Australasian Journal of Philosophy

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/rajp20

Beyond the universal Turing machine

B. Jack Copeland [a] & Richard Sylvan [a]

[a] University of Canterbury

Available online: 02 Jun 2006

To cite this article: B. Jack Copeland & Richard Sylvan (1999): Beyond the universal Turing machine, Australasian Journal of Philosophy, 77:1, 46-66

To link to this article: http://dx.doi.org/10.1080/00048409912348801

Australasian Journal of Philosophy
Vol. 77, No 1, pp. 46–67; March 1999

BEYOND THE UNIVERSAL TURING MACHINE

B. Jack Copeland and Richard Sylvan

I. Introduction

Two of our heresies—in the dictionary sense of 'opinions contrary to the accepted doctrine on any subject'—are these.[1]

Proposition 1. The so-called Church-Turing thesis is false.

The so-called Church-Turing thesis purports to draw a borderline between computability and noncomputability and is, it seems, pretty well universally accepted among computer scientists, cognitive scientists, and philosophers of mind. In point of fact neither Turing nor Church endorses, or even states, this thesis. *Their* theses, proved equivalent by Turing, concerned the functions that are in principle computable by an *idealised human being unaided by machinery.* Careful authors do use the term 'Church-Turing thesis' to refer to one or other of the various equivalent forms of the theses that Church and Turing themselves put forward. Proposition 1. concerns not *that* equivalence class of theses but a claim lying outside the class and widely but improperly termed 'the Church-Turing thesis'. We distinguish this thesis from members of the equivalence class by the use of riders such as 'so-called'. The so-called Church-Turing thesis is the claim that the class of well-defined computations is exhausted by the computations that can be carried out by Turing machines.

Proposition 2. Computability is a relative notion, not an absolute one.

There is no such thing as *the* class of well-defined computations. The extent of the computable functions is *resource*-relative. The richer the resources that are made available, the greater the extent of the computable. For example, if the resources run to a device (a 'black box', perhaps) that returns values of the halting function, then the classical first-order predicate calculus is decidable. The notion of relative computability is a familiar one in generalised recursion theory, where a function f is said to be computable relative to a set X just in case f is computable by a Turing machine with access to an additional resource that supplies on demand answers to queries of the form 'Is . . . in X?' We maintain that relative computation really is computation. There is no essential difference in kind between the machines that figure in the definition of the classicist's supposedly absolute set of computable functions, namely Turing machines with exactly

[1] Richard Sylvan did not live to see this paper written. He contributed a draft of section VIII and fragments of sections I, II and IX. Our collaboration began in August 1994, when Richard became interested in my descriptions of machines forbidden by the so-called Church-Turing thesis, and I in his ideas concerning the extent to which one's notion of computability is relative to one's logic (the topic of Sylvan and Copeland 199-).

the primitive capabilities specified by Turing in 1936, and the machines just described. The difference is merely one of degree, a matter of the extent of the primitive capabilities with which the machines are endowed. All computation, classical or otherwise, takes place relative to some set or other of primitive capabilities: *all* computation is relative computation. The primitives specified by Turing in 1936 occupy no privileged position. One may ask whether a function is computable relative to a subset of these resources or to a superset. The characterisations of the central terms of computability theory should make no reference to some *particular* set of resources nor to equivalence classes thereof.

In this relativist landscape two sets of functions are of special—although certainly not exclusive—interest. These are the functions that are computable by an idealised human being who is unaided by machinery, and the functions that are in principle computable in the real world, which is to say, are computable by machines, or organs, or in general entities, that physically could exist, given the resources on offer in the real world, even if they do not actually exist, nor ever do so. Turing argued, we think persuasively, that the first of these sets is coincident with the set of Turing-machine-computable functions. We believe that the extent of the second set is an open, empirical question.

Propositions 1. and 2. are connected, in that the second entails the first. For the so-called Church-Turing thesis seeks to provide an absolute set, of course. But the converse entailment does not hold. Someone (da Costa perhaps) can maintain that more is computable than the so-called Church-Turing thesis allows, for instance by analogue means, without adopting our relativist position.

In the present paper we focus largely on proposition 1. although much of what we say also bears intimately on proposition 2. Proposition 2. receives further discussion elsewhere (Sylvan and Copeland 199-).

II. Fundamentals and their Reorientation

Here are some typical statements of the so-called Church-Turing thesis (aka 'Church's thesis').

(i) '[A]nything computable is Turing-machine computable.' (Dennett 1978, 83.)

(ii) 'That there exists a most general formulation of machine and that it leads to a unique set of input-output functions has come to be called *Church's thesis*.' (Newell 1980, 150.)

(iii) '[S]ince all computers operate entirely on algorithms, the ... limits of Turing machines ... also describe the theoretical limits of all computers'. (McArthur 1991, 401.)

(iv) 'Turing's analysis of what is involved in computation ... seems so general that it is hard to imagine some other method which falls outside the scope of his description ... so ... anything which can be computed can be computed by a Turing machine.' (Phillips writing in the 'Handbook of Logic and Computer Science'; Abramsky, Gabbay and Maibaum 1992, 123.)

(v) Any 'problem for which we can find an algorithm that can be programmed in some programming language, *any* language, running on some computer, *any* computer, even one that has not been built yet but *can* be built, and even one that

will require unbounded amounts of time and memory space for ever-larger inputs, is also solvable by a Turing machine'. (Harel 1992, 233.)

(vi) 'Every algorithm can be implemented by a Turing machine'. (Franklin and Garzon 1991, 133.)

Franklin and Garzon expand on this as follows (and in so doing provide an excellent statement of some commonly held beliefs that we wish to call into question):

> By their rendering the notion of *algorithm* precise and unambiguous, Turing machines afford researchers in the theory of computability with a valuable tool for determining whether given tasks can be successfully completed, or for proving that they cannot be completely solved by any physical machine ... regardless of how much memory or computing time we may be willing to provide. (Ibid., 132–3.)

We will argue that the conceptions of computation and algorithm underlying the so-called Church-Turing thesis are biased in favour of a subset of digital and digitally emulable procedures. The thesis serves only to draw a boundary between what we shall call *orthodox* and *heterodox* computing devices (figure 1).

Some of the machines shown in figure 1 are real while others, such as universal Turing machines and the GPAC ('general-purpose analogue computer'; Shannon 1941, Pour-El 1974), are idealisations ('notional' machines). Accumulator machines, Scarpellini-type machines, O-machines, coupled Turing machines, asynchronous networks of Turing machines, and heterodox connectionist networks are described in what follows. For discussion of accelerating digital machines and the GPAC see Copeland 1997, 1998c, 1998d.[2] Our conjectural location of certain biological systems on the heterodox side of the

	ORTHODOX	HETERODOX
DIGITAL	Turing machines	Turing's O-machines
	von Neumann machines	Coupled Turing machines
		Asynchronous networks of Turing machines
		Accelerating digital machines
		Certain biological systems?
ANALOGUE	Shannon's GPAC	Accumulator machines
	Differential analysers (mechanical and electronic)	Scarpellini-type machines
		Certain connectionist networks

Figure 1

[2] Also of interest are Mark Hogarth's anti de Sitter machines (Hogarth 1992, 1994).

diagram reflects our belief that for all that is currently known, the human brain could turn out to be a heterodox computing device (see Copeland 1997 and 1998a for development of this theme).

There is an *algorithm* or *effective* procedure or method for solving a class of problems if and only if there is a program for some *computing machine* which will cause the machine to produce a correct answer for each problem in the class. The general requirements for a computing machine are simple. Part of the machine must be capable of being prepared in configurations that represent the problem to be solved, and part of the machine must be capable of coming to assume, as a result of activity within the machine, configurations that represent the solution. Subdevices of the machine—'black boxes'—must make available some number of primitive operations. The machine must have resources sufficient to enable these operations to be sequenced in some predetermined way.[3] Chief among these resources will be provision for 'feedback', whereby the results of previous applications of primitive operations may become the input for further applications of those selfsame operations. (In the case of some computing machines this will be feedback in the literal sense: the output of a black box effecting some primitive operation may return as the box's input, either immediately or after passing through some intervening sequence of operations.) This recursive application, or iteration, of primitives is the essence of computation (Copeland 1996a, 1997). A computing machine is not always finite in all respects: Post was the first explicitly to allow a machine with an infinite 'symbol space' (1936, 103; and see also Turing 1947, 107). A computing machine takes a finite time to execute any of its primitive operations, but there is no lower limit on the time taken, and while the program must consist of finitely many instructions, there is no upper limit on the number of them, nor on the length of time for which the computing machine may run. In short, the whole affair is, like the story of Turing machines, *idealised.*

Text after text offers the formal notion of execution by a Turing machine as an *explication* of the informal notion of a computation. This approach is no doubt very convenient. It absolves proponents of the so-called Church-Turing thesis from much, or any, effort as regards establishing the hypothesis. But sometimes *some* effort is made. For instance, it is demonstrated that Turing machines can compute a range of perhaps unlikely functions. Or, as in the remark by Phillips quoted above, the thesis is asserted on the basis of the claimed *generality* of 'Turing's analysis of what is involved in computation'. In fact Turing himself offered the Turing machine only as an analysis of the activity of an (idealised) human mathematician engaged in the process of computing a real number using (only) pencil and paper (1936, 231). (As Wittgenstein put it, 'Turing's . . . machines are *humans* who calculate' (1980, §1096).) Turing himself certainly did not claim for his analysis the generality that Phillips (and others) apparently discern in it. Turing's concern in his 1936 paper was with the theoretical limits of what a human mathematician can compute, the whole project being directed towards showing, in answer to a question famously raised by Hilbert, that there are classes of mathematical problems whose solutions cannot be discovered by a mathematician working 'mechanically'. Turing

[3] The term 'sequencing' should not be taken to imply a linear mode of operation. The sequence of operations may consist of cycles of parallel activity.

advanced, and argued for, a hypothesis much weaker than the so-called Church-Turing thesis:

> Any procedure that can be carried out by an idealised human clerk working mechanically with paper and pencil can also be carried out by a Turing machine.

(See, for example, Turing 1936, 232, 249, Turing 1948, 7; see also Copeland 1996b, 1998a.) This thesis (the Church-Turing thesis properly so called) concerns the limits only of what a human being can compute, and carries no implication concerning the limits of machine computation. Yet the myth has somehow arisen that in his paper of 1936 Turing discussed, and established important results concerning, the theoretical limits of what can be computed by machine.[4] The myth is quickly dispelled by a careful reading of Turing's actual words.

We will call the class of procedures available to an idealised human mathematician working mechanically the class of *manual methods*. One argument for the so-called Church-Turing thesis, regularly reiterated, concerns convergence: all the various accounts and analyses of this class of methods spun out in the 1930s, and later, turned out to be equivalent (in the sense that they define the same class of functions). It is easy to see that such equivalence results could have been guided (by a less than invisible hand): those who produced the analyses were all working in the same frame of reference and no doubt hoping for equivalence theorems. Moreover, as Kreisel has pointed out, those who wish to run this convergence argument have to be ready with an answer to the question 'What excludes the case of a *systematic* error?' (Kreisel 1965, 144). But, in any case, the convergence argument is beside the point. The convergence of various prima facie very different characterisations of the notion of a manual method is at best ground for thinking that the extent of *this* notion has been correctly delineated, but the convergence has no bearing on the quite different issue of the extent of what can be computed by machine. Equally worthless is the argument from computational experience, here stated by Börger (1989, 49):

> The experience of electronic computing machines confirms that each 'computable' function is Turing-computable.

[4] Here are some typical expressions of the myth. 'Turing had proven—and this is probably his greatest contribution—that his Universal Turing machine can compute any function that any computer, with any architecture, can compute' (Dennett 1991, 215). Turing's 'results entail something remarkable, namely that a standard digital computer, given only the right program, a large enough memory and sufficient time, can compute *any* rule-governed input-output function' (Churchland and Churchland 1990, 26). 'There are certain behaviours that are "uncomputable"—behaviours for which *no* formal specification can be given for a machine that will exhibit that behaviour. The classic example of this sort of limitation is Turing's famous *Halting Problem*: can we give a formal specification for a machine which, when provided with the description of *any* [Turing] machine together with its initial state, will . . . determine whether or not that machine will reach its halt state? Turing proved that no such machine can be specified' (Langton 1996, 46).

Of course every function computable by means of the electronic digital machinery actually in use is Turing-machine-computable, for this machinery is modelled on the Turing machine. Here is how Turing put the point:

> Electronic computers are intended to carry out any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner. (1950, 1.)

Likewise other customary arguments in favour of the so-called Church-Turing thesis are not compelling. There is *the challenge*, not really something one expects to encounter in the exact sciences: 'no-one has yet produced a procedure that is intuitively mechanical yet which cannot be translated into a program for a Turing machine'. This is sometimes misdescribed as an 'inductive argument'; perhaps those who borrow the term feel it enhances the cogency of their case. No doubt querulous students are crushed with: put up or shut up.

Finally, there is the weakest argument of them all, again expressed by Phillips, and a close cousin to the challenge: it is 'hard to imagine' some computational procedure that lies beyond the scope of the universal Turing machine. The following sections, which describe some simple machines that compute more than the universal Turing machine, give the lie to all these arguments.

III. Coupled Turing Machines

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 is the result of coupling a Turing machine to its environment via one or more input channels (Copeland 1997, 694). Each channel supplies a stream of symbols to the tape as the machine operates. The machine may also possess one or more output channels, which return symbols to the environment. (Turing considered coupled Turing machines of a specific sort in the course of a discussion of machine learning, calling these 'P-type' machines (Turing 1948, 17–20; see also Copeland and Proudfoot 1996, 369–371).)

Any coupled Turing machine that halts can be simulated by the universal Turing machine, since the streams of symbols supplied by the coupled machine's input channels are, in this case, finite in length, and so can be inscribed on the universal machine's tape before it commences its task. However, the universal Turing machine is not always able to simulate a coupled Turing machine that—like an idealised automatic teller machine or air traffic controller—never halts.

A computable real number (as defined by Turing) is a real number that a Turing machine can be programmed to 'churn out' digit by digit. π (3.14159...) is an example. A Turing machine computing π never halts (since there is no last digit in the representation of π); nevertheless each digit is produced by the machine after some finite amount of processing. Since the set of Turing machine programs is countable (which is to say, can be put into one-to-one correspondence with the integers, 1, 2, 3, ...), the set of all computable real numbers is also countable. So, the set of real numbers being uncountably

large, there are *uncomputable* real numbers: real numbers that no Turing machine can be in the process of 'churning out'.

Given this fact, the proof of the existence of coupled Turing machines that cannot be simulated by the universal Turing machine is trivial. Let T be a coupled Turing machine with a single input channel and no output channels. Let u be some uncomputable real number—between 0 and 1, say—and let the decimal representation of u be written $0.u_1u_2u_3\ldots$. T's input channel writes to a single square of T's tape and each successive symbol in the input stream overwrites its predecessor on this square. The first symbol in the input stream is u_1, the second is u_2, and so on. As each of these symbols arrives T performs some trivial computation with it—multiplies it by 2, say—and writes the result on some designated squares of the tape (the result of each of these multiplications being overwritten by the result of the next). No Turing machine can be in the process of producing the succession $2\times u_1$, $2\times u_2$, etc. (for if one could, it could also be in the process of producing the decimal representation of u). A coupled Turing machine can compute more than the universal Turing machine: an input stream makes all the difference. As in the case of the situated microrobots of modern AI, and the ant in Herb Simon's famous parable (Simon 1981, 63–5), T's complex behaviour is a function of relatively simple internal processing and complex input from the environment.

IV. Accumulator Machines

One function that no Turing machine can compute is addition over the real numbers. A Turing machine can add any pair of real numbers that are computable in Turing's sense, but cannot add any arbitrary pair of real numbers. This is easily shown. Let c be one of the real numbers that no Turing machine can be in the process of churning out. Let a and b sum to c. No Turing machine can be in the process of adding the numbers a and b. Accumulator machines (Copeland 1994, 1997) are able to compute addition over the real numbers, and other functions inaccessible to Turing machines. These machines, which are analogue in nature, compute in exactly the sense explained earlier.

Accumulator machines are notional machines—as, of course, are Turing machines, with their indefinitely long and so indefinitely massive tapes, and their components that never wear out, functioning reliably through all eternity. The question that we are presently addressing, namely whether the class of functions that are in principle computable by envisagable mechanisms is wider than the class of functions computable by Turing machines, is indeed one that concerns notional machines.

Any continuously-valued physical magnitude—for example, the rate at which an idealised neuron fires—can be used to represent arbitrary real numbers. Accumulator machines represent real numbers by means of a continuously-valued physical magnitude that, for vividness, will be termed 'charge' (no close analogy is intended with the concept of charge found in modern physical theory, which is discrete). Any real number can be represented by some quantity of charge.

An accumulator, as depicted in figure 2, is a device for storing quantities of charge. When two charges (of the same sign) are applied to the input lines i_1 and i_2 they accumulate in the device and their sum is available at the output line o. Consider a machine consisting of a single accumulator embedded in a programmable control structure. The machine's program is as follows (':=' may be pronounced 'becomes').

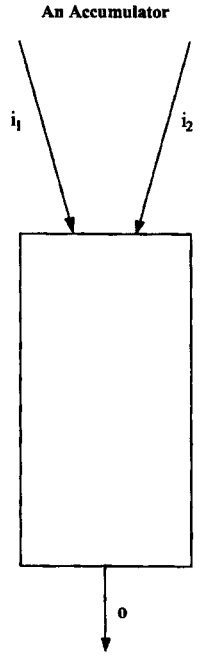

Figure 2

BEGIN

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

HALT

When the representation of any real number r is presented as input, the machine delivers a representation of 3r as output. The function over the reals of multiplication by 3 is not Turing-machine-computable, but the accumulator machine computes it.

It is a feature of this machine that if its inputs are restricted to computable numbers (in the sense defined earlier) then a Turing machine can exactly mimic its behaviour, for if the accumulator machine's primitive operations are restricted in their application to computable numbers, all remains within the bounds of Turing-machine-computability. It is also the case that (even without this restriction) a Turing machine can approximate the behaviour of the accumulator machine to any required degree of precision, in the sense

that if the universal Turing machine is given a representation of the inputs into the accumulator machine to n significant figures, for any n, then it can compute an approximation to the accumulator machine's output to n significant figures or better. Neither of these features holds generally of accumulator machines. To mention one example, a machine lacking both features can be obtained by adding a little extra hardware to the machine already described. (The machine is described in detail in Copeland 1997.) Essentially this hardware performs subtractions, exploiting the idea that charge may be negative in value. The machine computes a real function E defined: $E(x,y)=1$ if and only if $x=y$, and $E(x,y)=0$ if and only if $x\neq y$. Even where x and y range over only the computable real numbers, E(x,y) is not Turing-machine-computable (Rice 1954, 786; Aberth 1968, 287). (And obviously no Turing machine can reliably arrive at values of E(x,y) from approximations to x and y.)

V. Asynchronous Networks of Turing Machines

The standard textbook proof that any finite assembly of Turing machines can be simulated by one universal Turing machine—which involves the idea of the universal machine interleaving the processing steps performed by the individual machines in the assembly—is sound only in the case where the machines in the assembly are operating in synchrony (yet this restriction is seldom mentioned). Under certain conditions, a simple network of two non-halting Turing machines m_1 and m_2 writing binary digits to a common, initially blank, single-ended tape, T, cannot be simulated by the universal Turing machine.[5] m_1 and m_2 work unidirectionally 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. This is true also if m_1 and m_2 operate asynchronously and the *timing function* associated with each machine, Δ_1 and Δ_2 respectively, is Turing-machine-computable. $\Delta_1(n) = k$ ($n, k \geq 1$) if and only if k moments of operating time separate the n^{th} atomic operation performed by m_1 from the $n+1^{th}$; similarly for m_2 and Δ_2. Where Δ_1 and Δ_2 are both Turing-machine-computable, the universal machine can calculate the necessary values of these functions in the course of calculating each digit of the sequence being inscribed on T. If at least one of Δ_1 and Δ_2 is not Turing-machine-computable and m_1 and m_2 are not in synchrony, m_1 and m_2 may be in the process of inscribing an uncomputable number on T.

VI. A Classification of Algorithms

On the conceptualisation of matters presented in (Copeland 1997) the main divisions are as shown in figure 3.

An algorithm is *classical* just in case the function whose arguments are inputs into the algorithm and whose values are the corresponding outputs—the *characteristic function* of

[5] With thanks to Aaron Sloman for pointing this out in correspondence.

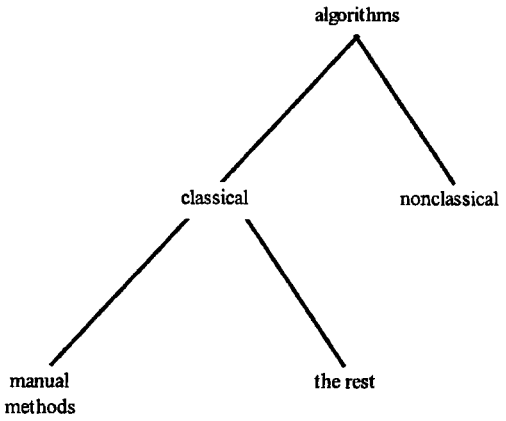

Figure 3

the algorithm—is Turing-machine-computable. Thus an algorithm may be classical even though it calls for the execution of primitive operations that no Turing machine can perform (quantum operations, for example). An algorithm is *nonclassical* just in case it is not classical. Derivatively, a nonclassical or heterodox computing machine is a computing machine that is capable of executing some nonclassical algorithm. As previously explained, *manual methods*—which were the object of Turing's investigation of 1936—are algorithms whose steps can be carried out by a human being working mechanically with pencil and paper, unaided by machinery, and idealised to the extent of having available unlimited amounts of time, internal memory, patience, scratchpad, and so forth. The Church-Turing thesis properly so called (section II) asserts, in effect, that all manual methods are classical algorithms.

It is perhaps surprising that not all classical algorithms are manual methods. That this is in fact the case has emerged from recent work on quantum computation (Deutsch 1985, Solovay and Yao 1996). Algorithms for quantum Turing machines are not in general manual methods, since not all the primitive operations made available by the quantum hardware can be performed by a person unaided by machinery. Nevertheless the algorithms executed by Deutsch-Solovay-Yao quantum Turing machines are all classical in the sense used here.[6]

VII. Real Hardware

It would be a surprise if an accumulator machine as here described were actually to be constructed and used to compute values of some non Turing-machine-computable function. Nevertheless, these machines serve their purpose in our a priori argument concerning the theoretical limits of computability. But our concern is not exclusively with

[6] On these points Copeland is indebted to conversations with Solovay.

a priori matters. We are also interested in the issue of whether it is practicable to build machines capable of computing functions that are uncomputable in the classicist's sense. The received view, certainly, is that such a project is akin to alchemy or worse. (For example, on the basis of the convergence argument, the argument from computational experience, and others of the arguments examined above, Börger hazards that just 'as we no longer seek for perpetual motion in physics, so today there are no good grounds for looking for algorithmically computable but non-[Turing-machine-computable] functions' (1989, 50).) We offer an all-too-brief discussion of this matter.

First, a possible objection. Does not any conceivable real machine have only a finite lifespan, no matter how long? If so, it delivers only finitely many outputs during its finite life: which is to say, computes only finitely many values of functions. And cannot a Turing machine always compute these same values, if only in virtue of embodying the appropriate lookup table, a finite array pairing inputs with outputs? So how can any real machine possibly compute more than a Turing machine?

This fallacious line of reasoning is discussed in detail elsewhere (Copeland 1997, 1998e; see also Copeland 1993, 237–8 and Proudfoot and Copeland 1994, 502–3 for more on the finitude argument). Notice that even if one were to agree that a Turing machine embodying a lookup table can always reproduce the actual input-output behaviour of any real machine, it certainly does not follow that the construction of nonclassical computing machines would be otiose. Let f be some non Turing-machine-computable function such that, for some reason—industrial or military, say—the calculation of some finite number of f's values is a matter of consequence. An orthodox computer that is given these values in the form of a lookup table is certainly capable of spitting them out again, but within orthodox computability theory there is no way of fully automating the process of calculating these values in the first place. However, the toil of human mathematicians may not be capable of producing the values fast enough to satisfy industrial or military demands. (One non Turing-machine-computable function that has received considerable attention is the so-called Busy Beaver function (Rado 1962). Despite more than three decades of investigation, only the first four values of the function have so far been calculated (Brady 1988, Machlin and Stout 1990).) A suitable nonclassical device might produce the required values of f with the customary swiftness of automatic computing machinery.[7]

Others, few as yet in number, share our interest in the construction of nonclassical hardware. Mike Stannett is investigating the physical implementability of devices logically similar to accumulator machines which he calls 'X-machines'. He restricts his search for an implementation to (theoretical) components that can be constructed and connected under the control of a system governed by a Turing machine. This restriction ensures that, if a design is forthcoming, the circuitry involved can be reproduced at will by processes governed by a controller no more esoteric than those already widely in use in industry. As Stannett says, 'an arbitrarily produced circuit might implement a normally

[7] Another springboard for objections is the issue of *noise*. Can a machine whose components are subject to noise compute more than the universal Turing machine? Chalmers, for one, assumes not (1996, 330–331), thereby begging delicate mathematical and empirical questions. See Copeland 1997, 703–4, Lokhorst and Copeland 199-, Maass and Orponen 1997, Maass and Sontag 1997.

uncomputable function, but this is of little practical use if the construction of this circuit cannot be repeated in a controlled manner' (1990, 339).

In an unpublished memo written in 1982 Jon Doyle wrote as follows:

> One of the most common abstract phenomena in our world is that of equilibriating systems: parts of the universe that settle into one of a spectrum of equilibrium states after certain boundary conditions are imposed. There are, in fact, many equilibriating systems with discrete spectra, for example the quantum states of molecules. Given the definiteness of these systems, we might take the operation of equilibriating as an effective one. Note carefully, I do not mean that equilibria are computable by Turing's operations, but that equilibriating can be so easily, reproducibly, and mindlessly accomplished that we grant it equal status with marking and moving slips of paper. My suspicion is that physics is easily rich enough so that ... the functions computable in principle given Turing's operations and equilibriating include non-recursive functions.[8] (Doyle 1982)

Some twenty years earlier Scarpellini wrote in German of related matters, in an article that has until recently received little attention. Scarpellini's specific concern was to illustrate the possibility (at the classical or non-quantum level) of natural processes that are not Turing-machine-computable. He wrote:

> One can ask the question whether it is possible to build a computing machine that is in a position to simulate functions f(x) so that the predicate $\int f(x) \cos nx\, dx > 0$ is not decidable, while the machine decides through direct measurement whether $\int f(x) \cos nx\, dx$ is greater than zero or not. (1963, 289.)

Kreisel, too, has written on the issue of whether there may be natural processes that are not Turing-machine-computable, his thoughts taking the form of scattered remarks throughout a series of papers spanning three decades (for example, Kreisel 1965, 1967, 1971, 1972, 1974, 1982, 1987). Like Scarpellini, he believes that it is an open question whether classical mechanics and classical electrodynamics—let alone quantum mechanics—is everywhere Turing-machine-computable. So too Pour-El and Richards (1979, 1981), who describe a possible classical physical system whose behaviour is in accordance with a function that is not Turing-machine-computable. Others who have speculated about the existence of physical processes that are not Turing-machine-computable include Geroch and Hartle 1986, Komar 1964, Penrose 1989, 1994, and Vergis *et al.* 1986.

Scarpellini makes it clear that, in his view, talk of 'building' such a machine is to be understood at the level of idealisation and thought experiment. He says

> Such a machine is naturally only of theoretical interest, since faultless measurement is assumed ... as well as, probably, ... the existence of infinitely thin resistance-free

[8] We first learned of Doyle's memo in January 1996 when Doyle wrote to Copeland after seeing a draft of Copeland 1997.

wires. All the same, the (theoretical) construction of such a machine would illustrate the possibility of non-recursive natural processes. (1963, 289.)

However, in a recent article da Costa and Doria raise the possibility that processes of the sort described by Scarpellini might be 'usefully harnessable' (contra a thesis which they attribute to Penrose to the effect that there can be no harnessable noncomputable physical processes at the classical level).[9]

VIII. Scarpellini-Type Machines

Da Costa and Doria suggest that the system they describe would solve not only an undecidability problem at the level of classical physics, but, more, 'leads to a sort of idealised solution to the Halting Problem for Turing machines' (1991, 363, 371). Let us separate these issues, addressing first the matter of deciding a classically undecidable class of physical problems, before turning, in section IX, to the halting problem.

What we consider (now rather slavishly following da Costa and Doria) are combinations of two (sorts of) physical systems, the evolution of each of which through time can be described by means of a certain function (in fact a Hamiltonian function). One system is a single free particle moving in a 2-dimensional flat space; let h be the function describing it. The other, with function h′, is a harmonic oscillator on a 2-dimensional flat phase space with real Cartesian co-ordinates. It is assumed that the harmonic oscillator is bounded, and that boundary conditions ensure this. We can also suppose that we are considering both systems within appropriately bounded regions of 2-dimensional space. Both systems are physically simple, integrable, and deterministic, and evidently quite distinct and easily distinguished.

Now define, for positive integers m, the following family $\{h_m\}$ of functions

$$h_m=\Theta_m h+(1-\Theta_m)h',$$

where $\Theta_m=\Theta_m(x)$ and where $\Theta_m(x)$ itself is a function from a family of functions $\{\Theta_m(x)\}$ (parametrized by the positive integers), the value of each of which is constant, 0 or 1, for all real numbers x. These functions are such that there is no general procedure, of a sort that can be performed by a Turing machine, for determining whether Θ_m is 0 or 1. There are various ways in which the family $\{\Theta_m(x)\}$ can be constructed, an important matter to which we return shortly.

Immediate salient points are these: If $\Theta_m=1$ then h_m represents a single free particle (since $h_m=h$), while if $\Theta_m=0$ then h_m represents an harmonic oscillator (since $h_m=h'$). These systems are topologically, and indeed qualitatively different, according as they

[9] Da Costa and Doria's project is to establish the falsity of this thesis of Penrose's, not to establish the falsity of the so-called Church-Turing thesis. Indeed, da Costa and Doria seem to subscribe to this thesis, infiltrating it into their translation of passages from Scarpellini (1991, 366). Penrose himself seems to consider his work to be anti-computational in nature, and he frequently stresses that in his view the brain cannot be a computer. He gives no serious discussion of the concept of relative computability and does not question the so-called Church-Turing thesis.

involve a bounded oscillator or a single free particle: in the one case the particle oscillates between the bounds and in the other it flies off to infinity. However, for arbitrary m, there is no general recursive procedure to decide for h_m whether it represents a free particle or an harmonic oscillator. This is the gist of the undecidability result on which we focus.[10] Still more exactly the result is as follows:

Classical undecidability proposition. Subject to a normal logistic assumption, there are explicit recursively characterised expressions for a family $\{h_m\}$ of Hamiltonian systems parametrized by the positive integers such that for each m, h_m represents a simple integrable system, either a single freely moving particle or a bounded harmonic oscillator. However, for arbitrary m, there is no recursive procedure to decide whether h_m represents a free particle or an oscillator.

The 'normal logistic assumption' is that the background logistic system ZFC, of Zremelo-Fraenkel set theory with the axiom of choice, is arithmetically consistent, that is to say that all the arithmetic theorems of ZFC are true in the standard model for arithmetic. This assumption is required for the underlying proposition from which the main proposition is derived, namely Richardson's undecidability result for the algebra A of real elementary functions—extended to include the absolute value function $|x|$. It is through a corollary of Richardson's result that the function $\Theta_m(x)$ is constructed.

For the record (and for essential notation for comprehending Θ_m), the underlying result is as follows:

Richardson's undecidability proposition. Subject to assumption:
There is a family $B_m(x)$ of smooth real functions in A parametrized by the positive integers such that given each positive integer m, either for all real x, $B_m(x)>1$, or there is a real x so that $B_m(x)\leq 1$. If the second alternative is the case, then for every real $\varepsilon>0$ there is a real x so that $B_m(x)<\varepsilon$. There is another family $C_m(x)$ of continuous, piecewise smooth, non-negative real functions in A parametrized by the positive integers such that given each positive integer m either for all real numbers $C_m(x)=0$ or there exist real numbers so that $C_m(x)>0$, but such that there is no general recursive procedure to decide, for an arbitrary m, which situation holds. Moreover, we can explicitly and algorithmically construct the expressions for both B_m and C_m.

For an algebra B extending A, the following corollary is obtained:

Corollary. Subject to assumption, we can explicitly and algorithmically construct the expression for a function $\Theta_m(x)\in B$ so that:

1. For all real x, $\Theta_m(x)=0$ *iff* there is no real x such that $C_m(x)>0$.
2. For all real x, $\Theta_m(x)=1$ *iff* there is a real x such that $C_m(x)>0$.

[10] It is part of the 'main result', proposition 2.5, of da Costa and Doria 1991, 370.

By Richardson's proposition there is no general recursive procedure to decide which situation obtains. (For proofs of these various propositions the reader may consult da Costa and Doria and works cited therein.) Now we are more or less prepared for intended applications.

As Scarpellini suggests, it is comparatively straightforward to describe analogue machines that decide for each h_m whether there is a free particle or a harmonic oscillator. For example, the machine could compare what transpires with inbuilt paradigms. A line recognition arrangement would suffice, or a line versus vibrator recognition system. For example, the machine might take a copy of a lower half of motion in the 2-dimensional space, and then check it for linearity. Call such a machine a *simple recognition* machine. Similarly, of course, observers can inspect the combined system in each case and in principle in every case, and decide whether it is a free particle, shooting off to infinity, or an harmonic oscillator, oscillating over a band. Either way, decidability ensues: what is there is decided in each case.

Nonclassical decidability, for instance analogue decidability, of classically undecidable problems has not so far been taken very seriously. Even those who have glimpsed nonclassical possibilities tend to be dismissive. Thus Scarpellini emphasizes the impracticality and mere theoretical interest of analogue electrical equipment. Da Costa and Doria are cagey:

> Is such a contraption actually feasible? ... Scarpellini has emphasized that his predicates would be decided by an analogue machine in ideal circumstances. In the concrete world, analogue devices are notoriously unreliable, and it seems to the authors that some new technology would have to be available in order to realistically implement such a [machine]. (1991, 372)

We believe that all the major questions concerning the limits of analogue computation in the real world are open. No one knows what might be achievable because there has been no investigation. (What exploration there was, in the days when analogue computation was still popular, was restricted to machines of the differential analyser type: see Copeland 1997 for more information.) Nor is anyone terribly likely to investigate while computer science and electrical engineering remain in thrall to the idea that the primitive operations specified by Turing in 1936 constitute a full answer to the question 'What can an information-processing machine do?'—an idea forcefully expressed in many textbooks.

IX. The Halting Problem

The halting function H(x,y) may be defined as follows. H(x,y)=1 if and only if the Turing machine whose program is the binary numeral representing the integer x eventually halts if set in motion with the binary numeral representing y inscribed on its tape; and H(x,y)=0 otherwise. Turing proved that no Turing machine can compute the halting function (1936, sect. 11). (This result is what is meant by the claim that no Turing machine can 'solve the halting problem'.)

There is certainly nothing outlandish in the idea that an analogue device may compute the halting function (and accordingly much else besides). For example, da Costa and Doria forge a connection between the halting function and Scarpellini-type predicates (da Costa and Doria 1994). Let $M_m(n)$ be the Turing machine that halts if and only if H(m,n)=1. Then:

Halting function proposition. One can explicitly and algorithmically construct an expression Θ(m,n) within the language of classical elementary analysis such that

1. Θ(m,n)=1 if and only if the machine $M_m(n)$ halts
2. Θ(m,n)=0 if and only if the machine $M_m(n)$ never halts.

And Θ(m,n) is $\Theta_m(n)$ (1994, 1918). A Scarpellini-type machine can in principle solve the halting problem.

Next we state some results concerning accumulator machines and the halting problem:

I (After Abramson 1971.) A machine consisting of an array of accumulators embedded in a control structure can compute the halting function provided the following capabilities are assumed (in addition to the primitive capabilities of the first accumulator machine described in section 4):

- The machine is able to multiply any pair of real numbers.
- Test-and-branch. The machine is able to test whether the number stored in a specified accumulator is zero, greater than zero, or less than zero. The test is carried out in such a way that its outcome determines which instruction in the program is executed next.
- The machine is able to take the greatest lower bound of a finite string of real numbers.

(Abramson himself—in a paper that has been unjustly neglected—considered not accumulator machines but what he called 'extended Turing machines' or ETMs. An ETM is able—by unspecified means—to store a real number on a single square of its tape. Interest in Abramson's paper has been stimulated by its rediscovery by Blum, Shub, and Smale (1989). The latter develop a general abstract framework for sequential computation over arbitrary ordered rings (e.g. the real numbers).)

II (Broadley 1996.) A network consisting of nothing but finitely many interconnected accumulators (and no control) can compute the halting function for Turing machines provided:

- The accumulators perform *weighted* addition. Where i and j are the inputs to an accumulator and m and w are two constants, the output is mi+wj. This is achieved by placing weights on the two input lines, as in a connectionist network. (m and w may both be 1.)

- Output from an accumulator is never negative. An accumulator will output 0 if the weighted sum of its inputs is less than 0. This cut-off introduces a non-linearity into the input-output function of an accumulator.[11]

X. Heterodox Connectionist Networks

A neural network is said to be *recurrent* (as opposed to *feedforward*) if there exist feedback loops among its hidden units. A *processor network* (Siegelmann and Sontag 1994, Siegelmann 1995, 1996) is a recurrent neural network of fixed physical structure consisting of a finite number of units ('neurons') with real-valued weights on the interconnections between the units.[12] In the special case where all the interconnection weights are rational, each such network is equivalent to a Turing machine (Siegelmann and Sontag 1992). If the connection matrix contains at least one irrational weight, the processor network can compute non Turing-machine-computable functions, even in polynomial time.[13]

Each neuron in the input and output layers of a processor network is a straightforward binary device and the input into and the output from a network is in binary code. Thus the non-discrete aspects of the network's functioning are invisible to the user, occurring only within the hidden layers of the network. This is in contrast to an accumulator machine, whose operation involves the user in preparing and presenting non-discrete input and identifying non-discrete output.

As previously mentioned, an accumulator machine can compute values of the function E(x,y). That is to say, given any pair of real numbers x and y the machine can compare them and determine whether or not they are identical. Such exact precision is not available in a processor network (not even among the hidden units). This is because the operation of the network is continuous in nature: discontinuities are excluded by design. Associated with each network is a precision constant, ε, determined by the interconnection weights of the network. That is to say, the network is able to distinguish between any two numbers that differ by at least ε. By appropriate choices of weights ε can be made arbitrarily small, though never zero.

There is an inverse relationship between the precision demands inherent in an architecture and the reliability of the corresponding piece of hardware. For example, the slightest leakages of charge from the accumulator will cause the machine described earlier to become grossly unreliable at its task of computing the function f(r)=3r. (In a strict sense it will no longer compute this function at all but some other function.) Processor networks do not suffer from this extreme brittleness. Within limits, errors in the values of the

[11] Without the restriction that the number of accumulators in the network be finite the result is trivial, since a network consisting simply of Boolean nodes can compute the halting function provided there are as many nodes as there are Turing machine programs. (A Boolean node is a node that computes a Boolean function, such as conjunction, disjunction, negation.)

[12] Other work concerning heterodox connectionist networks is reported in Garzon and Franklin 1989, MacLennan 1994, Wolpert and MacLennan 1993.

[13] In polynomial time, the recognition power of processor networks is identical to the recognition power of a certain subset of O-machines (section XI), the O-machines with so-called 'sparse oracles'. (An oracle is sparse just in case its ability to answer queries is constrained in a formally specified way by the lengths of the queries presented to it.) In exponential time, the recognition power of processor networks is unbounded.

interconnection weights make no difference to the set of functions computed by the network. This, and the discrete nature of the network's inputs and outputs, are pleasing properties. Although processor networks are purely notional machines, there are faint but encouraging tinges of realism in their specifications.

XI. O-Machines

That Turing was the first person to consider nonclassical computing machines is a fact that has been generally overlooked. (Even those working in the field speak inappropriately of 'super-Turing computation', 'computing beyond the Turing limit', 'breaking the Turing barrier' and the like.) This section gives a brief account of the O-machines to which Turing devoted section 4 of his PhD thesis (Turing 1939, 172–3; see also Copeland 1997, 1998a, 1998b, Copeland and Proudfoot, 1999).

O-machines, like Turing machines, are quintessentially digital devices. We will refer to functions whose arguments are integers and whose value is always 0 or 1 as 'binary-valued integer-functions'. An O-machine results when an ordinary Turing machine is augmented with a primitive operation (or several such) that returns the values of some binary-valued integer-function that is not Turing-machine-computable. Turing calls the subdevices that execute these nonclassical primitive operations 'oracles'. He says that an oracle works by 'unspecified means' and that we need 'not go any further into the nature of . . . oracle[s]'.

An O-machine has three of its states and one of the symbols of its alphabet reserved for special purposes, the *call state*, χ, the *1-state*, the *0-state*, and the *marker symbol*, μ. (Machines with n nonclassical primitive operations require n distinct call states.) The machine writes the symbols that are to form an input to the oracle on some convenient segment of its tape, using occurrences of μ to mark the start and finish of this string. When the execution of some instruction in the machine's program puts the machine into state χ, this input is delivered to the oracle and the oracle returns the value of the function for that argument, by placing the machine in either the 1-state or the 0-state. Obviously an O-machine can compute functions that are not computable by the classical Turing machine (and more such than the particular function(s) embodied in its nonclassical primitive(s)).

Turing postulated nonclassical primitive operations without providing any details of how these operations might be implemented, nor even offering an argument to the effect that their implementation in any form is a practical possibility. This procedure is perfectly sound. Turing was engaged in specifying the architecture of notional machines with a view to delineating classes of mathematical problems. The possibility of real existence, while an interesting matter, is beside the a priori point of such constructions. Certainly no one complains about the fact that in 1936 Turing paid no attention to the issue of how the primitive operations of a classical Turing machine might practically be implemented, despite the fact that it was by no means obvious how to do this, nor even clear that it could be done at all. Turing was once asked whether he thought a Turing machine could actually be constructed, and he replied that the machine would need to be as big as the Albert Hall.[14] Not until during the war, when Turing became acquainted with electronic

[14] This was related to Copeland by Robin Gandy in 1995.

technology developed for other purposes, did it become clear that the notional machines of his 1936 paper could be turned into a reality. Might the world likewise come to witness the construction of an O-machine? Are there, in other words, real physical processes that are harnessable to do the work of an oracle? This question is entirely open. If a sufficient momentum of investigation were to build up, perhaps the question might relatively swiftly be settled in the affirmative. Who would have suspected, when Deutsch first described a notional quantum Turing machine in 1985, that by 1995 pieces of experimental quantum hardware would be sitting on laboratory benchtops?

Issues of harnessability to one side, we would be profoundly surprised if the physics of the real world can be properly and fully set out without departing from the set of Turing-machine-computable functions. These functions have been the focus of intense interest during the brief six decades since Turing delineated them, but the explanation of this is surely their extreme tractability, together, of course, with the fact that they have made a considerable number of people very rich, rather than because some inherent suitability for exhaustively describing the structure and properties of matter is discernible in them. Moreover, as we have already related, these functions were the fruit of Turing's analysis of the activity of an idealised human mathematician working mechanically with pencil and paper. It is simple anthropomorphism to expect the same set of functions to be prominent in the behaviour of the world minus human mathematicians. In short it would—or should—be one of the greatest astonishments of science if the activity of Mother Nature were never to stray beyond the bounds of Turing-machine-computability.

University of Canterbury

Received: March 1997
Revised: August 1998

REFERENCES

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

Abramsky, S., Gabbay, D.M. and Maibaum, T.S.E. (eds.), *Handbook of Logic in Computer Science*, Vol.1. (Oxford: Clarendon Press, 1992).

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

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

Börger, E., *Computability, Complexity, Logic*. (Amsterdam: North Holland 1989).

Brady, A.H., 'The Busy Beaver Game and the Meaning of Life', in R. Herken (ed.), *The Universal Turing Machine: A Half-Century Survey* (Oxford: Oxford University Press, 1988).

Broadley, S., 'Relative Computation and Connectionist Networks'. Typescript, University of Canterbury, 1996.

Calude, C.S., Casti, J. and Dineen, M.J. (eds.) *Unconventional Models of Computation*. (Singapore: Springer-Verlag, 1998).

Chalmers, D.J., *The Conscious Mind: In Search of a Fundamental Theory*. (New York: Oxford University Press, 1996).

Copeland, B.J., *Artificial Intelligence: a Philosophical Introduction*. (Oxford: Blackwell, 1993).

Copeland, B.J., 'Beyond Turing Computability: New Foundations for a Computational Theory of Mind'. Talk given at Victoria University of Wellington, June 1994.

Copeland, B.J., 'What is Computation?', *Synthese* 108 (1996a), 335–359.

Copeland, B.J., 'The Church-Turing Thesis', in Perry, J., Zalta, E. (eds.), *The Stanford Encyclopaedia of Philosophy* [http://plato.stanford.edu, 1996b].
Copeland, B.J., 'The Broad Conception of Computation', *American Behavioral Scientist* 40 (1997), 690–716.
Copeland, B.J., 'Turing's O-machines, Penrose, Searle, and the Brain', *Analysis* 58 (1998a), 128–138.
Copeland, B.J. (ed.), 'A Lecture and Two Radio Broadcasts on Machine Intelligence by Alan Turing'. *Machine Intelligence* 15 (1998b), forthcoming.
Copeland, B.J., 'Even Turing Machines Can Compute Uncomputable Functions', in Calude, Casti, Dinneen 1998c.
Copeland, B.J., 'Super Turing-Machines'. *Complexity* 4 (1998d).
Copeland, B.J., 'Narrow Versus Wide Mechanism'. University of Canterbury Philosophy Research Papers no. 5 (1998e).
Copeland, B.J. and Proudfoot, D. 'Alan Turing's Forgotten Ideas in Computer Science'. *Scientific American*, April 1999.
da Costa, N.C.A. and Doria, F.A., 'Classical Physics and Penrose's Thesis', *Foundations of Physics Letters* 4 (1991), 363–374.
da Costa, N.C.A. and Doria, F.A., 'Undecidable Hopf Bifurcation with Undecidable Fixed Point', *International Journal of Theoretical Physics* 33 (1994), 1913–1931.
Dennett, D.C., *Brainstorms: Philosophical Essays on Mind and Psychology.* (Brighton: Harvester, 1978).
Deutsch, D., 'Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer', *Proceedings of the Royal Society* Series A 400 (1985), 97–117.
Doyle, J., 'What is Church's Thesis? An Outline'. Laboratory for Computer Science, MIT, 1982.
Franklin, S. and Garzon, M., 'Neural Computability', in O. Omidvar (ed.), *Progress in Neural Networks*, vol. 1 (Norwood, NJ: Ablex, 1991).
Garzon, M. and Franklin, S., 'Neural Computability II'. Abstract, *Proceedings (vol. I), IJCNN International Joint Conference on Neural Networks (1989),* 631–637.
Geroch, R. and Hartle, J.B., 'Computability and Physical Theories', *Foundations of Physics* 16 (1986), 533–550.
Harel, D., *Algorithmics: The Spirit of Computing.* (Reading, MA: Addison-Wesley, 1992).
Hogarth, M.L., 'Does General Relativity Allow an Observer to View an Eternity in a Finite Time?', *Foundations of Physics Letters* 5 (1992), 173–181.
Hogarth, M.L., 'Non-Turing Computers and Non-Turing Computability', *PSA 1994*, vol.1, 126–138.
Komar, A., 'Undecidability of macroscopically distinguishable states in quantum field theory', *Physical Review* second series, 133B (1964), 542–544.
Kreisel, G., 'Mathematical Logic', in T.L. Saaty (ed.), *Lectures on Modern Mathematics*, vol. 3 (New York: John Wiley, 1965).
Kreisel, G., 'Mathematical Logic: What Has it Done For the Philosophy of Mathematics?', in R. Schoenman (ed.), *Bertrand Russell: Philosopher of the Century* (London: George Allen and Unwin, 1967).
Kreisel, G., 'Some Reasons for Generalising Recursion Theory', in R.O. Gandy and C.M.E. Yates (eds.), *Logic Colloquium '69*, (Amsterdam: North-Holland, 1971).
Kreisel, G., 'Which Number Theoretic Problems can be Solved in Recursive Progressions on π^1_1-Paths Through 0?', *Journal of Symbolic Logic* 37 (1972), 311–334.
Kreisel, G., 'A Notion of Mechanistic Theory', *Synthese* 29 (1974), 11–26.
Kreisel, G., Review of Pour-El and Richards, *Journal of Symbolic Logic* 47 (1982), 900–902.
Kreisel, G., 'Church's Thesis and the Ideal of Formal Rigour', *Notre Dame Journal of Formal Logic* 28 (1987), 499–519.
Langton, C.R. 'Artificial Life', in M.A. Boden (ed.), *The Philosophy of Artificial Life.* (Oxford: Oxford University Press, 1996).
Lokhorst, G.J. and Copeland, B.J., 'O-Machines'. In preparation, 199-.
Machlin, R. and Stout, Q.F., 'The Complex Behaviour of Simple Systems', in S. Forrest (ed.), *Emergent Computation* (Cambridge, MA: MIT Press, 1990).
MacLennan, B.J., 'Continuous Symbol Systems: The Logic of Connectionism', in D.S. Levine and M. Aparicio IV (eds.), *Neural Networks for Knowledge Representation and Inference* (Hillsdale, NJ: Erlbaum, 1994).
Maass, W. and Orponen, P., 'On the Effect of Analog Noise in Discrete-Time Analog Computations'. NeuroColt Technical Report Series, NC-TR-97–042, 1997.
Maass, W. and Sontag, E.D., 'Analog Neural Nets with Gaussian or Other Common Noise Distributions Cannot Recognise Arbitrary Regular Languages'. NeuroColt Technical Report Series, NC-TR-97–043, 1997.

McArthur, R.P., *From Logic to Computing*. (Belmont, CA: Wadsworth, 1991).
Newell, A., 'Physical Symbol Systems', *Cognitive Science* 4 (1980), 135–183.
Penrose, R., *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. (Oxford: Oxford University Press, 1989).
Penrose, R., *Shadows of the Mind: A Search for the Missing Science of Consciousness*. (Oxford: Oxford University Press, 1994).
Pour-El, M.B., 'Abstract Computability and its Relation to the General Purpose Analog Computer', *Transactions of the American Mathematical Society* 199 (1974), 1–28.
Pour-El, M.B. and Richards, I., 'A Computable Ordinary Differential Equation Which Possesses No Computable Solution', *Annals of Mathematical Logic* 17 (1979), 61–90.
Pour-El, M.B. and Richards, I., 'The Wave Equation with Computable Initial Data such that its Unique Solution is not Computable', *Advances in Mathematics* 39 (1981), 215–239.
Proudfoot, D. and Copeland, B.J., 'Turing, Wittgenstein and the Science of the Mind', *Australasian Journal of Philosophy* 72 (1994), 497–519.
Rado, T., 'On Non-Computable Functions', *Bell Systems Technical Journal* 91 (1962), 877–884.
Rice, H.G., 'Recursive Real Numbers', *American Mathematical Society Proceedings* 5 (1954), 784–91.
Richardson, D., 'Some Undecidable Problems Involving Elementary Functions of a Real Variable', *Journal of Symbolic Logic* 33 (1968), 514.
Scarpellini, B., 'Zwei Unentscheitbare Probleme der Analysis', *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 9 (1963), 265–289.
Shannon, C.E., 'Mathematical Theory of the Differential Analyser', *Journal of Mathematics and Physics of the Massachusetts Institute of Technology* 20 (1941), 337–354.
Siegelmann, H.T. and Sontag, E.D., 'On the Computational Power of Neural Nets', *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory* (1992), 440–449.
Siegelmann, H.T., 'Computation Beyond the Turing Limit', *Science* 268 (1995), 545–548.
Siegelmann, H.T., 'Computability With Neural Networks', *Lectures in Applied Mathematics* 32 (1996), 733–747.
Siegelmann, H.T. and Sontag, E.D., 'Analog Computation via Neural Networks', *Theoretical Computer Science* 131 (1994), 331–360.
Simon, H.A., *The Sciences of the Artificial*, 2nd edition. (Cambridge, MA: MIT Press, 1981).
Solovay, R. and Yao, A., 'Quantum Circuit Complexity and Universal Quantum Turing Machines', to appear, 1996.
Stannett, M.. 'X-Machines and the Halting Problem: Building a Super-Turing Machine', *Formal Aspects of Computing* 2 (1990), 331–341.
Sylvan, R. and Copeland, B.J. 'Computability is Logic-Relative', forthcoming in D. Hyde and G. Priest (eds.), *Applications of Relevant Logics* (199-).
Turing, A.M., 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society* 42 (1936–37), 230–265.
Turing, A.M., 'Systems of Logic Based on Ordinals', *Proceedings of the London Mathematical Society* 45 (1939), 161–228.
Turing, A.M., 'Lecture to the London Mathematical Society on 20 February 1947', in B.E. Carpenter and R.W. Doran (eds.), *A.M. Turing's ACE Report of 1946 and Other Papers*. (Cambridge, MA: MIT Press, 1986).
Turing, A.M., 'Intelligent Machinery', National Physical Laboratory Report, 1948, in B. Meltzer and D. Michie (eds.), *Machine Intelligence 5* (Edinburgh: Edinburgh University Press, 1969).
Turing, A.M., 'Programmers' Handbook for Manchester Electronic Computer', University of Manchester Computing Laboratory, 1950.
Vergis, A., Steiglitz, K. and Dickinson, B., 'The Complexity of Analog Computation', *Mathematics and Computers in Simulation* 28 (1986), 91–113.
Wittgenstein, L., *Remarks on the Philosophy of Psychology*, vol.1. (Oxford: Blackwell, 1980).
Wolpert, D.H. and MacLennan, B.J., 'A Computationally Universal Field Computer That is Purely Linear', Santa Fe Institute Technical Report 93–09–056, 1993.