x_1(t_0))$ is computed, one time unit would have passed. At this point, the values of the $n-1$ remaining variables would have changed. The same problem occurs if the sequential computer attempts to first read all the x_i, one by one, and store them before calculating the f_i.

By contrast, a parallel computer consisting of n independent processors may perform all the computations at once: For $1 \leq i \leq n$, and all processors working at the same time, processor i computes $f_i(x_i(t_0))$, leading to a successful computation.

3.2 Time-Varying Computational Complexity

Here, the computational complexity of the problems at hand depends on *time* (rather than being, as usual, a function of the problem *size*). Thus, for example, tracking a moving object (such as a spaceship racing towards Mars) becomes harder as it travels away from the observer.

Suppose that a certain computation requires that n independent functions, each of one variable, namely, $f_1(x_1)$, $f_2(x_2)$, $\ldots$, $f_n(x_n)$, be computed. Computing $f_i(x_i)$ at time t requires $C(t) = 2^t$ operations, for $t \geq 0$ and $1 \leq i \leq n$. Further, there is a strict deadline for reporting the results of the computations: All n values $f_1(x_1)$, $f_2(x_2)$, $\ldots$, $f_n(x_n)$ must be returned by the end of the third time unit, that is, when $t = 3$.

It should be easy to verify that no sequential computer, capable of exactly one constant-time operation per step (that is, per time unit), can perform this computation for $n \geq 3$. Indeed, $f_1(x_1)$ takes $C(0) = 2^0 = 1$ time unit, $f_2(x_2)$ takes another $C(1) = 2^1 = 2$ time units, by which time three time units would have elapsed. At this point none of $f_3(x_3), \ldots, f_n(x_n)$ would have been computed. By contrast, an n-processor parallel computer solves the problem handily. With all processors operating simultaneously, processor i computes $f_i(x_i)$ at time $t = 0$, for $1 \leq i \leq n$. This consumes one time unit, and the deadline is met.

3.3 Rank-Varying Computational Complexity

Suppose that a computation consists of n stages. There may be a certain precedence among these stages, or the n stages may be totally independent, in which case the order of execution is of no consequence to the correctness of the computation. Let the *rank* of a stage be the order of execution of that stage. Thus, stage i is the ith stage to be executed. Here we focus on computations with the property that the number of operations required to execute stage i is $C(i)$, that is, a function of i only.

When does rank-varying computational complexity arise? Clearly, if the computational requirements grow with the rank, this type of complexity manifests itself in those circumstances where it is a disadvantage, whether avoidable or unavoidable, to being ith, for $i \geq 2$. For example, the precision and/or ease of measurement of variables involved in the computation in a stage s may decrease with each stage executed before s. The same analysis as in Section 3.2 applies by substituting the rank for the time.

3.4 Interacting Variables

A physical system has n variables, x_1, x_2, $\ldots$, x_n, each of which is to be measured or set to a given value at regular intervals. One property of this system is that measuring or setting one of its variables modifies the values of any number of the system variables *unavoidably*, *unpredictably*, and *irremediably*.

A sequential computer measures *one* of the values (x_1, for example) and by so doing it disturbs the equilibrium, thus losing all hope of recording the state of the system within the given time interval. Similarly, the sequential approach cannot update the variables of the system properly: Once x_1 has received its new value, setting x_2 disturbs x_1 unpredictably. A parallel computer with n processors, by contrast, will measure *all* the variables x_1, x_2, $\ldots$, x_n simultaneously (one value per processor), and therefore obtain an accurate reading of the state of the system within the given time frame. Consequently, new values $x_1, x_2, \ldots, x_n$ can be computed in parallel and applied to the system simultaneously (one value per processor).

3.5 Uncertain Time Constraints

In this paradigm, we are given a computation each of whose components, namely, the input phase, the calculation phase, and the output phase, needs to be computed by a certain deadline. However, unlike the standard situation in conventional computation, the deadlines here are not known at the outset. In fact, and this is what makes this paradigm truly unconventional, we do not know at the moment the computation is set to start, *what* needs to be done, and *when* it should be done. Certain physical parameters, from the external environment surrounding the computation, become spontaneously available. The values of these parameters, once received from the outside world, are then used to evaluate two functions, call them f_1 and f_2, that tell us precisely *what* to do and *when* to do it, respectively.

The difficulty posed by this paradigm is that the evaluation of the two functions f_1 and f_2 is itself quite demanding computationally. Specifically, for a positive integer n, the two functions operate on n variables (the physical parameters). Only a parallel computer equipped with n processors can succeed in evaluating the two functions on time to meet the deadlines.

3.6 Computations Obeying Mathematical Constraints

There exists a family of computational problems where, given a mathematical object satisfying a certain property, we are asked to transform this object into another which also satisfies the same property. Furthermore, the property is to be maintained throughout the transformation, and be satisfied by every intermediate object, if any. More generally, the computations we consider here are such that every step of the computation must obey a certain predefined constraint. (Analogies from popular culture include picking up sticks from a heap one by one without moving the other sticks, drawing a geometric figure without lifting the pencil, and so on.)

3.6.1 Rewriting Systems

An example of such transformations is provided by *rewriting systems.* From an initial string ab, in some formal language consisting of the two symbols a and b, it is required to generate the string $(ab)^n$, for $n > 1$. Thus, for $n = 3$, the target string is $ababab$. The rewrite rules to be used are: $a \rightarrow ab$ and $b \rightarrow ab$. Throughout the computation, no intermediate string should have two adjacent identical characters. Such rewrite systems (also known as $\mathcal{L}$-systems) are used to draw fractals and model plant growth [48]. Here we note that applying any *one* of the two rules at a time causes the computation to fail (for example, if ab is changed to abb, by the first rewrite rule, or to aab by the second).

A sequential computer can change only one symbol at once, thereby causing the adjacency condition to be violated. By contrast, for a given n, a parallel computer with n processors can easily perform a transformation on all the inputs collectively. The required property is maintained leading to a successful computation. Thus, the string $(ab)^n$ is obtained in $\log n$ steps, with the two rewrite rules being applied simultaneously to all symbols in the current intermediate string, in the following manner: ab, $abab$, $abababab$, and so on. It is interesting to observe that a successful generation of $(ab)^n$ also provides an example of a rank-varying computational complexity (as described in Section 3.3). Indeed, each legal string (that is, each string generated by the rules and obeying the adjacency property) is twice as long as its predecessor (and hence requires twice as many operations to be generated).

3.6.2 Sorting Variant

A second example of computations obeying a mathematical constraint is provided by a variant to the problem of sorting. For a positive even integer n, where $n \geq 8$, let n distinct integers be stored in an array A with n locations $A[1]$, $A[2]$, …, $A[n]$, one integer per location. Thus $A[j]$, for all $1 \leq j \leq n$, represents the integer currently stored in the jth location of A. It is required to sort the n integers in place into increasing order, such that:

1. After step i of the sorting algorithm, for all $i \geq 1$, no three consecutive integers satisfy $A[j] > A[j+1] > A[j+2]$, for all $1 \leq j \leq n-2$.
2. When the sort terminates we have $A[1] < A[2] < \cdots < A[n]$.

This is the standard sorting problem in computer science, but with a twist. In it, the journey is more important than the destination. While it is true that we are interested in the outcome of the computation (namely, the sorted array, this being the *destination*), in this particular variant we are more concerned with *how* the result is obtained (namely, there is a condition that must be satisfied throughout all steps of the algorithm, this being the *journey*). It is worth emphasizing here that the condition to be satisfied is germane to the problem itself; specifically, there are no restrictions whatsoever on the model of computation or the algorithm to be used. Our task is to find an algorithm for a chosen model of computation that solves the problem exactly as posed. One should also observe that computer science is replete with problems with an inherent condition on how the solution is to be obtained. Examples of such problems include: Inverting a nonsingular matrix without ever dividing by zero, finding a shortest path in a graph without examining an edge more than once, sorting a sequence of numbers without reversing the order of equal inputs, and so on.

An *oblivious* (that is, input-independent) algorithm for an $n/2$-processor parallel computer solves the aforementioned variant of the sorting problem handily in n steps, by means of predefined pairwise swaps applied to the input array A, during each of which $A[j]$ and $A[k]$ exchange positions (using an additional memory location for temporary storage) [2]. A sequential computer, and a parallel computer with fewer than $(n/2) - 1$ processors, both fail to solve the problem consistently, that is, they fail to sort all possible $n!$ permutations of the input while satisfying, at every step, the condition that no three consecutive integers are such that $A[j] > A[j+1] > A[j+2]$ for all j. In the particularly nasty case where the input is of the form $A[1] > A[2] > \cdots > A[n]$, any sequential algorithm and any algorithm for a parallel computer with fewer than $(n/2) - 1$ processors fail after the first swap.

It is interesting to note here that a Turing Machine with $n/2$ heads succeeds in solving the problem, yet its simulation by a standard (single-head) Turing Machine fails to satisfy the requirements of the problem. Indeed, suppose that the standard Turing Machine is presented with the input sequence $A[1] > A[2] > \cdots > A[n]$. It will do one of two things: Either use the given representation of the input, and proceed to perform an operation (a swap, for example), in which case it would fail after one step of the algorithm, or transform the given representation into a different encoding (perhaps one intended to capture the behavior of the Turing Machine with $n/2$ heads) in preparation for the sort, in which case it would again fail since the transformation itself will consist of more than one algorithmic step. This is a surprising result as it goes against the common belief that any computation by a variant of the Turing Machine can be effectively simulated by the standard model [37].

3.7 The Universal Computer Is A Myth

The Principle of Simulation is arguably the most important idea in computer science. It is at the heart of theoretical results and practical implements of the field such as programming languages, operating systems, and so on. The principle states that *any* computation that can be performed on *any* one general-purpose computer can be equally carried out through simulation on *any* other general-purpose computer [24, 29, 41]. 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 $\mathcal{U}$. Thus, Universal Computation, which clearly rests on the Principle of Simulation, is also one of the foundational concepts in the field [22].

In Section 1 we proved the NCT, namely, that there does not exist a *finite* computational device that can be called a Universal Computer. Our proof used the following reasoning. 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.

Evidence in support of this reasoning was presented in this section. Each of the computational problems described in Sections 3.1–3.6 *can* easily be solved by a computer capable of executing n operations at every step. Specifically, an n-processor parallel computer led to a successful computation in each case. However, *none* of these problems is solvable by any computer capable of at most $n - 1$ operations per step, for any integer $n > 1$. Furthermore, the problem size n itself is a variable that changes with each problem instance. As a result, *no* parallel computer, regardless of how many processors it has available, can cope with a growing problem size, as long as the number of processors is finite and fixed. This holds even if the computer purporting to be universal is endowed with an unlimited memory and is allowed to compute for an indefinite amount of time.

Therefore, it clearly follows from the NCT, and the fact that all computers (whether theoretical or practical) obey the *finiteness condition*, that the Universal Computer $\mathcal{U}$ is a myth. As a consequence, the Principle of Simulation itself (though it applies to most *conventional* computations) is, in general, a fallacy. In fact, the latter principle is responsible for many other myths in computing, such as the *Speedup Theorem*, the *Slowdown Theorem*, and *Amdahl's Law*. Counterexamples for dispelling these and other myths are presented in [3, 20].

4 Misconceptions and Replies

What follows are fifteen misconceptions relating to the nonuniversality result, and responses to them. They are presented as a dialog with an interlocutor who (one assumes, perhaps wishfully) has read the papers listed in the bibliography on the myth of universal computation. The title of each of Sections 4.1–4.15 summarizes the misconception described in that section.

4.1 Uncomputable functions are nothing new

Misconception 1: So, you describe a number of functions that are uncomputable. What is new about that? Uncomputable functions have been known since the time of Turing.

Response: This is incorrect. The functions described in Section 3 are eminently computable. In the papers listed in the bibliography (see, for example, [4] – [18], [27], [42] – [47]), every function F^n of n variables can be easily evaluated by a computer capable of at least n elementary operations per time unit (an n-processor parallel computer [13], for example). However, a computer capable of only $n-1$ or fewer elementary operations per time unit cannot compute F^n. Nonuniversality in computation immediately follows by simple induction.

4.2 Functions that cannot be simulated efficiently are not new

Misconception 2: You are proposing computations that cannot be simulated efficiently. What is new about that? Your own book [2] and your earlier papers presented such computations that can be performed in constant time by n processors, but require time exponential in n when executed on $n-1$ or fewer processors.

Response: The error in the preceding statement is in the phrase "cannot be simulated efficiently". Indeed, the nonuniversality result does not follow from computations that cannot be simulated efficiently. It follows from computations that *cannot be simulated at all.* Thus, for each of the functions F^n of n variables described in the papers listed in the bibliography (see, for example, [42]–[47]), no computer capable of fewer than n elementary operations per time unit can simulate the actions of a computer capable of n elementary operations per time unit. The latter computer is capable of evaluating F^n successfully; the former is not capable of doing so, even if given an infinite amount of time and memory space. It is this impossibility of simulation that leads to nonuniversality.

In every one of the examples in Section 3, the job simply cannot be done by simulation. The world does not stand still while the simulator is taking its sweet time. In some examples (the time-varying variables, say), physical time is the enemy: If a moment is not grasped, it is gone forever, and no amount of simulation will help. In other examples (the sorting variant, say), if a given condition is violated in the course of the simulation, the computation by the simulator is considered to have failed, regardless of whether the correct solution is eventually reached (simulating a plane landing after it has crashed is not useful to the unfortunate passengers, nor is simulating a surgery after the patient has died of any help to the deceased, and so on). Simulation is no more than a "mathematical exercise" in such cases.

4.3 Time deadlines are not part of the Turing Machine model

Misconception 3: On the issue of simulation. You have added a dimension of time deadlines that is not present in the Turing Machine model. The "Church-Turing Thesis" (I thought) says that if a Turing Machine can do it in any finite amount of time, it is a computable function. However, if the job is to get the function computed by some deadline, then some machines are not up to the job. Is this the essence of what you are saying?

Response: Let us start by saying that the Turing Machine confuses the issue because it is, by all measures, an inadequate model to capture what happens in the real world. The claims of universality that NCT disproves have been made about much more powerful physical devices (e.g. modern computers, robots that move about their environment, sensing and control devices, and so on). Now that the Turing Machine is out of the way, the answer to your question is both "Yes" and "No":

1. *Yes*: In [4] – [18], [27], and [42] – [47], unconventional computations are described, in some of which physical time plays a role. However, these are unlike the problems encountered in the field of real time computing, where deadlines are artificial ones imposed by the designer of the application (sometimes they are soft, sometimes they are hard, and they invariably refer to internal machine time). In the

unconventional computations described in Section 3, nature essentially is in control; for example, a physical variable takes on a value at time unit t which is lost forever at time unit $t + 1$; two physical variables may interact causing one or both to lose their value unpredictably, unavoidably, and irreversibly; computational complexity grows with the passage of time; and so on.

2. *No*: Not all of the counterexamples to universality use time deadlines. See, for example, the problems described in Section 3.6 in which a mathematical constraint must be satisfied at every step of the computation. In fact, a simple example that disproves computational universality is presented in [19]. Unlike previous counter-examples, it does not rely on extraneous phenomena, such as the availability of input variables that are time varying, computational complexity that changes with time or order of execution, physical variables that interact with each other, uncertain deadlines, or mathematical conditions among the variables that must be obeyed throughout the computation. All that is used is a single pre-existing global variable whose value is modified by the computation itself.

4.4 Can we use sensors?

Misconception 4: What do you think about this way of attacking your time-varying variables problem:

First of all, we should separate the tasks into sensing and computation tasks. That means, I assume that there are sensors equipped with memory which read the values of the variables at time t and write them into appropriate memory locations. So, for every t we have memory locations $x_1(t)$, $x_2(t)$, …, $x_n(t)$. Furthermore, assume that the values of n and t are stored in some other variables. Then, a universal machine would read the value of n (and t), read the values from memory, and perform the requested computations on these values, which is a rather simple task.

So, the main argument of this is that we allow for this separation. The sensors are peripheral components, which do not perform any computations, but 'just' provide the input for the computation, that is, they sense their values simultaneously at time t and store them in their respective memory locations, and the computations can access these values later on.

I guess that you will argue that this requires n values to be stored in the memory of the machine at the same time, which would contradict a specification of a machine which is independent of n. But if we can reduce the problem you posed to this separation of concerns, we would have the consistence with the traditional theory of computation except for the addition of these sensors, which are not considered to be a part of the universal computing device but of the input specification.

Response: Surely, you cannot "separate" part of the universal computer (in this case the input unit) from the rest of the computer just to fit the problem. The universal computer is one unit, and a computational step is: [read, calculate, write]. The definition of 'to compute' as 'the process of transforming information' applies to all three phases of computation, namely,

1. The input phase, where data such as keystrokes, mouse clicks, or temperatures are reduced, for example, to binary strings;

2. The calculation phase, where data are manipulated through arithmetic and logic operations, as well as operations on strings, and so on; and

3. The output phase, where data are produced as numbers on a screen, or rendered as images and sounds, for instance.

Each of the three phases represents information transformation; each is an integral part of every computation, and no computation worthy of that name can exclude any of them. In particular, input and output are fundamental in the design and analysis of algorithms. Input-sensitive and output-sensitive computations often play an especially important role in deriving lower and upper bounds on algorithmic complexity. One of the most dramatic illustrations of the interweaving of input and output with calculation is the well-known linear-time planarity testing algorithm [32]. In order to determine whether a graph with V vertices is planar, Step 1 of that algorithm avoids a potentially quadratic-time computation, by reading in the edges of the graph *one at a time* from the outside world; if there is one more edge than $3V - 6$, the graph is declared nonplanar.

But let's assume you have set up n sensors and succeeded in solving the problem. What happens when you discover, the next morning, that there are now, not n, but $n+1$ inputs? Do you think it is a fair solution for you to go to the sensor shop, buy one more sensor and rush back to attach it to the extra input source and to the computer? And even if you did, what if by the time you return, the time t_0 at which the result is needed would have passed? Finally, note that your proposal concerns only the computation in Section 3.1.

4.5 Universality is possible in theory

Misconception 5: Your argument is great. It is simple and effective. However, it does not show that there is something *theoretically* flawed with the concept of a universal computer; it shows that a universal computer could never be physically realized. So the "Church-Turing Thesis" is all right if we take it that the space/time needed is infinite. Having said that, I imagine that the distinction between the theoretical and the implementation claim is often overlooked, which makes the proof integral in making that distinction very sharp indeed.

Response: Even if given infinite space and infinite time (which we allow the Turing Machine anyway), no computer that you define once and for all (and are not allowed to change ever again) can solve the problems defined in Section 3 and in [4] – [18], [27], and [42] – [47]. The issue is not with infinite space and infinite time. The issue is: How many operations can you do in one slice of time? You have to define this for every theoretical (and of course practical) computer. Otherwise, analysis of algorithms becomes meaningless, and the running time of a program a void concept. For example, the Turing Machine can do one, two, or three operations per slice of time, depending on your definition, but it is always a finite number. It can read a symbol, then change state, or write a symbol, or move left or right or stay put. Then a new iteration starts. It cannot do an arbitrary number of these fundamental operations per slice of time (you may call the latter a step, an iteration, a time unit, and so on). In passing, it should be said that it is this finiteness of the number of operations per time unit that caused the great success of computer science, making the Computer the greatest invention of the 20th century: A machine designed once and for all that can simulate any operation by another machine. In theory, you should be able to buy a computer and use it for the rest of your life, for it will never be faced with a computable function that it cannot compute. Or so we thought ...

Suppose you have defined your machine once and for all (it can be a Turing Machine or a Supercomputer, or the chip in your digital camera or that in your toaster). Inevitably, there will come a day when your computer will be given a computable function that it will fail to compute. One example of such a computation is the problem of operating on variables that change with time (as described in Section 3.1). Even if given all the time in the world from here to eternity, even if given all the space in the known Universe and beyond, you could not solve the problem, not even in theory with pen and paper. Why? Because you defined your machine and fixed it once and for all (as one should if one claims to have a Universal Computer that can simulate anything that another computer can compute). And herein lies the Achilles heel of the Universal Computer: Your opponent, the designer of the problem, asks you for a definition of your purported Universal Computer, then concocts a computation that thwarts it. This computation, of course, can be easily carried out by another "Universal Computer", but then your opponent will give you another problem, and so on.

There is nothing here about implementation. No limits are placed on what your machine can do, except for one: It must do a finite number of operations per time unit. This number can be as big as you want, it can even be defined by a function of time, but it must be fixed the moment you define your model of computation, and it cannot be infinite. Once you define it, you cannot change it. The game can be played with paper and pencil. Your opponent will always win. The nonuniversality result is perfectly *theoretical* as well as being perfectly *practical*.

4.6 How about using only one sensor?

Misconception 6: While I could not make much sense of your "proof" (right off the bat, I do not think 'spatially and temporally connected variables' (see Section 1.1) is well defined, although I also think it suffers from many other flaws) I decided to take your challenge seriously, within the bounds and the scope of your restrictions, language and assumptions, in an attempt to define a reasonable sounding computational device that would solve the time-varying variables computation. I will assume that the variables appear each on a luminous display of some sort.

I am going to assume relativity (please let me know if spatially and temporally connected variables is supposed to mean something else?). There is space k between displays so that, when they update their values, the light from their screens arrive at my computational device at nearly, but not precisely the same time (determined by k and where my device is in the 'room'). I will move my computational device close to the displays so that I can exploit the hypotenuse the signals from each display must travel first to reach the sensor of my device. Then, it is simply a matter of setting the processing speed to a finite value around 'a constant $\times$ k meters / the speed of light.' In this way, my computational device is able to compute the function F^n as desired.

My device has one sensor. It only needs one sensor, since the signals from the displays come in sequentially. Remember that the displays are separated by a distance and the signal from each display, after they update, must travel this distance (at the speed of light). So my device uses its one sensor to read in a display and then it reuses that sensor some time later after the signal from the next display has reached it (some time later).

Response: Right off the bat, as is clear from the papers in the bibliography [4] – [18], [27], [42] – [47], the spatial and temporal relationship among the variables is such that:

1. The variables occupy the same region in physical space; specifically, the distance separating any two variables is no larger than a small constant (whose magnitude depends on the general paradigm under consideration);

2. The variables are constrained by a unique parameter representing true physical time.

Now turning to your solution, one could say it is original; unfortunately, you are missing the point of the exercise. The problems posed in Section 3 are to be treated independently of specific physical properties. For example, the time-varying variables could be anything one would want them to be (atmospheric pressures for instance, or humidity readings, etc.), and they should not necessarily be on display (they would need to be acquired, that is, measured, first). Light rays, the basis of your argument, should play no role in the solution (for they may not help in general). However, let's consider your one-sensor solution, assuming the variables are indeed displayed. The difficulty with such setups is that they inevitably break down at some point as the problem scales up. Specifically,

1. The dimensions of the sensor are fixed.

2. As the number of variables n, and hence displays, grows, the angle of the light rays from the furthest display approaches 0. There will not be enough real estate on the sensor for that light to impinge upon.

3. Thus, the problem poser can make n sufficiently large so as to render the sensor useless.

And one more thing: How does your sensor handle multiple simultaneous (or perhaps overlapping) inputs from displays all equidistant from it? Finally, please note that your proposal concerns only the computation described in Section 3.1.

4.7 This is an unconventional definition of computation

Misconception 7: Your definition of computation is unconventional.

Response: Absolutely, if one considers as *unconventional* the passage of time, or the interactions among the constituents of the universe subject to the laws of nature, or for that matter any form of information processing. In any case, the definition of computation embodied in the examples of Section 3 may be unconventional, but it is not unrealistic. Besides, it is important to realize that defining "to compute" as "what the Turing Machine does", which is quite commonly done (see, for example, [26]), leads to a logical reasoning that is viciously circuitous. If computation is what the Turing Machine does, then clearly the Turing Machine can compute anything that is computable. Furthermore, the "Church-Turing Thesis" would move immediately from the realm of "conjecture" to that of "theorem". The fact that it has not done so to this day, is witness to the uncertainty surrounding the widespread definition of "computation". As stated in Section 1.2, the counterexamples presented in Section 3, by contrast, *disprove* the "Church-Turing Thesis".

4.8 The Turing Machine is not intended for the computations you describe

Misconception 8: But the Turing Machine was not meant for this kind of computation.

Response: Precisely. Furthermore, the nonuniversality result covers all computers, not just the Turing Machine.

4.9 Abstract computational models ignore input and output considerations

Misconception 9: Abstract models of computation do not concern themselves with input and output.

Response: This opinion is held by those who believe that computation is the process that goes from input to output, while concerning itself with neither input nor output (see, for example, [26] cited in Section 4.7). By analogy, one might say that eating is all about digestion, and a Moonlight Sonata interpretation is nothing but the hitting of piano keys. Perhaps.

Perhaps not. A model of computation is useful to the extent that it is a faithful reflection of reality, while being mathematically tractable. In computer science, input and output are not cumbersome details to be ignored; they are fundamental parts of the computation process, which must be viewed as consisting of three essential phases, namely, input, calculation, and output. A computer that does not interact with the outside world is useless. In that sense, the thermostat in your house is more powerful than the Turing Machine. Please remember that the nonuniversality result goes beyond the Turing Machine (which, by the way, is a very primitive model of computation). To ask computer scientists to stick to the Turing Machine and not to look beyond, would be as if physics stopped progressing beyond the knowledge of the Ancient Greeks. In fact, looking ahead, if computers of the future are to be *quantum* in nature, their main challenge is expected to be, not calculation but, input and output.

4.10 In classical theory of computation, variables do not change with time

Misconception 10: Classical computability theory does not include variables that change with time.

Response: This is a serious lacuna in classical theory. The NCT shows that there are in fact many such lacunae. Their effect is to severely restrict the definition of computation. Indeed, to define computation merely as function evaluation, with fixed input and fixed output, is unrealistic and naive. It also trivializes the "Church-Turing Thesis" (turning it into a tautology), because it necessarily leads to the kind of sterile circular reasoning mentioned in the response to Misconception 7. Having said that, the time-varying variables counterexample is only one of many such refutations of universality in computation. Other examples arise in the computations described in Section 3 (for details, see [4] – [18], [27], [42] – [47]).

4.11 The CTT is concerned only with conventional computations

Misconception 11: The Church-Turing Thesis applies only to classical computations.

Response: This is certainly not the case. As the multitude of examples listed in [7] amply demonstrate, the commonly accepted statement of the "Church-Turing Thesis" [36] is essentially that there exists a Universal Computer capable of performing any computation that is possible on any other computer:

"Given a large but finite amount of time, the Turing machine is capable of any computation that can be done by any modern digital computer, no matter how powerful [31]."

There are no restrictions, exceptions, or caveats whatsoever on the definition of computation. In fact, a typical textbook definition of computation is as follows:

"A computation is a process that obeys finitely describable rules [49]."

Moreover, it is suggested in every textbook on the subject that, thanks to the fundamental and complementary notions of simulation and universality, every general-purpose computer is universal: A Turing Machine, a Random-Access Machine, a Personal Computer, a Supercomputer, the processing chip in a cell phone, are all universal. (The NCT shows this claim to be *false.*)

Going a little further, many authors consider all processes taking place in the Universe as computations. Interested readers may consult [16, 21, 24, 25, 28, 35, 38, 39, 49, 51, 52, 56, 57, 59, 61, 62, 63, 64, 65].

One may happen to agree with the authors listed in the previous paragraph on the pervasive nature of computation. However, in order to reach the conclusion about nonuniversality in computation, one does not in fact need to go that far. The counterexamples in Section 3 use very simple computations to refute universality: Computations whose variables change with time, computations whose variables affect one another, computations the complexity of whose steps changes with the passage of time, computations the complexity of whose steps depends (not on the time but) on the order of execution of each step, computations with uncertain time constraints, computations that involve variables obeying certain mathematical conditions, and so on. These are not uncommon computations. They may be considered unconventional only when contrasted with the standard view of computation as a rigid function evaluation, in which, given a variable x, it is required to compute $f(x)$, in isolation of the rest of the world.

4.12 Turing was aware of these difficulties

Misconception 12: Actually, Alan Turing knew that computations involving physical time would cause problems for his machine.

Response: This is the back-pedaling argument *par excellence.* Perhaps Turing knew about the computations described here, but this is doubtful. Indeed, Turing did propose the Oracle Machine (o-machine) [58] as an (unconventional) extension to his a-machine (today known as the Turing Machine). However, the o-machine has nothing to do with the problems proposed to counter universality in computation (see Section 3 and the references therein). Furthermore, the o-machine is a fanciful creation that appeals to some form of divine intervention in order to solve the problems it faces, while the computations that are used to prove nonuniversality are eminently executable on everyday computers, provided the latter are capable of performing the proper number of basic operations per time unit. The "Church-Turing Thesis" [36] is proof that both Turing and Church were convinced of the universality of the Turing Machine. In any case, there is no evidence of any writings by Turing, von Neumann, or anybody else, that hint to nonuniversality in computation, prior to the January 2005 paper on the myth of the universal computer [6].

4.13 Nonuniversality is not falsifiable

Misconception 13: The problem I see with the nonuniversality claim is that it does not appear to be falsifiable. It might be improved by thinking further about how to make the claim more specific, clear, testable, well-defined and worked out into detail. The entire claim would ideally be described in a few very worked out, specific, sentences - needing little external exposition or justification.

Response: Have you had a chance to read any of the papers [4] – [18], [42] – [47]? There is no lack of mathematical formalism there. For example in one of these papers, we use quantum mechanics to illustrate five of the six aforementioned paradigms [45]. But let me try this since you would like something simple and crisp: How about the best-understood problem in computer science, arguably that of sorting a sequence of numbers, but with a twist? See Section 3.6.2 for a description. For another simple exposition, see the computational problem presented in [19], and briefly introduced in Section 4.3; this is the most basic counterexample to universality in computation appearing in the literature.

4.14 A misrepresentation of past work

Misconception 14: A fine student just presented your NonUniversality in Computation work in my graduate theory course. Going simply on her presentation, I am not impressed.

Remark A: You start this paper by quoting Hopcroft and Ullman and saying that their statement is clearly false. You took this quote out of the context of all of theoretical computer science which clearly defines that a 'computation' is to start with the entire input presented and is given as much time as it wants to read this input and do its computation.

Remark B: It is true that since Turing, the nature of computation has changed to require real time interactions with the world. But you should not misrepresent past work.

Remark C: Having not studied your arguments at length, the only statement that I have gotten is that a machine is unable to keep up if you only allowed it to read in a fixed number of bits of information per time step and you throw at it in real time an arbitrarily large number of bits of information per time step. In itself, this statement is not very deep.

Response: For sure the student did a wonderful job, but somehow the message did not get across. Most likely, the reason for this is that you did not read any of the papers on nonuniversality, by your own admission.

Let us take your remarks (labeled for clarity in the preceding as *Remark A*, *Remark B*, and *Remark C*) one by one.

Response to Remark A:

1. Your definition of 'computation' cannot be found anywhere. It is certainly not in Hopcroft and Ullman's book [33], from which the quote in question was taken.

2. Your definition of 'computation' applies narrowly to some primitive computational models, such as the Turing Machine, Cellular Automata, and so on.

3. Because of (2), your definition trivializes the “Church-Turing Thesis”, rendering it a tautology: By defining ‘computation’ as ‘what the Turing Machine does’, it obviously follows that ‘the Turing Machine can compute everything that is computable’. As mentioned earlier (see the response to Misconceptions 7 and 10 in Sections 4.7 and 4.10, respectively), this is a typical example of circular reasoning.

4. Your definition is not sufficiently general to capture the richness and complexity of the notion of ‘computation’. Others have proposed more encompassing definitions of ‘computation’. Here are two typical quotes (others can be found in [55, 57, 62]):

   “Computation is physical; it is necessarily embodied in a device whose behaviour is guided by the laws of physics and cannot be completely captured by a closed mathematical model. This fact of embodiment is becoming ever more apparent as we push the bounds of those physical laws [54].”

   “Think of all our knowledge-generating processes, our whole culture and civilization, and all the thought processes in the minds of every individual, and indeed the entire evolving biosphere as well, as being a gigantic computation. The whole thing is executing a self-motivated, self-generating computer program [24].”

5. And one more thing about your point that a computation is to “start with the entire input presented”. Since *all* of the counterexamples in Section 3 indeed require the entire input to be “present”, one assumes that you mean “start with the entire input residing in the memory of the computer”. (Incidentally, some of the counterexamples in Section 3 assume the latter as well.) The relevant point here is this: Hopcroft himself has a well-known algorithm that makes no such assumption about the entire input residing in memory. As stated in response to Misconception 4 in Section 4.4, one of the most dramatic illustrations of the interweaving of input and output with calculation is the well-known linear-time planarity testing algorithm of Hopcroft and Tarjan [32]. In order to determine whether a graph with n vertices is planar, Step 1 of that algorithm avoids a computation that could potentially run in time at least quadratic in n, by reading in the edges of the graph one at a time from the outside world; if there is one more edge than the absolute maximum stipulated by Euler’s formula, namely $3n$ - 6 (the paper actually uses the loose bound $3n$ - 3 in order to allow for $n < 3$), the graph is declared nonplanar [32].

Response to Remark B:

1. It is good to see that we agree on the nature of computation. You should know, however, that the counterexamples to universality are not all about “real time interaction with the world”. There is a list of such counterexamples in Section 1, a brief description of some in Section 3, and a list of references in the bibliography [4] – [18], [27], [42] – [47]. One counterexample involving mathematical constraints (it is a variant of sorting, in which the entire input is available in memory at the outset of the computation) is described in Section 3.6.2. Note also that the nonuniversality result applies to putative ‘universal computers’ capable of interaction with the outside world. These ‘universal computers’ are endowed with unlimited memories and are allowed an indefinite amount of time to solve the problems they are given. They all still fail the test of universality.

2. However, I do take exception to the claim that I misrepresented past work. Below are quotes from famous computer scientists asserting, without caveat, exception, or qualification that a universal computer is possible, often explicitly stating that such a universal computer is the Turing Machine, and essentially taking for granted the unproven “Church-Turing Thesis” (see also [1], p. 123; [29], p. 233; [40], p. 152):

   “Church’s thesis: The computing power of the Turing Machine represents a fundamental limit on the capability of realizable computing devices [23], p. 2.”

   “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 [37], p. 223.”

   “It is possible to build a universal computer: A machine that can be programmed to perform any computation that any other physical object can perform. 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 [30], pp. 63–64."

Hundreds of such statements can be found in the literature; for a sample see [7].

3. Finally, please note that to correct previous mistakes is not to misrepresent the past. This is how science advances. Newton, Darwin, and Einstein were giants who built great scientific edifices. Each edifice, magnificent yet incomplete.

Response to Remark C:

1. As mentioned in Section 4.10, the "time-varying variables" computation is but one of many counterexamples to universality.

2. Surely you must know that a reasonable model of computation must be finite. The Turing Machine has a finite alphabet, a finite set of states, and a finite set of elementary operations per step. To assume otherwise would render the fields of complexity theory and algorithm design and analysis useless.

3. Your characterization of the nonuniversality result is erroneous. In Section 3, *computable functions* are described that no machine that claims to be universal can compute.

4. Let's try the following: *You* define a "universal computer" fulfilling the requirement of 'finite labor at each step' (to quote [37] again), and you will be given a computable function that it cannot compute.

4.15 Why are your counterexamples not Turing computable?

Misconception 15: I still find myself puzzled by things you say. I have no dispute that there are computations with various constrains such as real time constraints, time-varying aspects, global math constraints, and the like. I know that we have to deal with such computations already on existing machines. But since existing machines are Turing computable, I am puzzled why these constraints force us outside the limits of Turing computability. Can you help me understand?

Response: The Turing Machine of the 1930s as well as the 2015 machine on which this reply is being typed, operate in isolation of their environment. In particular, time for the problems we solve on these machines always means running time, and space means memory space. Imagine instead computations in which physical time and physical space play a central role, where the environment in which the computations take place affects the computations, and in turn is affected by them. Some of these computations, as you say, are starting to show up already. Many more will come as we expand the realm of computing. It is important to note here that this is not just about interaction. Even if equipped with the ability to interact with the outside world, unlike the Turing Machine but like all computers today, a machine that claims to be universal will fail to solve the problems that are proposed in Section 3 as counterexamples to universality.

How is this possible? Simply because physical time and physical space overwhelm any machine that satisfies the fundamental requirement of being fixed once and for all—the very definition of universality. Every problem described in Section 3 has a condition that must be satisfied in order for the computation to be said to have been carried out successfully. If this condition is not satisfied, the computation is judged to have failed, regardless of whether a correct output is later produced (by, for example, correcting the error, or by restarting the computation, or by simulation, and so on). Consider a computer that is supposed to control the reconfiguration of a physical structure of size n. It is required that the structure maintain its integrity at every step, otherwise the structure will collapse, and the operation will be declared to have failed. A universal computer will fail inevitably for a certain structure of a certain size.

In particular, a standard Turing Machine cannot solve the problem in the previous paragraph. By contrast, a Turing Machine with n tapes (and n heads) can. Interestingly, this contradicts the general belief that any computation by any variant of the standard Turing Machine can be simulated on the latter [37].

5 Conclusion

In conclusion, the reader is offered the following challenge. Anyone who still does not accept the NCT, namely, that universality in computation is a myth, has but one option: To prove it wrong. In order to do this, one must exhibit a universal computer capable of a finite and fixed number of operations per time unit, on which each one of the computations in the following classes (described in Section 3, and in [4] – [18], [27], and [42] – [47]), can be performed: Computations with time-varying variables, computations with time-varying computational complexity, computations with rank-varying computational complexity, computations with interacting variables, computations with uncertain time constraints, and computations with global mathematical constraints. It is important that the purported universal computer be able to execute successfully all aforementioned computations, since each one of them, by itself, is a counterexample to computational universality. For a simplified (and perhaps more colorful) version of the challenge, please see [5].

References

[1] Abramsky, S. et al.: Handbook of Logic in Computer Science. Clarendon Press, Oxford (1992)

[2] Akl, S.G.: Parallel Computation: Models and Methods. Prentice Hall, Upper Saddle River (1997)

[3] Akl, S.G.: Superlinear performance in real-time parallel computation. J. of Supercomputing. **29**, 89–111 (2004)

[4] Akl, S.G.: Non-Universality in Computation: The Myth of the Universal Computer. School of Computing, Queen's University.
`http://research.cs.queensu.ca/Parallel/projects.html`

[5] Akl, S.G.: A computational challenge. School of Computing, Queen's University.
`http://www.cs.queensu.ca/home/akl/CHALLENGE/A_Computational_Challenge.htm`

[6] Akl, S.G.: The myth of universal computation. In: Trobec, R., Zinterhof, P., Vajteršic, M., Uhl, A. (eds.) Parallel Numerics, pp. 211-236. University of Salzburg, Salzburg and Jozef Stefan Institute, Ljubljana (2005)

[7] Akl, S.G.: Universality in computation: Some quotes of interest. Technical Report No. 2006-511, School of Computing, Queen's University.
`http://www.cs.queensu.ca/home/akl/techreports/quotes.pdf`

[8] Akl, S.G.: Three counterexamples to dispel the myth of the universal computer. Parallel Proc. Lett. **16**, 381–403 (2006)

[9] Akl, S.G.: Conventional or unconventional: is any computer universal? In: Adamatzky, A., Teuscher, C. (eds.) From Utopian to Genuine Unconventional Computers, pp. 101-136. Luniver Press, Frome (2006)

[10] Akl, S.G.: Gödel's incompleteness theorem and nonuniversality in computing. In: Nagy, M., Nagy, N. (eds.) Proceedings of the Workshop on Unconventional Computational Problems, pp. 1-23. Sixth International Conference on Unconventional Computation, Kingston (2007)

[11] Akl, S.G.: Even accelerating machines are not universal. Int. J. of Unconventional Comp. **3**, 105–121 (2007)

[12] Akl, S.G.: Unconventional computational problems with consequences to universality. Int. J. of Unconventional Comp. **4**, 89–98 (2008)

[13] Akl, S.G.: Evolving computational systems. In: Rajasekaran, S., Reif, J.H. (eds.) Parallel Computing: Models, Algorithms, and Applications, pp. 1-22. Taylor and Francis, Boca Raton (2008)

[14] Akl, S.G.: Ubiquity and simultaneity: the science and philosophy of space and time in unconventional computation. Keynote address, Conference on the Science and Philosophy of Unconventional Computing, The University of Cambridge, Cambridge (2009)

[15] Akl, S.G.: Time travel: A new hypercomputational paradigm. Int. J. of Unconventional Comp. **6**, 329–351 (2010)

[16] Akl, S.G.: What is computation? Int. J. of Parallel, Emergent and Distributed Syst. **29**, 337–345 (2014)

[17] Akl, S.G., Nagy, M.: Introduction to parallel computation. In: Trobec, R., Vajteršic, M., Zinterhof, P. (eds.) Parallel Computing: Numerics, Applications, and Trends, pp. 43-80. Springer-Verlag, London (2009)

[18] Akl, S.G., Nagy: M.: The future of parallel computation. n: Trobec, R., Vajteršic, M., Zinterhof, P. (eds.) Parallel Computing: Numerics, Applications, and Trends, pp. 471-510. Springer-Verlag, London (2009)

[19] Akl, S.G. and Salay, N., On computable numbers, nonuniversality, and the genuine power of parallelism, Technical report no. 2015-625, School of Computing, Queen's University, Kingston, Canada, July 2015.

[20] Akl, S.G., Yao, W.: Parallel computation and measurement uncertainty in nonlinear dynamical systems. J. of Math. Model. and Alg. **4**, 5–15 (2005)

[21] Davies, E.B.: Building infinite machines. Br. J. for Phil. of Sc. **52**, 671–682 (2001)

[22] Davis, M.: The Universal Computer. W.W. Norton, New York (2000)

[23] Denning, P.J., Dennis, J.B., Qualitz, J.E.: Machines, Languages, and Computation. Prentice-Hall, Englewood Cliffs (1978)

[24] Deutsch, D.: The Fabric of Reality. Penguin Books, London (1997)

[25] Durand-Lose, J.: Abstract geometrical computation for black hole computation. Research Report No. 2004-15, Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon, Lyon (2004)

[26] Fortnow, L.: The enduring legacy of the Turing machine. `http://ubiquity.acm.org/article.cfm?id=1921573`

[27] Fraser, R., Akl, S.G.: Accelerating machines: a review. Int. J. of Parallel, Emergent and Distributed Syst. **23**, 81–104 (2008)

[28] Gleick, J.: The Information: A History, a Theory, a Flood. HarperCollins, London (2011)

[29] Harel, D.: Algorithmics: The Spirit of Computing. Addison-Wesley, Reading (1992)

[30] Hillis, D.: The Pattern on the Stone. Basic Books, New York (1998)

[31] Hopcroft, J.E. Turing Machines. Sci. Amer. **250**, 86–98 (1984)

[32] Hopcroft, J., Tarjan R.: Efficient planarity testing. J. of the ACM **21**, 549–568 (1974)

[33] Hopcroft, J.E., Ullman, J.D.: Formal Languages and their Relations to Automata. Addison-Wesley, Reading (1969)

[34] Hypercomputation. `http://en.wikipedia.org/wiki/Hypercomputation`

[35] Kelly, K.: God is the machine. Wired **10** (2002)

[36] Kleene, S.C.: Introduction to Metamathematics. North Holland, Amsterdam (1952)

[37] Lewis, H.R., Papadimitriou, C.H.: Elements of the Theory of Computation. Prentice Hall, Englewood Cliffs (1981)

[38] Lloyd, S.: Programming the Universe. Knopf, New York (2006)

[39] Lloyd, S., Ng, Y.J.: Black hole computers. Sci. Amer. **291**, 53–61 (2004)

[40] Mandrioli, D., Ghezzi, C.: Theoretical Foundations of Computer Science. John Wiley, New York (1987)

[41] Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967).

[42] Nagy, M., Akl, S.G.: On the importance of parallelism for quantum computation and the concept of a universal computer. In: Calude, C.S., Dinneen, M.J., Paun, G., Pérez-Jiménez, M. de J., Rozenberg, G. (eds.) Unconventional Computation, pp. 176-190. Springer, Heildelberg (2005).

[43] Nagy, M., Akl, S.G.: Quantum measurements and universal computation. Int. J. of Unconventional Comp. **2**, 73–88 (2006)

[44] Nagy, M., Akl, S.G.: Quantum computing: Beyond the limits of conventional computation. Int. J. of Parallel, Emergent and Distributed Syst. **22**, 123–135 (2007)

[45] Nagy, M., Akl, S.G.: Parallelism in quantum information processing defeats the Universal Computer. Par. Proc. Lett. **17**, 233–262 (2007)

[46] Nagy, N., Akl, S.G., Computations with uncertain time constraints: effects on parallelism and universality. In: Calude, C.S., Kari, J., Petre, I., Rozenberg, G. (eds.) Unconventional Computation, pp. 152-163. Springer, Heidelberg (2011)

[47] Nagy, N., Akl, S.G.: Computing with uncertainty and its implications to universality. Int. J. of Parallel, Emergent and Distributed Syst. **27**, 169–192 (2012)

[48] Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer, New York (1990)

[49] Rucker, R.: The Lifebox, the Seashell, and the Soul. Thunder's Mouth Press, New York (2005)

[50] Savage, J.E.: Models of Computation. Addison-Wesley, Reading (1998)

[51] Seife, C.: Decoding the Universe. Viking Penguin, New York (2006)

[52] Siegfried, T.: The Bit and the Pendulum. John Wiley & Sons, New York (2000)

[53] Sipser, M., Introduction to the Theory of Computation. PWS Publishing Company, Boston (1997)

[54] Stepney, S.: Journeys in non-classical computation. In: Hoare, T., Milner, R. (eds.) Grand Challenges in Computing Research, pp. 29-32. BCS, Swindon (2004)

[55] Stepney, S.: The neglected pillar of material computation. Physica D **237**, 1157–1164 (2004)

[56] Tipler, F.J.: The Physics of Immortality: Modern Cosmology, God and the Resurrection of the Dead. Macmillan, London (1995)

[57] Toffoli, T.: Physics and Computation. Int. J. of Th. Phys. **21**, 165–175 (1982)

[58] Turing, A.M.: Systems of logic based on ordinals. Proc. of the London Math. Soc. 2 **45**, 161–228 (1939)

[59] Vedral, V.: Decoding Reality. Oxford University Press, Oxford (2010)

[60] Wegner, P., Goldin, D., Computation beyond Turing Machines. Comm. of the ACM **46**, 100–102 (1997)

[61] Wheeler, J.A.: Information, physics, quanta: The search for links. In: Proc. of the Third Int. Symp. on Foundations of Quantum Mechanics in Light of New Technology, pp. 354-368. Tokyo (1989)

[62] Wheeler, J.A.: Information, physics, quantum: the search for links. In: Zurek, W. (ed.) Complexity, Entropy, and the Physics of Information. Addison-Wesley, Redwood City (1990)

[63] Wheeler, J.A.: At Home in the Universe. American Institute of Physics Press, Woodbury (1994)

[64] Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign (2002)

[65] Zuse, K.: Calculating space. MIT Technical Translation AZT-70-164-GEMIT, Massachusetts Institute of Technology (Project MAC). Cambridge (1970)