Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 317 (2004) 251–267
Theoretical Computer Science
www.elsevier.com/locate/tcs

Hypercomputation: philosophical issues

B. Jack Copeland*
Department of Philosophy, University of Canterbury, Private Bag 4800, Christchurch, New Zealand

Received 9 September 2003; received in revised form 1 October 2003

Abstract

A survey of the field of hypercomputation, including discussion of four a priori objections to the possibility of hypercomputation. An exegesis of Turing's pre- and post-war writings on the mind is given, and Turing's views on the scope of machines are discussed.

Keywords: Turing; Church; Super-Turing; Oracle; Mind

1. What is hypercomputation?

Hypercomputation is the computation of functions or numbers that cannot be computed in the sense of Turing [50], i.e. cannot be computed with paper and pencil in a finite number of steps by a human clerk working effectively.[1] A hypercomputer is any machine, notional or real, that is able to compute more than this human computer. Hypercomputers compute functions or numbers, or more generally solve problems or carry out tasks, that lie beyond the reach of the standard universal Turing machine.

The additional computational power of a hypercomputer may arise because the machine possesses, among its repertoire of fundamental operations, one or more processes 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

* Tel.: +64-3-3411053; fax: +64-3-3642889.
E-mail address: j.copeland@phil.canterbury.ac.nz (B.J. Copeland).
[1] On Turing's 1936 analysis of computation, see [9].

0304-3975/$ - see front matter © 2003 Published by Elsevier B.V.
doi:10.1016/j.tcs.2003.12.014

some finite number of steps. Some notional hypercomputers are discrete state machines while others involve infinite precision measurements or real-valued connection weights and the like.

Section 2 describes some elementary models of hypercomputation. These simple notional machines serve 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 capacities, richer or poorer. The capacities specified by Turing in 1936 [50] occupy no privileged position.

Section 3 deals with some objections to the possibility of hypercomputation. Section 4 concerns Turing's oracle machines or *o*-machines [51]. Section 5 discusses some exegetical issues concerning the writings of Turing and Church.

2. Elementary models

2.1. Coupled Turing machines

Coupled Turing machines are described in [10, pp. 694–695 and 19, pp. 51–52] (the term 'coupled Turing machine' is from the latter paper). A coupled Turing machine is a Turing machine coupled to an environment. In [59] 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.

The proof that with appropriate input a coupled Turing machine is hypercomputational is trivial. Let 'τ' (for Turing) be the number $0 \cdot h_1 h_2 h_3 \ldots$, where h_i is 1 if the ith machine halts when started with a blank tape, and is 0 otherwise [11,14]. Let T be a coupled Turing machine with a single input channel and let the digits of τ form the input. T's input channel writes to a single square of T's tape and each successive symbol h_i 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 $2 \times h_1,\ 2 \times h_2 \ldots$ is not Turing-machine-computable.

2.2. Partially random machines

A partially random machine (the term is from Turing's [53, p. 9] is a machine some of whose actions are the outcome of random influences but whose operations

are otherwise determined, e.g. by a program. Some partially random machines are hypercomputational [14, 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 [8, 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? One practical application of partially random machines lies in cryptography. Following the Second World War, Colossus—the first large-scale electronic computer (see [15,16])—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 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 3.4.)

2.3. Hypercomputation 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 stores 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 nth digit of the result. A and B together form an oracle for the problem: 'Determine, for any given n, whether or not the nth Turing machine halts.'

Is this arrangement of notional components a *machine*? In [14] 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 [2, p. 23] speak aptly of the mechanists' twin heuristic strategies of *decomposition* and *localisation*. 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 oracle just described,

no less than 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 oracle machine (see further [14]).

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. Accelerating Turing machines

A universal Turing machine that operates like Bertrand Russell's accelerating worker [42] is able to solve the Turing-machine halting problem [11,12,17]. In a discussion of a paper by Alice Ambrose [1] Russell wrote:

> Miss Ambrose says it is *logically* impossible [for a man] 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 [42, pp. 143–144].

An accelerating universal Turing machine (AUTM) speeds up in the manner described by Russell. 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

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

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 halt—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 Gold–Putnam machine [24,40]; see also [3,4,23,26].)

2.5. Asynchronous networks of Turing machines

Asynchronous networks of Turing machines are described in [19, p. 54] (we first learned of the idea from Aaron Sloman in correspondence).[2]

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. 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_i is defined as follows (where n, $k \geqslant 1$): $\Delta_i(n) = k$ if and only if k units of operating time separate the nth fundamental operation performed by m_i from the $n+1$th. 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 1*s*, m_2 prints only 0*s*, and that the sequence of printings on T consists of 1 followed by h_1 occurrences of 0—i.e. one or no occurrence—followed by 1 followed by h_2 occurrences of 0, etc.)

Synchrony is often assumed in biological modelling. Harvey and Bossomaier [25] 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 [25, p. 75].

3. Objections to the possibility of hypercomputation

This section answers four objections. Versions of each are commonly encountered. The first two objections are epistemological in nature, and the third and fourth concern finiteness. (Some other objections to the possibility of hypercomputation are answered in my [18].)

[2] Mark Burgin [5] argues that even a system of two asynchronous finite automata is hypercomputational.

3.1. First objection

"Suppose Laplace's genius says: 'Here is a black box for solving the Turing-machine halting problem.' Type in any integer n and the box will deliver the corresponding value of the halting function h_n—or so Laplace's genius tells us. But since there is no systematic method for calculating the values of the halting function, we have no means of testing whether or not the machine is producing correct answers. A computing machine that cannot be checked for reliability—useless!"

Yet the mathematical community can, in principle, check whether or not the machine has produced correct answers (assuming an unlimited supply of mathematicians). Since there is no uniform method for carrying out the check, from time to time new methods will have to be devised.

This response to the objection might be thought unrealistic, but given its purpose it is not. The response shows that the crucial step in the objection, namely the inference from 'there is no systematic method for calculating the values of the halting function' to 'we have no means of testing', is a non-sequitur. The absence of a systematic method is no block to calculation. Turing himself said as much, in a different connection:

> Let δ be a sequence whose nth figure is 1 or 0 according as n is or is not satisfactory. It is an immediate consequence of the theorem of § 8 that δ is not computable. It is (so far as we know at present) possible that any assigned number of figures of δ can be calculated, but not by a uniform process. When sufficiently many figures of δ have been calculated, an essentially new method is necessary in order to obtain more figures. [50, p. 253]

(n is satisfactory if it is the description number of a circle-free Turing machine, i.e., a Turing machine that prints an infinite number of binary digits, and is unsatisfactory otherwise.)

It might be thought that this answer to the objection simply begs the question, for does not the answer assume that mathematicians are *not* Turing machines? This is not presupposed, however. The fact that no single Turing machine can calculate the values of the halting function does not entail that a community of Turing machines cannot do so.

Turing made some remarks relevant to the latter issue in a letter to Max Newman. (The letter was written while Turing was working on Enigma at Bletchley Park. Newman taught Turing at Cambridge before the war. In 1942 Newman joined the codebreakers at Bletchley Park. There he mechanized Tutte's method of attack on the German Tunny machine and designed the Heath Robinson, precursor to Colossus, the first large-scale electronic computer (see [15]).) Turing said:

> I think you take a much more radically Hilbertian attitude about mathematics than I do. You say 'If all this whole formal outfit is not about finding proofs which can be checked on a machine it's difficult to know what it is about'. When you say 'on a machine' do you have in mind that there is (or should be or could be, but has not been actually described anywhere) 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 (and it is this one that seems to me so extreme Hilbertian) 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. On these lines my ordinal logics would make no sense. 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. Throughout my paper on ordinal logics [51] I have been assuming this too. …

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, which is interchangeable with the intuition required for finding an $\underline{\Omega}$ if one has an ordinal logic $\underline{\Lambda}$, or as a third alternative one may go straight for the proof and this again requires intuition: or one may go for a proof finding machine. I am rather puzzled why you draw this distinction between proof finders and proof checkers. It seems to me rather unimportant as one can always get a proof finder from a proof checker, and the converse is almost true: the converse fails if for instance one allows the proof finder to go through a proof in the ordinary way, and then, rejecting the steps, to write down the final formula as a 'proof' of itself. One can easily think up suitable restrictions on the idea of proof which will make this converse true and which agree well with our ideas of what a proof should be like. [3]

3.2. Second objection

The second epistemological objection concerns underdetermination. A version of the objection is attributed to Dana Scott, who is quoted by Davis as follows [21]:

70 years of research on Turing degrees has shown the structure to be extremely complicated. In other words, the hierarchy of oracles is worse than any political system. No one oracle is all powerful.

Suppose some quantum genius gave you an oracle as a black box. No finite amount of observation would tell you what it does and why it is non-recursive. Hence, there would be no way to write an algorithm to solve an understandable problem you couldn't solve before! Interpretation of oracular statements is a very fine art—as they found out at Delphi!

The problem that Scott is pointing to appears, however, to be a particular case of the general problem posed by Kripke's Wittgenstein [34]—and this is *everyone's* problem, having nothing to do with non-recursiveness per se. To illustrate: suppose God hands you a black box, informing you only that the box is able to tell you all the values of some Turing-machine computable two-place function on the natural numbers. No finite amount of observation would reveal what the black box does. Plus or quus? 'Quus' is Kripke's term for a 'bent' version of integer addition, behaving like normal addition

[3] Letter from Turing to Newman, undated, cc. 1940 (in the Modern Archive Centre, King's College, Cambridge (catalogue reference D 2)).

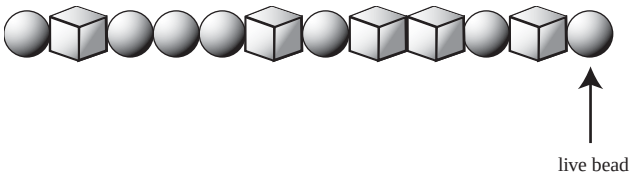

Fig. 1. An oracle-machine universe.

over some initial segment of the number sequence and then deviating in some way. Does the box implement plus, or $quus_1$, or $quus_2$, or … ? No matter how many pairs of numbers are tried out—finitely many—the box's behaviour will be consistent with many different rules.

Engineers will want to know why we are required to treat the device as a black box. Why can't we take it apart and study it in the light of our best physical theories? Once they have taken the plus/quus box apart and reassembled it, the engineers may tell us 'Given current physics, our best hypothesis is that this box will add any pair of non-negative integers (so long as you insert more tape whenever the red light goes on).' It is because we believe similar pronouncements by engineers that we entrust our savings, and in the case of air travel even our lives, to computers. The engineers' task is in principle no different in the case of Scott's box, assuming, as we may in this imaginary scenario, that the best quantum physics of their day is non-recursive.

3.3. Third objection

"Hypercomputation requires that an infinite amount of information exist in a finite volume of space. This is not physically possible. So hypercomputation is physically impossible."

Gert-Jan Lokhorst put the objection this way: hypercomputation infringes the 'Bekenstein bound'—Bekenstein argued that consideration of the maximum number of distinguishable quantum states of any system occupying a spherical region of space with finite radius and finite energy suggests that the system can contain only a finite amount of information [35].

To rebut this objection it suffices to consider a simple oracle-machine universe (see Fig. 1). An oracle-machine universe is a toy universe with discrete time (see [37, pp. 30–33 and 380]). The dynamical evolution of the universe is described by some function on the non-negative integers that is not Turing-machine computable. The universe depicted in Fig. 1 consists of nothing but a finite string of beads. Some beads are cubic and some are spherical. The bead at the right-hand end of the string (from the reader's perspective) is designated 'live'. The universe grows in discrete spurts, one spurt each second. A spurt consists of the live bead producing a single offspring bead. The offspring takes its place at the end of the string beside its parent, becoming the live bead and producing its own offspring at the next spurt.

Irrespective of the shape of the parent, the offspring may be spherical or cubic. The fundamental law of nature determining the shape of the offspring of the nth bead in

the string is:

Spherical if $h_n = 0$, cubic if $h_n = 1$.

The string that grows from a single initial bead is always finite, although unbounded. At no point in the evolution of the universe is there an infinite amount of information in a finite volume of space.

The growing string is in effect an oracle for the halting function.

3.4. Fourth objection

"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. This is so simply because any finite set is computable (which in turn is so in virtue of the fact that it is legitimate to code the members of any finite set onto the tape of a Turing machine as initial data). 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."

There is of course a parallel objection applying to non-discrete machines, and in that case 'perfectly simulated' is replaced by something like 'simulated to any required degree of accuracy'.

These objections are often advanced. This is surprising, for as long ago as 1992 Hilary Putnam dealt decisively with both forms.

> [*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. [41, p. 6].

As Putnam pointed out, a perspicuous representation addresses competence, whereas the objection in effect considers only performance [41, pp. 6–7]. Putnam continued:

> 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. [41, p. 7].

There is in addition a difficulty concerning the knowledge that is required in order for a Turing machine to carry out its simulation of the hypercomputational discrete state machine. How is this knowledge to be obtained? Equipped with some form of look-up table setting out the entire behaviour of the hypercomputer during its finite life, the Turing machine can certainly regurgitate the behaviour, so simulating the hypercomputer. But the table may be obtainable only post hoc, by running the hypercomputer and exhaustively logging its behaviour through its finite life.

The sheer fact that the finitely-lived hypercomputer can be simulated by a Turing machine tells you nothing of any real interest about the hypercomputer, and certainly does not imply that in practice the hypercomputer is redundant.

4. Turing's *o*-machines: a general framework for hypercomputation

Turing's abstract *o*-machines seem to be the earliest and most general form of hypercomputer to appear in the literature. (Turing introduced the concept of an *o*-machine in his Ph.D. thesis (Princeton, 1938); subsequently published as [51], this is a classic of recursive function theory.) An *o*-machine is a Turing machine augmented with an 'oracle'—a fundamental process that produces the values of some non Turing-machine-computable function (for example the Turing-machine halting function). The concept of oracular computation can profitably be employed in theorizing about hardware, about brains, and about minds.

Turing introduced oracle machines with the following words [51, pp. 172–173]:

> 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.

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 [50]. (This interpretation is to an extent supported by the fact that in the preceding pages Turing several times uses simply 'machine' where he clearly intends 'computing machine'.) Or perhaps his view might have been 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 might 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.)

In a critique of hypercomputation, Hodges [29] argues that according to Turing, *o*-machines are not literally machines. Hodges quotes Turing's statement that an oracle 'cannot be a machine' and asserts that an *o*-machine cannot therefore be a machine. Hodges does not quote Turing's statement that with 'the help of the oracle we could form a new kind of machine' and his interpretation latches onto the first of these statements while ignoring the second. Nor does he address the fact that Turing did repeatedly refer to *o*-machines as machines. For example [51, pp. 172–173]:

> 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 [50, pp. 231–232].) In any case, Hodges' inference from the statement that an oracle 'cannot be a machine' to the conclusion that *o*-machines are 'only *partly mechanical*' is logically on a par with: 'Ink is not a machine, therefore a Turing machine is only partly mechanical' (machines can

have parts that are not themselves machines). In short, Turing said that oracle machines are machines and there seems no reason not to take him at his word.

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 [50] he 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') [51, 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 [50]. 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 [51, p. 173]. Turing appealed to *o*-machines in discussing the question of the completeness of the 'ordinal logics' that he described [51, 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. There are by now in the literature numerous ways of filling out the idea of an oracle, some more physical in flavour than others.[4] To mention only a few examples, an oracle is in principle realisable by certain classical electrodynamical systems [43,44]; the physical process of equilibriation [22]; an automaton travelling through relativistic spacetime [30,31,39]; a quantum mechanical computer [6,32,33,48,49]; an inter-neural connection [46,47]; and a temporally evolving sequence of Turing machines, representing for example a learning mind (see [18]).

5. Exegetical issues: Turing and Church

5.1. The scope of machines: Hodges on Church on Turing

In his 1937 review of Turing's [50], Church said:

> The author [Turing] proposes as a criterion that an infinite sequence of digits 0 and 1 be 'computable' that it shall be possible to devise a computing machine,

[4] One of the first to speak of oracles in *physical* terms was Davis [20, 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' [20, p. 11]. 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 [20, pp. 11–12, 64].

> occupying a finite space and with working parts of finite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience, certain further restrictions are imposed on the character of the machine, but these are of such a nature as obviously to cause no loss of generality—in particular, a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a kind of Turing machine. It is thus immediately clear that computability, so defined, can be identified with (especially, is no less general than) the notion of effectiveness as it appears in certain mathematical problems … [7, pp. 42–43]

Hodges claims that this passage shows that Church 'equat[ed] the scope of computability with the scope of machines' [29]. Or to put it the other way round, according to Hodges, Church equated the scope of machines with the scope of computability in Turing's 1936 sense. (Hodges also claims in [29] that his own earlier statement, 'Alan had … discovered something almost … miraculous, the idea of a universal machine that could take over the work of *any* machine' [27, p. 109], 'reflected' what Church said in this quotation.)

Hodges [29] maintains that the (alleged) fact that Church 'equat[ed] the scope of computability with the scope of machines' is a strong indication that Turing did so too, given that the two men were in close contact at Princeton from 1936 to 1938. This argument is weak. The historian of mathematics Wilfred Sieg has long maintained that the passage in question shows that Church apparently misunderstood Turing [45, pp. 94–95]. 'Church's apparent misunderstanding [of Turing] is rather common', Sieg noted [45, p. 95].

In fact, however, there is no reason to think that Church was saying what Hodges claims he was saying. Had Church said simply 'devise a machine, occupying a finite space and with working parts of finite size', rather than 'devise a computing machine, occupying a finite space and with working parts of finite size', then Hodges might be on firmer ground. But Church did not say simply 'machine', he said 'computing machine'. And in the usage of the day, a computing machine was a machine that works in accordance with a systematic method or algorithm.

5.2. Hodges on Turing on discrete state machines

In a discussion of Turing's [54] Hodges says:

> Not quite made explicit, but implicit in every statement, is that the operation of a discrete state machine is *computable*. [28, p. 35]

Let us examine carefully Turing's argument in [54] concerning discrete state machines. After giving an example of a simple discrete state machine with a total of three states or configurations, Turing describes the behaviour of the machine by means of a finite table of the sort that would now be called a look-up table, and then says:

> This example is typical of discrete state machines. They can be described by such tables provided they have only a *finite number of possible states*. … Given the table corresponding to a discrete state machine it is possible to predict what it will do. There is no reason why this calculation should not be carried out by means of a digital computer. Provided it could be carried out sufficiently quickly

> the digital computer could mimic the behaviour of any discrete state machine. [54, pp. 440–441] (my italics)

Turing's point appears to be that any discrete state machine whose total number of configurations is finite can be mimicked by a digital computer (and therefore by a Turing machine) since the computer can be given a finite look-up table setting out the behaviour of the machine. It is unlikely that Turing thought the behaviour of every discrete state machine with an unlimited number of configurations is computable.

5.3. The pre- and post-war Turing on the mind

Concerning Turing's [51] Hodges writes:

> the evidence is that at this time [Turing] was open to the idea that in moments of 'intuition' the mind appears to do something outside the scope of the Turing machine. [28, p. 22]

but that:

> in the course of the war Turing dismissed the role for uncomputability in the description of mind, which once he had cautiously explored with the ordinal logics. [28, p. 51]

and:

> by 1945 Turing had come to believe computable operations had sufficient scope to include intelligent behaviour, and had firmly rejected the direction he had followed in studying ordinal logics. [28, p. 30]

What changed Turing's mind, Hodges suggests, was his experience at Bletchley Park:

> My guess is that there was a turning point in about 1941. After a bitter struggle to break U-boat Enigma, Turing could then taste triumph. Machines turned and people carried out mechanical methods unthinkingly, with amazing and unforeseen results. … [I] suggest that it was at this period that [Turing] abandoned the idea that moments of intuition corresponded to uncomputable operations. Instead, he decided, the scope of the computable encompassed … quite enough to include all that human brains did, however creative or original. [28, pp. 28–29]

The mathematician Peter Hilton, Turing's friend and colleague and a leading code-breaker at Bletchley Park, comments as follows (in a letter) on these passages by Hodges:

> I must say that, if Alan Turing's thinking was undergoing so dramatic a change at that time, he concealed the fact very effectively. [5]

There is no textual evidence for the supposed sea-change in Turing's thinking about the mind. What Turing [51] said is perfectly consistent with his post-war views (see [13, p. 45]). Indeed, Turing's later work on the mind, far from representing a rejection of his earlier ideas, appears to be a development of them. [6]

---

[5] Letter from Hilton to Copeland (16 May 2003). Hilton adds: 'I would never have said that we, working on Naval Enigma, "carried out mechanical methods unthinkingly", nor that our results were "amazing and unforeseen".')

[6] See [38] for a similar view.

Turing's wartime letters to Newman give a useful summary of Turing's view at that time of the role of intuition in mathematics. Different Turing machines, Turing said in the letter quoted above, allow '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'. If one selects a 'proof finding machine' for proving a particular theorem, intuition is required in making the selection, just as if one constructed the proof for oneself.

A human mathematician working according to the rules of a fixed logical system is in effect a proof-finding machine. When intuition supplies the mathematician with some new means of proof, he or she becomes a different proof-finding machine, capable of a larger set of proofs. If there is in principle a limit to the ability of human mathematicians to become transformed into successively more powerful proof-finding machines, no such limit has so far been discovered. (The above quotation from Turing's [50] beginning 'Let δ be a sequence' is relevant here.)

Neither in [51] nor in the wartime letters to Newman did Turing attempt to explain the 'activity of the intuition' [51, p. 214]. How does the mathematician manage the uncomputable transformation from one proof-finding machine to another? In 1939 Turing was content to leave this question to one side, making no 'attempt to explain this idea of "intuition" any more explicitly' [51, p. 215]. In his post-war work, on the other hand, Turing had a lot to say that is relevant to this question.

In his post-war writing on mind and intelligence [52–58] the term 'intuition' drops from view and what comes to the fore is the closely related idea of *learning*—in the sense of devising or discovering—new methods of proof. When a human mathematician is confronted by a problem that he or she is unable to solve, the mathematician 'would search around and find new methods of proof' [52, p. 123]. Turing argued forcefully that machines can do this too.

The limits of a proof-finding Turing machine are determined by its table of instructions. The mechanism for acquiring new methods of proof lying beyond this limit must therefore involve the modification of the table of instructions. Turing said in 1947:

> What we want is a machine that can learn from experience. The possibility of letting the machine alter its own instructions provides the mechanism for this. [52, p. 123]

And:

> One can imagine that after the machine had been operating for some time, the instructions would have altered out of all recognition. [52, p. 122]

(Ted Newman, one of the engineers at the National Physical Laboratory who built the Pilot Model ACE (Automatic Computing Engine), remarked that in working on the ACE, Turing's

> particular purpose was to permit the writing of programs that modify programs, not in the simple way now common but rather in the way that people think. [36, p. 12] [7]

Modifying the table of instructions in effect transforms the learning machine into a different Turing machine. So a machine with the ability to learn is able to traverse the

[7] I am grateful to Teresa Numerico for drawing this article to my attention.

space of proof-finding Turing machines. The learning machine successively mutates from one proof-finding Turing machine into another, becoming capable of wider sets of proofs as it searches for and acquires new, more powerful methods of proof.

Turing's discussions of the nature of learning emphasize two points, the importance of the learner's making and correcting mistakes, and the advantages of the involvement of a 'random element' in the learning process, resulting 'in the behaviour of the machine not being by any means completely determined by the experiences to which it was subjected' [56, p. 461]; see also [53–55]). The idea appears to be that the partially random learning machine emulates the 'activity of the intuition' in its walk through the space of proof-finding Turing machines. The trajectory of the learning machine through this space might indeed be uncomputable, in the precise sense that the function on the non-negative integers whose value at i is the ith Turing machine on the trajectory need not be computable by the universal Turing machine.

Acknowledgements

Research on which this article draws was supported in part by University of Canterbury Research Grant No. U6472 and Marsden Grant No. UOC905.

References

[1] A. Ambrose, Finitism in mathematics (I and II), Mind 35 (1935) 186–203; 317–340.
[2] W. Bechtel, R.C. Richardson, Discovering Complexity: Decomposition and Localization as Strategies in Scientific Research, Princeton University Press, Princeton, 1993.
[3] M.S. Burgin, Inductive Turing machines, Soviet Math. Dokl. 270 (1983) 1289–1293.
[4] M.S. Burgin, How we know what technology can do, Comm. ACM 44 (2001) 82–88.
[5] M.S. Burgin, From neural networks to grid automata, in: Proc. IASTED Internat. Conf. Modeling and Simulation, Palm Springs, California, 2003, pp. 307–312.
[6] C.S. Calude, B. Pavlov, Coins, quantum measurements, and Turing's barrier, Quantum Inform. Process. 1 (2002) 107–127.
[7] A. Church, Review, J. Symbolic Logic 2 (1937) 42–43.
[8] A. Church, On the concept of a random sequence, American Math. Soc. Bull. 46 (1940) 130–135.
[9] B.J. Copeland, The Church–Turing thesis, in: E. Zalta (Ed.), Stanford Encyclopedia of Philosophy, http://plato.stanford.edu.
[10] B.J. Copeland, The broad conception of computation, Amer. Behavioral Sci. 40 (1997) 690–716.
[11] B.J. Copeland, Super Turing-machines, Complexity 4 (1998) 30–32.
[12] B.J. Copeland, Even Turing machines can compute uncomputable functions, in: C. Calude, J. Casti, M. Dinneen (Eds.), Unconventional Models of Computation, Springer, London, 1998.
[13] B.J. Copeland (Ed.), A lecture and two radio broadcasts on machine intelligence by Alan Turing, in: K. Furukawa, D. Michie, S. Muggleton (Eds.), Machine Intelligence, Vol. 15, Oxford University Press, Oxford, 1999.
[14] B.J. Copeland, Narrow versus wide mechanism, J. Phil. 96 (2000) 5–32.
[15] B.J. Copeland, Colossus and the dawning of the computer age, in: R. Erskine, M. Smith (Eds.), Action This Day, Bantam Books, London, 2001.
[16] B.J. Copeland, Modern history of computing, in: E. Zalta (Ed.), Stanford Encyclopedia of Philosophy, http://plato.stanford.edu.
[17] B.J. Copeland, Accelerating Turing machines, Minds Mach. 12 (2002) 281–301.
[18] B.J. Copeland, Hypercomputation, Minds Mach. 12 (2002) 461–502.

[19] B.J. Copeland, R. Sylvan, Beyond the universal Turing machine, Austral. J. Phil. 77 (1999) 46–66.
[20] M. Davis, Computability and Unsolvability, McGraw-Hill, New York, 1958.
[21] M. Davis, The myth of hypercomputation (February 2003), in: C. Teuscher (Ed.), Alan Turing: Life and Legacy of a Great Thinker, Springer, Berlin, 2003, to appear.
[22] J. Doyle, What is Church's thesis? An outline, Minds Mach. 12 (2002) 519–520.
[23] R.V. Freyvald, Functions and functionals computable in the limit, Trans. Latvijas Vlasts Univ. Zinatn. Raksti 210 (1974) 6–19.
[24] E.M. Gold, Limiting recursion, J. Symbolic Logic 30 (1965) 28–48.
[25] I. Harvey, T. Bossomaier, Time out of joint: attractors in asynchronous random boolean networks, in: P. Husbands, I. Harvey (Eds.), Fourth European Conf. on Artificial Life, MIT Press, Cambridge, MA, 1997.
[26] J. Hintikka, A. Mutanen, An alternative concept of computability, in: Language, Truth, and Logic in Mathematics, Kluwer, Dordrecht, 1998.
[27] A. Hodges, Alan Turing: The Enigma, Vintage, London, 1992.
[28] A. Hodges, Turing, Phoenix, London, 1997.
[29] A. Hodges, What would Alan Turing have done after 1954? in: C. Teuscher (Ed.), Alan Turing: Life and Legacy of a Great Thinker, Springer, Berlin, 2003, to appear, http://www.turing.org.uk/philosophy/lausanne1.html.
[30] M.L. Hogarth, Does general relativity allow an observer to view an eternity in a finite time? Found. Phys. Lett. 5 (1992) 173–181.
[31] M.L. Hogarth, Non-Turing computers and non-Turing computability, Vol. 1, PSA, 1994.
[32] T.D. Kieu, Quantum hypercomputation, Minds Mach. 12 (2002) 541–561.
[33] A. Komar, Undecidability of macroscopically distinguishable states in quantum field theory, Phys. Rev. Ser. 2 133B (1964) 542–544.
[34] S. Kripke, Wittgenstein on Rules and Private Language: an Elementary Exposition, Blackwell, Oxford, 1982.
[35] G.J. Lokhorst, Why I am not a super-Turing machine, Hypercomputation Workshop, University College, London, 24 May 2000.
[36] E. Newman, Memories of the Pilot Ace, Resurrection 9 (1994) 11–14.
[37] R. Penrose, Shadows of the Mind: A Search for the Missing Science of Consciousness, Oxford University Press, Oxford, 1994.
[38] G. Piccinini, Alan Turing and the mathematical objection, Minds Mach. 13 (2003) 23–48.
[39] I. Pitowsky, The physical Church thesis and physical computational complexity, Iyyun 39 (1990) 81–99.
[40] H. Putnam, Trial and error predicates and the solution of a problem of Mostowski, J. of Symbolic Logic 30 (1965) 49–57.
[41] H. Putnam, Renewing Philosophy, Harvard University Press, Cambridge, MA, 1992.
[42] B.A.W. Russell, The limits of empiricism, Proc. Aristotelian Soc. 36 (1936) 131–150.
[43] B. Scarpellini, Zwei Unentscheitbare Probleme der Analysis [Two undecidable problems of analysis], Z. Math. Logik Grundlagen Math. 9 (1963) 265–289, Minds Mach. 13 (2003) (in English).
[44] B. Scarpellini, Comments on 'Two undecidable problems of analysis', Minds Mach. 13 (2003).
[45] W. Sieg, Mechanical procedures and mathematical experience, in: A. George (Ed.), Mathematics and Mind, Oxford University Press, Oxford, 1994.
[46] H.T. Siegelmann, Neural and super-Turing computing, Minds Mach. 13 (2003) 103–114.
[47] H.T. Siegelmann, E.D. Sontag, Analog computation via neural networks, Theoret. Comput. Sci. 131 (1994) 331–360.
[48] M. Stannett, X-machines and the halting problem: building a super-Turing machine, Formal Aspects Comput. 2 (1990) 331–341.
[49] M. Stannett, Computation and hypercomputation, Minds Mach. 13 (2003) 115–153.
[50] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proc. London Math. Soc., Ser. 2 42 (1936–1937) 230–265.
[51] A.M. Turing, Systems of logic based on ordinals, Proc. London Math. Soc., Ser. 2 45 (1939) 161–228.
[52] A.M. Turing, Lecture to the London Mathematical Society on 20 February 1947, in: B.E. Carpenter, R.W. Doran (Eds.), A.M. Turing's ACE Report of 1946 and Other Papers, MIT Press, Cambridge, MA, 1986.

[53] A.M. Turing, Intelligent Machinery, National Physical Laboratory, London, 1948, in: B. Meltzer, D. Michie (Eds.), Machine Intelligence, Vol. 5, Edinburgh University Press, Edinburgh, 1969, A digital facsimile of the original document is in The Turing Archive for the History of Computing, http://www.AlanTuring.net/intelligent_machinery.
[54] A.M. Turing, Computing machinery and intelligence, Mind 59 (1950) 433–460.
[55] A.M. Turing, Can digital computers think? (1951), in: B.J. Copeland, A lecture and two radio broadcasts on machine intelligence by Alan Turing, in: K. Furukawa, D. Michie, S. Muggleton (Eds.), Machine Intelligence, Vol. 15, Oxford University Press, Oxford, 1999.
[56] A.M. Turing, Intelligent machinery, a heretical theory (c.1951), in: B.J. Copeland, A lecture and two radio broadcasts on machine intelligence by Alan Turing, in: K. Furukawa, D. Michie, S. Muggleton (Eds.), Machine Intelligence, Vol. 15, Oxford University Press, Oxford, 1999.
[57] A.M. Turing et al., Can automatic calculating machines be said to think? (1952), in: B.J. Copeland, A lecture and two radio broadcasts on machine intelligence by Alan Turing, in: K. Furukawa, D. Michie, S. Muggleton (Eds.), Machine Intelligence, Vol. 15, Oxford University Press, Oxford, 1999.
[58] A.M. Turing, Chess, in: B.V. Bowden (Ed.), Faster Than Thought (part of Chapter 25), Sir Isaac Pitman & Sons, London, 1953.
[59] P. Wegner, Why interaction is more powerful than algorithms, Commun. ACM 40 (1997) 80–91.