The Myth of Hypercomputation

Martin Davis

To appear in a "Turing Festschrift" published by Springer

Summary. Under the banner of "hypercomputation" various claims are being made for the feasibility of modes of computation that go beyond what is permitted by Turing computability. In this article it will be shown that such claims fly in the face of the inability of all currently accepted physical theories to deal with infinite precision real numbers. When the claims are viewed critically, it is seen that they amount to little more than the obvious comment that if non-computable inputs are permitted, then non-computable outputs are attainable.

The Impossible As a Challenge

> *Why, sometimes I've believed as many as six impossible things before breakfast.* – Lewis Carroll

Despite the fact that it has been known for over a century that it is impossible to devise a construction for dividing a given angle into three equal parts using only straight-edge and compass, hopeful amateurs continue to bring forth constructions that purport to do exactly that. It is as though the word "impossible" is seen as a challenge. Although the laws of thermodynamics have made it plain that the search for a perpetual motion machine is an exercise in futility, inventors claiming to have constructed such a device still besiege patent offices and manage to obtain financial support from gullible investors ([16]).

Over the centuries, mathematicians had often found solutions to problems in the form of procedures that could be carried out in a step-by-step fashion, where each step was entirely mechanical, capable of being carried without any creative thought; such procedures are called *algorithms*.[1] During the 1930s, as a result of the work of a number of logicians, it became possible to explain with full precision what it means to say for some given problem that an algorithm exists providing a solution to that problem. Moreover it

[1] The word "algorithm" is derived from the twelfth century Arabic mathematician al-Khwarizmi. Originally used to refer to the rules for calculating with the "Arabic" numerals we use today (originating in India), the word gradually came to refer to any mechanical procedure. In particular, the procedure due to Euclid for finding the greatest common divisor of two integers by successive division has been called the "Euclidean algorithm" at least since the nineteenth century. Going further back one can find the word (or its variant "algorism") in the work of Fibonacci, Leibniz (cf. [19]), and Euler. Today every serious university computer science department offers courses in the design and analysis of algorithms.

then became feasible to prove that for certain problems no such algorithm exists, that it is *impossible* to specify an algorithm that provides a solution to those problems. Because the computing machinery that became available beginning in the 1950s were, in an important sense, physical realizations of the idealized computational models that had been developed by the logicians, it was generally held that it is impossible to construct a physical device capable of solving these "unsolvable problems". However in recent years a number of researchers, marching under the banner of "hypercomputation", unwilling to accept as fact this impossibility, have been making proposals to overcome this barrier. In assessing their claims, it will be important to be clear about the relation between the abstract mathematical theory of computability and modern computing machinery. A crucial and often ignored aspect of this relation is that while the abstract theory is involved essentially with the mathematical infinite, physical computers are necessarily finite objects.

Algorithms and Infinity

It is a fact of life that we are finite beings and that our calculations are carried out on data that is not only finite, but is sufficiently limited in extent to fit in the space available in such media as sheets of paper or computer disks. Nevertheless, it has turned out that the appropriate way to formulate algorithms is as though they are intended to apply to initial data of arbitrary size. We can see this even in the simple algorithm for adding two numbers that we all learned as children:

$$\begin{array}{r} 15 \\ +17 \\ \hline 32 \end{array} \qquad\qquad \begin{array}{r} 3456789234568921 \\ +8732198623456521 \\ \hline 12188987858025442 \end{array}$$

The same algorithm applies in these two cases although the numbers being added are of very different size. In fact it is clear that the same algorithm will work regardless of the size of the addends. One can imagine applying it to numbers so large that, written out, they would stretch from one end of our galaxy to the other! Almost all known algorithms have this same property: although intended to deal only with finite initial data, and always yielding finite results, they will behave correctly regardless of the size of the data. In fact one of the principal measures of the complexity of a given algorithm is based on its "asymptotic" behavior – that is, its behavior as the size of the initial data increases without limit.[2]

[2] For example, the complexity of algorithms designed to *sort* data into numerical or alphabetic order is usually measured in terms of the number of comparisons required as a function of the number of items being sorted. Thus the crudest algorithms for sorting n items require a number of comparisons proportional to n^2 , whereas more sophisticated algorithms manage with a number proportional to $n \log n$. For large n, this latter number is much smaller.

The use of the word "mechanical" in explaining what an algorithm is, suggests a machine, and indeed adding machines that may be said to *implement* the addition algorithm exhibited above were once a commonplace. But it is worth noting that unlike the abstract algorithm that countenances no limitation on the size of the numbers being added, a machine implementing this algorithm, being a finite physical object, is constrained to accept only numbers smaller than some definite amount.

Over the centuries, many algorithms have been developed to solve various problems, but until the 1930s mathematicians had no way to prove that for certain problems an algorithmic solution is impossible. This option only became available with the work of the logicians Gödel, Church, Post, and especially Turing who provided precise characterizations of algorithmic solvability [7]. As Robin Gandy explained:

> Both Church and Turing had in mind calculation by an abstract human being using some mechanical aids (such as paper and pencil). The word 'abstract' indicates that the argument makes no appeal to the existence of practical limits on time and space. ([14])

Although the various characterizations were superficially different, it turned out that they were equivalent to one another. The assertion that these equivalent notions provide a precise explication of the previously unanalyzed intuitive concept of algorithmic solvability has come to be called *Church's Thesis* or *The Church-Turing Thesis*. Making use of this work a considerable number of problems have been proved to be unsolvable – in the sense that no algorithmic solution for them is possible. In Turing's own classical paper [20] he proved the unsolvability in this sense of a problem in mathematical logic known as the Entscheidungsproblem.[3]

Turing introduced the term *computable* for his characterization of algorithmic solvability which he developed by imagining a human being carrying out a computation, and, by removing, one after another, successive layers of irrelevant complication, arriving at his celebrated notion of what has come to be called a *Turing machine*. These "machines" are mathematical abstractions that do not, and can not, exist in the physical world. Turing machines accommodate inputs of arbitrary size on an infinite linear "tape" ruled into individual cells on each of which a symbol can be written. At any instant the machine is in one of a finite number of "states", and is "scanning" one of these cells. The tape contents is modified step-by-step by the moves of the machine. These moves consist of changing the symbol on the currently scanned cell, causing the scanned cell to be the one either to the immediate right or to the immediate left of this cell, and finally entering a new state.

[3] The Entscheidunsproblem is, in effect, the problem of providing a general algorithm to determine whether some conclusion can be logically inferred from a given finite set of premises using the rules of classical logic; it should be mentioned that Church had also proved this problem to be unsolvable.

The precise action depends only on the symbol in the scanned cell and the machine's current state. Turing, in effect, held that any algorithmic process is equivalent to what some appropriate Turing machine can accomplish if an appropriate input string of symbols is placed on its tape.[4] A first example of an algorithmically unsolvable problem, the so-called halting problem, was then readily obtainable. One form of this result is: *There is no algorithm which will determine of a given Turing machine whether it will ever halt when started with a completely blank tape.*[5]

Although Turing had been led to think along these lines by his desire to show that the Entscheidungsproblem is unsolvable, he went on to obtain an additional result of great importance. He realized that it is possible to design a single "universal" Turing machine, which all by itself could do the work of any other Turing machine. Here's the idea: imagine placing on the tape of a Turing machine a symbolic representation or "code" for some arbitrary Turing machine $\mathcal{M}$ and in addition, an input to $\mathcal{M}$ as in the diagram below.

Code of Turing machine $\mathcal{M}$	**Input to $\mathcal{M}$**

The universal machine would then do exactly what $\mathcal{M}$ would have done if presented with that same input. Turing wrote out in full the tables showing the moves of his universal machine.[6] That such a single Turing machine could execute, as it were, any algorithm whatever pointed the way to building a genuine physical machine that would be all-purpose in this same way, subject only to limitations of space and time. The universal machine opened other vistas to be realized in practice only much later. As I wrote [10,11]:

> Before Turing the ... supposition was that ... the three categories, machine, program, and data, were entirely separate entities. The machine was a physical object ... hardware. The program was the plan for doing a computation ... The data was the numerical input. Turing's universal machine showed that the distinctness of these three categories is an illusion. A Turing machine is initially envisioned as a machine ..., *hardware*. But its code ... functions as a *program*, detailing the instructions to the universal machine ... Finally, the universal

[4] It is an unimportant detail that Turing's paper was mainly written in terms of algorithms for computing the successive 0s and 1s in the binary expansion of a real number.

[5] An unimportant technical detail: I assume a tape infinite in both directions as in my [6]. If, as with Turing, the tape is infinite in only one direction, one should specify in addition that the initial scanned cell is the one at the end of the tape.

[6] It is worth noting that given confidence in the success of Turing machines in capturing the concept of algorithmic solvability, the existence of such a universal machine is inevitable. This is because it is easy to provide an algorithm that using the table defining any given Turing machine $\mathcal{M}$ will step-by-step perform exactly like $\mathcal{M}$. Thus, if one believes that any algorithm can be carried out by a Turing machine, that would have to be the case for this algorithm as well.

> machine in its step-by-step actions sees the ... machine code as just more *data* to be worked on. This fluidity ... is fundamental to contemporary computer practice. A *program* ... is *data* to the ... compiler.

Turing Machines, The Church-Turing Thesis, and Modern Computers

Although no, necessarily finite, physical device can emulate a true universal Turing machine with its infinite tape and ability to deal with arbitrarily large data, the existence, even as a mathematical abstraction, of Turing's universal "machine", brought into focus the goal of building a machine that could usefully approximate universality. In effect such a machine would "implement" Turing's universal machine in much the same way that an adding machine implements the simple addition algorithm displayed above.[7] The myriad tasks that the computers on our desktops routinely accomplish attest to the great success in achieving that goal. But it was by no means clear ab initio that an all-purpose machine could be built that could do useful work. In the years just after the second world war, Turing and von Neumann each produced plans for such a machine. Before anything was actually built, von Neumann wrote a sophisticated sorting program for the EDVAC (a proposed computer to be built at the University of Pennsylvania) to see whether a machine designed principally for heavy-duty number crunching could also handle such an essentially logical task. Having succeeded, he stated with satisfaction that the machine was acceptably "all-purpose". A year later, in 1946, writing with Arthur Burks and Herman Goldstine, von Neumann commented on the relationship between Turing's abstract universal machine and the practical problem of designing a useful all-purpose computer:

> It is easy to see by formal-logical methods that there exist codes that are in abstracto adequate to control and cause the execution of any sequence of operations which are individually available in the machine and which are, in their entirety, conceivable by the problem planner. The really decisive considerations from the present point of view, in selecting a code, are of a more practical nature: simplicity of the equipment demanded by the code, and the clarity of its application to the actually important problems together with the speed of its handling those problems.[8]

[7] Of course, modern computers are not literally implementations of Turing's universal computer. The stripped-down design of a Turing machine is excellent for theoretical purposes, but would never do for practical computing. But of course it is not difficult to write programs to run on modern computers that simulate Turing machines. For example, see http://alexvn.freeservers.com/s1/turing.html

[8] See [9] for references and further discussion.

Nevertheless, there is no doubt that, from the beginning the logicians developing the theoretical foundations of computing were thinking also in terms of physical mechanism. Thus, as early as 1937, Alonzo Church reviewing Turing's classic paper wrote:

> [Turing] proposes as a criterion that an infinite sequence of digits 0 and 1 be 'computable' that it shall be possible to devise a computing machine, occupying a finite space and with working parts of finite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience, certain further restrictions are imposed on the character of the machine, but these are of such a nature as obviously to cause no loss of generality ... [2]

Turing himself speaking to the London Mathematical Society in 1947 said:

> Some years ago I was researching what now may be described as an investigation of the theoretical possibilities and limitations of digital computing machines. I considered a type of machine which had a central mechanism, and an infinite memory which was contained on an infinite tape. This type of machine seemed to be sufficiently general. One of my conclusions was that the idea of 'rule of thumb' process and 'machine process' were synonymous.

Referring to the machine he had designed for the British National Physics Laboratory, Turing went on to say:

> Machines such as the ACE (Automatic Computing Engine) may be regarded as practical versions of this same type of machine.[22]

Hava Siegelmann Ventures "Beyond the Turing Limit"

It is natural to seek and investigate models of computation suggested by the nervous systems, and especially the brains, of human beings and other animals. The brain presents itself as an extremely complicated network of intricately interconnected cells called neurons. Since the 1940s, investigators have been studying mathematical structures consisting of networks of neuron-like elements. The 1980s saw a resurgence of interest in this area after some years of neglect, with the hope that artificial networks of this kind, so-called neural nets, might lead to better understanding of our own brains.[9]

In 1995 an article by Hava Siegelmann appeared in *Science*, the entirely respectable journal of the American Association for the Advancement of Science, entitled "Computation beyond the Turing Limit" [17]. Presenting her

[9] The monograph [18] has an extensive bibliography of this field. The paper [15] suggests that these so-called "connectionist" models are unlikely to provide much speed-up compared to conventional computers.

own perfectly reasonable version of a neural net, she claimed that her nets could indeed achieve what had been thought to be impossible: among the things that they could compute were some that had been proved to be not Turing computable. A few years later she published a monograph [18] in which she studied her neural nets in some detail. Again the same claim was showcased: the book is subtitled "Beyond the Turing Limit." Shall we conclude that Siegelmann is indeed a pioneer of hypercomputation? Actually, as we shall see, there is much less to her claim than meets the eye.

Fortunately, to understand what is involved, it is not necessary to deal with the full technical definition of Siegelmann's neural nets. But it is necessary to understand what mathematicians mean when they speak of *real numbers*. In fact the crucial thing for our purposes is that each of Siegelmann's nets is associated with a finite number of real numbers called *weights*.[10] Among the data objects in which she frames her discussion are the so-called *languages* on an alphabet of two symbols, which we may conveniently take to be $\{a, b\}$. By such a "language" all that is meant is some set, finite or infinite, of strings of these two letters. For example, the language $\{ab, abb, abbb, abbbb, \ldots\}$ is the collection of all strings consisting of the letter a followed by some number of bs. To say that such a language is *computable* is just to say that there exists a Turing machine which, when a particular string on the alphabet $\{a, b\}$ is placed on its tape initially, will eventually halt and will reveal by what is then written on the tape, whether or not the given string belonged to the language in question. Siegelmann defines precisely what it means to say that such a language is *recognized* by one of her nets.

Siegelmann begins by restricting the numbers that are permitted to serve as weights, first to integers, and then to rational numbers.[11] She proves that when her weights are restricted to be rational numbers the languages recognized are precisely the computable languages. Next she considers what happens when arbitrary real numbers are permitted as weights. Lo and behold! For every language, there is now one of Siegelmann's nets that recognizes it! To understand why this is less remarkable than it may appear to be, it is necessary to understand the computational relationship between real numbers and languages on our two-letter alphabet. *A computationally transparent encoding can be used to represent each of these languages by a corresponding real number.* Here's how. First, by arranging all strings on the alphabet

[10] They are called weights because they participate in a weighted average that determines each successive step in the evolution of a neural net.

[11] A rational number is one like $7/11$ that can be written as a fraction with integers as numerator and denominator. Rational numbers are also characterized by the fact that their decimal expansions either consist of only a finite number of digits, or eventually begin repeating the same pattern over and over again. This is by contrast with irrational numbers like $\sqrt{2}$ or π.

$\{a, b\}$ in alphabetic order, we obtain the following coding that enables us to represent each such language as a set of positive integers:[12]

$$\begin{array}{cccccccccccccc} a & b & aa & ab & ba & bb & aaa & aab & aba & abb & baa & bab & bba & bbb & \dots \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \dots \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & \dots \end{array}$$

Finally any given set S of positive integers can be coded by the following real number written as an infinite decimal

$$0, c_1 c_2 c_3 c_4 \dots$$

where

$$c_n = \begin{cases} 4 \text{ if } n \in S \\ 5 \text{ otherwise.} \end{cases}$$

Let us see how this works out with an example. We begin with the language $\{a, ab, abb, abbb, \dots\}$ consisting of the strings with an initial a followed by a block of bs. The corresponding integers are $\{1, 4, 10, 22, \dots\}$.[13] So the real number that encodes this language is $0{,}4554555554555555555545\dots$[14] Although this number is irrational (because there is no repeating pattern), it is computable. Siegelmann doesn't consider what happens when all the weights in one of her nets are computable, but her proof that nets with rational weights recognize only computable languages readily extends to nets with computable weights. It's worth noting (as Turing already did in his classic [20]) that all the standard real numbers of mathematical analysis, including π, e, zeros of Bessel functions, etc. are computable. And *the only way Siegelmann's nets can hope to recognize a non-computable language is to use non-computable weights.* The neural nets can only go "beyond the Turing limit" if they are provided with weights that are already not computable!

Siegelmann is perfectly aware that languages can be coded by real numbers. In fact, her proof that when arbitrary real weights are permitted every language can be recognized works precisely by coding the desired language into a weight.[15] And she says in so many words, "...systems with infinitely precise constants cannot be built". Since the non-computability that Siegelmann gets from her neural nets is nothing more than the non-computability

[12] Although it is not necessary for understanding the encoding, readers may be interested to know that the string corresponding to a given integer in this listing is a kind of binary representation of the integer in which a represents 1 and b represents 2. So, for example, the string aba is associated with the number $1 \cdot 2^2 + 2 \cdot 2^1 + 1 \cdot 2^0 = 9$.

[13] The nth number in this sequence is $1 + 3n(n-1)/2$.

[14] The comma is used in Europe for the decimal point; in the U.S. and Britain it's a period. Obviously there's nothing special about the digits 4 and 5; their use is just a matter of convenience.

[15] The particular coding scheme she uses is different from the one we used above, but that is of no significance.

she has built into them, it is difficult to see in what sense she can claim to have gone "beyond the Turing limit".

For someone familiar with computability theory (also called recursion theory), it is clear that the language recognized by such a neural net is simply a computable function of the real weights. The well-developed theory of degrees of unsolvability makes it possible to classify such languages in various ways, depending on the degrees of unsolvability of the particular real weights used.[16] Siegelmann seems not to know (or not to care) about such fine distinctions. She observes that with rational weights, it is the computable languages that are recognized and that with arbitrary real weights all languages are recognized, and seems uninterested in the fact that between the rationals and arbitrary reals lie a complex taxonomy of subsets of the real numbers with a corresponding variety of resulting languages recognized when the weights are restricted to one of these subsets.

Siegelmann's only attempt in her monograph to connect neural nets with arbitrary real weights to the actual physical world is the following curious paragraph:

> In nature, the fact that the constants are not known to us, or cannot even be measured, is irrelevant for the true evolution of the system. For example, the planets revolve according to the exact values of G, π, and their masses.[17]

This statement presumably is referring to Newton's law of gravitation in which the force of attraction between two bodies is given by the formula

$$F = G\frac{m_1 m_2}{d^2}$$

where m_1, m_2 are the masses of the two bodies and d is the distance separating them. It is hard to know where to begin in criticizing this view of "nature". Ignoring the fact that Newtonian gravitation has been superceded by Einstein's General Theory of Relativity, we have to wonder what could possibly be meant by the "exact value" of the mass of a planet whose changing boundary is necessarily vague, and why Dr. Siegelmann imagines that (presumably in appropriate units) it is represented by an infinite precision (and uncomputable?) real number. In addition, if one were to propose measuring gravitational forces in the solar system to, say, 50 significant digits, one would have to take account of the masses of "nearby" stars.

[16] For example one, may use as a weight a real number that encodes the halting problem for Turing machines. If this is done, it can be proved that the languages recognized will be precisely those for which a Turing machine can be designed that provided with a string as input, will never halt, but will reach a final stable configuration that will reveal whether or not that string belongs to the given language. (However, an observer would in general have no way to know, at any point, whether that final configuration had actually been attained.)

[17] [18] p. 59.

The hope that physical theory will somehow lead to a non-computable real number appears to underlie much of the hypercomputation movement. More will be said about this later. But it may be worth noting that, if anything, physical science seems to be moving in the opposite direction. Chemists in the nineteenth century devoted much energy to computing the atomic weights of the elements to greater and greater precision. However, it has turned out that the atoms of which one of these elements consist typically come in a variety of "isotopes" each of which has a weight given by an *integer* number of protons and neutrons. What the chemists were measuring was an artifact of the proportion in which the different isotopes of a given element happen to occur in nature.

Turing's O-Machines

Jack Copeland, with his collaborator Diane Proudfoot has been an enthusiastic proponent of "hypercomputation"; indeed he is the person who baptized the movement with this name [3,5,4]. Copeland has based himself largely on a concept from Alan Turing's doctoral dissertation at Princeton University. It was on discovering that Alonzo Church in Princeton had found results similar to his own that the young Turing decided to spend some time in Princeton. Mainly for bureaucratic reasons, it seemed best that he enroll as a graduate student at the university. With Church as his advisor, Turing completed a doctoral dissertation that turned out to be an important and influential piece of work [21].

Gödel had made the somewhat paradoxical discovery that not only would every formal logical system (satisfying a few simple requirements) gives rise to a proposition U about the natural numbers that can not be decided within that system, but also, this very proposition U could be seen to be true from a perspective external to the system. A strengthened system in which U is provable can be obtained by simply adjoining U as a new axiom. But this new formal logical system will have its own undecidable proposition, and the whole process can be carried out over and over again. This leads naturally to the idea of progressions of stronger and stronger formal logical systems in which true propositions undecidable at a given place in the progression become provable in subsequent systems. The study of such progressions was the topic of Turing's dissertation.

Turing's 68 page paper contains a number of interesting digressions, and it is one of these that Copeland has enlisted in his cause. Turing introduced what he called "O-machines"; these were to be like the machines from his classic paper on computability, equipped with a linear tape and moving one square at a time, but with one significant difference. These new machines were to be provided access to the correct answers to problems known to be unsolvable,

in particular to a kind of problem Turing called "number-theoretic".[18] As Turing put it:

> Let us suppose that we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle as it were. We shall not go any further into the nature of this oracle apart from saying that it cannot be a machine.

Turing introduced these O-machines to solve a technical problem, specifically to produce an example of a problem that is *not* number-theoretic. He did this by carrying out for O-machines the same proof that in his classic paper on computability led to unsolvable problems. This was accomplished in one page, and except for a few sentences in a later part of the paper, was the only mention of O-machines in the entire 68 page paper.

It is perfectly plain in the context of Turing's dissertation, that O-machines were introduced simply to solve a specific technical problem about definability of sets of natural numbers. There is not the faintest hint that Turing was making a proposal about a machine to be built. In fact in 1938 when he was writing his dissertation, it required remarkable vision to see in his abstract universal machine, the prospect of actual all-purpose computers that could (subject to limitations of space and time) compute whatever is computable. It makes no sense to imagine that he was thinking about actual machines to compute the uncomputable. Turing advisedly used the term "oracle", a word redolent of the supernatural, as though to underline the purely abstract nature of his conception. Yet Copeland and Proudfoot, referring to O-machines insist that "Even among experts, Turing's pioneering theoretical concept of a hypermachine has largely been forgotten." This cooption of Turing to the fold of hypercomputation on the basis of these O-machines is without the slightest justification. The plain fact is that if one truly had an "oracle" that provided answers for an unsolvable problem, that, for example, specified for any given Turing machine whether or not it would eventually halt, then of course we could solve unsolvable problems. As with Siegelmann's nets, of course, if you imagine yourself provided with a solver of an unsolvable problem, you could solve unsolvable problems, that very one, for starters. One didn't need Turing to tell us that.

But Turing did show us how to make precise the notion of one problem being computable *relative* to a second: imagine solutions to that second problem provided by an "oracle" and study just what problems now become

[18] What Turing called "number-theoretic" were problems of the form "Does n belong to S?" where n is a natural number and S is a set that can be defined as consisting of those numbers n for which an equation $f(n, x) = 0$, with f a computable function, has infinitely many solutions in natural numbers x. Equivalently, these are the sets that can be defined as consisting of those n such that $(\forall x)(\exists y)[g(n, x, y) = 0]$ where g is computable. Today, such sets are called Π^0_2 sets, and they are seen as part of a hierarchy determined by the number and arrangement of the "quantifiers" $\forall$, $\exists$.

"solvable" with its aid. This idea gave rise to the fruitful study of what are called degrees of unsolvability or (in homage to Turing's dissertation) Turing degrees. These degrees have an intricate structure that researchers have spent decades uncovering. The notion of oracle has also played an important role in the theory of computational complexity, where it is not a question of computing the uncomputable, but rather of finding a scale for measuring the relative complexity of algorithms needed to solve different problems. So far from the truth is it that this work of Turing's "has largely been forgotten".

Copeland and Proudfoot have also been tempted by Siegelmann's infinite precision real numbers. Noting that the halting problem can be encoded by a real number (along the lines suggested above), they propose, as a possible "oracle", a physical device that makes the successive digits of the decimal representation of this number available. Without a clue as to what sort of physical device could actually serve in this connection, they visualize a capacitor that would hold this number in the form of electric charge. Although they are evidently not seriously proposing any such thing, the example serves to underline one of the pitfalls in attempting to make an infinite precision real number physically available: according to well established physical theory, electric charge in a capacitor consists of a difference in the number of free electrons present in its two plates. Since all electrons have the same charge, the Copeland-Proudfoot infinite precision real number is actually an integer![19] Of course, there's not much point in harping on what was just meant as an illustrative example, but it does serve to remind us that twentieth century physics has tended to see physical quantities as made up of discrete units.

Of course physical theory is in constant flux. Can we really utterly exclude the possibility of some new development leading to a physical realization of an uncomputable quantity? Of course not. In 1958 I wrote:

> For how can we can we ever exclude the possibility of our being presented, some day (perhaps by some extraterrestrial visitor) with a ... device or "oracle" that "computes" a noncomputable function?[20]

For that matter can we really and definitively rule out perpetual motion machines? Isn't it possible that some future development in physical theory will give us access to unlimited energy from some other universe? Well, one would have to say, "Possible but most unlikely." However that may be, such a development will not be the result of someone tinkering in a garage. It could only happen as the result of a revolutionary transformation of physical theory. One would surely look askance at philosophers proclaiming that "the

[19] I'm indebted to Andrew Hodges for reminding me of this fact.

[20] See [6], p. 11. In view of the Copeland-Proudfoot suggestion that Turing's O-machines had been forgotten, it may not be amiss to mention that this book (which has been called a "classic") remains in print, and that Turing machines with oracle are treated in its first chapter.

search is on" for a perpetual motion machine. Yet Copeland-Proudfoot use those very words referring to the quest for an oracle.

Until now, all physical theory has been content with predictions that can be verified to within less than, say, 50 significant digits. A physical theory leading to uncomputable quantities, is certainly not out of the question. But such a revolutionary development will not come from exhortations assuring us that "the search is on"; if at all, it could only arise from the work of theoretical physicists seeking, not an "oracle", but deeper understanding of the universe. Moreover, even if such uncomputable physics were to be developed, making use of it for computational purposes would hardly be automatic. As Dana Scott has remarked:

> 70 years of research on Turing degrees has shown the structure to be extremely complicated. In other words, the hierarchy of oracles is worse than any political system. No one oracle is all powerful.
>
> Suppose some quantum genius gave you an oracle as a black box. No finite amount of observation would tell you what it does and why it is non-recursive. Hence, there would be no way to write an algorithm to solve an understandable problem you couldn't solve before! Interpretation of oracular statements is a very fine art - as they found out at Delphi![21]

Consider what would be involved in harnessing a putative noncomputable physics. What is usually taken to be the ultimate test of a physical theory, agreement with measurements to the extent that instruments permit, would be of no use, because no finite amount of information can verify the value of an infinite precision real number. All experience suggests that every physical theory is accepted only provisionally with every expectation of its eventual replacement. But for a useable oracle to be obtainable, one would require absolute certainty that a real number provided by a particular theory will not have its value changed if and when the theory is upgraded. Finally, even if one knew that some such number is not computable, in order to use it as an oracle, one would also have to know its degree of unsolvability. If indeed "the search is on" for such a number, one can only pity those engaged in this misguided enterprise.

Computing with Randomness and Quantum Computation

There has been considerable success in using randomness to find more efficient algorithms for solving various problems. In recent years, the use of quantum mechanical principles in computation has been shown to lead to efficiency. One might be led to wonder whether one or both of these might not lead to "hypercomputation" after all.

[21] Personal correspondence.

The computing power of Turing machines provided with a random number generator was studied in the classic paper [12]. It turned out that such machines could compute only functions that are already computable by ordinary Turing machines.

The case of quantum computers is similar. Quantum algorithms can provide an exponential speed-up. However, they can only compute computable functions.[22]

Mechanism

The role of mechanism in human cognition was much discussed in the 17th century, in particular by Descartes, Hobbes , and La Mettrie. The question has been the subject of renewed interest in the context of the possibility of machine intelligence. Of course one is very far from understanding the workings of the human mind, but there is every reason to believe that one of the things our brains do is to execute algorithms. Whether that is all that they do remains unknown although Okham's razor does suggest that as a parsimonious thesis.

In [4], Jack Copeland discusses these matters in the context of the theoretical adequacy of Turing computability. He proposes that one should permit a "wide" mechanism that allows for hypercomputation, and he minimizes the relevance of computability theory. A detailed discussion of these questions is beyond the scope of this article. However, it is strange that despite his extensive references, he fails to mention Judson Webb's outstanding monograph [25].

Algorithms: Universality vs. Complexity

I well remember writing code for vacuum tube (British: "valve") computers in the early 1950s. We early programmers found it delightful to see that we could "code" any algorithm to run on our machines. And indeed it is this application of Turing's discovery of universality that underlies the enormous range of tasks that computers are asked to perform in the contemporary world. It didn't take very long for the realization to sink in that care was needed in the allocation of resources if calculations were to be completed in an acceptable time period using the available data storage. At one point I had undertaken to run a program that had been written by a physics graduate student to compute the moments of a function that occurred in the theory of cosmic ray "showers". The first three moments were obtained in an hour. The fourth required that the machine run all night devoted exclusively to this task. It was clear that the fifth moment was unobtainable.

[22] See for example, [13] p. 210.

Early work on automated theorem proving ran into exponential explosions. There seemed to be no way to find an algorithm avoiding such blowups for the simple problem of testing a logical expression in the connectives $\neg$, $\vee$, $\wedge$ for the existence of a truth-value assignment that would evaluate the given expression as "true". This *satisfiability problem* eventually assumed the role of a paradigmatic "hard" problem – one for which no really feasible algorithm was to be expected. While there is still no proof that this is indeed the case, the important subject of computational complexity has developed around this question. A proof of the "million dollar" proposition P $\neq$ NP would settle the matter.[23]

All of this is to point out that the enthusiasts for "hypercomputation" have quite ignored questions of complexity. Copeland's supposed oracles not only store information regarding unsolvable problems, but apparently spew out the information with no significant delay. Of course, in reality, even if, despite all that has been said above, an actual oracle materializes, it will be quite useless if, for example, the time needed for the answer to a query to the oracle is an exponential function of the size of the query.

We may summarize: the positive evidence provided by enthusiasts for hypercomputation amounts to no more than the trivial remark that given a physical "oracle" that somehow makes uncomputable information available, it will become possible to compute other uncomputable functions as well. The great success of modern computers as all-purpose algorithm-executing engines embodying Turing's universal computer in physical form, makes it extremely plausible that the abstract theory of computability gives the correct answer to the question "What is a computation?" and, by itself, makes the existence of any more general form of computation extremely doubtful. In any case, a useable physical representation of an uncomputable function, would require a revolutionary new physical theory, and one that it would be impossible to verify because of the inherent limitations of physical measurement. Finally, the real problems in learning to carry out computations currently regarded as unfeasible lie in a quite different direction – overcoming the exponential explosions in the straightforward algorithms for such problems. It is in this direction that quantum computation may make a real contribution.

[23] Certain problems that are algorithmically solvable, nevertheless have resisted every attempt to find an algorithm that is feasible in practical terms. The proposition P $\neq$ NP may be thought of as asserting that no such feasible algorithms exist. A prize of one million dollars will be awarded by the Clay Mathematics Institute for a proof of this proposition. In an important paper [1], Baker, Gill, and Solovay discussed the "relativization" of this question to an oracle. They showed that depending on which particular oracle was used the relativized proposition could be made to be true or to be false, so that no method of proof that continued to work when relativized to an oracle could possibly resolve this question. *It is worth noting that the oracles used in this work are* **computable**. *In any case, this example can serve to emphasize how far from the truth it is that Turing's notion of oracle "has been largely forgotten".*

References

1. Baker, Theodore P., John Gill, and Robert Solovay (1975) Relativizatons of the P =? NP Question. SIAM J. Comput. **4**, 431-442.
2. Church, Alonzo (1937) Review of [20]. J. Symbolic Logic. **2**, 42-43.
3. Copeland, B. Jack (1998) Turing's O-Machines, Penrose, Searle, and the Brain. Analysis. **58**, 128-38.
4. Copeland, B. Jack (2000) Narrow versus Wide Mechanism: Including a Reexamination of Turing's Views on the Mind-Machine Issue. J. of Phil. **96**, 5-32.
5. Copeland, B. Jack and Diane Proudfoot (1999) Alan Turing's Forgotten Ideas in Computer Science. Scientific American. **253:4**, 98-103.
6. Davis, Martin (1958) Computability and Unsolvability. McGraw-Hill; reprinted with an additional appendix, Dover 1983.
7. Davis, Martin (1982) Why Gödel Didn't Have Church's Thesis. Information and Control. **54**, 3-24.
8. Davis, Martin, ed. (1965) The Undecidable. Raven Press.
9. Davis, Martin, (1987) Mathematical Logic and the Origin of Modern Computers. Studies in the History of Mathematics, pp. 137-165. Mathematical Association of America. Reprinted in The Universal Turing Machine – A Half-Century Survey, Rolf Herken, editor, pp. 149-174. Verlag Kemmerer & Unverzagt, Hamburg, Berlin 1988; Oxford University Press, 1988.
10. Davis, Martin (2000) The Universal Computer: The Road from Leibniz to Turing. W.W. Norton.
11. Davis, Martin (2001) Engines of Logic: Mathematicians and the Origin of the Computer. W.W. Norton (paperback edition of [10]).
12. De Leeuw, K., E.F. Moore, C.E. Shannon, and N. Shapiro (1956) Computability by Probabilistic Machines. Automata Studies, Shannon, C. and J. McCarthy, eds., Princeton University Press, 183-212.
13. Deutsch, David (1997) The Fabric of Reality. Allen Lane, The Penguin Press, New York.
14. Gandy, Robin (1980) Church's Thesis and Principles for Mechanisms. In: The Kleene Symposium. Jon Barwise, ed. North-Holland, Amsterdam.
15. Hong, J.W. (1988) On Connectionist Models. Comm. Pure and Applied Math. **41**, 1039-1050.
16. Park, Robert (2001) Voodoo Science. Oxford.
17. Siegelmann, Hava T. (1995) Computation Beyond the Turing Limit. Science **268**, 545-548.
18. Siegelmann, Hava T. (1999) Neural Networks and Analog Computation: Beyond the Turing Limit. Birkhäuser Boston.
19. Smith, David Eugene (1929) A Source Book in Mathematics. McGraw-Hill.
20. Turing, A.M. (1937) On Computable Numbers, with an Application to the Entscheidungsproblem. Proc. London Math. Soc. **42**, 230-265. Correction: Ibid. **43**, 544-546. Reprinted in [8] 155-222, [24] 18-56.
21. Turing, A.M. (1939) Systems of Logic Based on Ordinals. Proc. London Math. Soc. **45**, 161-228. Reprinted in [8] 116-154, [24] 81-148.
22. Turing, A.M. (1947) Lecture to the London Mathematical Society on 20 February 1947. In: A.M. Turing's ACE Report of 1946 and Other Papers. B.E. Carpenter and R.N. Doran, eds. MIT Press 106-124. Reprinted in [23] 87-105.

23. Turing, A.M. (1992) Collected Works: Mechanical Intelligence. D.C. Ince, ed. North-Holland.
24. Turing, A.M. (2001) Collected Works: Mathematical Logic. R.O. Gandy and C.E.M. Yates, eds. North-Holland.
25. Webb, Judson C.(1980) Mechanism, Mentalism, and Metamathematics. D. Reidel Dordrecht.