Computational Nonuniversality: Philosophical and Artistic Perspectives

Selim G. Akl[1]

This paper draws connections from the science of computation to philosophy and the visual arts. The motivation for this endeavor is computational nonuniversality, a fundamental theorem in theoretical computing established relatively recently. Two distinct mathematical proofs of this result are offered, one proof by counterexample, and one proof by contradiction. Both proofs show that simulation, upon which rests the principle of universality, is not always possible, thereby making the existence of a universal computer a myth. Both proofs are inspired by philosophy of science. Both are illustrated using an artist's conception.

Introduction

The science of computation has witnessed a tremendous success since its inception in the middle of the 20th century. Today, computers are crucial in every aspect of modern society, transforming communication, transportation, education, business, health care and entertainment, to name just a few of the areas benefiting from information technology. The idea behind this extraordinary impact is a simple one, namely, *universality*. The *principle of universality* states that given sufficient time and memory space, any computation that can be performed by a general-purpose computer can, through simulation, be performed by any other general-purpose computer (regardless of any architectural differences between the simulating and simulated computers, or how efficient or inefficient is the simulation). This is accomplished by having the second computer simulate (that is, imitate exactly in order to obtain the same effect) every step executed by the first computer, using its own (hardware and software) resources. Thus, for example, an email program that runs on a laptop can be made to run equally well on a mobile phone. This applies to far more complex computations, from studying the subatomic realm to exploring the far reaches of our universe. It is indeed fair to say that simulation is the engine that would make universality possible. Consequently, according to the principle of universality, every general purpose computer is universal, capable of carrying out through simulation any computation that is possible on any other computer. And yet, it seems natural to ask: is simulation *always* possible?

In reality, the principle of universality is but a conjecture, sometimes referred to as the Church-Turing thesis. This conjecture is impossible to prove in general because an all encompassing and agreed upon definition of what constitutes a computation does not exist. This is the case, despite the overwhelming number of instances providing evidence of its validity. An abundance of examples confirming a claim is not a proof, however. By contrast, it is actually possible to *disprove* the universality conjecture, and this is precisely what this paper is about.

It was shown recently that, in fact, the principle of universality is false in general; it does not apply to all computations. The reason for this is that, as it turns out, simulation is the weakest link, the Achilles heel in the quest for universality in computation. Put simply, *simulation is not always possible*. It immediately follows that a universal computer cannot exist. This result, to which we refer as computational nonuniversality (or nonuniversality in computation) is established using two distinct approaches:

1. A proof by counterexample, whereby an otherwise *computable* function, cannot be computed on a putative 'universal' computer, that is, on a computer that is finite and fixed once and for all.

2. A proof by contradiction, in which it is assumed that there indeed exists a 'universal' computer, and this assumption is then shown to lead to an absurd situation whereby this computer embarks on an unending computation.

These two approaches to disproving the existence of a universal computer are reviewed in this paper. The role of philosophy in interpreting these results is highlighted. The equally important contribution of the visual arts in illustrating the proofs is put in evidence.

Nonuniversality and incompleteness

In this section we draw an analogy between Gödel's Incompleteness Theorem in mathematical logic, and the impossibility of achieving a Universal Computer in computer science, that illustrates the similarities in the formal structure and philosophical implications of the two results. 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.

Proving nonuniversality in computation by counterexample

Let U_1 be a general-purpose computer. For the purpose of this proof, we suppose further that time is divided into discrete time units, and that U_1 is capable of $V(t)$ elementary operations at time unit number t, where t is a positive integer, $t = 1, 2, 3, \ldots$ 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). In order to make the nonuniversality result even stronger, in what follows 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 $V(t) = 2^t$, and so on.

Finally, 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 U_1 to perform a computation.

Nonuniversality Theorem: U_1 cannot be a universal computer.

Proof: Let us define a computation P_1 requiring $W(t)$ operations during time unit number t, for $t = 1, 2, 3, \ldots$ If these operations are not performed by the end of time unit t, the computation P_1 is said to have failed. Let $W(t) > V(t)$, for some t. Clearly, U_1 cannot perform P_1. However, P_1 is successfully completed by another computer U_2 capable of $W(t)$ operations during the t^{th} time unit, for $t = 1, 2, 3, \ldots$

It is important to note that, by the definition of universality, U_1, once its features have been specified, is fixed and cannot change during the computation. 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), U_1 still fails to perform P_1. The computer U_2 on the other hand is especially tailored to carry out P_1 and succeeds in doing so. This establishes

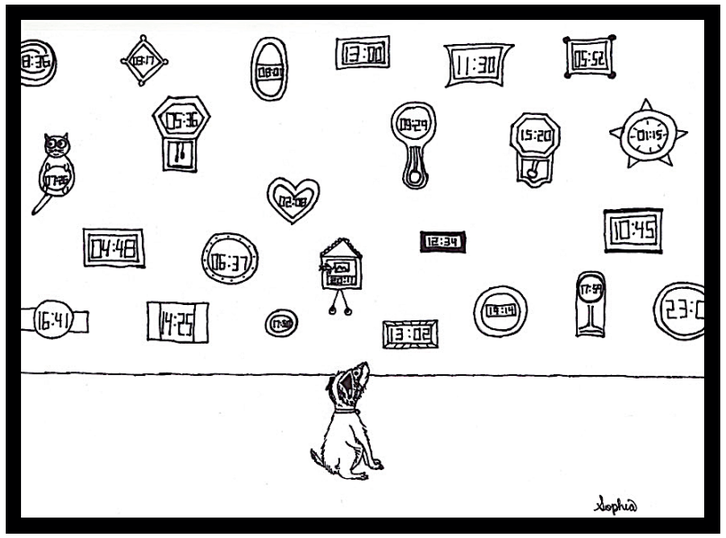

Fig. 1. Computing a function of *N* clocks.

that P_1 is definitely computable. Yet surprisingly, U_1 is unable to simulate the actions of U_2, notwithstanding the fact that no limit is placed on its memory or the time it is allowed to run. Would U_2 be the new Universal Computer? Of course not, as we can easily define a computation P_2 requiring $X(t) > W(t)$ operations during time unit t, that U_2 cannot perform. A more powerful computer U_3 can execute P_2, but is in turn defeated by a third computation P_3, and so on forever.

A pictorial example

As shown in Fig. 1, N clocks are hanging on a wall, $N > 1$. The clocks are digital, each displaying the time as a quadruple of digits AB:CD; for example, 19:48. All the clocks are working, ticking away synchronously. At every tick, each clock displays a new, but random, quadruple AB:CD—a time perhaps different from the ones displayed by the other $N - 1$ clocks. No clock has a memory; therefore, when at the following tick a new time is generated and displayed, the previous quadruple is lost forever. The wall is long at will, allowing N to be big at will. The problem to be solved is the following: For an arbitrary number of clocks N, it is required to compute a function (for example, the average) of the N times displayed at a given moment T. A computer capable of exactly N operations per time unit, and claiming to be 'universal', readily solves the problem. It does so by reading the times displayed by the N clocks, and computing a function of these values all in one time unit (before time unit $T + 1$ when the clocks update their displays). This computer, however, is thwarted if even one clock is added to the wall!

An algorithmic counterexample to universality

Consider the well-known quintessential computational problem of sorting a sequence of numbers stored in the memory of a computer. For a positive even integer n, where $n \geq 8$, let n distinct integers be stored in an array S with n locations $S[1]$, $S[2]$, …, $S[n]$, one integer per location. Thus $S[j]$, for all $1 \leq j \leq n$, represents the integer currently stored in the j^{th} location of S. In a variant of the standard sorting problem, 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:
   $S[j] > S[j + 1] > S[j + 2]$,
   for all $1 \leq j \leq n - 2$.

2. When the sort terminates we have:
   $S[1] < S[2] < \ldots < S[n]$.

An algorithm for a computer M capable of $n/2$ operations per time unit solves the aforementioned variant of the sorting problem handily in n steps, by means of predefined pairwise swaps applied to the input array S, during each of which $S[j]$ and $S[k]$ exchange positions (using an additional memory location for temporary storage). Thus, for example, the input array

7 6 5 4 3 2 1 0

would be sorted by the following sequence of comparison/swap operations (each pair of underlined numbers are compared to one another and swapped if necessary to put the smaller first):

7 6 5 4 3 2 1 0
6 7 4 5 2 3 0 1
6 4 7 2 5 0 3 1
4 6 2 7 0 5 1 3
4 2 6 0 7 1 5 3
2 4 0 6 1 7 3 5
2 0 4 1 6 3 7 5
0 2 1 4 3 6 5 7
0 1 2 3 4 5 6 7

However, an alleged 'universal' computer capable of fewer than $(n/2)$ operations per time unit, cannot solve the problem consistently. Confronted with the input array shown above, it fails to satisfy the requirement that at no time three consecutive values are listed in decreasing order. Is M universal? Certainly not, for it too cannot solve the sorting problem when the input sequence has length greater than n.[2]

A philosophical precursor to nonuniversality

In 1931, the twenty-five year old Austrian logician Kurt Gödel published his famous Incompleteness Theorem, arguably the most important result in the history of mathematical logic. The theorem established that there exist nontrivial formal systems of mathematics that, if consistent, cannot be complete.[3]

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 A_1, consider the following proposition G_1, expressible within A_1:

G_1 = < This statement cannot be proved within A_1 >.

Stepping outside of A_1, Gödel proved that G_1 cannot be proved within A_1. Indeed, proving it true within A_1 would mean that a false statement is true, while proving it false within A_1 would mean that a true statement is false. Since G_1 cannot be proved within A_1, it follows that G_1 is true. This means that A_1 is incomplete as it contains a true statement that cannot be proved within A_1.

To appreciate the significance of this result, consider adding the recalcitrant proposition G_1 to A_1, thus obtaining a new system A_2. Is the latter now complete? Surely not, for now we can create a proposition G_2 not provable within A_2. We can prove G_2 in a new system A_3, which in turn has its own problem proposition G_3 not provable within it, and so on forever. This result became known as Gödel's Incompleteness Theorem.

The infinite ascent of formal systems A_1, A_2, A_3, …, in the Incompleteness Theorem, is directly paralleled by the infinite ascent of aspiring 'universal' computers U_1, U_2, U_3, …, in the Nonuniversality Theorem.

It is interesting to note in passing the way in which various philosophical movements took hold of the result as a validation of their agenda. 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 mathematicians and an assortment of thinkers in various disciplines 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.

Nonuniversality and unending recursion

Here the philosophical concept of infinite regress is used to develop a proof of nonuniversality that is distinct from the proof by counterexample. The proof itself is presented within the general framework of logic known as proof by contradiction. When invoking contradiction to prove a theorem, we begin by assuming that the claim in the theorem is false; we then show that this assumption leads logically

to an absurdity (hence the Latin name *reductio ad absurdum* for this style of proof in mathematics).

Proving nonuniversality in computation by contradiction

An alternative proof of the Nonuniversality Theorem is provided in what follows.[4] Let U_1 be a general-purpose computer.

> **Nonuniversality Theorem.** U_1 cannot be a universal computer.
> **Proof:** Let us assume, as is commonly the case in computer science, that there exists a 'universal' computer capable of simulating any possible computation *C*, the latter being the result of another computer *M* executing a certain program on an input *I*. For brevity, in what follows we use *M* to represent both the computer as well as the program being simulated.

In order to be specific, and without any loss of generality, let the assumed 'universal' computer be U_1. We write $U_1(M, I)$ to express the fact that U_1 takes *M* and *I* as input and simulates the computation *C* by performing the actions of *M* on *I*. Note that U_1 is used here, by definition, as a *simulating computer*. Such a computer needs the description of another computer (*M*) in order to simulate *that* computer as it runs a program on its input (*I*). In other words, U_1 does not act directly on an input (such as *I*).

In what follows, let $C = (M, I)$ be a terminating computation, meaning that *M* runs on *I* and halts in a finite number of computational steps. We write $U_1[C]$ as a shorthand for $U_1(M, I)$, the simulation of *C* by U_1, the latter also a terminating computation.

It is evident, from the principle of universality, that the actions of U_1 itself should be possible to simulate. The question is: 'Who' is to simulate a computation performed by a universal computer? Specifically, how are the actions of U_1–as it simulates *C*, that is, $U_1[C]$–*themselves* to be simulated?

There are two options.

> **Option 1.** U_1 simulates itself. We write $U_1[U_1[C]]$ to indicate that U_1 is simulating $U_1[C]$. This means that U_1 is executing the actions of itself (U_1, the computer), on the simulation ($U_1[C]$, the input); we write:
>
> $$U_1[U_1[C]] = U_1(U_1, U_1[C]).$$
>
> The right hand side of the above expression, has three U_1s: the first is the simulator, the second is the computer being simulated, and the third, $U_1[C]$, is the input. We therefore have:
>
> $$\begin{aligned} U_1[U_1[C]] &= U_1(U_1,U_1[C]) \\ &= U_1(U_1[U_1[C]]) \\ &= U_1(U_1(U_1,U_1[C])) \\ &= U_1(U_1(U_1[U_1[C]])) \\ &= U_1(U_1(U_1(U_1,U_1[C]))) \\ &\ldots \end{aligned}$$
>
> This leads to a *self-referential* infinite regress, with U_1 simulating itself, simulating itself, simulating itself,… The computation just described is a non-terminating process (one could say that it is not even a computation, by definition, since it does not halt, but that is not important for our purpose). It follows that U_1 has failed to simulate itself, while executing a *terminating* computation *C*.
>
> **Option 2.** We can stipulate the existence of a more powerful universal computer U_2 that simulates $U_1[C]$. Again, this leads, in turn, to an *ascending* infinite regress, featuring a sequence of ever more powerful computers U_2, U_3, U_4, … , U_n, U_{n+1}, U_{n+2}, …, performing simulations C_1, C_2, C_3, … , C_n, C_{n+1}, C_{n+2}, …
>
> Formally,
>
> $$\begin{aligned} C_1 &= U_2[U_1[C]], \\ C_2 &= U_3[U_2[U_1[C]]], \\ C_3 &= U_4[U_3[U_2[U_1[C]]]], \\ &\ldots \\ C_{n+1} &= U_{n+2}[U_{n+1} \ldots U_4[U_3[U_2[U_1[C]]]]\ldots], \\ &\ldots \end{aligned}$$
>
> and so on *ad infinitum*. In the unending sequence of computations described here, computer U_{n+1} is needed to simulate U_n, for $n = 1, 2, 3, \ldots$, given that computer U_n cannot simulate itself (as shown in Option 1 for U_1).

Both Option 1 and Option 2 are computationally absurd, leading to one conclusion: the assumption regarding the existence of a universal computer U_1 is false. No computer is universal.

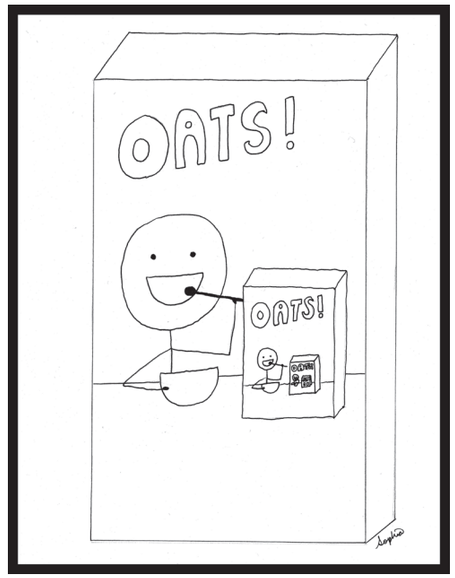

Fig. 2. Self reference and unending recursion.

Absolute idealism and self-referential cartography

As the sun was setting on the 19th century, idealist philosopher Josiah Royce proposed an interesting thought experiment. Imagine, he suggested, we could draw a detailed map of England. The map would be so precise as to contain every road, every river, every hill, every plain, and so on. A flat terrain will be chosen and the map inscribed on the surface of the English countryside. But then something unexpected would happen. Since the map is now part of the landscape, in order for it to be exact, it would necessarily have to contain a copy of itself. That copy would therefore inevitably contain a copy of itself as well, and the copy of the copy a copy, …; the process will continue indefinitely. This self-referential map-within-a map construction is known as the map of England problem and is sometimes used in studies of consciousness.[5] A whimsical illustration of self-reference and the potential for infinite regress is shown in Fig. 2.

Conclusion

On a hot August day in the year 1900, the illustrious German mathematician David Hilbert addressed the International Congress of Mathematicians assembled at the 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.

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, the Nonuniversality Theorem shows that no finite computer can be universal. A new machine will always be needed to cope with the next challenge. The resemblance between the Nonuniversality Theorem and the Incompleteness Theorem is captured by the

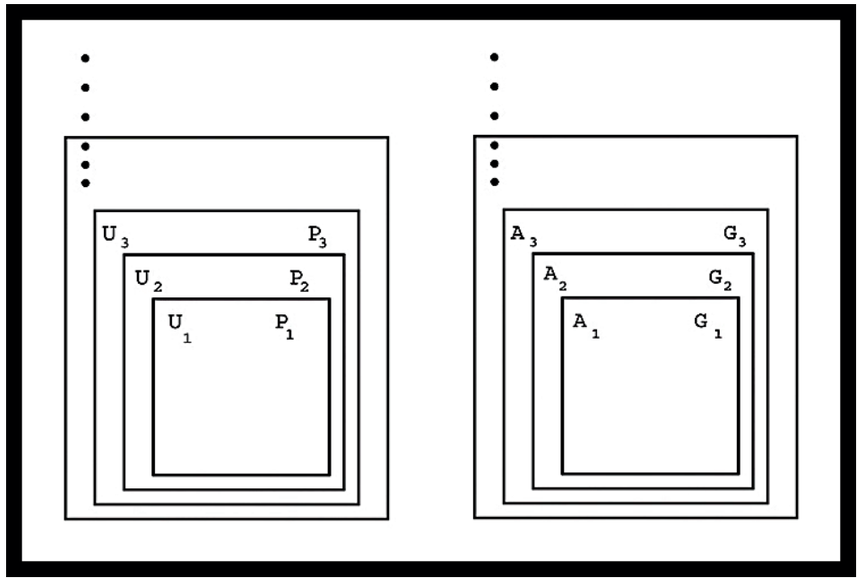

Fig. 3. Nonuniversality and incompleteness.

drawing in Fig. 3. For every computer thought to be universal, there exists a computational problem that it cannot solve, even if given unbounded time and space; a more powerful computer is required. Similarly, for every set of axioms thought to be complete, there exists a proposition that it cannot prove; an augmented set of axioms is required.

The idea of nonuniversality in computation is not a new one. Several proofs *by counterexample* of the non-existence of a universal computer have been presented since early in this century. They described different unconventional computational paradigms that falsify the idea of a universal computer. Examples of such paradigms include computations with: time varying variables, time varying computational complexity, rank varying computational complexity, interacting variables, uncertain time constraints, mathematical constraints, global variables, and so on. These computational problems 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.

By contrast with the proof by counterexample, the proof of nonuniversality in computation *by contradiction* is a more recent result. It was motivated when an obvious and logical question was asked, apparently for the first time ever: If simulation is the bedrock of computing, and the supposed 'universal' computer can simulate any computation, how can the actions of the 'universal' computer *itself* be simulated? The answer led once again to the collapse of the notion of universality. This, in turn, leads here to the following interesting observation. The invention in 1899 of the self-referential map-of-England metaphor, bringing about an inevitable infinite regress along with it, predates by 121 years the discovery of a proof, by self-simulation, of computational nonuniversality. Miraculously, the former (old result) serves as a striking illustration of the latter (new result), as demonstrated in Fig. 4.

We note in closing that nonuniversality in computation applies to all known computational models and existing conventional computers, both sequential and parallel, as well as applying to all future unconventional computers, including quantum computers, biomolecular

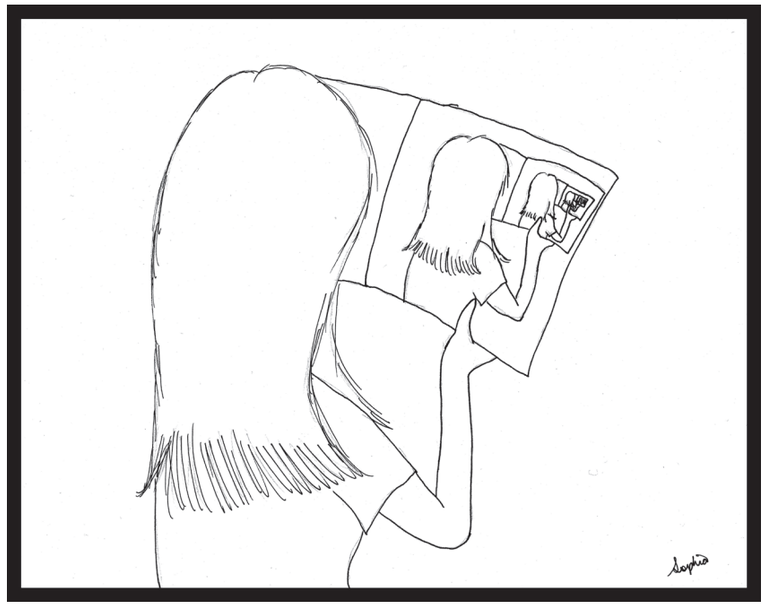

Fig. 4. Self-simulation and the ensuing infinite regress.

computers, chemical computers, and so on. Like the Halting Problem in Computer Science, Incompleteness in Mathematics, and the Uncertainty Principle in Physics, Nonuniversality in Computation is a limiting theorem, an impossibility result.

I am grateful to my daughter Sophia for the beautiful drawings in this paper.

[1] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada, akl@cs.queensu.ca.

[2] S. G. Akl, "Unconventional computational problems," in R. A. Meyers (ed.), *Encyclopedia of Complexity and Systems Science*, New York, Springer, 2018, 631-639.

[3] K. Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I," *Monatshefte für Methematik und Physik 38*, 1931, 173-198.

[4] S. G. Akl, "A map of England, the simulator simulated, and nonuniversality in computation," to appear in: *International Journal of Unconventional Computing*.

[5] J. Royce, *The World and The Individual, First Series: The Four Historical Conceptions of Being*, New York, Dover Publications, 1959, 504-505.