Technical Report No. 2006-508

EVEN ACCELERATING MACHINES ARE NOT UNIVERSAL*

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

March 7, 2006

Abstract

We draw an analogy between Gödel's Incompleteness Theorem in mathematics, and the impossibility of achieving a Universal Computer in computer science. Specifically, Gödel proved that there exist formal systems of mathematics that are consistent but not complete. In the same way, we show that there does not exist a general-purpose computer that is universal in the sense of being able to simulate any computation executable on another computer.

Keywords: Incompleteness, simulation, unconventional computation, universality.

1 Introduction

In 1931, the Austrian logician Kurt Gödel published his famous Incompleteness Theorem, arguably the most important result in the history of mathematics. The theorem established that there exist nontrivial formal systems of mathematics that, if consistent, cannot be complete. Seventy-five years after Gödel, we prove that universality in computer science cannot be achieved. Specifically, we show that no general-purpose computer can be built that claims to be universal, in the sense of being able to simulate *any* computation that is executable on another computer. We also draw an analogy between the two results that illustrates the similarities in their formal structure and philosophical implications.

Lest there be any misunderstanding, we wish to state at the outset, absolutely clearly and unequivocally: This paper is not about the Turing Machine [22]. Indeed, the inadequacy of the Turing Machine as a universal model of computation has been previously demonstrated amply, eloquently, and definitively. These demonstrations are well documented elsewhere (see, for example, [13, 14, 24, 25, 33, 37, 43, 48, 52, 54, 58]). One of many limitations of the Turing Machine is the fact that it is incapable of performing any computation that requires feedback from the outside world *during* the computation (by contrast, such computations are routinely performed on today's computers, from those in our cars to those in our airplanes).

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

We do not intend to belabor this point here any further. Rather, our purpose is to prove that (like the Turing Machine) no *other* computer, regardless of how powerful, be it theoretical or practical, whether existing or contemplated, can achieve universality, as long as it is designed and fixed once and for all (as required by the very definition of a universal computer). Having said that, we note that the Turing Machine is often referred to throughout the paper; these references, however, are only for historical and pedagogical purposes.

An overview of Gödel's theorem is provided in Section 2. In Section 3 we prove that, given a computer $\mathcal{U}_1$ with a claim to universality, a computation P_1 can be defined that cannot be carried out on $\mathcal{U}_1$. While P_1 is easily performed on another computer $\mathcal{U}_2$, the latter's algorithm is *impossible to simulate* on $\mathcal{U}_1$. Examples of such computations P_1 are presented in Section 4. Some final remarks are offered in Section 5.

2 Gödel's Result

On a hot August day in the year 1900, the illustrious German mathematician David Hilbert addressed the International Congress of Mathematicians assembled at the Université Sorbonne in Paris. Hilbert presented his colleagues with a list of problems on which, he believed, they should spend their time in the new century. Among these problems was the question of whether there exists a fixed set of true mathematical statements that can be used to prove *automatically* any new mathematical statement. Hilbert's objective was the formalization of mathematics.

A formal mathematical system consists of an alphabet of symbols, rules for combining these symbols into well-formed formulas, and a special set of well-formed formulas, the axioms of the system. Examples of such systems are arithmetic, geometry and so on. A formal system is *consistent* if it does not yield any logical contradiction. It is *complete* if all logically true propositions expressible in the system are provable within the system. Finally, a formal system is *decidable* if it is possible to determine (without providing a proof) whether a proposition expressible in the system is true. The hope was that, with a few axioms and just by mechanically moving symbols around, one could prove any mathematical system to be consistent, complete, and decidable.

With their monumental work [60], Whitehead and Russell felt that they had fulfilled Hilbert's dream. In it, they supplied rigid rules for manipulating symbols to produce theorems from axioms. Paradoxes such as *Russell's Paradox* (also known by many other names, including the *Barber's Paradox*), were completely avoided. Unfortunately, the approach had a flaw which was to be discovered by the twenty-five year old Kurt Gödel. His celebrated incompleteness result shows that there exist nontrivial formal systems that, provided they are consistent, must be incomplete.

In order to make his point, Gödel chose the formal system of simple arithmetic, that is, the natural numbers with equality, addition, and multiplication. Denoting this system by $\mathcal{A}_1$, consider the following proposition G_1, expressible within $\mathcal{A}_1$:

$$G_1 =< \text{This statement cannot be proved within } \mathcal{A}_1 >$$

Stepping outside of $\mathcal{A}_1$, Gödel proved that G_1 cannot be proved within $\mathcal{A}_1$. Indeed, proving it true within $\mathcal{A}_1$ would mean that a false statement is true, while proving it false within $\mathcal{A}_1$ would mean that a true statement is false. Since G_1 cannot be proved within $\mathcal{A}_1$, it follows that G_1 is true. This means that $\mathcal{A}_1$ is incomplete as it contains a true statement that cannot be proved within $\mathcal{A}_1$.

Technically, Gödel needed to accomplish three things. He had to be able to: express G_1 within $\mathcal{A}_1$, that is, using natural numbers, remove any self-referential ambiguity from G_1, and step out of $\mathcal{A}_1$ to prove that G_1 is true. All this he did through an ingenious technique, now known as *Gödel numbering* [39]. Every symbol, every well-formed formula, every proof, was mapped to (and henceforth represented by) a unique natural number. Suppose that G_1 maps to some number y. Thus y stands for the proposition which says: "There does not exist a proof whose number is x, such that x proves y". Gödel then proved that indeed no such x exists.

To appreciate the significance of this result, consider adding the recalcitrant proposition G_1 to $\mathcal{A}_1$, thus obtaining a new system $\mathcal{A}_2$. Is the latter now complete? Surely not, for now we can create a proposition G_2 not provable within $\mathcal{A}_2$. We can prove G_2 in a new system $\mathcal{A}_3$, which in turn has its own problem proposition G_3 not provable within it, and so on for ever. This is illustrated in Fig. 1.

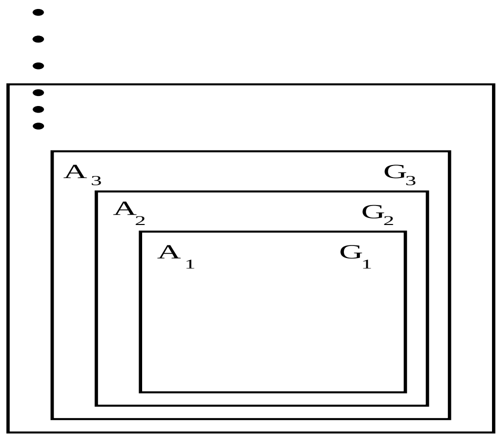

Figure 1: An incomplete formal system of mathematics.

This result became known as Gödel's Incompleteness Theorem, though it was one of two incompleteness theorems in his paper [27]. The second theorem stated that it is impossible to prove the consistency of a formal system of arithmetic within that same formal system.

It is interesting to note in passing the way in which various intellectual movements took hold of the result as a validation of their agenda [28]. Thus, for example, to the postmodernists, Gödel's Incompleteness Theorem implied that no firm foundation exists for any system of logic. The existentialists saw in it an end to rational and objective thought. Some philosophers [38] and mathematicians [41] argued on the strength of the theorem, that humans are superior to machines. One physicist even suggested that, thanks to Gödel's work, it is now obvious that the human brain is not a deterministic computer; rather, it is a quantum computer [48, 49].

Having dispatched completeness and consistency, Gödel then moved on to other pursuits. It was a pioneer of computing who would then tackle the third and only remaining component of Hilbert's question, namely, decidability.

3 No Computer Is Universal

Alan Turing's scientific legacy is rich, varied, and well documented [18, 55]. Our concern here is with two of his most profound contributions, namely, simulation and universality. We begin by stating the twin principles of Simulation and Universality as they are understood today. A few quotes gleaned from the literature in connection with these two principles are offered, whose role is to provide a current context for our subsequent discussion. We then pose the question as to whether these principles, which have so far remained unchallenged, are still valid in today's world of computing. The section concludes with our main result, namely, that no computer, however powerful, can ever fulfill the ideal of universality.

3.1 Turing, Simulation, and Universality

Aiming to address the question of decidability in mathematics, Turing needed to formally define what it means "to compute". His first and great insight was to invent a hypothetical computer that we now call the *Turing Machine* [21]. It consists of a control unit that can be in one of a finite set of *states*, the *symbols* 0 and 1, an infinite one-dimensional *tape* divided into squares each of which can hold a 0 or a 1, and a *head* for moving on the tape, reading and writing symbols. Turing settled the decidability question by proving that there exist problems that cannot be computed on this machine, and hence formal mathematical systems that cannot be decided [57].

While this was an important result, Turing's true gift to the field of computing (which he had by now created) is the idea of *simulation*. He showed how one machine can simulate another, leading to a universal machine that can simulate the actions of all others. This is a profound idea. Today, simulation and its primary consequence *universality*, are the main reasons behind the success of the computer as the most influential invention of the 20th century. Take *any* algorithm designed in any part of the world, it can be instantly programmed into *any* programming language, and before the day is over it can be running on *any* computer anywhere (even in outer space).

Simulation and universality are so entrenched as foundational ideas that they are often stated as principles:

1. **Simulation:** Any computation that can be performed on some computer A can be simulated exactly on any other computer B.

2. **Universality:** There exists a Universal Computer $\mathcal{U}$, such that any computation that can be performed on any other computer can also be performed on $\mathcal{U}$.

In order to dispel any possible doubt (and at the risk of stating the obvious), we make precise the meaning of *simulation* before going any further. Let an algorithm for computer A consist of the sequence of instructions I_0, I_1, $\ldots$, I_{N-1}, for some positive integer N. Computer B simulates this algorithm by executing each instruction I_i, either directly if its repertoire of instructions includes I_i, or indirectly by performing a sequence of its own instructions I'_0, I'_1, $\ldots$, I'_{k-1}, for some positive integer k, whose effect is equivalent to that of I_i. Thus, for example, suppose that I_i calls for multiplying two variables x_i and x_j, and computer B does not have the multiplication operation built in its repertoire. Assuming addition is available to B as an elementary (or primitive) operation, it can obtain the product of x_i and x_j by computing the sum of x_j copies of x_i.

The evidence for the veracity of the two aforementioned principles is, of course, overwhelming. Computers are everywhere and none of them (provided it has enough time, memory, and software) ever fails to do what another computer does.

3.2 Universality in the Literature

Every student of computing learns early on in his or her education that there exists a Universal Computer that can compute everything that can be computed. If this sounds as a tautology, recall what we have all been taught: There exists a computer that can imitate exactly the actions of any other computer. Most frequently, though not always, the Turing Machine is used as this omnipotent computer. Consider, for example, the following quotes:

> "The reader will find it incredible, at first sight, that some of these sets of simple operations could give rise to the full range of possible computations. [...] As we will see, it is possible to execute the most elaborate possible computation procedures with Turing machines whose fixed structures contain only dozens of parts. [...] Accepting Turing's thesis, we conclude that the universal machine can simulate any effective process of symbol-manipulation, be it mathematical or anything else; it is a completely general instruction-obeying mechanism." [40]

> "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." [31]

> "The computing power of the Turing machine represents a fundamental limit on the capability of realizable computing devices." [23]

> "[...] 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. [...] 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." [35]

> "**The Turing principle**
> (for physical computers simulating each other)
> *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]

> "[...] any algorithmic problem for which we can find an algorithm that can be programmed in some programming language, *any* language, running on some computer, *any* computer, even one that has not been built yet but *can* be built, and even one that will require unbounded amounts of time and memory space for ever-larger inputs, is also solvable by a Turing machine." [29]

> "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." [30]

In summary, what these statements are saying is that it is possible to build, once and for all, a certain device–the Universal Computer–henceforth capable of performing *any* computation that *any* other physically realizable device can perform. Such quotes are typical and can be found in more or less the same form in most books on computer science, as well as in the logic, physics, psychology, philosophy of the mind, and general science literature (for a small sample, see [5]). Two points are worth noting. First, while the Turing Machine is used as the 'universal computer' in most of these quotes, it should be clear that the same claims apply to any general-purpose computer:

> "That is to say, even the lowliest of today's home computers can be programmed to solve any problem, or render any environment, that our most powerful computers can, provided only that it is given additional memory, allowed to run for long enough, and given appropriate hardware for displaying its results." [24]

Second, these claims are indeed true for the vast majority of conventional computations fitting the following general paradigm: Given a datum x in the computer memory, compute $y = f(x)$, for some function f. Do they still hold when the essential *nature of computation* changes?

In fairness, it must be said that some computer scientists prefer to think of universality as a *conjecture* and properly refer to it as a 'thesis' (specifically, the so-called 'Church-Turing Thesis', when it comes to the Turing Machine). They state, quite correctly, that this conjecture may be *dis*proved (as we, in fact, do in this paper for *any* computer, not just the Turing Machine), but never proved, for lack of an all encompassing definition of what it means "to compute" [35]. (Indeed, defining *computation* in terms of the operations of the Turing Machine would clearly constitute a circular argument.) There is, however, at least one physicist who has elevated universality to the rank of 'fundamental principle of nature', evidently *without* proof (and not only in connection with the Turing Machine, but for all computers). Referring to the Turing principle quoted above, he writes [24]: "It is on par with the principle of conservation of energy and the other laws of thermodynamics: that is, it is a constraint that, to the best of our knowledge, all other theories conform to."

3.3 The Main Theorem

Thus we ask the two questions: Is simulation always possible? Is universality true? In this section we answer these questions in the negative. Specifically, there are computations that are executable on a given computer but impossible to simulate on any putative universal computer. We provide a short proof of this result in what follows. Examples of the computation used in our proof are presented in the following section.

Suppose that time is divided into discrete time units, and let $\mathcal{U}_1$ be a computer capable of $V(t)$ elementary operations at time unit number t, where t is a positive integer. Here, an elementary *computational* operation may be any one of the following:

1. Obtaining the value of a fixed-size variable from an external medium (for example, reading an input, measuring a physical quantity, and so on),
2. Performing an arithmetic or logical operation on a fixed number of fixed-size variables (for example, adding two numbers, comparing two numbers, and so on), and
3. Returning the value of a fixed-size variable to the outside world (for example, displaying an output, setting a physical quantity, and so on).

Each of these operations can be performed on every conceivable machine that is referred to as a *computer*. Together, they are used to define, in the most general possible way, what is meant by *to compute*: the acquisition, the transformation, and the production of information.

Now all computers today (whether theoretical or practical) have $V(t) = c$, where c is a constant (often a very large number, but still a constant). This is the assumption made in [6] to demonstrate the impossibility of a universal computer. That result is generalized here: We do not restrict $V(t)$ to be a constant. Thus, $V(t)$ is allowed to be an increasing function of time, such as $V(t) = t$, or $V(t) = 2^{2^t}$, and so on, as is the case for some hypothetical *accelerating machines* [7, 17]. (The idea behind these machines goes back to Bertrand Russell, Ralph Blake, and Hermann Weyl [15]; recent work is surveyed in [26].)

Finally, $\mathcal{U}_1$ is allowed to have an unlimited memory in which to store its program, as well as its input data, intermediate results, and outputs. Furthermore, no limit whatsoever is placed on the time taken by $\mathcal{U}_1$ to perform a computation.

Theorem U: $\mathcal{U}_1$ cannot be a universal computer.

Proof: Let us define a computation P_1 requiring $W(t)$ operations during time unit number t. If these operations are not performed by the beginning of time unit $t + 1$, the computation P_1 is said to have failed. Let $W(t) > V(t)$, for all t. Clearly, $\mathcal{U}_1$ cannot perform P_1. However, P_1 is computable by another computer $\mathcal{U}_2$ capable of $W(t)$ operations during the tth time unit. ■

It is important to note that, by the definition of universality, $\mathcal{U}_1$, once its features have been specified, is fixed and cannot change during the computation:

> "Although the Universal Turing Machine – as its name suggests – is universal, in the sense that it can simulate any specialized Turing machine (i.e., any computational task), the machine cannot be reconfigured and its architecture cannot be changed during operation." [56]

> "But the point of universality is that it should be possible to program a single machine, specified once and for all, to perform any possible computation, or render any physically possible environment." [24]

Despite being allowed extraordinary powers (such as, for example, the ability to increase the number of operations it can do at every consecutive time unit), $\mathcal{U}_1$ still fails to perform P_1. The computer $\mathcal{U}_2$ on the other hand is especially tailored to carry out P_1 and succeeds in doing so. This establishes that P_1 is definitely computable. Yet surprisingly, $\mathcal{U}_1$ is unable to simulate the actions of $\mathcal{U}_2$, notwithstanding the fact that no limit is placed on its memory or the time it is allowed to run. Would $\mathcal{U}_2$ be the new Universal Computer? Of course not, as we can easily define a computation P_2 that it cannot perform. A more powerful computer $\mathcal{U}_3$ can execute P_2, but is in turn defeated by a third computation P_3, and so on for ever. This is illustrated in Fig. 2.

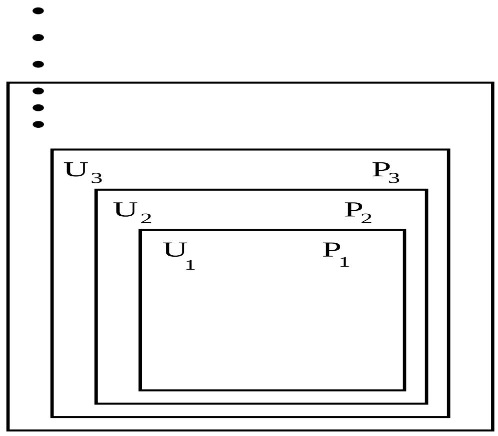

Figure 2: No general-purpose computer is universal.

The similarity between Fig. 1 and Fig. 2 is remarkable, and suggests a definite link between the underlying formal structures of the two results. The infinite ascent in the search for completeness in mathematics and in the quest for universality in computer science are very much alike.

4 Counterexamples To Universality

Theorem U tells us that for *any* computer $\mathcal{U}_1$ that purports to be universal (be it the Turing Machine, a supercomputer, or massively parallel processors), there exists a computational task P_1 that can easily be performed on another computer $\mathcal{U}_2$, but *not* on $\mathcal{U}_1$. Clearly, Theorem U applies in turn to $\mathcal{U}_2$ for which a task P_2 exists that it cannot compute. However, P_2 is computable on some other machine $\mathcal{U}_3$, and so on *ad infinitum*. In this section we provide examples of the computation P_1 used in Theorem U. Each of these computations is, in some sense, unconventional. (Other examples of such unconventional computations are described in [6, 7].)

In what follows we assume that $\mathcal{U}_1$ (unlike the Turing Machine, but like any ordinary computer today) is capable of communicating with the outside world during its computation.

Furthermore, the number of operations that $\mathcal{U}_1$ can perform during the tth time unit, that is $V(t)$, is a function that grows with t. This is not only unlike the Turing Machine, but also unlike *any* of today's computers (even the most powerful ones). But, according to Theorem U, computer $\mathcal{U}_1$, with all its power, still cannot be universal.

For definiteness, we assume henceforth that $V(t) = 2^{t-1}$, that is, $\mathcal{U}_1$ is able to *double* the number of operations that it can execute at every consecutive time unit, for $t = 1, 2, \ldots$. Now consider the second computer mentioned in the poof of Theorem U, namely, $\mathcal{U}_2$, which is used to express the fact that P_1, though it foils $\mathcal{U}_1$, is in fact executable on $\mathcal{U}_2$. Henceforth, we assume that $W(t)$, the number of operations that $\mathcal{U}_2$ can execute in time unit t, is always equal to a positive integer n, a measure of the size of P_1. Specifically, we will take $\mathcal{U}_2$ to be a parallel computer equipped with n processors, each of which able to perform one operation per time unit [3]. Of course, a computation P_2 of size $n+1$ or more renders $\mathcal{U}_2$ powerless.

4.1 Time-Varying Variables

For $n \geq 2$, there are n functions $f_0, f_1, \ldots, f_{n-1}$, each of one variable, and n physical variables, each of which changes as a function of time, namely, $x_0(t), x_1(t), \ldots, x_{n-1}(t)$. For all practical purposes, the change in the value of x_i as time moves from t to $t+1$ is random, unstoppable, and irreversible.

The computational problem that concerns us here is to evaluate $f_i(x_i(1))$, for $i = 0, 1, \ldots, n-1$. Each function evaluation requires one time unit. Since $\mathcal{U}_1$ can do one operation when $t = 1$, it can compute $f_0(x_0(1))$, for example. At this point, one time unit has elapsed, $t = 2$, and the remaining variables have changed to $x_1(2), x_2(2), \ldots, x_{n-1}(2)$. Clearly, $\mathcal{U}_1$ fails to perform the computation. Note that $\mathcal{U}_2$ succeeds in computing $f_i(x_i(1))$, for $i = 0, 1, \ldots, n-1$ during the first time unit. With all of the processors of $\mathcal{U}_2$ operating simultaneously, each processor evaluates one of the functions. However, simulating this algorithm on $\mathcal{U}_1$ is impossible, due to the relentless march of time. Two examples of computations with time-varying variables follow.

4.1.1 Computing with qubits

A quantum bit, or *qubit*, is a physical entity in a state of *superposition* of the values 0 and 1 [44]. Through 'prolonged' exposure to its environment, the qubit undergoes *decoherence*: Its superposition is said to collapse, and it loses its quantum properties. Usually, decoherence takes place in times significantly smaller than one second.

Suppose that $x_0(1), x_1(1), \ldots, x_{n-1}(1)$, represent the values of n independent qubits, each in a state of superposition, at time $t = 1$. Now $\mathcal{U}_1$ proceeds to compute $f_0(x_0(1))$, by which time the $n-1$ remaining qubits would most likely have undergone decoherence.

4.1.2 Identifying genes

In a living cell, the genes $x_0(t), x_1(t), \ldots, x_{n-1}(t)$, expressed as proteins change over time as the cell ages or becomes infected, and can evolve from simple regulatory processes to a full out war against invaders [59]. A biochemical application requires the number and types of genes expressed in a cell to be determined. However, since drastic cellular changes can occur in extremely small amounts of time, $\mathcal{U}_1$ has no hope of telling exactly which proteins are coded for in a given cell and how they interact with one another.

4.2 Interacting Variables

In this computation, x_0, x_1, ..., x_{n-1}, where $n \geq 2$, are the variables of a physical system. The variables are related to one another, in the sense that the value of each variable is a function of the values of all the other variables. Thus, $x_i = g_i(x_0, x_1, \ldots, x_{n-1})$, for some functions g_i, and $i = 0, 1, \ldots, n-1$. When the physical system reaches a state of equilibrium, the variables acquire values that remain unchanged so long as they are not disturbed. However, any operation on any one of the physical variables, affects its value and the values of all the other variables. This means that in the next time unit, the variables would have shifted to new values, unpredictably, uncontrollably, and irremediably.

Once again, given n functions f_i of one variable each, suppose we wish to compute $f_i(x_i)$, for $i = 0, 1, \ldots, n-1$, when the system is in a state of equilibrium. It takes one time unit to evaluate each function. Computer $\mathcal{U}_1$ will attempt this computation by evaluating $f_0(x_0)$. This disturbs the equilibrium, meaning that, one time unit later, all the variables x_0, x_1, ..., x_{n-1}, would have shifted to new values. In particular, $\mathcal{U}_1$ has no hope of obtaining $f_1(x_1)$, $f_2(x_2)$, ..., $f_{n-1}(x_{n-1})$, as the original values of x_1, x_2, ..., x_{n-1}, are lost forever. Nor does $\mathcal{U}_1$ have any way of simulating the actions of $\mathcal{U}_2$, which is capable of executing n operations in one time unit and thus easily computes all of $f_0(x_0)$, $f_1(x_1)$, ..., $f_{n-1}(x_{n-1})$, simultaneously, before the variables change value due to the disturbance. Two examples of such physical systems follow.

4.2.1 Sorting out an entanglement

In the same way as a single qubit is in a superposition of 0 and 1, an entire register of n qubits x_0, x_1, ..., x_{n-1}, can be put in a superposition of two states. There are 2^n such superpositions, in which the n qubits are said to be *entangled*: operating any one of them causes the superposition to collapse into one of the two original states. Any subsequent operation on the remaining $n-1$ qubits will yield a value that agrees with the state to which the superposition collapsed [42, 43].

4.2.2 Dependence for survival

Here n living organisms $x_0, x_1, \ldots, x_{n-1}$, are housed in a closed environment. The organisms depend on one another for survival. Performing any operation on one of the organisms in exclusion of the others may have the effect of disturbing the equilibrium sufficiently to provoke a serious adverse effect on the remaining organisms.

4.3 Time-Varying Computational Complexity

In traditional computational complexity theory, the number of operations required to solve a problem is a function of the *size* of the problem [4, 19]. In this section, we consider a paradigm that differs from the traditional one. Here, the computational complexity of a problem depends on *time*. Time-varying computational complexity is everywhere around us. For example, as software viruses spread with time they become more difficult to deal with. A spaceship racing away from Earth becomes ever harder to track. We concentrate in what follows on such computational tasks whose complexity, as illustrated by the aforementioned examples, grows with the passage of time.

4.3.1 Computing with deadlines

Given n variables, x_0, x_1, ..., x_{n-1}, it is required to compute $f_i(x_i)$, where the n functions f_0, f_1, ..., f_{n-1}, are entirely independent, and computing any one of them during time unit t requires 2^{t-1} operations. Furthermore, there is a deadline for reporting the results of the computations: All n values $f_i(x_i)$ are to be returned by the end of the third time unit. We assume $n > 3$.

How does $\mathcal{U}_1$ fare? It computes $f_0(x_0)$, which requires one operation, during the first time unit. During the second time unit, $f_1(x_1)$, now requiring two operations, is computed. Finally, $\mathcal{U}_1$ performs the four operations needed by $f_2(x_2)$ in the third time unit. At this point the deadline has been reached and none of

$$f_3(x_3), f_4(x_4), \ldots, f_{n-1}(x_{n-1}),$$

has been computed.

While $\mathcal{U}_1$ has failed, $\mathcal{U}_2$ succeeds. It evaluates all n values $f_i(x_i)$ during the first time unit, handily meeting the deadline.

4.3.2 Computing without deadlines

An alternative to the paradigm of Section 4.3.1 does away with the deadline, while still being able to defeat $\mathcal{U}_1$. Suppose it is the case that computing $f_i(x_i)$, for $i = 0, 1, \ldots, n-1$, requires 3^{t-1} operations if executed during the tth time unit. Now $\mathcal{U}_1$ can compute $f_0(x_0)$, but is unable to keep up beyond that point. Once again $\mathcal{U}_2$ succeeds.

5 Conclusion

The purpose of Hilbert's program in formalizing mathematics was twofold. The first goal was to contain infinity. Proofs for all true statements of a formal system were to be produced from a finite set of axioms. The second goal was to eliminate intuition from mathematics. By mechanizing proof generation, serendipity would no longer be part of the process of doing mathematics. Gödel's work demonstrated that, on the contrary, infinity is an integral part of mathematics and cannot be tamed. Mathematicians will always use their intuition to reason about the infinite.

Likewise, Theorem U shows that no finite computer can be universal. A new machine will always be needed to cope with the next challenge. The computational problems described in this paper imply that computation is a fundamental category of Nature, and as such it has no bounds. Its parameters are limitless. Time passes, inexorably, changing everything in its path. The constituents of our physical space constantly interact with one another, mutually affecting each other. As our world evolves, computations are taking place everywhere, all the time. The genie simply does not fit in the bottle.

It is important to realize that Theorem U applies to all known models of computation, both theoretical and practical. These include the Turing Machine, the Random Access Machine, and other idealized models [50], as well as all of today's general-purpose computers, including existing conventional (electronic) computers (both sequential and parallel), as well as contemplated unconventional ones [8, 11, 12], such as optical [36], biological [2, 20, 53], chemical [1] and quantum [9, 33, 45] computers. As long as it is defined at the outset as a contender for the title of 'universal', and its properties established in advance and fixed once and for all, no computer can be universal. This is true of hypercomputers [10, 14, 16], quantum hypercomputers [32, 46], quantum adiabatic hypercomputers [34], relativistic computers [25], black hole computers [37], interactive computers [58], X-machines [54], and analog neural networks [51, 52]. Even accelerating machines are not universal.

In this respect, we note that one way to realize an accelerating machine is to have a computer that is able to acquire new processors as needed during its computation. An original and intriguing way of doing this is through a biological approach known as *membrane computing*, which models the computations occurring in a living cell. Membrane computers can increase the number of their "processing units" while they are computing [13, 47]. This is accomplished by applying certain rewrite rules, a process which itself represents the computation. These computers, however, have a fixed initial configuration and a fixed set of rewrite rules. By Theorem U, they cannot be universal. For example, it takes time to apply the rules in order to achieve the required number of processing units, and this would be a problem in a computation with time-varying variables. Similar difficulties are easy to imagine in computations involving interacting variables (where a–necessarily perturbing–measurement is needed to find out how many processing units are needed) and computations with time-increasing computational complexity (where there may be no way of catching up, however many processing units are generated).

In his book *The Fabric of Reality* [24], David Deutsch formulates the following variant of the 'Turing principle' quoted in Section 3.2:

> "**The Turing principle**
>
> *It is possible to build a virtual-reality generator whose repertoire includes every physically possible environment.*" [24]

He goes on to write: "Is there any fundamental limit on how accurately any given era could be rendered? The Turing principle says that a universal virtual-reality generator can be built, and could be programmed to render any physically possible environment, so clearly it could be programmed to render any environment that did once exist physically." It should be clear by now that, unfortunately, this is not true. Theorem U and the three examples of Section 4 represent a refutation of this principle. Like a camera that can 'see' only part of a landscape, no finite device can capture at time t_0 a faithful snapshot of all the details of an arbitrary environment, needed for a correct playback at time $t_0 + 1$. If the present cannot be recorded, the past cannot be reproduced. Even if it were allowed unlimited 'time travel' to the past in order to recover what it missed (an extraordinary assumption in its own right), the finite device could not simulate time t_0 at time $t_0 + 1$. This remains true, even if we assume the presence of an infinite number of parallel worlds in which every moment of time 'exists' and is available for retrieval (another unsubstantiated claim). Even accelerating machines that can travel in time and space cannot recreate an accurate virtual reality of a bygone moment.

One cannot help but wonder how for seventy years (ever since Turing), we computer scientists have believed that a simple, finite, and fixed 'universal machine' can fully capture the complexity, immensity and ever changing nature of the whole Universe.

References

[1] A. Adamatzky, B. DeLacy Costello, and T. Asai, *Reaction-Diffusion Computers*, Elsevier Science, New York, 2005.

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

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

[4] S.G. Akl, The design of efficient parallel algorithms, in: *Handbook on Parallel and Distributed Processing*, Blazewicz, J., Ecker, K., Plateau, B., and Trystram, D., Eds., Springer Verlag, Berlin, 2000, pp. 13–91.

[5] S.G. Akl, Some quotes of interest, Technical Report, School of Computing, Queen's University, Kingston, Ontario, Canada, 2005. `http://www.cs.queensu.ca/Parallel/technical.html#Technical`

[6] S.G. Akl, The myth of universal computation, in: *Parallel Numerics*, Trobec, R., Zinterhof, P., Vajteršic, M., and Uhl, A., Eds., Part 2, Systems and Simulation, University of Salzburg, Austria and Jožef Stefan Institute, Ljubljana, Slovenia, 2005, pp. 211–236.

[7] S.G. Akl, Evolving computational systems, in: *Parallel Computing: Models, Algorithms, and Applications*, Rajasekaran, S. and Reif, J.H., Eds., Taylor and Francis, CRC Press, Boca Raton, Florida, 2006.

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

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

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

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

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

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

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

[15] B.J. Copeland, Even Turing machines can compute uncomputable functions, in [11], pp. 150–164.

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

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

[18] B.J. Copeland, Ed., *The Essential Turing*, Clarendon Press, Oxford, United Kingdom, 2004.

[19] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2001.

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

[21] M.D. Davis and E.J. Weyuker, *Computability, Complexity, and Languages*, Academic Press, New York, 1983.

[22] M. Davis, *The Universal Computer*, W.W. Norton, New York, 2000, p. 151.

[23] P.J. Denning, J.B. Dennis, and J.E. Qualitz, *Machines, Languages, and Computation*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[24] D. Deutsch, *The Fabric of Reality*, Penguin Books, London, England, 1997.

[25] 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.

[26] R. Fraser and S.G. Akl, Accelerating machines, Technical Report No. 2006-510, School of Computing, Queen's University, Kingston, Ontario, Canada, 2006.

[27] K. Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für Methematik und Physik*, Vol. 38, 1931, pp. 173–198.

[28] R. Goldstein, *Incompleteness: The Proof and Paradox of Kurt Gödel*, W.W. Norton, New York, 2005.

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

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

[31] J.E. Hopcroft and J.D. Ullman, *Formal Languages and their Relations to Automata*, Addison-Wesley, Reading, Massachusetts, 1969.

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

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

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

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

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

[37] S. Lloyd and Y.J. Ng, Black hole computers, *Scientific American*, Vol. 291, No. 5, November 2005, pp. 52–61.

[38] J.R. Lucas, Minds, machines and Gödel, *Philosophy*, Vol. 36, 1961, pp.112–127.

[39] R. McNaughton, *Elementary Computability, Formal Languages, and Automata*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[40] M.L. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, New Jersey, 1967.

[41] E. Nagel and D.R. Newman, *Gödel's Proof*, New York University Press, New York, 2001.

[42] 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.

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

[44] M. Nagy and S.G. Akl, Quantum computation and quantum information, *International Journal of Parallel, Emergent and Distributed Systems*, Vol. 21, No. 1, February 2006, pp. 1–59.

[45] M.A. Nielsen and I.L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, United Kingdom, 2000.

[46] 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.

[47] Gh. Păun, *Computing with Membranes*, Springer Verlag, Berlin, 2002.

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

[49] R. Penrose, *Shadows of the Mind*, Oxford University Press, New York, 1994.

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

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

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

[53] T. Sienko, A. Adamatzky, N.G. Rambidi, M. Conrad, Eds., *Molecular Computing*, MIT Press, Cambridge, Massachusetts, 2003.

[54] 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.

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

[56] C. Teuscher, Turing's connectionism, in [55], pp. 506–507.

[57] 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.

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

[59] G.H. Wei, D.P. Liu, and C.C. Liang, Charting gene regulatory networks: strategies, challenges and perspectives, *Biochemical Journal*, Vol. 381, Part 1, July 2004, pp. 1–12.

[60] A.N. Whitehead and B. Russell, *Principia Mathematica*, 3 vols, Cambridge University Press, Cambridge, 1910, 1912, 1913.