Hypercomputation

B. JACK COPELAND
Department of Philosophy, Canterbury University, Private Bag 4800, Christchurch, New Zealand; E-mail: j.copeland@phil.canterbury.ac.nz

Abstract. A survey of the field of hypercomputation, including discussion of a variety of objections.

Key words: analog, computability, hypercomputation, mind, neural network, oracle, recursion theory, super-Turing, Turing, Turing machine

Hypercomputation is the computation of functions or numbers that cannot be computed in the sense of Turing (1936), i.e., cannot be computed with paper and pencil in a finite number of steps by a human clerk working effectively.

As has often been remarked, in 1936 a computer was not a machine at all, but a human being, a mathematical assistant (Copeland, 1997a, 2000). The human computer calculated by rote, in accordance with an effective method supplied by an overseer, prior to the calculation. The overseer would also supply all requisite data, in the form of symbols on paper. Many thousands of human computers were employed in business, government, and research establishments. It was in that sense that Turing used the term 'computer' and its cognates in (1936), saying, for example, 'Computing is normally done by writing certain symbols on paper' (1936, p. 249) and 'The behaviour of the computer at any moment is determined by the symbols which he is observing, and his "state of mind"' (1936, p. 250). The Turing machine (or as Turing called it, 'computing machine') is an idealization of the human computer:

> We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions ... The machine is supplied with a 'tape' ... (1936, p. 231)

A number (respectively, function) is Turing-machine-computable if and only if there is a Turing machine that can write down each successive digit (value) of the number (function), starting from a blank tape and producing each digit (value) in a finite number of steps.

The Church–Turing thesis connects this technical sense of computability with the sense just explained: all numbers and functions computable in that sense — 'by human clerical labour, working to fixed rules, and without understanding' (Turing, 1945, pp. 38–39) — are computable by the universal Turing machine. This thesis and its converse together state that the predicates 'is Turing-machine-computable' and 'is computable by a human rote-worker unassisted by machinery' are equivalent in extension (assuming that the rote-worker is free of limitations on time, paper, pencils, etc).

Minds and Machines **12:** 461–502, 2002.

A hypercomputer is any information-processing machine, notional or real, that is able to achieve more than the traditional human clerk working by rote. Hypercomputers compute functions or numbers, or more generally solve problems or carry out tasks, that lie beyond the reach of the universal Turing machine of 1936. (Some working in the field speak of 'breaking the Turing barrier', 'computing beyond the Turing limit', 'escaping the Turing tar pit', and the like.) The additional computational power of a hypercomputer may arise because the machine possesses, among its repertoire of fundamental operations, one or more operations that no human being unaided by machinery can perform. Or the additional power may arise because certain of the restrictions customarily imposed on the human computer are absent in the case of the hypercomputer — for example, the restrictions that data take the form of symbols on paper, that all data be supplied in advance of the computation, and that the rules followed by the computer remain fixed for the duration of the computation. In one interesting family of hypercomputers, what is relaxed is the restriction that the human computer produce the result, or each digit of the result, in some finite number of steps.

The literature now contains a diverse array of notional hypercomputers. Some of these are noted in Section 1 of this introduction, which briefly surveys key papers and developments in the history of the field. In Section 2, some of the simplest models of hypercomputation are described. Section 3 answers some frequently asked questions about hypercomputation.

The simple notional machines described in Section 2 function as aids to the imagination and intuition. They serve also to make the point that computability is a relative notion, not an absolute one. All computation takes place relative to some set or other of primitive capacities. The richer the capacities that are available, the greater the extent of the computable. The primitive capacities specified by Turing in 1936 occupy no privileged position. One may ask whether a function is computable relative to a subset of these capacities, or to a superset.

One set of functions (or numbers) is of special interest: the functions (or numbers) that are in principle computable in the real world. The exact membership of this set is an open question. Speculation that there may be physical processes — and so, potentially, machine-operations — whose behaviour cannot be simulated by the universal Turing machine of 1936 stretches back over a number of decades. Could a machine, or natural system, deserving the name 'hypercomputer' really exist? Indeed, is the mind — or the brain — some form of hypercomputer? These are two of the central questions addressed in this collection.

1. History of the Field

When Proudfoot and I introduced the term 'hypercomputation' in 1999, we were naming an emerging field which nevertheless has a substantial history (Copeland and Proudfoot, 1999a, p. 77). The brief historical overview given in this section is highly selective.

1.1. TURING (1938)

Turing's abstract 'oracle machines' or *o*-machines seem to be the earliest form of hypercomputer to appear in the literature. Turing introduced the concept of an *o*-machine in his PhD thesis (Princeton, 1938). Subsequently published as Turing (1939), this is a classic of recursive function theory.

The fundamental processes of the standard Turing machine are: (1) move left one square; (2) move right one square; (3) identify the symbol in the scanner; (4) write a symbol on the square of tape in the scanner (first deleting the symbol already inscribed there, if the square is not blank); (5) change internal state. These fundamental processes are made available by unspecified subdevices of the machine — 'black boxes'. (The question of what mechanisms might appropriately occupy these black boxes is not relevant to the machine's logical specification.)

An *o*-machine is a Turing machine augmented with a fundamental process that produces the values of some non Turing-machine-computable function (for example the Turing-machine halting function, explained in Section 1.6 below). The new fundamental process is carried out by a black box called an 'oracle'. When a function g is computable by an *o*-machine whose oracle produces the values of function f, then g is sometimes said to be computable *relative* to f.

How is the *o*-machine organised? As in the case of the ordinary Turing machine, the behaviour of the *o*-machine is governed by a table of instructions (program). The table provides an exhaustive specification of which fundamental processes the machine is to perform when in such-and-such internal state with such-and-such symbol in the scanner. Let p be the new fundamental process. The machine inscribes the sequence of symbols that is to be input into p on any convenient block of squares of its tape, using some symbol to mark the beginning and end of the sequence. p is called into action by means of a special internal state χ. When an instruction in the program puts the machine into state χ, the marked sequence is delivered to the subdevice that effects p, which then returns the corresponding value of the function, 0 or 1 (Turing considered two-valued functions). On Turing's way of handling matters, the value is not written on the tape; a pair of states is employed in order to record values of the function. Thus a call to p ends with a subdevice placing the machine in one or other of these two states, according to whether the value of the function is 1 or 0.

Turing introduced oracle machines with the following words (1939, p. 173):

> 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. With the help of the oracle we could form a new kind of machine (call them *o*-machines), having as one of its fundamental processes that of solving a given number-theoretic problem.

Turing's statement that an oracle 'cannot be a machine' might be taken to indicate that, in Turing's view, oracle machines are not really machines at all. On the other

hand, he did say that oracle machines are 'a new kind of machine' and he did repeatedly refer to *o*-machines as *machines*. For example (ibid.):

> Given any one of these machines . . .
>
> If the machine is in the internal configuration . . .
>
> These machines may be described by tables of the same kind as those used for the description of *a*-machines . . .

(*a*-machines are what we now call Turing machines (1936, pp. 231–232).) By the statement that an oracle 'cannot be a machine', Turing perhaps meant that an oracle cannot be a machine of the kind so far considered in his discussion, viz. a machine that calculates effectively — a Turing machine. His statement is then nothing more than a reiteration of what he himself had shown in 1936. Or perhaps his view was simply that *o*-machines are machines one of whose fundamental processes is implemented by a component that is not in turn a machine. Not every component of a machine need be a machine. (One could even take a more extreme view and say that the atomic components of machines — e.g., squares of paper — are never machines.)

Modern writers who believe that there can be non-mechanical physical action might well choose to speak of machines with components that, while physical, are not themselves machines.

Turing's description of *o*-machines is entirely abstract; he introduced them in order to exhibit an example of a certain type of mathematical problem (the section in which *o*-machines are introduced is entitled 'A type of problem which is not number-theoretic'). In 1936 he had exhibited a problem which cannot be solved by effective means, the problem of determining, given any Turing machine, whether or not it prints infinitely many binary digits. All problems equivalent to this one he termed 'number-theoretic' (noting that he was using the term 'number-theoretic' in a 'rather restricted sense') (1939, pp. 168–170). The *o*-machine concept enabled him to describe a new type of problem, not solvable by a uniform process even with the help of a number-theoretic oracle. The class of machines whose oracles solve number-theoretic problems is, he showed, subject to the same diagonal argument that he used in 1936. The problem of determining, given any such machine, whether or not it prints infinitely many binary digits is not one that can be solved by any machine in the class and, therefore, is not number-theoretic (1939, p. 173).

Turing appealed to *o*-machines when discussing the question of the completeness of the 'ordinal logics' described in his thesis (1939, p. 200). Via a suitable class of oracle machines and the diagonal argument, a logic itself forms the basis for the construction of a problem with which it cannot deal.

Turing, as he said, did 'not go any further into the nature' of an oracle. One way to conceptualise an oracle — perhaps the simplest — is as a device accessing an infinite internal tape upon which there have been inscribed, in order, all the infinitely many arguments and values of whatever function it is that the oracle generates. This device can produce any of the function's values after only a finite search along the tape. Various other conceptualisations of an oracle are introduced in what follows.

There is considerable diversity among them, some being more physical in flavour than others.[1] To mention only a few examples, an oracle is in principle realisable by certain classical electrodynamical systems (Section 1.2); the physical process of equilibriation (Section 1.10); an automaton travelling through relativistic spacetime (Sections 1.15, 1.23); a quantum mechanical computer (Sections 1.3, 1.19, 3.10); an inter-neural connection (Section 1.25); and a temporally evolving sequence of Turing machines, representing for example a learning mind (Sections 1.18.1, 2.7).

The concept of oracular computation can profitably be employed in theorizing about hardware, about brains, and about minds.

1.2. SCARPELLINI (1960–1963)

As early as 1960, in an article published in German in 1963, Bruno Scarpellini speculated that non-recursive (i.e., non-Turing-machine-computable) processes might occur in nature (Scarpellini, 1963). Scarpellini wrote:

> [O]ne may ask whether it is possible to construct an analogue-computer which is in a position to generate functions $f(x)$ for which the predicate $\int f(x) \cos nx\mathrm{d}x > 0$ is not decidable [by Turing machine] while the machine itself decides by direct measurement whether $\int f(x) \cos nx\mathrm{d}x$ is greater than zero or not.

Scarpellini made it clear that he intended talk of 'constructing' the machine that he characterised to be understood at the level of thought-experiment. He said:

> Such a machine is naturally only of theoretical interest, since faultless measuring is assumed, which requires the (absolute) validity of classical electrodynamics and probably such technical possibilities as the existence of infinitely thin perfectly conducting wires. All the same, the (theoretical) construction of such a machine would illustrate the possibility of non-recursive natural processes.

Scarpellini's article is published here in English translation for the first time, together with some comments by Scarpellini written specially for this collection. In these comments Scarpellini examines Penrose's thesis that classical physics is computable (Penrose, 1989, 1994). Scarpellini argues that, given what appears to be Penrose's intended mathematical interpretation of the thesis, then, even if true, the thesis 'leaves plenty of room for a variety of nonrecursive phenomena which may occur in classical physics'.

Scarpellini ends his comments with some speculations about hypercomputation in the brain. He writes:

> [I]t does not seem unreasonable to suggest that the brain may rely on analogue-processes for certain types of computation and decision-making. Possible candidates which may give rise to such processes are the axons of nerve cells. . . . [I]t is conceivable that the mathematics of a collection of axons may lead to undecidable propositions like those discussed in my paper.

1.3. KOMAR (1964)

In 1964 Komar showed that the behaviour of a quantum system having an infinite number of degrees of freedom may be hypercomputational, proving that, in general, the universal Turing machine cannot determine whether or not two arbitrarily chosen states of such a system are macroscopically distinguishable. He wrote (1964, pp. 543–544):

> Given two arbitrary quantum states of a physical system having an infinite number of degrees of freedom, we know either that they are macroscopically distinguishable, or that they are not ... What we have shown is that there is in general no effective procedure for deciding whether they [are] or not. That is, there exists no [effective] procedure for determining whether two arbitrarily given physical states can be superposed to show interference effects characteristic of quantum systems. ... [I]t is rather curious or surprising that the issue of the macroscopic distinguishability of quantum states should be among the undecidable questions.

1.4. KREISEL (1965–1967)

Kreisel emphasised that it is an open question whether there are natural processes that are not Turing-machine-computable. Scattered remarks throughout a series of papers spanning three decades discuss this theme in relation to classical mechanics, classical electrodynamics, and quantum mechanics (see, for example, Kreisel, 1965, 1967, 1971, 1972, 1982, 1987, and especially 1970, 1974). Among his early statements concerning quantum mechanics are:

> [E]xcepting collisions as in the 3-body problem, which introduce discontinuities ... the theory of partial differential equations shows that the behavior of discretely described (finite) systems of classical *mechanics* is recursive. But this may not be so in the quantum theory ... (1965, p. 144)
>
> [T]he hypothesis that reasoning is not mechanistic is by no means anti-materialist or anti-physicalist. *There is no evidence that* even present day *quantum theory is a mechanistic, i.e. recursive theory* in the sense that a recursively described system has recursive behaviour. (1967, p. 270)

1.5. PUTNAM AND GOLD (1965)

In papers published in the same year, Gold and Putnam independently described the variant of the Turing machine known variously in the literature as a trial-and-error machine, a guessing machine, or simply a Putnam–Gold machine (Gold 1965; Putnam 1965).[2] Here is Putnam's description of the new idea (1965, p. 49):

> [W]e modify the notion of a decision procedure by (1) allowing the procedure to 'change its mind' any finite number of times (in terms of Turing Machines:

> we visualize the machine as being given an integer (or an *n*-tuple of integers) as input. The machine then 'prints out' a finite sequence of 'yesses' and 'nos'. The *last* 'yes' or 'no' is always to be the correct answer.); and (2) we give up the requirement that it be possible to tell (effectively) if the computation has terminated? I.e., if the machine has most recently printed 'yes', then we know that the integer put in as input must be in the set *unless the machine is going to change its mind*; but we have no procedure for telling whether the machine will change its mind or not. ... [I]f we always 'posit' that the most recently generated answer is correct, we will make a finite number of mistakes, but we will eventually get the correct answer. (Note, however, that even if we have gotten to the correct answer (the end of the finite sequence) we are never *sure* that we have the correct answer.)

A Putnam–Gold machine computes a function if and only if, given any argument of the function, the machine will sooner or later produce the correct value. The computable functions include functions that are not Turing-machine-computable. An example is the Turing-machine halting function, defined in the next section.

Putnam–Gold machines are discussed in Sections 1.13 and 2.5 below.

1.6. ABRAMSON (1971)

In 1971, in a paper that has been unjustly neglected, Abramson introduced what he called the Extended Turing Machine or ETM. An ETM is able to store a real number on a single square of its tape (by unspecified means). Abramson wrote (1971, p. 33):

> We extend the notion of effective computability to functions on the real numbers by adding [to the Turing machine] abilities to store real numbers, to move them about, to compare two given numbers, and to perform the elementary arithmetic operations of addition and multiplication.

Since, as Turing proved in 1936, not all real numbers are Turing-machine-computable, it follows immediately that an ETM is able to carry out computations that no Turing machine can carry out. To give one simple example, an ETM can add any pair of real numbers that are given as input. A Turing machine, on the other hand, may not even be able to represent the numbers in question, let alone add them.

The famous Turing-machine halting function (Davis 1958, p. 70) may be defined as follows. $H(x,y)=1$ if and only if x represents the program of a Turing machine that eventually halts if set in motion with the number y inscribed on its otherwise blank tape; $H(x,y)=0$ otherwise. A machine able to 'solve the halting problem' can inform us, concerning any given Turing machine program x and any finite amount of data y, whether or not $H(x,y)=1$.

The halting function for ETMs is defined analogously. Having shown that the halting problem for ETMs is not solvable by an ETM, Abramson augmented the

ETM with a greatest-lower-bound (GLB) operation, producing what he called a level one GLB Automaton (1971, pp. 33–34). He proved that a machine of this type is able to solve the halting problem for ETMs.

Abramson introduced an infinite hierarchy of GLB Automata by allowing the GLB operation of machines at the nth level to operate over sets defined by machines at m–1th level. He proved that the halting problem for the GLB Automata of any given level is always solvable by a machine higher in the hierarchy (1971, pp. 35–36).

1.7. BOOLOS AND JEFFREY (1974)

In a discussion of Zeno's paradox of the race-course, Russell said: 'If half the course takes half a minute, and the next quarter takes a quarter of a minute, and so on, the whole course will take a minute' (Russell, 1915, pp. 172–173). Later, in a discussion of a paper by Alice Ambrose (Ambrose, 1935), he wrote:

> Miss Ambrose says it is *logically* impossible to run through the whole expansion of π. I should have said it was *medically* impossible. ... The opinion that the phrase 'after an infinite number of operations' is self-contradictory, seems scarcely correct. Might not a man's skill increase so fast that he performed each operation in half the time required for its predecessor? In that case, the whole infinite series would take only twice as long as the first operation. (1936, pp. 143–144.)

This same temporal patterning and the attendant idea of completing an infinite number of operations in a finite time was described independently by a number of writers, including Blake (1926) and Weyl (1927).

Boolos and Jeffrey imagined Zeus working in this manner in order to 'simulate every step of an unending computation in 1 second' (Boolos and Jeffrey, 1974, p. 40). They pointed out that Zeus is therefore able to determine, of any arbitrary Turing machine, whether or not it halts (ibid.). In the computability literature, machines operating in accordance with Russell's formula have become known as *Zeus machines*.

Later in this issue, Hamkins supplies a general theory of what computations can be done by a Turing machine that is able to survey its own infinitely long computations.

1.8. POUR-EL AND RICHARDS (1979–1981)

In 1979 Pour-El and Richards published their paper 'A computable ordinary differential equation which possesses no computable solution', followed in 1981 by their now famous 'The wave equation with computable initial data such that its unique solution is not computable' (Pour-El and Richards, 1979, 1981; see also their book 1989).

In the first paper, Pour-El and Richards proved that there exists a 'computable ... function $F(x,y)$ defined on a rectangle R of the plane such that the differential equation $y'(x) = F(x,y)$ has no computable solution on any neighbourhood within R' (1979, p. 61). They explained (1979, p. 63):

> Our results are related to some remarks of Kreisel. In [1974], Kreisel concerns himself with the following question. Can one *predict theoretically* on the basis of some current physical theory — e.g. classical mechanics or quantum mechanics — the existence of a physical constant which is not a recursive real? Since physical theories are often expressed in terms of differential equations, it is natural to ask the following question: Are the solutions of $\varphi' = F(x,\varphi)$, $\varphi(0) = 0$, computable when F is?

Introducing the second paper, Pour-El and Richards referred to the same problem posed by Kreisel, remarking that their results are similar to those presented in the first paper but are obtained with respect to 'an equation which is more familiar', the three-dimensional wave equation (1981, p. 216). They proved that the behaviour of a system with Turing-machine-computable initial conditions and evolving in accordance with their equation is not Turing-machine-computable. The function describing the evolution of the system has real-number values that are not Turing-machine-computable.

In a review of the two papers, Kreisel described the results in almost racy terms (1982, p. 901):

> The authors suggest, albeit briefly and in somewhat different terms, that they have described an analogue computer that — even theoretically — cannot be simulated by a Turing machine. Here 'analogue computer' refers to any physical system, possibly with a discrete output, such as bits of computer hardware realizing whole 'subroutines'. (Turing's idealized digital computer becomes an analogue computer once the physical systems are specified that realize — tacitly, according to physical theory — his basic operations.)

1.9. KARP AND LIPTON (1980)

In a conference presentation in 1980, Karp and Lipton discussed infinite families of digital circuits, for example circuits consisting of boolean logic gates or McCulloch–Pitts neurons (Karp and Lipton, 1982). Each individual circuit in an infinite family is finite. A family may be regarded as a representation of a piece of hardware that grows over time, each stage of growth consisting of the addition of a finite number of new nodes (e.g., neurons). The behaviour of any given circuit in a family can be calculated by some Turing machine or other (since each circuit is finite), but there may be no single Turing machine able to do this for all circuits in the family. In the case of some families, that is to say, the function computed by the growing hardware — successive members of the family computing values of the function for increasingly large inputs — is not Turing-machine-computable.

1.10. DOYLE (1982)

Written in 1982, Doyle's classic 'What is Church's Thesis? An Outline' has been in private circulation. It is published here for the first time. Doyle suggested that the physical process of equilibriating — for example, a quantum system's settling into one of a discrete spectrum of states of equilibrium — is 'so easily, reproducibly, and mindlessly accomplished' that it be granted equal status alongside the operations usually termed effective. Doyle wrote:

> My suspicion is that physics is easily rich enough so that . . . the functions computable *in principle* given Turing's operations and equilibriating, include non-recursive functions. For example, I think that chemistry may be rich enough that given a diophantine equation . . . we plug values into [a] molecule as boundary conditions, and solve the equation iff the molecule finds an equilibrium.

1.11. RUBEL (1985–1989)

Rubel emphasised that aspects of brain function are analog in nature and suggested that the brain be modelled in terms of continuous mathematics, as against the discrete mathematics of what he called the 'binary model'. He wrote (1985, pp. 78–79):

> It is fashionable nowadays to downgrade analog computers, largely because of their unreliability and lack of high accuracy . . . But analog computers, besides their versatility, are extremely fast at what they do . . . In principle, they act instantaneously and in real time. . . . Analog computers are still unrivalled when a large number of closely related differential equations must be solved.

Rubel noted that the digital simulation of analog computers can offer 'some of the best of both worlds' (1985, p. 79). In digital simulation, the various 'black boxes' making up the analog computer — integrators, adders, multipliers, etc. — are 'replaced by digital counterparts, but from this point, the logic of the computer is analog' (ibid.). Rubel noted, however, that digital simulation may not always be possible, even in principle (Rubel, 1989, p. 1011):

> One can easily envisage other kinds of black boxes of an input–output character that would lead to different kinds of analog computers. . . . Whether digital simulation is possible for these 'extended' analog computers poses a rich and challenging set of research questions.

1.12. GEROCH AND HARTLE (1986)

Geroch and Hartle examined the nature of prediction in theories describing physical systems not simulable by Turing machine. They argued that 'such a theory should be no more unsettling to physics than has the existence of well-posed prob-

lems unsolvable by any algorithm been to mathematics', suggesting that such theories 'may be forced upon us' in the quantum domain (1986, pp. 534, 549).

> [O]ur experience with present-day physical theories [is that in] both classical and quantum physics ... the implementation of the theory consists of selecting some algorithm ... for solving the equations. But not all theories of physics need be of this type. We may someday in physics be confronted with a situation in which ... the activity of applying the theory is not so different from that of finding the theory in the first place. Specifically, we may reach a point at which there exist no algorithms whatever for applying a theory mechanically to specific circumstances. (1986, p. 534)

Geroch and Hartle drew an analogy between the exercise of deriving predictions from such a physical theory and the process of calculating approximations to some real number that is not Turing-machine-computable. One algorithm may deliver the first n digits of the decimal representation of the real number, another the next m digits, and so on. Discovering each algorithm is akin to finding the theory. They said (1986, p. 549):

> To predict to within, say, 10%, one manipulates the mathematics of the theory for a while, arriving eventually at the predicted number. To predict to within 1%, it will be necessary to work much harder, and perhaps to invent a novel way to carry out certain mathematical steps. To predict to 0.1% would require still more new ideas. ... The theory certainly 'makes definite predictions', in the sense that predictions are always in principle available. It is just that ever increasing degrees of sophistication would be necessary to extract those predictions. The prediction process would never become routine. Clearly, this would not be a particularly desirable state of affairs. But it would seem to be merely an inconvenience — far from a disaster for physics.

1.13. KUGEL (1986)

Kugel suggested that, as a matter of empirical fact, the mind is hypercomputational, citing evidence from a number of areas of psychology (Kugel, 1986). He emphasised that this claim by no means entails that the 'uncomputable parts of thinking' cannot be studied scientifically (1986, pp. 137–138). His central idea is that those parts of the mind that are not computational — in the 1936 sense — may be modelled as consisting of families of non-halting processors, e.g., 'non-halting recognizers' (1986, p. 149). Once a processor succeeds at its task, it turns itself off. Otherwise the processor runs on forever. Kugel bases his non-halting processors on Putnam–Gold machines (see above).

Kugel acknowledged that any 'finite behavior we observe can be modelled by a finite automaton' (1986, p. 148). He commented (ibid.):

> [O]ur choice of one kind of model, rather than another, depends not on what it can account for, but on what it can account for smoothly. The arguments I

> have given suggest (to me at least) that non-computing machines [i.e. hypercomputers — ed.] might give us a smoother account of our observations than computing machines.

Kugel's paper in this collection further explores this model of the mind. He makes the important point that at one level — the hardware level, so to speak — a Putnam–Gold machine is simply a Turing machine, consisting of tape and scanner, nothing more. The difference between the two types of machine lies in the interpretation that is placed on the activities of the hardware. (See also the discussion of accelerating Turing machines in Section 2.5.)

1.14. KRYLOV (1986)

Building on the work of Bogdanov (1980), Krylov developed the mathematical framework that he calls Formal Technology (Krylov 1986; see also Krylov 1996, 1997). A technology is an ordered pair <**E**, **O**>, where **E** is a finite set of basic components and **O** is a finite set of operations of synthesis. The operations in **O** are applicable to members of **E** and to constructions already obtained by the application of these operations to members of **E**. Possible constructions include mathematical formulae, codes, real or abstract machines, buildings, molecules, electromagnetic signals, and social groups. Krylov countenances operations such that the process of construction cannot be simulated by the universal Turing machine.

1.15. PITOWSKY (1987)

In a conference presentation given in 1987, Pitowsky considered the question 'Can a physical machine compute a non-recursive function?' (Pitowsky, 1990, p. 82). Referring to the thesis that no non-recursive function is physically computable as *Wolfram's thesis* (after Wolfram, 1985), Pitowsky said (1990, p. 86):

> The question of whether Wolfram's thesis is valid is a problem in the physical sciences, and the answer is still unknown. Yet there are very strong indications that Wolfram's thesis may be invalid.

Generalising the proposal by Weyl mentioned earlier, Pitowsky independently introduced the idea of a Zeus machine (Section 1.7), under the name 'Platonist computer'. He said 'My concern is with physical possibilities: Are Platonist computers physically possible?' (1990, p. 81). Pitowsky concluded: 'as far as computation time is concerned, the existence of Platonist computers is compatible with general relativity' (1990, p. 84).

In their contribution to this collection Shagrir and Pitowsky pursue this idea, describing a physical hypercomputer that is consistent with General Relativity.

1.16. MacLennan (1987)

In 1987 MacLennan proposed a new model of computation in which the individual processing steps are field-transformations. A large aggregate of data is thought of as forming a continuous scalar or vector field (analogous to an electrical field). The fundamental processes of the computation operate on entire fields to yield entire fields.

MacLennan wrote (1987, p. 39):

> AI is moving into a new phase characterized by biological rather than psychological metaphors. Full exploitation of this new paradigm will require a new class of computers characterized by massive parallelism: parallelism in which the number of computational units is so large it can be treated as a continuous quantity. We suggest that this leads to a new model of computation based on the transformation of continuous scalar and vector fields. We describe a class of computers, called *field computers*, that conform to this model, and claim that they can be implemented in a variety of technologies (e.g., optical, artificial neural network, molecular).

In describing field computers, 'continuous mathematics (such as the infinitesimal calculus) can be applied, which is much more tractable than discrete mathematics (e.g., combinatorics)' (MacLennan, 1987, p. 40). The design of field computers can be carried out independently of the implementation technology (ibid.):

> [W]hether the implementation technology is ... discrete or continuous (or nearly so, as in molecular computing) ... the design of the computer can be described by continuous mathematics. Then, if the intended implementation technology is discrete, we can select out of the continuum of points a sufficiently large finite number.

In his contribution to this collection, 'Transcending Turing Computability', MacLennan argues that Turing-machine computation is a poor model for what he calls 'natural computation', computation occurring in nature or inspired by computation in nature.

1.17. Blum, Shub and Smale (1989)

Blum, Shub and Smale described abstract machines for performing sequential computations with arbitrary real numbers (Blum et al., 1989; see also Blum et al., 1998). They described their work as 'close in spirit' to Abramson's (1989, p. 4). Blum et al. (1989) developed a rich general theory of such computations, including a proof of the existence of a universal machine.

Part of their motivation was to 'bring the theory of computation into the domain of analysis, geometry and topology' (1989, p. 1). Arguably, the discrete idealisations of computation prevalent in computer science have served to isolate the theory of computation from the rest of mathematics. The suggestion is that the concept of

a machine able to take arbitrary real numbers as input is for some purposes a more appropriate idealisation of actual computing machinery than discrete models like the Turing machine.

Smale argued (1988, p. 92–93):

> I am quite critical of [the] idea of the computer as a finite instrument. Turing modelled his theory on logic, which is a very specific and narrow part of mathematics. This has kept it away from the mainstream of mathematics and hindered its development. John von Neumann . . . [wrote] 'We are very far from possessing a theory of automata which deserves that name'. He said the reason for this lack was that logic has very little contact with the continuous concept of a real or complex number, that is, with mathematical analysis. He went on to say: 'A detailed, highly mathematical and more specifically analytical, theory of automata and of information is needed'. Computer scientists still rest their theories on the discrete idea expressed by Turing machines. Why I object to that, and why von Neumann expected more, has to do with scientific computing . . . Numerical analysis, the theoretical side of scientific computation, provides some basis for solving differential equations using algorithms. Some scientific people — engineers in scientific computation or theoreticians of numerical analysis — have very little respect for the Turing machine. The Turing theory is seen as old-fashioned and limited and remote from the kind of explanation they need. There is also a move away from the use of calculus in the discrete mathematics of computer science. What this has caused is an unhappy lack of unity between two disciplines, computer science and numerical analysis, two subjects which should be intimately related.
>
> What I want to suggest is a different model for computer theory based on the idea of using real numbers. It will be an idealization just as the Turing machine model is an idealization, and it provides a different model to explain the same phenomena. Different models explain different aspects of the same physical reality. One should not look for a single model of the computer, but rather look for models which will help us understand scientific computation on the one hand and classical computer science on the other.

1.18. PENROSE (1989)

In a well-known book published in 1989, Penrose suggested that 'non-algorithmic action' may 'have a role within the physical world of very considerable importance' and that 'this role is intimately bound up with . . . "mind"' (1989, p. 557; see also Penrose, 1994). In a précis of the book, Penrose wrote (1990, p. 653):

> I have tried to stress that the mere fact that something may be scientifically describable in a precise way does not imply that it is computable. It is quite on the cards that the physical activity underlying our conscious thinking may be governed by precise but nonalgorithmic physical laws and our conscious think-

ing could indeed be the inward manifestation of some kind of nonalgorithmic physical activity.

Penrose famously appealed to Gödel incompleteness in the course of his attempt to establish that non-algorithmic action must occur at the quantum level (1989, p. 538ff; see also 1994). Penrose's Gödelian argument is highly controversial and hypercomputationalists who believe that the mind is not Turing-machine-computable may have no truck with the argument (for discussion see Copeland, 1998a; Copeland and Proudfoot, 1999b).

1.18.1. *Turing on the Gödelian Argument*

Turing himself discussed the argument from 'the theorem of Gödel and related results' (which of course include his own result of 1936) to the conclusion that 'if one tries to use machines for such purposes as determining the truth or falsity of mathematical theorems ... then any given machine will in some cases be unable to give an answer at all' (Turing, 1948, p. 4; see also 1947, 1950, c. 1951). Turing had an interesting response to the argument. He pointed out that the argument depends 'on the condition that the machine must not make mistakes', which is 'not a requirement for intelligence' (ibid.). Turing envisaged a machine able both to make mistakes and to 'correct itself' through the ability to 'learn by experience' (c. 1951, p. 459).

Fundamentally, this response involves lifting the restriction, mentioned previously, that the program must remain fixed during the course of the calculation. Learning modifies the program (see, e.g., Turing, 1947, pp. 122–123). Each time that learning brings about a modification to the program, the calculation in effect passes from one Turing machine to another (each Turing machine having a fixed program or 'machine table'). One might put it like this: At every moment the learning machine — or mind — is equivalent to a Turing machine, but not necessarily always to the *same* Turing machine.

In general, there is no effective procedure (i.e., Turing machine) for determining which Turing machine will be next in this sequence. For example, there is no such effective procedure in cases where the transition from program to program involves a genuinely random element. In several places Turing emphasised the advantages of involving a random element in the learning procedure, which he said 'would result in the behaviour of the machine not being by any means completely determined by the experiences to which it was subjected' (Turing, c. 1951, p. 461; see also 1948, 1950, 1951).

If, by such means, the learning mind is always able in principle to find new methods of mathematical proof, then it is able in principle to reach a decision about the truth-value of any given mathematical formula (cf. Turing, 1947, p. 123).

In the Karp and Lipton model, a single piece of growing hardware can be represented by means of a family of circuits (Section 1.9). Similarly, a single learning machine can be represented by a family of Turing machines. If the machine contin-

ues learning indefinitely, the representing family of Turing machines is infinite. At each stage in its development, the learning machine is equivalent to some Turing machine in the family. Nevertheless, the universal Turing machine may be unable to simulate the behaviour of the learning machine over time.

A connection between this picture of the learning mind and the concept of an oracle is drawn in Section 2.7.

Piccinini's contribution to this collection, 'Alan Turing and the Mathematical Objection', gives a detailed exegesis of Turing's treatment of the Gödelian argument.

1.19. STANNETT (1990)

In 1990 Stannett described a form of hypercomputer that he called an 'analogue X-machine'. He wrote (1990, p. 331):

> We describe a novel machine model of computation, and prove that this model is capable of performing calculations beyond the capability of the standard Turing machine model. In particular, we demonstrate the ability of our model to solve the Halting problem for Turing machines. We discuss the issues involved in implementing the model as a physical device, and offer some tentative suggestions.

Stannett ended his paper with some speculations concerning quantum X-machines (1990, p. 340):

> [W]e envisage the eventual development of a quantum computer which takes full advantage of available quantum properties. In particular, if one considers electronic energy level transitions within atoms, it is arguably the case that natural systems can and do implement such features as infinite nondeterminism, in which case one would expect quantum computers to exhibit super-Turing potential.

In his contribution to this collection Stannett continues his discussion of quantum hypercomputation and other possible forms of physical hypercomputation. Quantum hypercomputation is also discussed in the contributions by Kieu and Calude. Kieu offers a quantum algorithm for solving Hilbert's tenth problem (like the Turing-machine halting problem, unsolvable by a standard Turing machine).

1.20. STEWART (1991)

Stewart independently described a machine that, by working ever faster, is able to carry out an infinite number of computational steps in a finite time (Stewart, 1991a, b). He wrote (1991b, pp. 8–9):

> [T]he Rapidly Accelerating Computer (RAC) whose clock accelerates exponentially fast, with pulses at (say) times $1–2^{-n}$ as $n \to \infty$ ... can cram an infinite number of computations into a single second. ... It can, therefore,

> solve the halting problem for Turing machines ... by running a computation in accelerating time and throwing a particular switch if and only if the Turing machine halts. ... [The RAC] can prove or disprove the Riemann hypothesis by computing all zeroes of the zeta function. ... [I]t can run a computation that will prove all possible theorems in its Industry Standard Second, by pursuing *all* logically valid chains of deduction from the axioms of set theory.

Stewart pointed out, contra Penrose's thesis to the effect that classical physics is computable (mentioned above), that the RAC 'appears to be entirely possible within classical mechanics' (1991b, p. 9).

A form of accelerating machine that is specifically a Turing machine is considered in Section 2.5.

1.21. KAMPIS (1991–1995)

Kampis considered the idea of self-modifying programs, particularly in a biomolecular context (Kampis, 1991, 1995). He argued that, even in the absence of random influences, a self-modifying system need not be equivalent to any single, static Turing machine (see also Section 1.18.1 above). Kampis described an abstract biochemical mechanism to illustrate the point (1995, pp. 104–106).

1.22. PUTNAM (1992)

In the 1960s Putnam defended and popularised the idea that the mind is a Turing machine (see for example Putnam, 1960). By 1992, he had changed his view (1992, p. 4):

> [M]aterialists are committed to the view that a human being is — at least metaphorically — a machine. It is understandable that the notion of a Turing machine might be seen as just a way of making this materialist idea precise. Understandable, but hardly well thought out. The problem is the following: a 'machine' in the sense of a physical system obeying the laws of Newtonian physics need not be a Turing machine.

Referring to Pour-El and Richards (1981) and Kreisel (1982), Putnam continued (1992, p. 5):

> [I]t has been proved that there exist possible physical systems whose time evolution is not describable by a recursive function, even when the initial condition of the system is so describable. (The wave equation of classical physics has been shown to give rise to examples.) In less technical language, what this means is that there exist physically possible analogue devices which can 'compute' non-recursive functions. Even if such devices cannot actually be prepared by a physicist (and Georg Kreisel has pointed out that no theorem has been proved *excluding* the preparation of such a device), it does not follow that they do not occur in nature. Moreover, there is no reason at all why the real

numbers describing the condition at a specified time of a naturally occurring physical system should be 'recursive'. So, for more than one reason, a naturally occurring physical system might well have a trajectory which 'computed' a non-recursive function.

1.23. HOGARTH (1992–1994)

Like Pitowsky and Stewart, Hogarth explored the idea of infinite yet surveyable computation. His proposal for what he called a 'non-Turing computer' was based on Pitowsky's observation (in private communication) that 'there is no reason why [a] computer user must remain beside the computer' (Hogarth, 1992, p. 173). Hogarth considered the physically possible anti-de Sitter spacetime and showed that in a possible world with this spacetime the Turing-machine halting function is computable (Hogarth, 1994; see also Earman and Norton, 1993, 1996).

Hogarth's computing machine consists simply of a universal Turing machine fitted with a signalling device. The user programs the universal machine to simulate the particular machine t for which the value of the halting function is required and then launches the universal machine plus signalling device along a certain spacetime curve. The properties of the spacetime are such that the infinite lifespan of the computing machine can be surveyed by the user in a finite amount of time. (Hogarth dubs spacetimes allowing this 'Pitowsky'.) The signalling device sends a signal to a predetermined point p on the user's world-line if and only if the simulation of t halts. Thus a signal at p tells the user that the value of the halting function is 1 in this case, and no signal at p that the value is 0.

Hogarth showed that this arrangement can be used to settle Goldbach's conjecture and to decide the predicate calculus, and that in a more complicated spacetime, a similar arrangement can be used to decide arithmetic. He concluded that the 'limit of computation' is a 'thoroughly contingent' matter (1994, p. 133 and abstract).

1.24. CLELAND (1993)

Cleland in effect discussed the notion of a rote procedure over arbitrary objects (Cleland, 1993). She laid particular stress on procedures over mundane objects, calling these 'mundane procedures'. Examples are recipes, knitting patterns, street directions, and a set of instructions for lighting a fire. Cleland argued — in the same vein as Doyle — that mundane procedures stand alongside the procedures normally classed as effective. She argued for the possibility that 'by following an effective mundane procedure ... a physical system is able to compute a function which couldn't be computed by a Turing machine' (1993, p. 307).

1.25. SIEGELMANN AND SONTAG (1994)

A neural network is *recurrent* (as opposed to *feedforward*) if there are feedback loops among its hidden units. Siegelmann and Sontag described a type of recurrent neural network consisting of a finite number of neurons with real-valued weights on their interconnections. In the special case where all the interconnection weights are rational, each such network is equivalent to a Turing machine (Siegelmann and Sontag, 1992). However, if the weights are arbitrary real numbers, the networks can compute functions that are not Turing-machine-computable, and moreover can do so in polynomial time. It suffices that there be in the interconnection matrix a single connection whose weight is a non Turing-machine-computable real number (Siegelmann and Sontag, 1994). (The proof given by Siegelmann and Sontag that their networks have hypercomputational power builds on the work of Karp and Lipton mentioned above.)

Siegelmann's contribution to this collection surveys the field of neural hypercomputation.

2. Some Simple Models

2.1. COUPLED TURING MACHINES

Coupled Turing machines are described in Copeland (1997b, pp. 694–695) and Copeland and Sylvan (1999, pp. 51–52); the term 'coupled Turing machine' originates in the latter paper. See also Wegner (1997), where the term 'interaction machine' is used for essentially the same idea.

Turing machines accept no input while operating. A finite amount of data may be inscribed on the tape before the computation starts, but thereafter the machine runs in isolation from its environment. A coupled Turing machine results from coupling a Turing machine to its environment via an input channel (or any finite number of input channels). Each channel supplies a stream of symbols to the tape as the machine operates. (The machine might also possess one or more output channels, which return symbols to the environment.) Depending on the nature of the input, the activity of a coupled Turing machine may not be simulable by the universal Turing machine.

Any coupled Turing machine whose activity ceases after a finite number of steps can be simulated by the universal Turing machine. This is because the stream of symbols supplied to the tape by the coupled machine's input channel (or channels) is, in this case, finite, and so can be inscribed on the universal machine's tape before the simulation commences. Where the coupled Turing machine never halts — as, for example, in the case of an idealised automatic teller machine or air traffic controller — simulation by the universal machine may or may not be possible. If the unending stream of input can itself be generated 'on the fly' by the universal Turing machine — as in the case, for example, of the digits of π (3.14159...) —

then the universal machine is able to simulate the coupled machine. If, however, the unending stream of input consists of the digits of some real number that is not Turing-machine-computable, then the coupled machine is hypercomputational.

There are infinitely many real numbers that are not Turing-machine-computable. This follows straight away from the fact that there are only countably many distinct Turing machine programs (i.e., there are no more programs than there are integers, 1, 2, 3, . . .) and, therefore, only countably many Turing-machine-computable numbers — whereas there are uncountably many real numbers. I call the following example of a non Turing-machine-computable real number "τ", for Turing (Copeland, 1998b, 2000). τ is defined like this. Assume the Turing machines to be ordered in some way, so that we may speak of the first Turing machine in the ordering, the second, and so on (there are various standard ways of accomplishing such an ordering). Let h_1 be a constant associated with the first Turing machine in the ordering. h_1 is 1 if the first machine in the ordering eventually halts when started with a blank tape; and h_1 is 0 if the first machine runs on forever when started with a blank tape. Similarly for h_2, h_3, etc. τ is the number: $0{\cdot}h_1h_2h_3 \ldots$. (So the first few digits of τ might be $0{\cdot}000000011 \ldots$.) Like π, τ is a definite — irrational — number.[3]

The proof that with appropriate input a coupled Turing machine is hypercomputational is trivial. Let T be a coupled Turing machine with a single input channel and let the digits of τ form the input; the first digit to appear on the input channel is h_1, the second is h_2, and so on. T's input channel writes to a single square of T's tape and each successive symbol in the input stream overwrites its predecessor on this square. As each input arrives, T performs some minor computation with it — multiplies it by 2, say — and writes the result on some designated squares of the tape (in order to keep the time of operation constant, the next result always overwrites its predecessor). The succession $2{\times}h_1$, $2{\times}h_2$, etc., is not Turing-machine-computable (if it were, τ would be).

2.2. PARTIALLY RANDOM MACHINES

A partially random machine (the term is from Turing, 1948, p. 9) is a machine some of whose actions are the outcome of random influences but whose operation is otherwise determined, e.g., by a program. Some partially random machines are hypercomputational (Copeland, 2000, pp. 28–31).

One of the simplest examples of a hypercomputational partially random machine consists of a coupled Turing machine with a single input line carrying an infinite sequence of binary digits that is random. As Church argued, if a sequence of digits $r_1, r_2, \ldots r_n, \ldots$ is random, then there is no function $f(n){=}r_n$ that is calculable by the universal Turing machine (Church, 1940, pp. 134–135). Let the coupled machine be T, as above. T prints out on its tape the sequence of digits $2{\times}r_1, \ldots$. Since this sequence is itself random, the universal Turing machine cannot produce it.

One might be inclined to think that randomness is not a useful route to hypercomputation. For what work of practical value could such a machine perform? The thought is far from correct. Section 1.18.1 has already mentioned Turing's suggestion concerning the use of a random element in learning. My (2000) discusses Turing's views on randomness and freedom of the will, and argues that Turing viewed the mind as a partially random machine.

One practical application of partially random machines lies in cryptography. Following the Second World War, Colossus — the first large-scale electronic computer (see Copeland, 2001a, b) — was used in conjunction with a random device, endearingly code-named the 'Donald Duck', for the high-volume production of one-time pad in the form of punched paper tape. The overall set-up — modified Colossus plus the Donald Duck — is perspicuously represented as a hypercomputer whose output is an endless supply of one-time pad. (For more on the concept of perspicuous representation, see Section 3.17.)

In both cases described in Sections 2.1 and 2.2, *T*'s input channel is a form of oracle, although *T* does not conform exactly to Turing's description of an *o*-machine (the state χ and the two states indicating the 'pronouncement' of the oracle being absent).

2.3. ORACULAR COMPUTATION VIA PERFECT MEASUREMENT

According to some classical physical theories, the world contains continuously-valued physical magnitudes (for example, the theory of continuous elastic solids in Euclidean space). The magnitude of some physical quantity might conceivably be exactly τ units. Suppose that some mechanism *A* does store exactly τ units of such a physical quantity, which for the sake of vividness one might call 'charge'. Suppose further that a mechanism *B* can measure the quantity of 'charge' stored in *A* to any specified number of significant figures. *B* determines h_n by measuring *A*'s charge to sufficiently many significant figures and outputting the n^{th} digit of the result. *A* and *B* together form an oracle for the number-theoretic problem: 'Determine, for any given *n*, whether or not the n^{th} Turing machine halts'.

This oracle can be combined with a Turing machine to form an *o*-machine. The Turing machine inscribes a number *n* on its tape and when the machine enters state χ a subdevice passes *n* to *B*. *B* determines h_n and a subdevice places the Turing machine in one or other of two reserved states, according to whether h_n is 0 or 1.

Is this arrangement of notional components a *machine*? In 2000 I argue that it is so in the sense of 'machine' crucial to the historical debate between mechanists and anti-mechanists about physiological and psychological mechanism (a debate involving such figures as Descartes, Hobbes, and de la Mettrie). Bechtel and Richardson speak aptly of the mechanists' twin heuristic strategies of *decomposition* and *localisation* (1993, p. 23). The former heuristic seeks to decompose the activity of the system whose functioning is to be explained into a number of subordinate activities; the latter attributes these subordinate activities to specific

components of the system. The core of the claim, as put forward by the historical mechanists, that such-and-such naturally occurring item — a living body, say — is a machine is this: the item's operation can be accounted for in monistic, materialist terms and in a manner analogous to that in which the operation of an artefact, such as a clockwork figure or church organ, is explained in terms of the nature and arrangement of its components. The *o*-machine just described, like the universal Turing machine, is a machine in the sense that its behaviour is the product of the nature and arrangement of its material parts.

The claim that the mind is a machine, in the sense of 'machine' used by the historical mechanists, is evidently consistent with the hypothesis that the mind is a form of *o*-machine (see further Copeland, 2000).

Of course, the perfect measuring device B is thoroughly hypothetical. Nevertheless, this model hypercomputer serves to illustrate the point that, in discussing whether oracular computation can be carried out in the real world, we are discussing an out-and-out *empirical* matter.

2.4. Accumulator machines: computation with arbitrary reals

Consider a device like A, above, but with two input lines, i_1 and i_2, and an output line, o (Copeland, 1997b, pp. 698–670). If two charges are applied to i_1 and i_2 they accumulate in the device and their sum is available at the output line o. The device may conveniently be called an 'accumulator'. Let the accumulator be embedded in a programmable control structure. The control has the ability to clear the input lines of charge, to apply charge to the input lines, and to transfer the charge on the output line back to an input or to an external device. The following simple program can be set into the control (':=' is read 'becomes'):

BEGIN	
INPUT TO i_1	(A charge is received from some external device and applied to line i_1.)
$i_2 := i_1$	The control clears i_2 and applies the charge on i_1 to i_2.)
ADD (i_1, i_2)	(The charges on i_1 and i_2 enter the accumulator.)
$i_2 := o$	(i_2 is cleared and the charge on o is applied to i_2.)
ADD (i_1, i_2)	
OUTPUT o	(The output of the accumulator is delivered to some external device.)
HALT	

When the representation of any real number x is presented as input, the machine delivers a representation of $3x$ as output. Since x may not be Turing-machine-computable, this machine cannot be simulated by the universal Turing machine.

Charge may be negative in value and so quantities of charge can be used to represent negative numbers. Introducing additional hardware produces a machine that, given any two real numbers x and y, prints '1' if $x = y$ and prints '0' otherwise.

The machine's program is below. The additional hardware is a device able to test whether or not the charge on the output line is null.

BEGIN
INPUT TO i_1 (A charge representing x is received from an external device and placed on line i_1.)
INPUT TO i_2 (A charge representing $-y$ is received from an external device and placed on line i_2.)
ADD (i_1, i_2)
IF $o = 0$ PRINT '1'
IF $o \neq 0$ PRINT '0'
HALT

This machine cannot be simulated by the universal Turing machine even if x and y are restricted to Turing-machine-computable numbers, since the identity function over Turing-machine-computable numbers is not Turing-machine-computable (Aberth, 1968).

2.5. Accelerating Turing Machines

In my (1998b) I showed how to implement Stewart's Rapidly Accelerating Computer (see above) in the universal Turing machine, so enabling the universal Turing machine to solve the Turing-machine halting problem (see also Copeland, 1998c, 2002).

An AUTM, or Accelerating Universal Turing Machine, speeds up in the manner described by Russell (see above). Since Turing imposed no restrictions on the temporal patterning of a Turing machine's operations, an AUTM is a Turing machine within the full meaning of the act. Because, as mentioned previously,

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2}^{n-1} + \cdots$$

is less than 2, the AUTM requires less than two units of running time to do everything that the program on its tape instructs it to do. This is true even in the case of a program that does not terminate, for example a program that runs on forever calculating each successive digit of π. Each of the infinite number of operations that the non-halting program instructs the machine to perform will be completed before the end of the second unit of running time.

Given any Turing machine program, the AUTM is able to determine, in a finite amount of time, whether or not the program halts. The program is inscribed on the tape of the AUTM. The initial square of the AUTM's tape is reserved for a display of the outcome of the AUTM's computation, 0 for 'does not halt' or 1 for 'halts'. The AUTM begins its work by writing 0 on the initial square; it then proceeds, in the usual manner of the universal machine, to simulate the machine whose program it has been given. If the program halts then the scanner of the

AUTM returns to the initial square of the tape and replaces the 0 written there during the setting-up procedure by 1. If, on the other hand, the program does not halt, the scanner of the AUTM never returns to the start of the tape. Either way, at the end of the second unit of operating time the initial square contains the desired answer. (Without acceleration, this machine is a Putnam–Gold machine.)

Any air of paradox — the computation by Turing machine of a function known not to be Turing-machine-computable — vanishes when a distinction is made between two senses in which a function may be computable by a given machine, the *internal* sense and the *external* sense (Copeland, 1998c, 2002). A function is computable by a machine in the internal sense just in case the machine can, when given any argument of the function, produce and then *indicate* the corresponding value of the function, where what counts as 'indicating' can be specified in terms of features internal to the machine and without reference to the behaviour of some device or system — e.g., a clock — that is external to the machine. Various behaviours on the part of a machine can satisfy this condition, for example the machine's printing a specified symbol on its tape immediately to the right of the value once it has produced it. A function is computable by a machine in the *external* sense just in case the machine can, given any argument, be displaying the corresponding value of the function at a designated location some pre-specified number of time units after the start of the job. For example, it is in the external sense that a given function may be computable by a logic circuit. The value of the function is displayed at some designated node some pre-specified number of time units after the argument is presented at the input nodes; before and after that critical moment, the activity of the output node may afford no clue as to the desired value. Even where the logic circuit never stabilises (in the sense of eventually producing an output signal that remains constant until such time as the input signal alters) the circuit nevertheless computes values of a function in the external sense if it displays them at the designated location at the pre-specified times.

No Turing machine can compute the Turing-machine halting function in the internal sense, but the AUTM does compute this function in the external sense.

2.6. Asynchronous networks of Turing machines

Asynchronous networks of Turing machines are described in Copeland and Sylvan (1999, p. 54); we first learned of the idea from Aaron Sloman in correspondence.

The standard textbook proof that any finite assembly of Turing machines can be simulated by the universal Turing machine involves the idea of the universal machine *interleaving* the processing steps that are performed by the individual machines in the assembly, so forming a single sequence of steps. For example, if there are only two machines in the assembly, the universal machine might use odd-numbered squares of its tape to simulate the tape of one machine and even-numbered squares to simulate the tape of the other. The proof assumes that the machines in the assembly are operating in synchrony: in the case of asynchronously

operating machines, there may be no effective way of interleaving the steps. Under certain conditions, a simple network of two non-halting Turing machines writing binary digits to a common, initially blank, single-ended tape cannot be simulated by the universal Turing machine.

Let the machines in the network be m_1 and m_2 and let the additional common tape be T; m_1 and m_2 work uni-directionally along T, never writing on a square that has already been written on, and writing only on squares all of whose predecessors have already been written on. If m_1 and m_2 attempt to write simultaneously to the same square, a refereeing mechanism gives priority to m_1.

If m_1 and m_2 operate in synchrony, the evolving contents of T can be calculated by the universal machine. Where m_1 and m_2 operate asynchronously, the same is true if the *timing function* associated with each machine, Δ_1 and Δ_2 respectively, is Turing-machine-computable. The timing function Δ_1 is defined as follows (where $n, k \geq 1$): $\Delta_1(n) = k$ if and only if k units of operating time separate the n^{th} fundamental operation performed by m_1 (e.g., write) from the $n{+}1^{th}$ (e.g., move left). Similarly for m_2's timing function Δ_2. Where Δ_1 and Δ_2 are both Turing-machine-computable, the universal machine can calculate the necessary values of these functions in the course of calculating each digit of the sequence being inscribed on T.

If, however, the machines are not in synchrony and at least one of the two timing functions is not Turing-machine-computable, then the machines can inscribe a number on T that is not Turing-machine-computable. For example, suppose that m_1 prints only 1s, m_2 prints only 0s, and that the sequence of printings on T consists of 1 followed by h_1 occurences of 0 — i.e., one or no occurrence — followed by 1 followed by h_2 occurrences of 0, and so on.

Synchrony is often assumed in biological modelling. Harvey and Bossomaier (1997) suggest that modellers have been deterred from considering asynchronous models — for example, asynchronous boolean networks — because of their assumed intractability. Harvey and Bossomaier remark that, in the absence of empirical justification for the 'assumption of synchrony', the value of synchronous models is in doubt for many biological systems (1997, p. 75).

2.7. COMMUNITIES OF MATHEMATICIANS

No single Turing machine can produce (in the internal sense) each answer to the sequence of questions (asked relative to a given ordering of the Turing machines): 'Does the 1^{st} Turing machine halt?', 'Does the 2^{nd} Turing machine halt?', and so on. There is, however, no difficulty in the claim that each question can be answered (in the internal sense) by one or another Turing machine. As Turing said in a letter to Newman:

> When you say 'on a machine' do you have in mind that there is ... some fixed machine on which proofs are to be checked, and that the formal outfit is, as it were about this machine. If you take this attitude ... there is little more

> to be said: we simply have to get used to the technique of this machine and resign ourselves to the fact that there are some problems to which we can never get the answer. ... However I don't think you really hold quite this attitude because you admit that in the case of the Gödel example one can decide that the formula is true i.e. you admit that there is a fairly definite idea of a true formula which is quite different from the idea of a provable one. ... If you think of various machines I don't see your difficulty. One imagines different machines allowing different sets of proofs, and by choosing a suitable machine one can approximate 'truth' by 'provability' better than with a less suitable machine, and can in a sense approximate it as well as you please. The choice of a proof checking machine involves intuition ... [4]

Let Σ be an infinite community of mathematicians. The mathematicians learn. Each mathematician can be represented by a family of Turing machines (Section 1.18.1); at each stage in the mathematician's development, the mathematician is equivalent to some Turing machine in the family. The transition from one stage of development to the next corresponds to the learning of new methods of proof. As previously mentioned (Section 1.18.1), the transition function between the machines in a given family need not be Turing-machine-computable; and will not be so if, for example, the learning process associated with the family involves some random sequence of digits $r_1, r_2, \ldots r_n, \ldots$, so that for some or all values of n, the transition from the n^{th} to the $n{+}1^{th}$ machine in the family depends on r_n.

Development brings differentiation: different mathematicians in Σ are equivalent at a given time to different Turing machines capable of different sets of proofs. (Perhaps there was some differentiation at the initial stage also.) One may consistently suppose that there comes a point in the development of Σ at which each of the above questions can be answered by one or another of the mathematicians.

At that point, Σ forms an oracle for the number-theoretic problem of determining, given any Turing machine, whether or not the machine eventually halts when started on a blank tape. If the community is asked 'Does the n^{th} machine halt?', some member of the community will produce the correct answer in a finite number of steps. This can be achieved by each mathematician enumerating the statements that it can prove at that stage. Eventually one will produce either 'Machine n halts' or its negation.

Arguably such an oracle forms an appropriate idealisation, relative to a given number-theoretic problem, of an always finite but indefinitely growing community of learners.

3. The Very Idea of Hypercomputation: Objections and Replies

3.1. Any task that can be made completely precise can be programmed for the universal Turing machine. In other words, given enough memory and sufficient time, a standard digital computer can compute any rule-governed input-output function.

That is what Turing and Church showed. Therefore the notion of hypercomputation is otiose.

I shall assume throughout this section that when the objector speaks about what can be achieved by the universal Turing machine, attention is restricted to what can be computed by the machine in the internal sense (Section 2.5).

Turing and Church are sometimes said to have shown that a standard digital computer can, given enough memory and sufficient time, compute any rule-governed input-output function (see, for example, Churchland and Churchland, 1990, p. 26; Dreyfus, 1992, p. 72). In fact, they showed the opposite. There is nothing imprecise about the halting problem. The halting function is certainly rule-governed.

3.2. Turing showed in 1936 that every mechanical process can be carried out by the universal Turing machine. Therefore 'hypercomputers' are not machines of any sort — let alone computing machines.

Let's be clear about what Turing showed (see further Copeland, 1997a). He argued for the following thesis (1936, p. 249):

> the 'computable' numbers include all numbers which would naturally be regarded as computable.

Turing meant by this statement that the Turing-machine-computable numbers include all numbers that are calculable effectively — i.e., calculable by the ideal human clerk, described above, who works with paper and pencil, reliably but without insight or ingenuity.

In logic, 'effective' and 'mechanical' are terms of art. Neither carries its everyday sense, and the two are used interchangeably. Using 'mechanical' in its technical sense, Turing's thesis can be expressed: the Turing-machine-computable numbers include all numbers that are calculable mechanically — but this means no more and no less than that the Turing-machine-computable numbers include all numbers that are calculable by a human being working in the way described, for this is the technical meaning of 'mechanical'.

This thesis carries no implication concerning the extent of what can be calculated by a *machine*, for among the machine's repertoire of fundamental processes there may be those that a human rote-worker unaided by machinery cannot perform.

3.3. Hypercomputation seems to amount to the claim that there might be mechanical processes that are not mechanical!

True — so long as 'mechanical' means something different at the two occurrences. At the second occurrence, 'mechanical' has its technical sense: 'not mechanical' means 'cannot be done by a human computer'. At the earlier occurrence, 'mech-

anical process' means simply 'process that can be carried out by a machine'.

3.4. Over the years, a number of alternative analyses have been given of the notion of a mechanical process. Apart from Turing's analysis in terms of Turing machines, and Church's analyses in terms of lambda-definability and recursiveness, there are analyses, e.g., in terms of register machines, Post's canonical and normal systems, combinatory definability, Markov algorithms, and Gödel's notion of reckonability. The striking thing is that these various analyses all turn out to be provably equivalent in extension. Because of the prima facie diversity of the various analyses, their equivalence is strong evidence that whatever can be done by a machine, mathematically speaking, can be done by the universal Turing machine.

This is nothing more than a confusion (which I have elsewhere termed the 'equivalence fallacy' (Copeland, 2000, pp. 20, 23–25)). The analyses under discussion are all analyses of the notion of an effective method. Each seeks to characterise the processes that are mechanical in the sense that they can be carried out by a human computer. The equivalence of these analyses is evidence for the truth of Turing's thesis that the Turing-machine-computable numbers include all numbers that are calculable effectively. But the equivalence tells us nothing about the extent of processes that are mechanical in the sense that they can be carried out by a machine. Reflecting on the equivalence will not help us to decide whether, for example, a device or organ whose behaviour is the product of the nature and arrangement of its material parts can calculate more than a human computer.

3.5. 'Computable' means 'effectively calculable'. You have already pointed out that this is how Turing used the term (3.2), and in so doing he was following normal usage. Something is computable if a human computer can calculate it. A computing machine is a machine able to do the work of a human computer. This is how the words were used in Turing's day and it merely invites confusion to begin using them differently. Turing's thesis can be expressed: Whatever can be done by any computing machine can be done by the universal Turing machine. Some functions are absolutely uncomputable — uncomputable by any past, present or future machine. The halting function is an example. If hypercomputation is the computation of functions or numbers that cannot be computed with paper and pencil by a human clerk working effectively, then there is no such thing.

It is seldom productive to fight for a word. Freezing usage as it was in 1936 will not settle any of the substantive questions — the questions merely have to be rephrased. Let us say that a function f is *generated* by a machine m if and only if it is the case that, for each of the function's arguments, x, if x is presented to m as input, m will produce the corresponding value of the function, $f(x)$. Then instead of saying, for example, that a coupled Turing machine or an accelerating Turing machine can compute functions that are not computable in Turing's sense, one can, if one

wishes, say that these machines are able to generate functions not computable in Turing's sense. The whole discussion can be rephrased without loss.

3.6. It seems that according to hypercomputationalists, every function is computable (or generatable by some machine). Each number-theoretic function is computable by a machine accessing an infinite tape on which are listed all the arguments of the function and the corresponding values. ETMs (Section 1.6) even permit an entire real number to be stored on a single square of the machine's tape. And there is no reason to stop there — additional fantasy brings additional computable functions. On the new way of speaking, 'computable function' means simply 'function'. Hypercomputationalism comes down to this: the term 'computable' is redundant.

Hypercomputationalists believe that statements concerning computability are explicitly or implicitly indexed to a set of capacities and resources (see Copeland and Sylvan, 1999). When classicists say that some functions are absolutely uncomputable, what they mean is that some functions are not computable relative to the capacities and resources of a standard Turing machine. That particular index is of paramount interest when the topic is computation by effective procedures. In the wider study of computability, other indices are of importance. As the objection indicates, some indexed statements of computability are entirely trivial — for example, the statement that each number-theoretic function is computable relative to itself. This is not generally so, however. Mathematical theorems of the form 'f is computable relative to r' are often hard-won. Questions about which functions are computable relative to certain physical theories are seldom trivial. The question of which functions are computable relative to the theories that characterise the real world is of outstanding interest.

3.7. The physical version of the Church–Turing thesis rules out hypercomputation.

Let's be clear about what the Church–Turing thesis is (see further Copeland, 1997a, 2000). *Church's thesis* (Church, 1936) is:

> every function of positive integers whose values can be calculated by an effective method is lambda-definable (or recursive).

As stated above (Section 3.2), *Turing's thesis* is:

> The Turing-machine-computable numbers include all numbers computable by a human computer.

The name 'Church–Turing thesis', now standard, seems to have been introduced by Kleene (1967, p. 232):

> So Turing's and Church's theses are equivalent. We shall usually refer to them both as *Church's thesis*, or in connection with that one of its . . . versions which deals with 'Turing machines' as *the Church–Turing thesis*.

It has already been explained (Section 3.2) why the Church–Turing thesis does not rule out hypercomputation. Therefore the 'physical version' of the Church–Turing thesis alluded to in the objection cannot be the Church–Turing thesis properly so called, but is some other claim. This is misleading, not least because the Church–Turing thesis properly so called is already a physical claim, in that it places a limit on what can be done by any machine that can be simulated by a human computer.

In the literature, various non-equivalent claims are described as being physical versions of the Church–Turing thesis. This terminology is unfortunate, not only for the reason just given, but also because it tends to suggest that Church and Turing themselves either embraced, or were implicitly committed to, these claims. Examples are:

> The behaviour of any discrete physical system evolving according to local mechanical laws is recursive.
>
> Every finitely realizable physical system can be perfectly simulated by the universal Turing machine.

One should distinguish clearly between the Church–Turing thesis and the following, which I term the *maximality thesis* (Copeland, 2000, p. 15):

> The Turing-machine-computable functions include all functions that can be generated by any machine that works on finite input in accordance with a finite program of instructions and produces $f(x)$ from x by finitely many applications of fundamental processes of the machine.

The maximality thesis admits of a weaker and a stronger interpretation. On the weaker, the thesis abstracts from the issue of whether or not the machine in question is countenanced by the actual laws of nature. Any one of a number of the machines described above suffices to show that the thesis is false under this interpretation. On the stronger interpretation, the phrase 'can be generated by machine' is taken in the this-worldly sense of 'can be generated by a machine conforming to the actual laws of nature'. The universal Turing machine is presumably an example of such a machine. None of its atomic components, nor their configuration in the machine, nor the machine's mode of operation involves a violation of the laws of nature. The qualification 'physical version' will be used to indicate that the thesis is being interpreted in this second way.

Charitably understood, the objection amounts to this: the physical version of the maximality thesis rules out hypercomputation. This statement is true, but as an objection begs the entire question. The physical version of the maximality thesis is an empirical proposition whose truth value is unknown.

3.8. There is an epistemological problem with the idea of hypercomputation. Suppose Laplace's genius says 'Here is a black box for solving the Turing-machine halting problem' (The problem arises no matter which non Turing-machine-computable function is considered.) Type in any integer x and the box will deliver the

corresponding value of the halting function H(x) — or so Laplace's genius assures you. Since there is no systematic method for calculating the values of the halting function, you have no means of checking whether or not the machine is producing correct answers. Even simulating the Turing machine in question will not in general help you, because no matter how long you watch the simulation, you cannot infer that the machine will not halt from the fact that it has not yet halted.

In principle, the mathematical community is able to check whether or not the machine has produced correct answers. Since there is no systematic method for doing so, the mathematicians' deliberations will sometimes involve ingenuity and from time to time new methods of proof will have to be devised.

3.9. Nevertheless an epistemological problem remains. There is no empirical means of distinguishing the hypothesis that the box is a hypercomputer able to solve the Turing-machine halting problem from the hypothesis that the box is a Turing machine. Suppose you type one million integers into the box and for each one you receive as output the corresponding value of the halting function. This is consistent with the box being a Turing machine. No matter how large the finite subset of integers that you test, you have no way of telling whether or not the box is a Turing machine.

Exactly the same difficulty can be posed if Laplace's genius says 'Here is a black box for adding any pair of integers'. No matter how large the finite subset of pairs of integers that one tests, the run of confirming instances is consistent with the negation of the hypothesis that the box adds any pair of integers. The objection does not raise a difficulty peculiar to hypercomputation. The problem that function is underdetermined by a finite sampling of behaviour is everyone's problem — if it is a problem at all.

Engineers, of course, take boxes to bits and examine their inner workings. After an examination of a certain box's internal workings, the engineers might say: Given current physics, our best hypothesis is that this box will add any pair of integers (so long as you insert more paper tape whenever the red light goes on). It is because we credit similar prouncements by engineers that we entrust our savings and, in the case of air travel, even our lives, to computers.

The engineers may say: Given current physics, our best hypothesis is that this box will output H(x) when given any integer x.

3.10. One suggestion made by hypercomputationalists is that some form of quantum computer may be able to compute non Turing-machine-computable functions. However, the originator of the universal quantum computer, David Deutsch, states that this is not so (1985, p. 97):

> [T]he universal quantum computer . . . would have many remarkable properties not reproducible by any Turing machine. These do not include the computation of non-recursive functions, but they do include 'quantum parallelism'.

Thanks to quantum parallelism, a quantum computer requires only polynomial time in order to complete tasks for which the classical universal Turing machine requires exponential time. However, every function computable by a quantum computer is also computable, if more slowly, by the universal Turing machine.

A number of different quantum computational architectures have been proposed. Some are not hypercomputational, some are. In a paper in this collection, Kieu outlines a hypercomputational quantum computer that is able to solve Hilbert's tenth problem.

Despite what Deutsch says, his universal quantum computer is able to compute non-recursive functions, since an entire non-recursive function can be encoded into one of the real-valued parameters figuring in the quantum-mechanical description of the machine (Solovay, personal communication). For example, the Turing-machine halting function can be coded into the machine by allowing a parameter to take the value τ (Section 2.1).

3.11. One suggestion made by hypercomputationalists is that some form of analog computer employing continuous representations may be able to compute more than the universal Turing machine. But Claude Shannon proved a theorem long ago to the effect that the action of any analog computer can be approximated by the universal Turing machine to any required degree of precision.

There is no such theorem. Shannon's theorem (Shannon, 1941) states that this is so for a *particular* type of analog computer, the GPAC, or general-purpose analog computer. The GPAC is Shannon's idealisation of a differential analyser. (The first differential analyser, which was mechanical, was built in 1931 (Bush, 1931, 1936). In subsequent versions mechanical components were replaced by electromechanical, and finally by electronic, devices (Bush and Caldwell, 1945).) A differential analyser may be conceptualised as a collection of black boxes connected together in such a way as to allow considerable feedback. Each box performs a fundamental process, among which are addition, multiplication, and integration. Programming the machine consists of wiring together boxes in such a way that the desired sequences of fundamental processes are executed.

There was a crucial lacuna in the proof of Shannon's theorem. The situation was improved by Pour-El (1974), whose approach involved a significant modification to Shannon's idealised machine. Given this modification, Pour-El showed (1) that there are Turing-machine-computable functions that cannot be computed by the GPAC, and (2) that the action of the GPAC can always be approximated by the universal Turing machine (Pour-El, 1974; Theorems 3 and 7). (Pour-El's own proof

was also incomplete and was refined by Lipshitz and Rubel (1987, Rubel, 1988, 1989).)

The Shannon–Pour-El theorem does not imply that the action of *any* analog computer can be approximated by the universal Turing machine to any required degree of precision. The GPAC is an idealisation of a museum piece. If new black boxes are added to perform fundamental processes foreign to the differential analyser, then by definition the resulting machine is not a GPAC, and so falls outside the scope of the theorem.

3.12. Physical encodings of real numbers cannot be used in practice to give hypercomputational power. This is because the human operator will not be able to distinguish real numbers from arbitrarily close rational numbers. Take the first accumulator machine described in Section 2.4: a human operator could not distinguish a physical instantiation of this machine from a physical instantiation of a Turing machine.

A human operator could not so distinguish by direct inspection of the output from the accumulator. However, it does not follow that any physical instantiation of the machine might as well for all practical purposes be replaced by an instantiation of a Turing machine (any more than it follows from parallel considerations that the solar system might indistinguishably be replaced by a system in which the gravitational constant, π, and the masses of the planets are 'rounded down'). The accumulator might pass its output to some other device that is capable of doing work plainly visible to a human being and which it would not have been able to do, or do so well, had it been connected to an instantiation of a Turing machine. This is illustrated by the second toy example given in Section 2.4: the output reader takes the output of the accumulator and prints 1 or 0.

3.13. To claim that an analog or partly analog device — for example, a neural network with real-valued interconnection weights — is hypercomputational is to claim that the device is sensitive enough to be able to distinguish real numbers from arbitrarily close rational numbers. However, the real world contains noise and the presence of noise means that only a limited amount of precision is available, not the unlimited precision implied by sensitivity of this order. Maass and Orponen (1997) have shown that in the presence of noise, neural networks with real-valued weights lose their hypercomputational power (see also Maass and Sontag, 1997). The same goes for any analog or partly analog device. Noise will wash out the precision required for hypercomputational power.

The objection begs empirical questions concerning the nature of the noise that could occur in real-world neural networks and the effects that the noise could have on the functioning of the network. These empirical questions are at present open.

Siegelmann has shown that a neural network subject to *stochastic* noise remains hypercomputational (see her paper in this collection).

Even if it were true that, in noisy conditions, a hypercomputational network would degrade to a finite state automaton, this would not mean that we should stop thinking of it as having real-valued weights, or stop thinking of the network as hypercomputational. In Putnam's terminology (Section 3.17), the network is not perspicuously representable as a finite state automaton. This is illustrated by considering what happens as the noise fluctuates over time (Copeland, 1997b, pp. 673–674). As the noise fluctuates, the system shifts its identity from one automaton to another. It cannot be assumed that the function mapping a measure of the noise to finite state automata is Turing-machine-computable. The situation resembles that described by Geroch and Hartle (Section 1.12).

3.14. As Turing said, the behaviour of any discrete state machine — by which he meant machines that 'move by sudden jumps or clicks from one quite definite state to another' — can be perfectly simulated by the universal Turing machine (Turing, 1950, pp. 439, 441). Therefore no discrete state machine is a hypercomputer.

Turing did say that a 'digital computer could mimic the behaviour of any discrete state machine'; and by 'digital computer' he meant a machine 'intended to carry out any operations which could be done by a human computer' (1950, pp. 441, 438). However, his statement must be taken in context. The discussion in which the statement is embedded (1950, pp. 440–441) makes it clear that Turing intended the statement to apply only in the case of those discrete state machines that have 'a finite number of possible states' (i.e., a finite number of possible configurations) (1950, p. 440). He pointed out that when this condition is satisfied, the behaviour of the machine can be described exhaustively by a finite table of the sort nowadays commonly called a 'lookup' table (ibid.):

> discrete state machines . . . can be described by such tables provided they have only a finite number of possible states.

It is, Turing said, on the basis of being '[g]iven the table corresponding to a discrete state machine' that a digital computer is able to mimic the discrete state machine (1950, p. 441).

Several of the machines described above are examples of discrete state machines that cannot be mimicked by the universal Turing machine. These include types of coupled Turing machine, types of partially random machine, types of asynchronous network, and the learning machines described in Section 1.18.1. Except in the special case where the activity of the machine ceases when the machine has passed through some finite number of configurations, no lookup tables are forthcoming for machines of these types.

3.15. Gandy proved that no discrete state machine satisfying four physical postulates, which called Principles I–IV, can calculate more than the universal Turing machine (Gandy, 1980).

It is an open empirical question whether discrete state machines contravening Gandy's principles are permitted by the physics of the real world. Such systems are certainly permitted by Newtonian physics. (Gandy remarked: 'I am sorry that Principle IV does not apply to machines obeying Newtonian mechanics' (1980, p. 145).)

Even discrete state machines *satisfying* Gandy's principles may be hypercomputational if embedded in a universe whose physical laws have uncomputability, in the Turing sense, built into them. An asynchronous network of Turing machines provides an example.

It is important to note that Gandy's argument does not apply to partially random discrete state machines and nor to coupled Turing machines. He ruled such machines out of consideration for the purposes of his proof (1980, pp. 126–127).

Concerning his four principles, Gandy argued: 'if any of the principles be significantly weakened in (almost) any way then every function becomes calculable' (1980, p. 130). He appeared to consider this a *reductio ad absurdum*. As Israel (2002, p. 197) put the point, each of Gandy's conditions

> is necessary to avoid a certain kind of absurdity or vacuity. The form of the result is as follows: if we allow machines that satisfy any three of the ... conditions, but not all four, we can show that for any number-theoretic function f, there is a machine M_f such that for any given (notation for) n, M_f will output (a notation for) $f(n)$. That is ... every number-theoretic function is computable.

As explained above, however, hypercomputationalists need not agree that this argument is a *reductio* (Section 3.6). Each number-theoretic function *is* computable, relative to some set of capacities and resources.

Gandy's proof is discussed later in this collection by Shagrir and Pitowsky (see also Shagrir, 2002). Shagrir and Pitowsky describe a model of hypercomputation that is consistent both with General Relativity and with Gandy's assumption (part of his Principle IV) that the speed of signal propagation is bounded.

Doyle argues in his paper in this collection that Gandy in effect shows that if 'slight variations' to the procedures traditionally considered effective are permitted, then non-recursive functions are effectively computable. The variations are achieved 'simply by allowing the "same" physical operations to involve more information or information paths than usual'.

In unpublished work Gandy argued for the impossibility of using analog machines to calculate non Turing-machine-computable functions. In this collection Kieu comments on Gandy's unpublished argument, arguing that it fails to apply to his own model of quantum computation.

3.16. Hypercomputation requires that an infinite amount of information exist in a finite volume of space. This is not physically possible, since it infringes the 'Beckenstein bound' (Lokhorst, 2000). Beckenstein has shown that, given certain assumptions, any spherical region of space with finite radius and finite energy contains only a finite amount of information. This is shown by considering the maximum number of distinguishable quantum states that a system occupying the sphere could have.

The Beckenstein bound is conjectural. Some models of hypercomputation do infringe it, for example the accumulator machines described in Section 2.4. But not all do. Examples of those that do not include coupled Turing machines, asynchronous networks of Turing machines, partially random machines, and the learning machines described in Section 1.18.1.

To show that the Beckenstein bound is consistent with hypercomputation, it suffices to consider what Penrose calls 'oracle-machine universes' (1994, pp. 380, 30–33). An oracle machine universe is a toy universe with discrete time and whose dynamical evolution is described by some function on the integers that is not Turing-machine-computable. For example, imagine a universe consisting only of a finite string of beads. Some beads in the string are spherical, some cubic (there are no other shapes). The bead at one end of the string is designated 'live'. The universe 'updates' once each second. Updating consists of the live bead producing a single offspring bead, which takes its place at the end of the string beside its parent, becoming the live bead and producing its own offspring at the next update. Irrespective of the shape of the parent, the offspring may be either spherical or cubic. The 'fundamental law of nature' determining the shape of the offspring of the n^{th} bead in the string is:

$$\text{Spherical if } h_n = 0, \text{ cubic if } h_n = 1.$$

An always finite, but unbounded, string grows from a single bead. The growing string is in effect a machine that produces the values of the halting function. At no point in the evolution of the universe is the Beckenstein bound infringed.

3.17. No finite discrete system can be hypercomputational. Any discrete state machine whose total number of configurations is finite can be perfectly simulated by a Turing machine. In the real world, no machine can have an infinite number of configurations. Artefacts wear out and break down, brains die. Even a machine that in and of itself could continue running forever would pass through only a finite number of configurations before the heat death of the universe. Any real-world discrete state machine is equivalent to a Turing machine.

It is certainly true that if, in its finite lifespan, a machine produces a finite number of outputs, then following the machine's demise all the ouputs can be gathered together in a finite table. The table can be given to a Turing machine. Equipped

with this crib, the Turing machine is able to simulate the machine in question by regurgitating the outputs listed in the table. However, the important point is that if the machine is a hypercomputer, there may in fact be no way of arriving at this table other than by employing the hypercomputer itself to generate the outputs. This table, produced by the hypercomputer, may be the *only* effective procedure for obtaining the machine's outputs.

In the objection two quite different issues are conflated, that of the in-principle post-hoc simulability by Turing machine of any discrete state machine situated in a bounded environment (bounded time, bounded energy, bounded tape, etc.); and, on the other hand, that of whether the machine in question is perspicuously represented as being a Turing machine (see further Copeland, 1997b, p. 677; 2000, pp. 32).

Knowing that a machine is equivalent to a Turing machine *if* the action of the machine is arbitrarily terminated after some finite number of configurations leaves us no wiser as to whether the machine, when abstracted out from its bounded environment, is a finite state automaton, a universal Turing machine, or a hypercomputer of some sort. For example, if the issue is whether the human cognitive architecture, abstracted out from sources of inessential boundedness (such as mortality), is perspicuously represented as a generator of (one or more) Turing-machine-uncomputable functions, then the fact that the brain is simulable by Turing machine when death is assumed tells us nothing either way.

In answering a similar objection, Putnam said (1992, p. 6):

> *[E]very* physical system whose behavior we want to know only up to some specified level of accuracy and whose 'lifetime' is finite can be simulated by [a Turing machine]! [This] does not prove that such a simulation is in any sense a *perspicuous representation* of the behavior of the system.

As Putnam pointed out, the objection in effect considers performance, whereas what is at issue is competence.

Putnam continued (1992, p. 7):

> In sum, it does not seem that there is any principled reason why we must be perspicuously representable as Turing machines ... Or any reason why we must be representable in this way at all — even non-perspicuously — under the idealization that we live forever and have potentially infinite external memories.

Notes

[1] One of the first to speak of oracles in *physical* terms was Davis (1958, p. 11). He said: 'For how can we ever exclude the possibility of our being presented, some day (perhaps by some extraterrestrial visitors), with a (perhaps extremely complex) device or "oracle" that "computes" a noncomputable function?' Acknowledging that the possibility cannot be ruled out, Davis said: 'However, there are fairly convincing reasons for believing that this will never happen' (ibid.). The only argument that he offered is conspicuously weak, consisting of the observation that certain modifications to a Turing machine — he mentioned machines able to insert squares into their own tape, machines able to move

left or right more than a single square in a single operation, and machines that operate on two- (or higher) dimensional tape — result in a machine no more powerful than a Turing machine (1958, pp. 11–12, 64).

[2]The term 'Putnam–Gold' is appropriate since Putnam's paper was received by the JSL in 1963, Gold's in 1964.

[3]*Pace* the intuitionists. Turing assumes a classical framework.

[4]The undated letter is in the Modern Archive Centre, King's College Library, Cambridge. It was written in about 1940.

References

Aberth, O. (1968), 'Analysis in the Computable Number Field', *Journal of the Association of Computing Machinery* 15, pp. 275–299.

Abramson, F.G. (1971), 'Effective Computation over the Real Numbers', *Twelfth Annual Symposium on Switching and Automata Theory*, Northridge, CA: Institute of Electrical and Electronics Engineers.

Ambrose, A. (1935), 'Finitism in Mathematics (I and II)', *Mind* 35, pp. 186–203 and 317–340.

Bechtel, W. and Richardson, R.C. (1993), *Discovering Complexity: Decomposition and Localization as Strategies in Scientific Research*, Princeton: Princeton University Press.

Blake, R.M. (1926), 'The Paradox of Temporal Process', *Journal of Philosophy* 23, pp. 645–654.

Blum, L., Shub, M. and Smale, S. (1989), 'On a Theory of Computation and Complexity Over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines', *Bulletin of the American Mathematical Society*, New Series, 21, pp. 1–46.

Blum, L., Cucker, F., Shub, M. and Smale, S. (1998), *Complexity and Real Computation*, New York: Springer.

Bogdanov, A.A. (1980), *Essays in Tektology: The General Science of Organisation*, trans. G. Gorelik, Seaside, CA: Intersystems.

Boolos, G.S. and Jeffrey, R.C. (1974), *Computability and Logic*, Cambridge: Cambridge University Press.

Bush, V. (1931), 'The Differential Analyser: A New Machine for Solving Differential Equations', *Journal of the Franklin Institute* 212, pp. 447–488.

Bush, V. (1936), 'Instrumental Analysis', *Bulletin of the American Mathematical Society* 42, pp. 649–669.

Bush, V. and Caldwell, S.H. (1945), 'A New Type of Differential Analyser', *Journal of the Franklin Institute* 240, pp. 255–326.

Cleland, C.E. (1993), 'Is the Church–Turing Thesis True?', *Minds and Machines* 3, pp. 283–312.

Church, A. (1936), 'An Unsolvable Problem of Elementary Number Theory', *American Journal of Mathematics* 58, pp. 345–363.

Church, A. (1940), 'On the Concept of a Random Sequence', *American Mathematical Society Bulletin* 46, pp. 130–135.

Churchland, P.M. and Churchland, P.S. (1990), 'Could a Machine Think?', *Scientific American* 262, pp. 26–31.

Copeland, B.J. (1997a), 'The Church–Turing Thesis', in E. Zalta, ed., *Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu>.

Copeland, B.J. (1997b), 'The Broad Conception of Computation', *American Behavioral Scientist* 40, pp. 690–716.

Copeland, B.J. (1998a), 'Turing's O-machines, Penrose, Searle, and the Brain', *Analysis* 58, pp. 128–138.

Copeland, B.J. (1998b), 'Super Turing-Machines', *Complexity* 4, pp. 30–32.

Copeland, B.J. (1998c), 'Even Turing Machines Can Compute Uncomputable Functions', in C. Calude, J. Casti and M. Dinneen, eds., *Unconventional Models of Computation*, London: Springer.
Copeland, B.J. (2000), 'Narrow Versus Wide Mechanism', *Journal of Philosophy* 96, pp. 5–32.
Copeland, B.J. (2001a), 'Colossus and the Dawning of the Computer Age', in R. Erskine and M. Smith, eds., *Action This Day*, London: Bantam Books.
Copeland, B.J. (2001b), 'Modern History of Computing', in E. Zalta (ed.), *Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu>.
Copeland, B.J. (2002), 'Accelerating Turing Machines', *Minds and Machines* 12, pp. 281–301.
Copeland, B.J. and Proudfoot, D. (1999a), 'Alan Turing's Forgotten Ideas in Computer Science', *Scientific American* 280, pp. 76–81.
Copeland, B.J. and Proudfoot, D. (1999b), 'The Legacy of Alan Turing', *Mind* 108, pp. 187–195.
Copeland, B.J. and Sylvan, R. (1999), 'Beyond the Universal Turing Machine', *Australasian Journal of Philosophy* 77, pp. 46–66.
Davis, M. (1958), *Computability and Unsolvability*, New York: McGraw-Hill.
Deutsch, D. (1985), 'Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer', *Proceedings of the Royal Society*, Series A, 400, pp. 97–117.
Dreyfus, H.L. (1992), *What Computers Still Can't Do: A Critique of Artificial Reason*, Cambridge, MA: MIT Press.
Earman, J. and Norton, J.D. (1993), 'Forever is a Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes', *Philosophy of Science* 60, pp. 22–42.
Earman, J. and Norton, J.D. (1996), 'Infinite Pains: The Trouble with Supertasks', in A. Morton and S.P. Stich, eds., *Benacerraf and his Critics*, Oxford: Blackwell.
Gandy, R. (1980), 'Church's Thesis and Principles for Mechanisms', in J. Barwise, H.J. Keisler and K. Kunen, eds., *The Kleene Symposium*, Amsterdam: North-Holland.
Geroch, R. and Hartle, J.B. (1986), 'Computability and Physical Theories', *Foundations of Physics* 16, pp. 533–550.
Gold, E.M. (1965), 'Limiting Recursion', *Journal of Symbolic Logic* 30, pp. 28–48.
Harvey, I. and Bossomaier, T. (1997), 'Time Out of Joint: Attractors in Asynchronous Random Boolean Networks', in P. Husbands and I. Harvey, eds., *Fourth European Conference on Artificial Life*, Cambridge, MA: MIT Press.
Hogarth, M.L. (1992), 'Does General Relativity Allow an Observer to View an Eternity in a Finite Time?', *Foundations of Physics Letters* 5, pp. 173–181.
Hogarth, M.L. (1994), 'Non-Turing Computers and Non-Turing Computability', *PSA 1994* 1, pp. 126–138.
Israel, D. (2002), 'Reflections on Gödel's and Gandy's Reflections on Turing's Thesis', *Minds and Machines* 12, pp. 181–201.
Kampis, G. (1991), *Self-Modifying Systems in Biology and Cognitive Science: A New Framework for Dynamics, Information and Complexity*, Oxford: Pergamon.
Kampis, G. (1995), 'Computability, Self-Reference, and Self-Amendment', *Communications and Cognition-Artificial Intelligence* 12, pp. 91–110.
Karp, R.M. and Lipton, R.J. (1982), 'Turing Machines that Take Advice', in E. Engeler et al., eds., *Logic and Algorithmic*, Genève: L'Enseignement Mathématique.
Kleene, S.C. (1967), *Mathematical Logic*, New York: Wiley.
Komar, A. (1964), 'Undecidability of Macroscopically Distinguishable States in Quantum Field Theory', *Physical Review*, second series, 133B, pp. 542–544.
Kreisel, G. (1965) 'Mathematical Logic', in T.L. Saaty, ed., *Lectures on Modern Mathematics*, Vol. 3, New York: John Wiley.
Kreisel, G. (1967), 'Mathematical Logic: What Has it Done For the Philosophy of Mathematics?', in R. Schoenman, ed., *Bertrand Russell: Philosopher of the Century*, London: George Allen and Unwin.

Kreisel, G. (1970), 'Hilbert's Programme and the Search for Automatic Proof Procedures', in M. Laudet et al., eds., *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics, Vol. 125, Berlin: Springer.

Kreisel, G. (1971), 'Some Reasons for Generalising Recursion Theory', in R.O. Gandy and C.M.E. Yates, eds., *Logic Colloquium '69*, Amsterdam: North-Holland.

Kreisel, G. (1972), 'Which Number Theoretic Problems Can Be Solved in Recursive Progressions on π_1^1-Paths Through 0?', *Journal of Symbolic Logic* 37, pp. 311–334.

Kreisel, G. (1974), 'A Notion of Mechanistic Theory', *Synthese* 29, pp. 11–26.

Kreisel, G. (1982), Review of Pour-El and Richards, *Journal of Symbolic Logic* 47, pp. 900–902.

Kreisel, G. (1987), 'Church's Thesis and the Ideal of Formal Rigour', *Notre Dame Journal of Formal Logic* 28, pp. 499–519.

Krylov, S.M. (1986), 'Formal Technology and Universal Systems', *Cybernetics*, Part 1: No. 4, pp. 85–89, Part 2: No. 5, pp. 28–31.

Krylov, S.M. (1996), 'Formal Technology and Cognitive Processes', *International Journal of General Systems* 24, pp. 233–243.

Krylov, S.M. (1997), *Formal Technology in Philosophy, Engineering, Bio-evolution and Sociology*, Samara State Technical University.

Kugel, P. (1986) 'Thinking May Be More Than Computing', *Cognition* 22, pp. 137–198.

Lipshitz, L. and Rubel, L.A. (1987), 'A Differentially Algebraic Replacement Theorem, and Analog Computability', *Proceedings of the American Mathematical Society* 99, pp. 367–372.

Lokhorst, G.J. (2000), 'Why I am Not a Super-Turing Machine', Hypercomputation Workshop, University College, London, 24 May 2000.

Maass, W. and Orponen, P. (1997), 'On the Effect of Analog Noise in Discrete-Time Analog Computations', NeuroColt Technical Report Series, NC-TR-97-042.

Maass, W. and Sontag, E.D. (1997), 'Analog Neural Nets with Gaussian or Other Common Noise Distributions Cannot Recognise Arbitrary Regular Languages', NeuroColt Technical Report Series, NC-TR-97-043.

MacLennan, B.J. (1987), 'Technology-Independent Design of Neurocomputers: The Universal Field Computer', *IEEE First International Conference on Neural Networks*, Vol. 3, San Diego, CA: Institute of Electrical and Electronics Engineers.

Penrose, R. (1989), *The Emperor's New Mind Concerning Computers, Minds, and the Laws of Physics*, Oxford: Oxford University Press.

Penrose, R. (1990), *Précis of The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*, *Behavioural and Brain Sciences* 13, pp. 643–655 and 692–705.

Penrose, R. (1994), *Shadows of the Mind: A Search for the Missing Science of Consciousness*, Oxford: Oxford University Press.

Pitowsky, I. (1990), 'The Physical Church Thesis and Physical Computational Complexity', *Iyyun* 39, pp. 81–99.

Pour-El, M.B. (1974), 'Abstract Computability and its Relation to the General Purpose Analog Computer', *Transactions of the American Mathematical Society* 199, pp. 1–28.

Pour-El, M.B. and Richards, J.I. (1979), 'A Computable Ordinary Differential Equation Which Possesses No Computable Solution', *Annals of Mathematical Logic* 17, pp. 61–90.

Pour-El, M.B. and Richards, J.I. (1981), 'The Wave Equation with Computable Initial Data such that its Unique Solution is not Computable', *Advances in Mathematics* 39, pp. 215–239.

Pour-El, M.B. and Richards, J.I. (1989), *Computability in Analysis and Physics*, Berlin: Springer.

Putnam, H. (1960), 'Minds and Machines', in S. Hook, ed., *Dimensions of Mind*, New York: New York University Press.

Putnam, H. (1965), 'Trial and Error Predicates and the Solution of a Problem of Mostowski', *Journal of Symbolic Logic* 30, pp. 49–57.

Putnam, H. (1992), *Renewing Philosophy*, Cambridge, MA: Harvard University Press.

Rubel, L.A. (1985), 'The Brain as an Analog Computer', *Journal of Theoretical Neurobiology* 4, pp. 73–81.

Rubel, L.A. (1988), 'Some Mathematical Limitations of the General-Purpose Analog Computer', *Advances in Applied Mathematics* 9, pp. 22–34.

Rubel, L.A. (1989), 'Digital Simulation of Analog Computation and Church's Thesis', *Journal of Symbolic Logic* 54, pp. 1011–1017.

Russell, B.A.W. (1915), *Our Knowledge of the External World as a Field for Scientific Method in Philosophy*, Chicago: Open Court.

Russell, B.A.W. (1936), 'The Limits of Empiricism', *Proceedings of the Aristotelian Society* 36, pp. 131–150.

Scarpellini, B. (1963), 'Zwei Unentscheitbare Probleme der Analysis', *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 9, pp. 265–289.

Shagrir, O. (2002), 'Effective Computation by Humans and Machines', *Minds and Machines* 12, pp. 221–240.

Shannon, C.E. (1941), 'Mathematical Theory of the Differential Analyser', *Journal of Mathematics and Physics of the Massachusetts Institute of Technology* 20, pp. 337–354.

Siegelmann, H.T. and Sontag, E.D. (1992), 'On the Computational Power of Neural Nets', *Proceedings of the* 5^{th} *Annual ACM Workshop on Computational Learning Theory*, Pittsburgh, pp. 440–449.

Siegelmann, H.T. and Sontag, E.D. (1994), 'Analog Computation via Neural Networks', *Theoretical Computer Science* 131, pp. 331–360.

Smale, S. (1988), 'The Newtonian Contribution to Our Understanding of the Computer', *Queen's Quarterly* 95, pp. 90–95.

Stannett, M. (1990), 'X-Machines and the Halting Problem: Building a Super-Turing Machine', *Formal Aspects of Computing* 2, pp. 331–341.

Stewart, I. (1991a), 'Deciding the Undecidable', *Nature* 352, pp. 664–665.

Stewart, I. (1991b), 'The Dynamics of Impossible Devices', *Nonlinear Science Today* 1, pp. 8–9.

Turing, A.M. (1936–1937), 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society*, Series 2, 42, pp. 230–265.

Turing, A.M. (1938), 'Systems of Logic Based on Ordinals'. Dissertation presented to the faculty of Princeton University in candidacy for the degree of Doctor of Philosophy. Published in *Proceedings of the London Mathematical Society* 45 (1939), pp. 161–228.

Turing, A.M. (1945), 'Proposal for Development in the Mathematics Division of an Automatic Computing Engine (ACE)', in B.E. Carpenter and R.W. Doran, eds., *A.M. Turing's ACE Report of 1946 and Other Papers*, Cambridge, MA: MIT Press. A digital facsimile of the original document may be viewed in The Turing Archive for the History of Computing <http://www.AlanTuring.net/proposed_ electronic_calculator>.

Turing, A.M. (1947), 'Lecture to the London Mathematical Society on 20 February 1947', in B.E. Carpenter and R.W. Doran, eds., *A.M. Turing's ACE Report of 1946 and Other Papers*, Cambridge, MA: MIT Press.

Turing, A.M. (1948), 'Intelligent Machinery', in B. Meltzer and D. Michie, eds., *Machine Intelligence 5*, Edinburgh: Edinburgh University Press. A digital facsimile of the original document may be viewed in The Turing Archive for the History of Computing <http://www.AlanTuring.net/intellige nt_machinery>.

Turing, A.M. (1950), 'Computing Machinery and Intelligence', *Mind* 59, pp. 433–460.

Turing, A.M. (1951), 'Can Digital Computers Think?', in B.J. Copeland, ed., 'A Lecture and Two Radio Broadcasts on Machine Intelligence by Alan Turing', in K. Furukawa, D. Michie and S. Muggleton, eds., *Machine Intelligence 15*, Oxford: Oxford University Press.

Turing, A.M. (c. 1951), 'Intelligent Machinery, A Heretical Theory', in B.J. Copeland, ed., 'A Lecture and Two Radio Broadcasts on Machine Intelligence by Alan Turing', in K. Furukawa, D. Michie and S. Muggleton, eds., *Machine Intelligence 15*, Oxford: Oxford University Press.

Wegner, P. (1997), 'Why Interaction is More Powerful than Algorithms', *Communications of the ACM* 40, pp. 80–91.
Weyl, H. (1927), *Philosophie der Mathematik und Naturwissenschaft*, Munich: R. Oldenbourg.
Wolfram, S. (1985), 'Undecidability and Intractability in Theoretical Physics', *Physical Review Letters* 54, pp. 735–738.