Accelerating Turing Machines*

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

Abstract. Accelerating Turing machines are Turing machines of a sort able to perform tasks that are commonly regarded as impossible for Turing machines. For example, they can determine whether or not the decimal representation of π contains n consecutive 7s, for any n; solve the Turing-machine halting problem; and decide the predicate calculus. Are accelerating Turing machines, then, logically impossible devices? I argue that they are not. There are implications concerning the nature of effective procedures and the theoretical limits of computability. Contrary to a recent paper by Bringsjord, Bello and Ferrucci, however, the concept of an accelerating Turing machine cannot be used to shove up Searle's Chinese room argument.

Key words: accelerating Turing machine, Chinese room argument, Church–Turing thesis, decision problem, effective procedure, halting problem, hypercomputer, hypercomputation, infinity machine, π-machine, oracle machine, super-task

1. Effective and effective

One might use Turing's own words to define the topic of this volume: An *effective procedure* is 'any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner' (1950, p. 1). Such a human operator is a *computer* in the original sense of the term – a clerk whose task is to calculate in accordance with instructions provided by a supervisor. The clerk works with-nothing but pen and paper. The instructions must not demand insight or ingenuity from the clerk, must lead unambiguously from one step to the next, and must be such that the clerk can complete them in a finite span of work. The Turing machine is a model, idealised in certain respects, of a human computer, and was introduced as such by Turing (1936, p. 231).

In her "Is the Church–Turing Thesis True?" (Cleland, 1993; see also Cleland, 1995), Cleland develops an alternative analysis of the concept of an effective procedure. The present paper tills the same field, but without following Cleland in her rejection of the analysis of effective procedures in terms of Turing machines.

The requirement that the instructions encapsulating an effective procedure be capable of being carried out in a finite span of work is usually expressed by saying that the complete execution of the instructions should demand only a finite number of primitive (or 'atomic' or 'fundamental') *computing steps*. But an alternative account is possible: complete execution of the instructions should demand only a finite *amount of time*. The two accounts are sometimes treated as being the same (e.g., by Hofstadter, 1980, pp. 40–41). However, they are not. In order to mark the

Minds and Machines **12:** 281–301, 2002.
© 2002 *Kluwer Academic Publishers. Printed in the Netherlands.*

distinction without introducing new terminology, procedures satisfying the finite-time account will be said to be 'Effective', always with capital 'E'.

2. The Russell–Blake–Weyl Temporal Patterning

Neither Turing nor Post, in their descriptions of the devices that we now call Turing machines, made much mention of time (Turing, 1936; Post, 1936). They listed the primitive operations (or fundamental processes) that their devices perform – read a square of the tape, write a single symbol on a square of the tape, move one square to the right, and so forth – but they said nothing about the *duration* of each primitive operation. Temporal considerations are not relevant to the functioning of the devices as described, nor to the soundness of the proofs that Turing gave concerning them. Things are very different in the case of the boolean networks of McCulloch and Pitts (1943) and Turing (1948), where the duration of each primitive operation is a critical factor.[1] Turing's boolean networks were synchronised by a central clock and each primitive operation took one 'moment' of clock time (1948, p. 10). When working with Turing machines it is no doubt intuitive to imagine each primitive operation to be similarly fixed in duration (or even to be instantaneous), but this is no part of the original conception. No conditions were placed on the temporal patterning of the sequences of primitive operations.

Bertrand Russell, Ralph Blake and Hermann Weyl independently described one extreme form of temporal patterning. Weyl considered a machine (of unspecified architecture) that is capable of completing

> an infinite sequence of distinct acts of decision within a finite time; say, by supplying the first result after 1/2 minute, the second after another $1\frac{1}{4}$ minute, the third 1/8 minute later than the second, etc. In this way it would be possible ... to achieve a traversal of all natural numbers and thereby a sure yes-or-no decision regarding any existential question about natural numbers. (Weyl, 1927, p. 34; English translation from Weyl, 1949, p. 42.)

It seems that this temporal patterning was first described by Russell, in a lecture given in Boston in 1914. 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 [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. (1936, pp. 143–144.)

Blake, too, argued for the possibility of completing an infinite series of acts in a finite time:

> A process is perfectly conceivable, for example, such that at each stage of the process the addition of the next increment in the series 1/2, 1/4, 1/8, etc., should take just half as long as the addition of the previous increment. But ... then the addition of all the increments each to each shows no sign whatever of taking forever. On the contrary, it becomes evident that it will all be accomplished within a certain definitely limited duration. ... If, e.g., the first act ... takes 1/2 second, the next 1/4 second, etc., the [process] will ... be accomplished in precisely one second. (1926, pp. 650–651)

The human computer may carry out the steps of an Effective procedure in accordance with the Russell–Blake–Weyl temporal patterning (that this is 'medically impossible' is of no more relevance than that real human computers, unlike their idealised counterparts, suffer from wear and tear, run out of paper, and die). Imposing the same temporal patterning upon a Turing machine produces what I have termed an *accelerating* Turing machine (Copeland, 1998b, c). These are Turing machines that perform the second primitive operation called for by the program in half the time taken to perform the first, the third in half the time taken to perform the second, and so on. Let the time taken to perform the first primitive operation called for by the program be one 'moment'. Since

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

is less than 1, an accelerating Turing machine – or human computer – can perform infinitely many primitive operations before two moments of operating time have elapsed. Because accelerating Turing machines are Turing machines (pace Steinhart's treatment in this volume), the restricted quantifiers 'all Turing machines', 'some Turing machines' and 'no Turing machines' have accelerating Turing machines among their range.

Stewart (1991, pp. 664–665) gives a cameo discussion of accelerating Turing machines. Related to accelerating Turing machines are the anti de Sitter machines of Hogarth (1992, 1994) (concerning which see also Earman and Norton, 1993, 1996) and the Zeus machines of Boolos and Jeffrey (1980, pp. l4–15).[2] Also related are the trial-and-error machines of Putnam (1965) and Gold (1965). Hamkins and Lewis (2000) give a mathematical treatment of the computability theory associated with accelerating Turing machines (which they term 'infinite-time Turing machines'). Copeland and Hamkins (in preparation) discuss the physical plausibility of accelerating Turing machines with respect to Newtonian physics, relativistic physics, and quantum theory. Sorensen (1999, Section 6) contains a discussion of accelerating Turing machines based on Copeland (1998b).

The Church–Turing thesis for Effective procedures is:

> Each Effective procedure can be performed by a Turing machine.

Accelerating machines are counted in, of course. So long as 'can be performed by a Turing machine' is taken in what is termed in Section 7 its *external* sense, the

thesis would seem to be true. Even if the thesis is accepted, however, there is scope for disagreement about the extent of Effective procedures, as noted in Section 8. My own view is that the Effective procedures do not extend beyond a proper subset of the procedures that can be carried out by the machine $O_{\mathbb{H}}$ defined in Section 6.

3. π-Machines

To say that a Turing machine computes a number is to say that if the machine is set in motion with its tape blank, it will write out successively the digits of the decimal, or binary, representation of the number, halting when it reaches the last digit if there is one. Where the number is irrational – π, for example – or is a recurring rational, its representation has no last digit, and so each act of writing is followed by further operations culminating in another act of writing. Nevertheless, each digit of the representation is written down by the machine after some finite number of primitive operations has been performed.

Since a Turing machine can be programmed to compute π, an accelerating Turing machine can execute each act of writing that is called for by this program before two moments of operating time have elapsed. That is to say, for every n, the accelerating machine writes down the nth digit of the decimal representation of π within two moments of operating time. Such a machine can be further programmed to indicate whether or not there exist, say, three consecutive 7s in the decimal representation of π (a question famously discussed by Wittgenstein). Such a machine constitutes an Effective procedure for settling Wittgenstein's question.

A number of philosophers have maintained that (to use Ambrose's words) it is 'logically impossible to run through the entire expansion [of π]' (1935, p. 320). Some of these philosophers have even put forward arguments for this view. (Ambrose herself is not among them. She was content to let the claim just quoted rest on the assertion that 'the phrase "to run through the entire expansion" is self-contradictory' (loc.cit.).) In the 1950s and 1960s, there was a vigorous debate over whether such a 'π-machine' and other machines that perform an infinite number of operations in a finite time – collectively termed 'infinity machines' – are logically possible.[3] If it is correct that a π-machine is a logical impossibility, then accelerating Turing machines stand exposed as logically impossible devices. However, there is little doubt that defeat went to those who attempted to defend versions of the position staked out by Ambrose: no inconsistency in the notion of a π-machine was ever demonstrated.

In 1954 Thomson introduced the term 'super-task' for a task whose completion involves carrying out all of an infinite number of subtasks (1954, p. 2). He believed at the time that there are 'reasons for supposing that super-tasks are not possible of performance' (1954, p. 5). He offered the following example in support of his view.

> There are certain reading-lamps that have a button in the base. If the lamp is off and you press the button the lamp goes on, and if the lamp is on and you press

> the button the lamp goes off. So if the lamp was originally off, and you pressed the button an odd number of times, the lamp is on, and if you pressed the button an even number of times the lamp is off. Suppose now that the lamp is off, and I succeed in pressing the button an infinite number of times, perhaps making one jab in one minute, another jab in the next half-minute, and so on, according to Russell's recipe. After I have completed the whole infinite sequence of jabs, i.e., at the end of the two minutes, is the lamp on or off? It seems impossible to answer this question. It cannot be on, because I did not ever turn it on without at once turning it off. It cannot be off, because I did in the first place turn it on, and thereafter I never turned it off without at once turning it on. But the lamp must be either on or off. This is a contradiction. (*Loc. cit.*)

Although Thomson did not say so explicitly, it seems clear that his purpose in introducing the 'Thomson lamp', as it became known, was to illustrate a particular *form* of argument, applicable to any device that allegedly completes a super-task, and turning on the question of what the state of the device could be *after* it completes its task. Concerning the π-machine in particular he made the following remarks. (A parity machine scans a numeral and indicates whether the number represented by it is odd or even.)

> This type of argument refutes also the possibility of a machine built according to Russell's prescription that say writes down in two minutes every integer in the decimal expansion of π. For if such a machine is (logically) possible so presumably is one that records the parity, 0 or 1, of the integers written down by the original machine as it produces them. Suppose the parity-machine has a dial on which either 0 or 1 appears. Then, what appears on the dial after the first machine has run through all the integers in the expansion of π? (*Loc. cit.*)

The latter argument is weak. From the fact that a device composed of two subdevices cannot possibly carry out its advertised function – in the present case, that of coming to rest with the parity of the last digit of the decimal representation of π displayed on its dial – it hardly follows that the subdevices themselves cannot carry out *their* advertised functions. For instance, there can be a machine satisfying the specification 'serves to keep the voltage across wire AB constant and high' and a machine satisfying the specification 'serves to keep the voltage across wire AB constant and low', but there can be no machine that satisfies both these specifications simultaneously. Any attempt to build one – e.g., by connecting the 'high' machine and the 'low' machine in parallel – will inevitably result in a machine that, whatever else it does (burn out, for example), fails to satisfy the conjunctive specification. Even supposing that the π-machine and the (accelerating) parity machine are both logical possibilities, it remains the case that the composite machine described by Thomson is a logical impossibility. There is no entailment from the impossibility of the composite machine to the impossibility of the component π-machine.

The argument concerning the Thomson lamp is also unsuccessful, although for a more subtle reason. In a marvellous reply to Thomson's paper, Benacerraf argued

that 'The lamp is on at the end of the two minutes' and 'The lamp is off at the end of the two minutes' are both logically consistent with the statement that the super-task in question has been carried out (1962, pp. 767–770). (Benacerraf's point was partially anticipated by Blake (1926, pp. 651–652).) The instruction issued to the super-lamplighter is essentially this: Use the button to make the lamp cycle between its two states, never allowing the one state to go unfollowed by the other, the first half-cycle to take one minute, and each subsequent half-cycle to take half as long as its predecessor. Call this the Thomson instruction. Since each change of state brought about by the lamplighter's obeying the Thomson instruction must occur *before* the end of the second minute, nothing about the state of the lamp *at* the end of the second minute or thereafter is logically entailed by the statement that the Thomson instruction has been obeyed. So Thomson cannot derive the contradiction that he said he could derive. His previous conclusion (a few sentences earlier in the quotation), that it seems *impossible to answer* the question 'What is the state of the lamp at the end of the two minutes?', is slightly nearer the mark. Given only the information that the Thomson instruction has been successfully obeyed, it *is* impossible to answer that question. But that is not to say that the question need not have a perfectly good answer (an answer dictated, perhaps, by a sudden failure of the power supply at the end of the second minute, or in general by causes other than the lamplighter's obeying of the instruction).

In a subsequent paper, Thomson graciously acknowledged that Benacerraf had faulted his argument. He wrote:

> I thought that there was some conceptual difficulty about the idea of a lamp having been turned on and off infinitely often, because, roughly speaking, of the question about the state of the lamp immediately afterwards. Unfortunately, I tried to make out that there was a difficulty here by arguing that the lamp could not be in either of its two states. This argument was worthless ... I am now inclined to think that there are no simple knock-down arguments to show that the notion of a completed [super-task] is self-contradictory. (1970, pp. 130–131.)

Nevertheless, Thomson's query as to what state an infinity machine may consistently be supposed to be in *after* it completes its super-task is a good one. As Thomson points out, Benacerraf's successful critique of his earlier arguments hardly relieves the advocate of one or another type of infinity machine of the burden of supplying an answer to this form of question (1970, pp. 133–134). Thomson does not suggest (in the later paper) that the question is always unanswerable, but he does emphasise that any adequate treatment of the subject matter must confront the question squarely.

In 1965 Chihara urged the unintelligibility of a π-machine:

> The difficulty, as I see it, is ... the inconceivability of how the machine could actually finish its super-task. The machine would supposedly print the digits on tape, one after another, while the tape flows through the machine, say from right to left. Hence, at each stage in the calculation, the sequence of digits will extend to the left with the last digit printed being 'at center'. Now when the machine

> completes its task and shuts itself off, we should be able to look at the tape to see what digit was printed last. But if the machine finishes printing all the digits which constitute the decimal expansion of π, no digit can be the last digit printed. And how are we to understand this situation? (1965, p. 80.)

Chihara's specification of the π-machine is simply inconsistent. He requires the machine to print the unending decimal representation of π and yet he also requires it to halt having performed a last printing. It is not surprising that, having described a flatly impossible device, Chihara is able to discern 'something unintelligible' about his 'hypothetical machine' (*loc. cit.*).

Let us see what can be done to render intelligible a device reasonably similar to Chihara's, in that the device prints the decimal of π and then halts. The first point to notice is that this device cannot be a Turing machine. A Turing machine halts when it finds an instruction to do so in its program and, by the nature of a Turing machine, if an instruction to halt is obeyed one may ascertain which instruction was obeyed immediately prior to it, and also which square the scanner was resting on when the prior instruction was obeyed, and then ascertain which instruction was obeyed immediately prior to that one, and so forth. Since this process cannot discover a printing of a digit of π – for that would be a printing of the last digit of π – it follows that either no digit of π was ever printed by the machine or that the machine carried out some further super-task, for example the task of printing infinitely many 0s, in between carrying out the π super-task and halting. But, taking the latter case, exactly the same reasoning can be applied to this second super-task. Once the total task has been completely specified – for example, the machine is to write out the digits of π and halt, or is to write out the digits of π followed by the digits of e and halt – then it can be shown, in the above fashion, that no Turing machine can carry out the task.

The solution of this particular difficulty is to embed a Turing machine in some further structure. The program of the Turing machine effects the printing of the digits of π and contains no instruction to halt. The cessation of operation that Chihara desires is brought about by the larger machine shutting itself and all its components off by some means. For example, the machine may contain a clock and arrangements be made so that the machine disconnects itself from its energy source at the end of the second moment of operation.

Further difficulty is occasioned by Chihara's demand for a tape with a first and a last square and infinitely many squares in between. The tape of a Turing machine – which as Post put it is 'ordinally similar to the series of integers' (Post, 1936, p. 103) – cannot possibly be like that. But the machine need not print successive digits on successive squares. Since the point at issue is the intelligibility of the proposition that the machine carries out an infinite number of *printings* and then halts, we may allow the machine to carry out each of its printings of digits of π on the same square, to be called the π-printing square (a printing involves first deleting any symbol previously inscribed on the square to be printed upon). The auxiliary printings performed by the machine in the course of computing any digit

of π are carried out on a strip of tape to be called the auxiliary strip. (Digits printed on the auxiliary strip during the computations leading to the production of the nth digit of π are overwritten during the computations leading to the production of the $n + 1$*th* digit.)

By hypothesis, the printing operations performed by the machine on the π-printing square have no last member. What, then, is inscribed on the π-printing square after the machine ceases its activity? The lesson of Benacerraf's argument is that the specification of the machine as so far given entails no answer to this question. For an answer to exist, the specification must be extended. Not every way of doing this will produce a specification of a possible machine. For example, if it is set down that the only manner in which information can be introduced onto the π-printing square is by means of a printing operation performed by the Turing machine's scanner, and that information can be lost from the square only during the deleting phase of one of these printing operations, then we may grant that no machine can possibly satisfy the extended specification. But not all ways of extending the specification so as to incorporate an answer to the question at issue will end in impossibility. For example, let it be set down that the information introduced onto the square during an act of printing decays as soon as that printing is completed (rendering the deleting phase of the next printing operation obsolete). One might further require that the information persists undecayed throughout a final 'resting' phase of each printing operation (perhaps the resting phase occupies 10% of the time taken by the entire printing operation). The latency of the auxiliary strip is so arranged that the digits inscribed on it during the computations leading to the printing of the nth digit of π decay as soon as the printing operation that produces the nth digit of π is completed. Once the machine's activity has ceased, the tape bears no traces of any of the printings that were performed. It is blank.

Where the specification for the machine provides for loss of information from the tape, the specification can be satisfied, but if the specification is altered so as to prohibit loss of information, other things remaining the same, then the specification cannot possibly be satisfied. So the protasis of the question 'What *would* have been written on the tape if the information had not decayed?' (understood as elliptical for 'if the machine in question were to satisfy an amended specification in which information loss is prohibited, other things remaining the same') is a supposition to the effect that a logical impossibility be true. The question seeks a reply that is a true counterfactual statement with a logically impossible antecedent. Many spring to mind!

Equally, impossibility is bound to be the fruit of any attempt to extend the specification of the π-machine to a specification for a compound device in which the π-machine operates in conjunction with an accelerating scanner-cum-memory that compensates for the non-retentiveness of the tape. As in the case of the parity machine, two device-specifications that are satisfiable individually may be impossible to satisfy if taken jointly.

So logically consistent answers can be given to Thomsonian questions about the condition of a halting π-machine after it has halted. However, as a specimen of a π-machine, the foregoing is a shade disappointing. It performs its advertised super-task but leaves no record of its printings. Moreover, it is not a Turing machine (as previously discussed). The Turing machine Ω described next leaves its user holding (at the end of the second moment) the end of an infinitely long tape on which each digit of the decimal of π is written.

The Newtonian equations of motion have what are known as 'escape solutions' whereby a body disappears from the universe in consequence of its velocity increasing without bound (Geroch, 1977, p. 82, Earman, 1986, p. 34). For example, let the velocity of a body β increase in accordance with the Russell–Blake–Weyl formula: β takes 1 second to cover the first metre of its trajectory, a 1/2 second to cover the next metre, and so on. β's velocity is always finite but is unbounded. At the end of the second second of motion, β disappears from the universe. Take any Euclidean coordinate system originated on any point of β's trajectory: the axes specify every location in the universe and β is to be found at none of them!

Ω is an accelerating Turing machine programmed to churn out the decimal of π, the digits to be written on successive squares of the tape. Ω's scanner moves in tandem with a tape generator which supplies a further square of tape each time the scanner reaches the end of the strip of tape already generated (perhaps the tape generator is bolted to the scanner). Each square of the tape is of side 1 unit. The user holds the free end of the tape and presses the start button. With each move-right operation called for by the program, the scanner travels one unit further from the user along a linear trajectory. By the end of the second moment the user is holding an endless tape on which can be found each digit of the decimal of π. The answer to the Thomsonian question 'Where is the scanner at that point?' is: Nowhere.

4. Solving the Halting Problem by Clerical Labour

Every Turing machine has a program of instructions 'hard wired' into its scanner. By using some suitable system of coding conventions, the instructions of any given Turing machine can be represented by means of a single (large) binary number. Call this the machine's *program number*.

Before a Turing machine is set in motion some sequence of binary digits may be inscribed on its tape.[4] This is the input, the data upon which the machine is to operate, encoded in binary form. Call this number the machine's *data number*. Naturally the data number will alter from run to run of the machine. One may speak of the machine being set in motion *bearing* such-and-such a data number. (The data number of an initially blank tape is zero.)

The famous halting function H takes pairs of integers as arguments and returns the value 0 or 1. H may be defined as follows, for any pair of integers x and y: $H(x, y) = 1$ if and only if x is the program number of a Turing machine that eventually halts if set in motion bearing data number y; $H(x, y) = 0$ otherwise.

Notice that if the integer x is not a program number of a Turing machine then $H(x, y) = 0$ for every choice of y; and if x is a program number, say of Turing machine t, then $H(x, y) = 0$ if and only if t fails to halt when set in motion bearing y.

A machine able to 'solve the halting problem' can inform us, concerning any given Turing machine, whether or not that machine would halt when set in motion bearing any given data number. The *halting theorem* states that no Turing machine can compute the value of the halting function for each pair of integers x, y. The theorem notwithstanding, an accelerating Turing machine can produce the values of the halting function.

Let t be any Turing machine. A universal Turing machine that is set in motion bearing the data number formed by writing out the digits of t's program number p followed by the digits of t's data number d will simulate t. The universal machine performs every operation that t does, in the same order as t (although interspersed with sequences of operations not performed by t), and halts just in case t does. A machine $\mathbb{H}$ that computes the values of the halting function for all integers x and y will result if one equips an accelerating universal Turing machine (or a human clerk) with a signalling device – a hooter, say – such that the hooter blows when and only when the machine halts.[5] To obtain the value of the halting function for a given machine t, one writes out the digits of t's program number p followed by the digits of t's data number d on $\mathbb{H}$'s tape and sets $\mathbb{H}$ in motion. If the hooter blows within two moments of the start of the simulation then $H(p, d) = 1$, and if it remains quiet then $H(p, d) = 0$.[6] Given any Turing machine bearing any data number, $\mathbb{H}$ can inform us whether or not that machine halts.

The next section describes a modified form of $\mathbb{H}$ which decides the full first-order predicate calculus. $\mathbb{H}$ is not, within the meaning of the act, an Effective procedure, since according to the characterisation of effectiveness (section 1), the human computer has no access to any machinery other than a pen, not even a hooter. No matter. Section 7 describes a minor variant of $\mathbb{H}$, $\mathbb{H}_T$, which is an Effective procedure for solving the halting problem, and another variant, $\mathbb{D}_T$, which is an Effective procedure for deciding the predicate calculus. $\mathbb{H}_T$ and $\mathbb{D}_T$ are Turing machines. There is, of course, an air of paradox about this, to be addressed in Section 7.

5. Deciding the Predicate Calculus

Church explains the *Entscheidungsproblem*, or decision problem, as follows:

> By the Entscheidungsproblem of a system of symbolic logic is here understood the problem to find an effective method by which, given any expression Q in the notation of the system, it can be determined whether or not Q is provable in the system. (Church, 1936, p. 41.)

Famously, Turing (1936) argued that there is no such effective method in the case of the predicate calculus, proving that no Turing machine can compute the function

D(x) whose value is 1 if formula x is a theorem of the calculus and 0 if x is not a theorem. Nevertheless, an accelerating machine is able to compute this function.

A Turing machine can be programmed to derive theorems of the first order predicate calculus. The machine is given a set of axioms for the calculus (e.g., the Bernays–Hilbert–Ackermann axioms (Hilbert and Ackermann, 1928)) and is programmed to apply the rules of the calculus – such as modus ponens – iteratively, both to axioms and to formulae already derived. The simplest form of such a program works blind: the rule-applications are not selected by the machine in such a way as to lead to some particular theorem but are carried out according to an inflexible and exhaustive regime specified by the programmer. Nevertheless, each and every rule-application produces a theorem of the calculus, and if the machine runs on forever, each theorem of the calculus is produced sooner or later.

As Stewart remarks, an accelerating machine 'could prove all possible theorems in a finite period of time, by pursuing all logically valid chains of deduction from the axioms' (1991, p. 665). Let an accelerating Turing machine equipped with a hooter be given a program of the sort just described for churning out the theorems of the predicate calculus. Each time the accelerating machine derives a formula it compares the formula to a formula x of the predicate calculus that was written on its tape by the user before setting the machine in motion. The machine hoots if and only if it proves a formula that matches x symbol-for-symbol. Thus, if a hoot comes, x is a theorem of the calculus, and if no hoot comes by the end of the second moment of operating time, x is not a theorem.

6. Oracle Machines with Accelerating Components

Turing introduced the concept of an *O-machine* in Section 4 of his PhD thesis (Turing, 1938). An O-machine is a Turing machine equipped with an additional device – a black box – that, when presented with arguments of some non Turing-machine-computable function, returns the corresponding values of this function. For example, the black box may respond to an input of a pair of integers, x and y, with the corresponding value of the halting function, H(x,y), for every x and y. Turing called such black boxes 'oracles' and described O-machines as 'a new kind of machine'. As in the case of an ordinary Turing machine, the behaviour of an O-machine is determined by a table of instructions (or program). The table provides an exhaustive specification of which primitive operations the machine is to perform when it is in such-and-such state and has such-and-such symbol in its scanner.

The tables of the two sorts of machine differ only in the following respect: an O-machine's table may contain instructions of the form '*TRANSFORM* #*'.[7] '#*' refers to some particular string of symbols on the machine's tape, the beginning of the string being marked by the presence on the tape of a reserved symbol '#' and the end of the string being marked by a reserved symbol '*'. The instruction causes the portion of tape so marked to be presented to the black box. The symbols on this portion of tape constitute a specification of an argument of whatever function

it is that the box generates (or of a series of n arguments in case the function is n-ary). The box replaces the symbols on the tape with a specification of the corresponding value of the function. The *transform* operation performed by the oracle is, in Turing's expression, one of the 'fundamental processes' of the machine (*op. cit.*, p. 173).[8] (He gave no indication of how this fundamental process might conceivably be carried out, saying only that an oracle works by 'unspecified means' and that 'we shall not go any further into the nature of [an] oracle' (*op. cit.*, pp. 172–173).)

In summary: An O-machine consists of a 'head' and a paper tape of unbounded length bearing discrete symbols. The 'head' contains various sub-devices, at least one of which is an oracle. The O-machine manipulates symbols on the tape in a serial, step-by-step manner in accordance with the rules specified by its program. Every primitive operation of an O-machine that is not among the primitive operations of a Turing machine is a formal operation on discrete symbols (in essence an operation that replaces a binary string with 1 or 0).

One way of specifying a mechanism by means of which an oracle can do its work is in terms of $\mathbb{H}$. Let $O_{\mathbb{H}}$ be an O-machine of which $\mathbb{H}$ itself serves as oracle. $O_{\mathbb{H}}$ consists of a (non-accelerating) universal Turing machine $\mathbb{T}$ operating in conjunction with $\mathbb{H}$, and including a clock, and resources for delivering digits from $\mathbb{T}$'s tape to $\mathbb{H}$'s tape, for setting $\mathbb{H}$ in motion at the appropriate time, and for detecting and recording the presence or absence of a signal within two moments of $\mathbb{H}$'s work commencing. $O_{\mathbb{H}}$ will be described as being set in motion bearing data number x just in case $\mathbb{T}$ is so set in motion, and to halt (with 1 [0] under its head) just in case $\mathbb{T}$ halts (with 1 [0] under its scanner). In case $O_{\mathbb{H}}$'s computation requires more than one 'call to the oracle', $O_{\mathbb{H}}$ has resources to create fresh copies of $\mathbb{H}$ ad libitum (the equivalent of a potential infinity of accelerating human clerks). In this way, the Thomsonian question 'How is $\mathbb{H}$'s scanner to be brought back to the starting position again?' is avoided. (In my 2000 paper I describe some ways of realising an oracle that do not involve the use of accelerating components, e.g., by means of a device capable of storing a real number; see also Copeland and Sylvan, 1999.)

$O_{\mathbb{H}}$ is able to compute the halting function and all 'uncomputable' functions reducible to the halting function. There is certainly no paradox here. $O_{\mathbb{H}}$ is not a Turing machine. $O_{\mathbb{H}}$ consists of two Turing machines plus additional equipment (most conspicuously the clock and signalling equipment): $O_{\mathbb{H}}$ is *more* than a Turing machine. The same is true of $\mathbb{H}$: $\mathbb{H}$ is a Turing machine plus a signalling device. $O_{\mathbb{H}}$'s program of instructions is not a Turing machine program. Since $O_{\mathbb{H}}$ is not a Turing machine, the fact that $O_{\mathbb{H}}$ computes the halting function cannot contradict the halting theorem.

But might not the considerations used in the proof that a Turing machine cannot compute the halting function also apply to $O_{\mathbb{H}}$, enabling one to infer that $O_{\mathbb{H}}$ is a logical impossibility?

The proof of the halting theorem proceeds by reductio ad absurdum. One assumes that there *is* a Turing machine h that computes the halting function. The

argument may proceed via the introduction of a further machine, h_2, derived from h by two simple modifications (see for example Minsky, 1967, pp. 148–149). The first modification produces a machine h_1 which, when set in motion bearing the same data number as h, halts with 0 under its scanner if and only if h does so, but which never halts with 1 under its scanner. The modification consists of adding some instructions to h's program in order to make the scanner shuffle endlessly back and forth between some pair of adjacent squares of the tape in the case where it would otherwise have halted with 1 beneath it. A second modification produces h_2 from h_1. h_2 is identical to h_1 except that h_2 first writes out a copy of its data number (beginning on the square immediately following the last digit of the initial occurrence of the data number). Thereafter h_2 behaves exactly as h_1. So if h_2 is set in motion bearing the data number m, then once the copying phase is completed, h_2 will behave exactly as h_1 behaves when set in motion bearing the data number $m\widehat{\ }m$ ($x\widehat{\ }y$ – x concatenated with y – is the number formed by first writing out the digits of x and then the digits of y).

To set h in motion bearing the data number $n\widehat{\ }n$ formed from the program number n of some Turing machine t is to ask h whether or not t will halt if t is set in motion bearing its own program number as data number. Suppose, then, that we were to set h in motion bearing the data number $k\widehat{\ }k$, k being h's own program number. Doing so would tie a knot of self-reference every bit as tangled as those exhibited in the more familiar semantic paradoxes. The point of the construction leading to h_2 is to bring the contradiction contained in this knot right out into the open. Writing p for h_2's program number and writing $h_2 \downarrow p$ to mean that h_2 halts when set in motion bearing p, each of the following is entailed:

$$h_2 \downarrow p \leftrightarrow H(p, p) = 0 \qquad H(p, p) = 0 \leftrightarrow \neg h_2 \downarrow p.$$

If h exists, then h_2 exists; but h_2 exists on pain of contradiction.

It is the fact that $O_{\mathbb{H}}$'s program number, r say, differs from the program number of each Turing machine that stalls this train of reasoning as applied to $O_{\mathbb{H}}$. The halting behaviour of h and its derivatives bears messages concerning the halting behaviour of Turing machines. Thus the means of tying the self-referential knot and producing the contradiction. The halting behaviour of $O_{\mathbb{H}}$ (and suitable derivatives) also bears such messages about Turing machines, but since $O_{\mathbb{H}}$ is not a Turing machine, these messages say nothing about $O_{\mathbb{H}}$ itself. Investigating a machine $\mathbb{H}_2$ that bears the same relationship to $O_{\mathbb{H}}$ as h_2 bears to h will make these considerations concrete. Let $O_{\mathbb{H}}$ be arranged so that its halting behaviour when set in motion bearing a data number $x\widehat{\ }y$ (which is to say, the halting behaviour of its component machine $\mathbb{T}$ when set in motion bearing this data number) is as follows: $O_{\mathbb{H}}$ halts with 1 under its head just in case the oracle $\mathbb{H}$ delivers a signal when fed $x\widehat{\ }y$ and $O_{\mathbb{H}}$ halts with 0 under its head just in case the oracle delivers no signal when fed $x\widehat{\ }y$. Writing '$m \downarrow 0n$' ['$m \downarrow 1n$'] to mean 'machine m halts with 0[1] under its head when set in motion bearing data number n':

$$\text{for every } x, y : O_{\mathbb{H}} \downarrow 0x\widehat{\ }y \leftrightarrow H(x, y) = 0 \;\&\; O_{\mathbb{H}} \downarrow 1x\widehat{\ }y \leftrightarrow H(x, y) = 1. \qquad (1)$$

$\mathbb{H}_1$ is obtained from $O_{\mathbb{H}}$ by exactly the moves that yield h_1 from h. So:

$$\text{for every } x : \mathbb{H}_1 \downarrow x \leftrightarrow O_{\mathbb{H}} \downarrow 0x. \tag{2}$$

$\mathbb{H}_1$ is modified to produce $\mathbb{H}_2$ in just the way that h_1 is modified to produce h_2. So:

$$\text{for every } x : \mathbb{H}_2 \downarrow x \leftrightarrow \mathbb{H}_1 \downarrow x\widehat{x}. \tag{3}$$

As with h_2, let us consider the effect of setting $\mathbb{H}_2$ in motion bearing its own program number, r, as data number:

$$\mathbb{H}_2 \downarrow r \leftrightarrow \mathbb{H}_1 \downarrow r\widehat{r} \qquad \text{(from 3)} \tag{4}$$
$$\mathbb{H}_1 \downarrow r\widehat{r} \leftrightarrow O_{\mathbb{H}} \downarrow 0r\widehat{r} \qquad \text{(from 2)} \tag{5}$$
$$O_{\mathbb{H}} \downarrow 0r\widehat{r} \leftrightarrow H(r, r) = 0 \qquad \text{(from 1)} \tag{6}$$
$$\mathbb{H}_2 \downarrow r \leftrightarrow H(r, r) = 0 \qquad \text{(4-6, transitivity).} \tag{7}$$

Since r is not the program number of a Turing machine, $H(r, r)$ is in fact 0. Thus one can conclude that $\mathbb{H}_2$ does halt if set in motion bearing its own program number as data number.

The inference (above) to:

$$H(p, p) = 0 \leftrightarrow \neg h_2 \downarrow p$$

requires the knowledge that p (h_2's program number) is the program number of a Turing machine (for it is only if this is so that $H(p, p)$ being 0 entails that the Turing machine so numbered does not halt when fed p). It is exactly this inference that is blocked in the case of $\mathbb{H}_2$.

O-machines form an infinite hierarchy. At the bottom are the *first-order* O-machines: the O-machines whose only oracle produces the values of the halting function $H(x, y)$. The halting function for first-order O-machines, $H_1(x, y)$, is defined as follows (for all integers x and y): $H_1(x, y) = 1$ if and only if x is the program number of a first-order O-machine that halts if set in motion bearing data number y; $H_1(x, y) = 0$ otherwise. *Second-order* O-machines have an additional oracle, producing the values of $H_1(x, y)$. That $O_{\mathbb{H}}$ lies right at the bottom of this hierarchy will be important in Section 8.

Perhaps the biological brain – abstracted out from sources of inessential boundedness, such as mortality – is an O-machine (although doubtless not one with accelerating components). Elsewhere I dub this view *wide mechanism* (Copeland, 2000).

7. Internal and External Computability

$\mathbb{H}$ is not a Turing machine. However, as this section explains, $\mathbb{H}$ can be mimicked by a Turing machine pure and simple, $\mathbb{H}_T$. The same is true in the case of the accelerating machine for deciding the predicate calculus. It might be thought that the

negations of the halting theorem and Turing's undecidability theorem are thereby entailed, yielding a logically rigorous demonstration that – just as those in the Ambrose camp always suspected – a contradiction lurks in the proposition that infinitely many operations can be performed in a finite time. The thought, however, is mistaken.

As with $\mathbb{H}$, $\mathbb{H}_T$ is set in motion bearing the concatenation of a pair of integers. $\mathbb{H}_T$'s first actions are to position the scanner over the first square to the left of the input string – called the designated square – and print 0 there. The remainder of $\mathbb{H}_T$'s program is identical to $\mathbb{H}$'s except that the instruction to blow the hooter is replaced by a block of instructions that causes the scanner to move back to the designated square and change the 0 to 1 before halting. $\mathbb{H}$ computes the halting function and $\mathbb{H}_T$ mimics $\mathbb{H}$ exactly (save for writing 1 on the designated square instead of hooting): $\mathbb{H}_T$ – a Turing machine fair and square – computes the halting function. For any x and y, if $\mathbb{H}_T$ is set in motion bearing the data number $x\widehat{\ }y$, the value H(x, y) can be read from the designated square at the end of the second moment of operating time. But according to the halting theorem, the halting function is not Turing-machine-computable. A blatant contradiction, it might be said.

At bottom, the reason that the contradiction just touted cannot validly be obtained is that the halting theorem is in fact a somewhat weaker proposition than is sometimes supposed. It is time to be more precise in stating the halting theorem.

Standardly, to say that a function f(x) each of whose values is 0 or 1 is Turing-machine-computable is to say that there is a Turing machine which, if set in motion bearing x as data number (for any x in the function's domain), will eventually halt with its scanner resting on the corresponding value of the function. (Similarly for functions of more than one argument and for functions whose values include integers other than 0 and 1. In the latter case, it is stipulated that the scanner should halt resting on, say, the last digit of the function's value.) The halting theorem is this: *in the sense just given* the halting function is not Turing-machine-computable. (Only this much was proved by the reductio described earlier, for it is crucial to the construction employed in the reductio that the machine h should halt with the value of the function, 0 or 1, beneath its scanner. Unless this is so, the recipe for obtaining h_1 from h is inapplicable.)

One must distinguish between two senses in which a function may be said to be computable, which I term the *internal* sense and the *external* sense.[9] A function is computable by a machine in the internal sense just in case the machine can produce values from arguments (for all arguments in the domain), halting once any value has been produced, where what counts as halting 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. (This condition on the manner of production of values of the function will be referred to as the internalist condition.) Numerous behaviours on the part of a machine can count as halting, for example complete cessation of activity, or emitting a hoot, or writing any sequence of digits in a certain location.

A function is computable by a machine in the *external* sense just in case the machine can produce values from arguments (for all arguments in the domain), displaying each value at a designated location some pre-specified number of moments after the corresponding argument is presented. The machine may or may not halt once a value has been displayed. For example, it is in the external sense that a given function may be computable by a logic circuit (one of Turing's own boolean networks, for instance). The value of the function is displayed at some designated node – the output node – n moments after the argument is presented at the input nodes (mutatis mutandis for functions of more than one argument and functions whose values require more than one binary node for their expression). Before and after that critical time, the activity of the output node may afford no clue as to the desired value. (n, which is a constant for that circuit and that function, is known as the *delay time*.)

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 prespecified times. The same is true of neural networks. A particular network may compute the values of a certain function in the external sense even though the network never stabilises (a network stabilises, or 'halts', if and only if after some point there is no further change in the activity level of any of its units).

Both $\mathbb{H}$ and $\mathbb{H}_T$ compute the halting function in the external sense, but not in the internal sense, and $O_{\mathbb{H}}$ computes the function in the internal sense. Any machine that computes a function in the external sense can always be converted into one that computes the function in the internal sense by the addition of some additional equipment. Of course, adding the extra equipment may result in a machine not of the same type.

The halting theorem speaks only of computability by Turing machine in the internal sense, not of computability in the external sense. This is what was meant by the earlier statement that the halting theorem is weaker than is sometimes supposed. If the internalist condition is lifted, a Turing machine will compute functions that are not Turing-machine-computable when the internalist condition is in place. A Turing machine liberated from the internalist condition is an example of what I have elsewhere termed a *hypercomputer* (Copeland and Proudfoot, 1999).

A Turing machine $\mathbb{D}_T$, similar to $\mathbb{H}_T$, decides the predicate calculus. The user inscribes a formula X of the predicate calculus on $\mathbb{D}_T$'s tape, leaving the leftmost square of the tape (the designated square) blank, sets $\mathbb{D}_T$ in motion, and at the end of the second moment of operating time finds either 1 or 0 inscribed on the designated square, 1 indicating that X is a theorem, 0 indicating that it is not. There is no contradiction with the result of Turing's mentioned earlier, for Turing proved that the function D(x) is not Turing-machine computable in the internal sense.

Each of $\mathbb{H}_T$ and $\mathbb{D}_T$ constitutes an Effective procedure. Actual human computers, who computed only in the internal sense, would at the end of their labour

screw the tops back on their pens and hand their results to a supervisor. An accelerating human computer has no such ceremony to look forward to, and the supervisor must watch the clock, awaiting a signal.

8. The Chinese Room

Searle has repeatedly emphasised that the fact that computers have 'no more than just formal symbols' entails that programs 'cannot constitute the mind' and that this entailment is demonstrated by the Chinese room argument (1989, p. 33, 1992, p. 200).

> The whole point of the [Chinese room] example was to argue that symbol manipulation by itself couldn't be sufficient for understanding Chinese. (1980, p. 419.)

> [F]ormal syntax ... does not by itself guarantee the presence of mental contents. I showed this a decade ago in the Chinese room argument. (1992, p. 200.)

As I argue in my 1998a paper, O-machines point up the fact that the concept of a programmed machine whose activity consists of the manipulation of formal symbols is *more general* than the restricted notion of formal symbol-manipulation targetted in the Chinese room argument. The Chinese room argument depends upon the occupant of the room – a human clerk working by rote and unaided by machinery; call him or her Clerk – being able to carry out by hand each operation that the program in question calls for (or in one version of the argument, to carry them out in his or her head). Yet an O-machine's program may call for fundamental symbol-manipulating processes that rote-worker Clerk is incapable of carrying out. In such a case, there is no possibility of Searle's Chinese room argument being deployed successfully against the functionalist hypothesis that the brain instantiates an O-machine – a hypothesis which Searle will presumably find as 'antibiological' (1990, p. 23) as other functionalisms. If there is a general implication from 'is defined purely formally or syntactically' to 'is neither constitutive of nor sufficient for mind', it is not one that could possibly be established by the Chinese room argument.

Bringsjord, Bello and Ferrucci (2001) attempt to defend Searle against my objection. Clerk (or Searle) can hand-work the program of an O-machine, they say, by accelerating in the Russell–Blake–Weyl fashion.

> Jack Copeland ... has recently argued that oracle machines ... are immune to Searle's (1980) CRA-based claim that mere symbol manipulation is insufficient for genuine mentation ... Unfortunately, oracle machines are *not* immune to the Chinese room ... It's easy to imagine that Searle in the Chinese Room manipulates symbols in order to parallel the operation of a Zeus machine [a machine operating in accordance with Russell–Blake–Weyl] ... The only difference is that in (what we might call) the 'Zeus room,' Searle works much faster. But this speed-up makes no difference with respect to true understanding.

The problem with this effort to shore up Searle's argument is that it leaves what is supposed to go on in the 'Zeus room' radically under-described. Bringsjord, Bello and Ferrucci (2001) owe us answers to some difficult Thomsonian questions. How is it with Clerk after the period of acceleration? Suppose the program calls for $\mathbb{H}$ to run twice, with different inputs, or even infinitely many times – how is Clerk supposed to manage this? It is not certainly not enough that Clerk carry out the simulation of $\mathbb{H}$ and (in the manner of Section 3) sail out of the universe, a hoot, if there is one, returning to the waiting crowd. One way or another, Clerk must *be there* in order to participate in the necessary facts. If the Chinese room argument is not to be subverted by the new twist to the story, it must be a fact about Clerk that, having carried out all the symbol-manipulations called for by the program, he or she does not understand Chinese. Moreover, this fact must be epistemologically accessible to us.

Thomsonian difficulties multiply where the machine that Clerk is supposed to mimic lies not at the very bottom of the infinite hierarchy mentioned earlier, as $O_{\mathbb{H}}$ does, but higher up. Bringsjord, Bello and Ferrucci (2001) do not mention this case at all, yet it is crucial to their attempt to defend the 'CRA-based claim that mere symbol manipulation is insufficient for genuine mentation'. They must demonstrate – to good Thomsonian standards – that Clerk is able to simulate not only $O_{\mathbb{H}}$ but *every* programmed symbol-manipulator in the ascending hierarchy.

The stakes are high. If Bringsjord, Bello and Ferrucci (2001) can meet this challenge, they will show that the extent of Effective procedures is very considerable – greater, perhaps, than anybody ever dreamed.

Notes

*Research on which this article draws was supported in part by University of Canterbury Research Grant no. U6271. Thanks to David Armstrong, Chris Bullsmith, Keith Campbell, Philip Catton, Peter Farleigh, Diane Proudfoot and Neil Tennant for valuable comments and discussion.

[1] 'Turing's boolean networks and his connectionist project involving them are described in Copeland and Proudfoot (1996, 1999).

[2] The term 'anti de Sitter machine' is from Copeland and Sylvan (1999). Boolos and Jeffrey envisage Zeus being able to act so as to exhibit (what is here called) the Russell–Blake–Weyl temporal patterning (1980, p. 14). By an extension of terminology (which Boolos and Jeffrey do not make) a Zeus *machine* is any machine exhibiting the Russell–Blake–Weyl temporal patterning. All accelerating Turing machines are Zeus machines, but not vice versa. For example, an O-machine that exhibits the Russell-Blake-Weyl patterning – such as the machine $O_{\mathbb{H}}$ of Section 4 – is a Zeus machine but is not a Turing machine.

[3] See, for example, Benacerraf (1962), Black (1951), Chihara (1965), Grünbaum (1968), Hinton and Martin (1954), Taylor (1951), Thomson (1954), (1970) and Watling (1952).

[4] Each Turing machine is equivalent to a Turing machine employing the binary alphabet.

[5] Turing and his colleagues enjoyed the possibilities afforded by the hooter of the Manchester Mark I computer (the world's first fully electronic stored-program digital computer). Turing's programming manual for the Manchester machine describes the hooter as producing 'a steady note, rich in harmonics' (1950, p. 24). One of the machine's instructions would send a single pulse to the hooter; a train of pulses, timed correctly, would produce a note. Turing displays a loop of two instructions

producing middle C, and a loop of three instructions 'which gives a slightly louder hoot a fifth lower in frequency' (ibid.). The first program of any significant size to run on the machine – written by Strachey at Turing's behest – brought its activity to a close by playing the British National Anthem on the hooter.

[6]Since, by definition, H(x,y) is 0 whenever x is *not* the program number of a Turing machine, $\mathbb{H}$ must not halt and blow its hooter in this case. This is achieved by adding some instructions that drive $\mathbb{H}$ into an infinitely repeated loop if it determines x not to be the program number of some Turing machine.

[7]For ease of exposition, the present account departs from Turing's own in various matters of detail.

[8]In Turing's original exposition, these new fundamental processes produce the values only of π_2^0 functions. In the subsequent technical literature, the notion of an O-machine has been widened to include fundamental processes that produce values of *any* function on the integers that is not Turing-machine-computable. I employ this extended notion here.

[9]The distinction is from Copeland (1998b).

References

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

Benacerraf, P. (1962), 'Tasks, Super-Tasks, and the Modern Eleatics', *Journal of Philosophy* 59, pp. 765–784.

Black, M. (1951), 'Achilles and the Tortoise', *Analysis* 11, pp. 91–101.

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

Boolos, G.S., Jeffrey, R.C. (1980), *Computability and Logic*, 2nd edition, Cambridge: Cambridge University Press.

Bringsjord, S., Bello, P. and Ferrucci, D. (2001), 'Creativity, the Turing Test, and the (Better) Lovelace Test', *Minds and Machines* 11, pp. 3–27.

Chihara, C.S. (1965), 'On the Possibility of Completing an Infinite Process', *Philosophical Review* 74, pp. 74–87.

Church, A. (1936), 'A Note on the Entscheidungsproblem', *Journal of Symbolic Logic* 1, pp. 40–41.

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

Cleland, C.E. (1995), 'Effective Procedures and Computable Functions', *Minds and Machines* 5, pp. 9–23.

Copeland, B.J. (1997), '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), 'Even Turing Machines Can Compute Uncomputable Functions', in C. Calude, J. Casti, and M. Dinneen, eds., *Unconventional Models of Computation*, London: Springer-Verlag, pp. 150–164.

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

Copeland, BJ. (2000), 'Narrow Versus Wide Mechanism', *Journal of Philosophy* 96, pp. 5–32.

Copeland, B.J. and Hamkins, J.D. (in preparation), 'Infinitely Fast Computation'.

Copeland, B.J. and Proudfoot, D. (1996), 'On Alan Turing's Anticipation of Connectionism', *Synthese* 108: pp. 361–377.

Copeland, B.J. and Proudfoot, D. (1999), 'Alan Turing's Forgotten Ideas in Computer Science', *Scientific American* 280 (April), pp. 76–81.

Copeland, B.J. and Sylvan, R. (1999), 'Beyond the Universal Turing Machine', *Australasian Journal of Philosophy* 77, pp. 46–66.

Earman, J. (1986), *A Primer on Determinism*, Dordrecht: Reidel.

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.

Geroch, R. (1977), 'Prediction in General Relativity', in J. Earman, C. Glymour and J. Stachel, eds., *Foundations of Space-Time Theories*, Minnesota Studies in the Philosophy of Science, 8, Minneapolis: University of Minnesota Press.

Gold, E.M. (1965), 'Limiting Recursion', *Journal of Symbolic Logic* 30, pp. 28–48.

Grünbaum, A. (1968), *Modern Science and Zeno's Paradoxes*, London: Allen and Unwin.

Hamkins, J.D. and Lewis, A. (2000), 'Infinite Time Turing Machines', *Journal of Symbolic Logic* 65, pp. 567–604.

Hilbert, D. and Ackermann, W. (1928), *Grundziige der Theoretischen Logik*, Berlin: Springer.

Hinton, J.M and Martin, C.B. (1954), 'Achilles and the Tortoise', *Analysis* 14, pp. 56–68.

Hofstadter, D.R. (1980), *Gödel, Escher, Bach: An Eternal Golden Braid*, Harmondsworth: Penguin.

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.

McCulloch, W.S., and Pitts, W. (1943), 'A Logical Calculus of the Ideas Immanent in Nervous Activity', *Bulletin of Mathematical Biophysics* 5, pp. 115–33.

Minsky, M.L. (1967), *Computation: Finite and Infinite Machines*, Englewood Cliffs, NJ.: Prentice-Hall.

Post, E.L. (1936), 'Finite Combinatory Processes – Formulation 1', *Journal of Symbolic Logic* 1, pp. 103–105.

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

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.

Searle, J. (1980), 'Minds, Brains, and Programs', *Behavioral and Brain Sciences* 3, pp. 417–424, 450–456.

Searle, J. (1989), *Minds, Brains and Science*, London: Penguin.

Searle, J. (1990), 'Is the Brain's Mind a Computer Program?' *Scientific American* 262(1), pp. 20–25.

Searle, J. (1992), *The Rediscovery of the Mind*, Cambridge, MA: MIT Press.

Sorensen, R. (1999), 'Mirror Notation: Symbol Manipulation without Inscription Manipulation', *Journal of Philosophical Logic* 28, pp. 141–164.

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

Taylor, R. (1951), 'Mr. Black on Temporal Paradoxes', *Analysis* 12, pp. 38–44.

Thomson, J.F. (1954), 'Tasks and Super-Tasks', *Analysis* 15, pp. 1–13.

Thomson, J.F. (1970), 'Comments on Professor Benacerraf's Paper', in W.C. Salmon, ed., *Zeno's Paradoxes*, Indianapolis: Bobbs-Merrill.

Turing, A.M. (1936), 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society*, Series 2, 42 (1936–37), 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. (1948), 'Intelligent Machinery', National Physical Laboratory Report, 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/intelligent_machinery>.

Turing, A.M. (1950), 'Programmers' Handbook for Manchester Electronic Computer', University of Manchester Computing Laboratory. A digital facsimile of the original may be viewed in The

Turing Archive for the History of Computing document.
<http://www.AlanTuring.net/programmers_handbook>.
Watling, J. (1952), 'The Sum of an Infinite Series', *Analysis* 13, pp. 39–46.
Weyl, H. (1927), *Philosophie der Mathematik und Naturwissenschaft*, Munich: R. Oldenbourg.
Weyl, H. (1949), *Philosophy of Mathematics and Natural Science*, Princeton: Princeton University Press.