THE BROAD CONCEPTION OF COMPUTATION

B. Jack Copeland

ABSTRACT

A myth has arisen concerning Turing's paper of 1936, namely that Turing set forth a fundamental principle concerning the limits of what can be computed by machine - a myth that has passed into cognitive science and the philosophy of mind, to wide and pernicious effect. This supposed principle, sometimes incorrectly termed the 'Church-Turing thesis', is the claim that the class of functions that can be computed by machines is identical to the class of functions that can be computed by Turing machines. In point of fact Turing himself nowhere endorses, nor even states, this claim (nor does Church). I describe a number of notional machines, both analogue and digital, that can compute more than a universal Turing machine. These machines are exemplars of the class of *nonclassical* computing machines. Nothing known at present rules out the possibility that machines in this class will one day be built, nor that the brain itself is such a machine. These theoretical considerations undercut a number of foundational arguments that are commonly rehearsed in cognitive science, and gesture towards a new class of cognitive models.

1. Introduction

Turing's famous (1936) concerns the theoretical limits of what a human mathematician can compute. As is well known, Turing was able to show, in answer to a question raised by Hilbert, that there are classes of mathematical problems whose solutions cannot be discovered by a person acting solely in accordance with an algorithm. Nowadays this paper is generally taken to have achieved rather more than Turing himself seems to have intended. It is commonly regarded as a treatment - indeed, the definitive treatment - of the limits of what *machines* can compute. This is curious, for it is far from obvious that the theoretical limits of human computation and the theoretical limits of machine computation need coincide. Might not some machine be able to perform computations that a human being cannot - even an idealised human being who never makes mistakes and who has unlimited resources (time, scratchpad, etc.)? This is the question I shall discuss here. I shall press the claim - a claim which, as I explain, originates with Turing himself - that there are (in the mathematical sense) computing machines capable of computing *more* than any Turing machine. These machines form a diverse class, some being digital in nature and some analogue. For want of a better name I will refer to such computing machines as *nonclassical* computers. It may in the future be possible to build such machines. Moreover, it is conceivable that the brain is a nonclassical computer. This possibility has, it seems, been widely overlooked both by proponents of traditional versions of the computational theory of mind and by opponents of the view that the brain is a computer.

In order to keep the discussion as self-contained as possible, I will begin by briefly reviewing some essential concepts, in particular those of a Turing machine, a (Turing-machine-) computable number and a (Turing-machine-) computable function.

A Turing machine is an idealised computing device consisting of a read/write head with a paper tape passing through it. The tape is divided into squares, each square bearing a single symbol - '0' or '1', for example. This tape is the machine's general purpose storage medium, serving both as the vehicle for input and output and as a working memory for storing the results of intermediate steps of the computation. The tape is of unbounded length - for Turing's aim was to show that there are tasks that these machines are unable to perform *even given* unlimited working memory and unlimited time.

Nevertheless, Turing required that the input inscribed on the tape should consist of a finite number of symbols. Later I discuss the effect of lifting this requirement.

FIG 1 ABOUT HERE ('A Turing machine').

The read/write head is programmable. It may be helpful to think of the operation of programming as consisting of altering the head's internal wiring by means of a plugboard arrangement. To compute with the device first program it, then inscribe the input on the tape (in binary or decimal code, say), place the head over the square containing the leftmost input symbol, and set the machine in motion. Once the computation is completed the machine will come to a halt with the head positioned over the square containing the leftmost symbol of the output (or elsewhere if so programmed).

The head contains a subdevice that I will call the *indicator*. This is a second form of working memory. The indicator can be set at a number of 'positions'. In Turing machine jargon the position of the indicator at any time is called the *state* of the machine at that time. To give a simple example of the indicator's function, it may be used to keep track of whether the symbol last encountered was '0' or '1': if '0', the indicator is set to its first position, and if '1', to its second position.

Fundamentally there are just six types of operation that a Turing machine performs in the course of a computation. It may (i) read (i.e. identify) the symbol currently under the head; (ii) write a symbol on the square currently under the head (after first deleting the symbol already written there, if any); (iii) move the tape left one square; (iv) move the tape right one square; (v) change state; (vi) halt. These are called the *primitive* operations of the machine.

Commercially available computers are hard-wired to perform primitive operations considerably more sophisticated than those of a Turing machine - add, multiply, decrement, store-at-address, branch, and so forth. (The precise constitution of the list of primitives varies from manufacturer to manufacturer.) However, the remarkable fact is that none of these machines can outdo a Turing machine: despite its austere simplicity a Turing machine is capable of computing anything that e.g. a

Cray supercomputer can compute. Indeed, a Turing machine can compute *more* than a Cray, since (a) the Cray has access to only a bounded amount of memory, and (b) the Cray's speed of operation is limited by various real-world constraints. (It is often said, incorrectly, that a Turing machine is necessarily slow (for example Haugeland 1985: 140). A Turing machine is an idealised device and has no real-world constraints on its speed of operation.)

A program or 'machine table' for a Turing machine is a finite collection of instructions each calling for certain primitive operations to be performed if certain conditions are met. Every instruction is of the form:

> If the current state is n and the symbol currently under the head is x, then write y on the square currently under the head [y may be identical to x], go into state m [m may be n], and - - - .

In place of - - - may be written either 'move left one square' or 'move right one square' or 'halt'.

There are in fact two ways of arranging for a Turing machine to act in accordance with a program. One, as already mentioned, is to program the head. The other is to translate the instructions in the program into, say, binary code and inscribe them on a machine's tape. Turing was able to show that there is a program U such that if the head of a machine is programmed in accordance with U and if any Turing machine program whatever, P, is written on the machine's tape, then the machine will behave *as if* its head had been programmed in accordance with P. A Turing machine whose head has been programmed in accordance with U is called a *universal* machine. This idea of modifying the function of a computing machine by storing a program of symbolic instructions in its working memory - as opposed to reconfiguring the machine's hardware - was Turing's greatest contribution to the development of the digital computer. There are many different programs that will do the work of U and thus many distinct universal Turing machines, all equivalent in computational power. (The 'smallest' universal Turing machine known is due to Minsky (1967: 277 ff). It uses an alphabet of 4 symbols and has just 28 instructions programmed into its head!)

A Turing machine program is said to terminate just in case any Turing machine running the program will halt no matter what the input. An easy way to write a non-terminating program is simply to omit an instruction to halt. Computer programs that never terminate by design are commonplace. Air traffic control systems, automated teller machine networks, and nuclear reactor control systems are all examples of such. A non-terminating Turing machine program that is of importance for the present discussion consists of a list of instructions for calculating sequentially each digit of the decimal representation of π (say by using one of the standard power series expressions for π). A Turing machine that is set up to loop repeatedly through these instructions will spend all eternity writing out the decimal representation of π digit by digit, 3.14159 . . .

Turing called the numbers that can be written out in this way by a Turing machine the *computable* numbers. That is, a number is computable, in Turing's sense, if and only if there is a Turing machine that calculates in sequence each digit of the number's decimal representation. (There is nothing special about decimal representation here: I use it because everyone is familiar with it. This necessary and sufficient condition can equally well be stated with 'binary representation', for example, in place of 'decimal representation'.) Straight off, one might expect it to be the case that *every* number that *has* a decimal representation, either finite or infinite - that is, every real number - is computable. (The real numbers comprise the integers, the rational numbers - which is to say, numbers that can be expressed as a ratio of integers, for example $^1/_2$ and $^3/_4$ - and the irrational numbers, such as $\sqrt{2}$ and π, which cannot be expressed as a ratio of integers.) For what could prevent there being, for each particular real number, a Turing machine that 'churns out' that number's decimal representation digit by digit? However, Turing did indeed show that not every real number is computable. The decimal representations of some real numbers are so completely lacking in pattern that there simply is no finite list of instructions, of the sort that can be followed by a finite-input Turing machine, for calculating the n^{th} digit of the representation for arbitrary n. Indeed, most real numbers are like this. There are only countably many computable numbers whereas, as Cantor showed last century, there are uncountably many real numbers. (A set is countable if and only if either it is finite or its members can be put into a one-to-one correspondence with the integers.)

It is common to speak of Turing machines computing *functions* (in the mathematical sense of 'function', not the biological). These may be functions over any objects that are capable of being represented by means of finite strings of symbols inscribed on the machine's tape. Following Turing, to say that a function, for example addition, is *computable* is to say that there is a Turing machine such that if, for any pair of numbers x and y, the machine is given x and y as input, it will print out the value of x+y and halt. As I shall explain shortly, addition over the real numbers is in fact *not* a computable function.

Addition over the integers clearly is a computable function. Is addition over the computable numbers a computable function? (That is, is x+y computable whenever x and y are both computable numbers?) The answer is 'yes', but off the top of one's head one might think otherwise, for if x (or y) is a computable number having an infinite decimal representation, how can x be input, given Turing's restriction that the input inscribed on the tape must consist of a finite number of symbols? The solution is to input x in the form of a program which, if inscribed on the (otherwise blank) tape of some universal Turing machine, would cause the machine to calculate the decimal representation of x digit by digit. Nothing tricky is going on here. The program inscribed on the tape is, after all, a sequence of symbols, and this particular sequence of symbols has no less a claim to be counted as a representation of the number, x, than '14' and '1110' have to be counted as representations of the number fourteen. In short, Turing has given us a new method for representing numbers; and in this system there are finite representations of some of the numbers which, in the decimal system, can be represented only by means of an infinite sequence of symbols. To return to the machine that is to add x and y, where x and y are any computable numbers: once the representations, in Turing's ingenious system, of x and y have been inscribed on the tape, the machine is set in motion. First it calculates the first digit of the decimal representation of x and remembers it; next it calculates the first digit of the representation of y; then it adds these two digits, using a left-to-right addition procedure; then it turns to the second digits of the representations, and so on. Each digit of the decimal representation of x+y is produced by the machine in some finite number of steps.[1] (Exercise: convince yourself that it *is* possible to do decimal addition from left to right!)

I hope the discussion so far has afforded some insight into what a computable number is, in Turing's restricted sense of 'computable number'. A number is computable, in his sense, just in case the number can be expressed by means of a certain finite string of symbols, namely a string of symbols which, if inscribed on the otherwise blank tape of some universal Turing machine, will cause the machine to churn out the decimal representation of the number. If 'computable' is to mean 'machine-computable' then it is by no means obvious that the only computable numbers are those that are computable in Turing's sense. For why should a necessary condition for the computability of a number by some machine or other be that the number is expressible in a finite number of symbols in this notation of Turing's? Indeed, I shall shortly be giving some toy examples of idealised computing machines that compute numbers and functions that are not Turing-machine-computable. Even a Turing machine can be modified to compute such numbers, simply by lifting the restriction that the input must consist of a finite number of symbols. Let x and y be two numbers that are not Turing-machine-computable and let M be a Turing machine whose tape is infinite in one direction only. The first digit of the decimal representation of x appears on the first square of M's tape, the first digit of the representation of y appears on the second square, the third square is blank, the second digit of x's representation is on the fourth square and of y's on the fifth square, the sixth square is blank, and so on. When the machine is set in motion it invokes the left-to-right addition procedure previously mentioned, writing the resulting digits on the blank squares of the tape. Equivalently one may consider a Turing machine that has been modified by the addition of two input lines (see Turing 1948: 17ff, Copeland and Proudfoot 1996). This arrangement removes the need to inscribe the representations of x and y on the tape prior to the commencement of the computation. If a digit is applied to an input line (e.g. in the form of an electrical pulse) this results in the digit being inscribed on the tape in a square associated with this input line (the inscription resulting from the immediately preceding input on this line is overwritten). At the first step of the computation the first digit of the representation of x is applied to the first input line and the first digit of the representation of y to the second input line; and so on. In either case M does *compute* the output on the basis of the input, for M performs a sequence of primitive Turing-machine operations under the control of a program, and this is the paradigm example of what it is for a machine to compute. Yet the number whose decimal representation M is engaged in churning out, x+y, is not a computable number in Turing's sense.

Henceforward I will abbreviate 'computable in Turing's sense' by 'Computable' (and similarly 'Uncomputable'). By the unqualified expression 'computable' I will always mean 'computable by some machine'.

The nonclassical nature of the computing machines described in the next section arises from the fact that addition over the real numbers, as opposed to addition over the Computable real numbers, is not a Computable function. This is easy to establish. Let z be an Uncomputable real. Many pairs of numbers will sum to z; let x and y be one such pair. Since z is not Computable nor is x+y: no Turing machine can be in the process of churning out the decimal representation of x+y. (There is, of course, the additional difficulty, previously discussed, of entering x and y as input, since at least one of them must itself be Uncomputable.)

Not all functions over the Computable numbers are Computable. The task of computing the values of such a function is beyond a Turing machine, even though the function's arguments (or inputs), being Computable numbers, can be represented on the machine's tape by means of finitely many symbols. Turing gave a famous example of such a function, the so-called halting function, which I will discuss in section 3. In the meantime, it will be useful at this stage to introduce, without proof, a rather simple example of one of these functions, to be written E(x,y). Where x and y are any Computable numbers, $E(x,y)=1$ if and only if $x=y$, and $E(x,y)=0$ if and only if $x\neq y$. (A proof to the effect that E(x,y) is not Computable is given by Rice 1954: 786 and Aberth 1968: 287.)

Turning from concepts that Turing himself explicitly presented, I will now introduce the general notions of a *computing machine* and an *algorithm*. The general requirements for a computing machine are simple. Part of the machine must be capable of being prepared in configurations that represent the arguments of whatever functions are to be computed, and part of the machine must be capable of coming to assume, as a result of activity within the machine, configurations that represent values of these functions. 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.[2] 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.

Any machine functioning in this way can usefully be described as following a collection of *instructions*. Each instruction calls for one or more primitive operations to be performed, either conditionally or unconditionally. The machine acts in accordance with the instructions either because a symbolic representation of them has been entered into it, as in the case of a universal Turing machine, or simply because, as in the case of 'rewiring' the head of a Turing machine, the hardware has been configured that way by hand, either temporarily or permanently. Typically, digital computers are programmed in the first way and analogue computers in the second way (although the early electronic and electromechanical digital computers were programmed by rewiring). It is usual to call machines that operate in the second way 'program controlled' and those that operate in the first way 'stored program'.

Extending standard terminology, I will call any collection of rules or instructions that determines the behaviour of a computing machine an *algorithm*. In full: an algorithm is a finite set of instructions such that, for some computing machine M, each instruction calls for one or more of M's primitive operations to be performed, either unconditionally or if certain conditions, recognisable by the machine, are met. (The instructions are said to be an algorithm *for* achieving some specified result *R* just in case following the instructions is certain to bring about *R* .) The process, *computing*, may be characterised as consisting in a computing machine's acting in accordance with an algorithm (see my 1996 for an elaboration and defence of this characterisation). An algorithm is *classical* just in case it can be implemented by a machine that computes only Computable functions. A *nonclassical* algorithm determines a machine to compute a function that is Uncomputable. An *effective procedure* is an algorithm each instruction of which can be carried out by a *human being*, unaided by any machinery (save for pencil and paper), and without the exercise of insight or ingenuity.

In the literature the word 'algorithm' is rarely used in the way in which it is used here. Standardly, 'algorithm' is used as a synonym for 'effective procedure' (see, for example, the glossary in Minsky 1967: 311), there being no accepted term for machine-executable procedures that are not effective (i.e. which cannot also be carried out by an obedient human clerk who is devoid of any imagination or talent and is unaided by any machinery save paper and pencil). No doubt part of the explanation for this lack is that the existence of such procedures has commonly been overlooked. Their existence tends to be obscured by the fact that the word 'mechanical' is standardly used as another synonym for 'effective'. (Gandy 1988 outlines the history of this use of the word 'mechanical'.) On this way of speaking, the issue of whether there are machine-executable procedures that are not effective becomes hidden, for the question 'Are there mechanical procedures that are not mechanical?' appears self-answering. A further reason to revise traditional terminology in the manner suggested here is provided by the recent emergence of the field of quantum computation. Algorithms for quantum Turing machines (Deutsch 1985, Solovay and Yao 1996) are not effective procedures, since not all the primitive operations of a quantum Turing machine can be performed by a person unaided by machinery. (While the algorithms executed by Deutsch-Solovay-Yao quantum Turing machines are not effective procedures, they are nonetheless classical algorithms in the sense of that term introduced above, since these machines compute only Computable functions. On this point I am indebted to conversations with Solovay.)

As is always remarked, the concept of an effective procedure is an 'informal' one and characterisations given of it are 'vague', 'intuitive' and 'lacking in precision'. One of Turing's aims in his paper of 1936 was to present a precise concept with which the imprecise concept of an effective procedure might be replaced. Thus his famous thesis that whenever there is an effective procedure for obtaining the values of a function, the function can be computed by a Turing machine. Turing's thesis is a large, and probably true, claim, with implications concerning the limitations of the use of effective procedures, but it is a claim that says nothing about the limitations of machine computation.

Turing introduced his machines with the intention of providing an idealised description of a certain human activity, the tedious one of *numerical computation*, which until the advent of automatic computing machines was the occupation of many thousands of people in commerce, government, and research establishments. These human workers were called 'computers', and this is how Turing himself uses that word in his early papers. The Turing machine is a model, idealised in certain respects, of a human being engaged in computation. Turing prefaces his first description of a Turing machine with the words: 'We may compare a man in the process of computing a ... number to a machine' (1936: 231). Wittgenstein put this point in a striking way: 'Turing's "Machines". These machines are *humans* who calculate' (Wittgenstein 1980, §1096). It is a point that Turing was to emphasise, in various forms, again and again (see further my 1997a). Of course, it was not some deficiency of imagination that led Turing to model the Turing machine on what could be achieved by a human computer. The purpose for which it was invented demanded it. Turing's aim was to show, in response to a question raised by Hilbert, that there are well-defined mathematical problems that cannot be solved by a human being working in accordance with an effective procedure. Turing's strategy was simple. First, he argued for his thesis that there are no effective procedures that Turing machines are incapable of carrying out (see especially 1936: 249-54). Second, he gave a specific example of a mathematical problem that no Turing machine can solve. He proved there can be no Turing machine that answers all questions of the form 'Does statement p follow from the Bernays-Hilbert-Ackermann axioms for first-order predicate logic?' (1936, sect. 11).

I will bring this introductory section to a close with some comments on how the issue of nonclassical computation bears on the way some of the principal battle lines are currently drawn in cognitive science and the philosophy of mind. Johnson-Laird is surely right when he discerns three mutually exclusive positions in current thinking concerning the relationship between human mentality and computation (1987: 252).

(1) The human brain (or, variously, mind or mindbrain) is a classical computer. Cognition is classical computation. This is the traditional computational theory of mind. There are, of course, innumerable fine-grained renderings of the theory: the brain is variously thought to be a digital

computer, an analogue computer, a stored-program machine, a program-controlled machine, a massively parallel distributed processor, and so on.

(2) The cognitive activity of a human brain can be simulated perfectly by a Turing machine but (1) is false. Analogies offered in the literature include hurricanes, the motion of the planets around the sun, and the digestion of pizza. None of these phenomena is itself computational in nature yet, supposedly, all can be simulated perfectly by a Turing machine, in the sense that the machine can compute correct descriptions of such phenomena on any temporal grid. (Opinions differ among adherents of position (2) as to whether a Turing machine simulating the cognitive activity of a brain could itself properly be taken to be *thinking*, or whether to do so would be as foolish as supposing 'that a computer simulation of a storm will leave us all wet' (Searle 1989: 37-38).)

(3) The human brain is not a computing machine and nor can the brain's cognitive activity be simulated in its entirety by a computing machine.[3]

Johnson-Laird maintains that the only alternative to these three positions is that mental processes are not 'scientifically explicable' (1987: 252). But if the considerations put forward here are correct there is conceptual space for two further positions.

(4) The human brain is a nonclassical computing machine.

(5) The cognitive activity of a human brain can be simulated perfectly by a nonclassical computing machine but the brain is not itself a computing machine; such simulation cannot be effected by a classical computing machine.

A prominent advocate of position (2) is Searle (see, for example, 1992: 200; an extended argument for this position is given in his 1997). Searle argues for (2) on the basis of a proposition that he calls 'Church's thesis' and his belief that (1) is false. In section 4 I will be maintaining that his argument is unsound, and instructively so. At this point it is worth remarking that his separate argument for the falsity of (1), the Chinese room argument, is powerless when faced with (4).[4] This argument depends upon the occupant of the Chinese room being able in principle to 'handwork' a computer program, or in a later version, upon the occupants of the Chinese gym being able in principle to enact the steps of the algorithm that determines the behaviour of some connectionist network (Searle 1980, 1990). Thus in both cases the argument demands that the algorithm in question be an effective procedure. Indeed, one way of explaining the notion of an effective procedure is as a machine program that can be carried out by the clerk in the Chinese room, who works obediently but without imagination and who uses no machinery other than paper and pencil. By its nature, the argument is inapplicable to machine programs that are not effective procedures, even machine programs that, like Turing machine programs, consist of instructions to perform operations on discrete symbols. Section 3 describes a class of nonclassical machines that run such programs. First, some analogue nonclassical machines are considered.

2. Nonclassical Analogue Computing Machines

In analogue representation, properties of the representational medium ape (or reflect or model) properties of the represented state-of-affairs. Analogue representations form a diverse class. To list some examples: the longer a line on a motorway map, the longer the road the line represents; the greater the number of clear plastic squares in an architect's model, the greater the number of windows in the building represented; the higher the pitch of an acoustic depth meter, the shallower the water. In a typical electrical analogue computer, numerical quantities are represented by potential difference. Thus, for example, the output voltage of the machine at a time might represent the momentary speed of the object being modelled. Notice how different this is from representing speed in terms of decimal code. Strings of decimal digits obviously do not represent by means of possessing a physical property - such as length - whose magnitude varies in proportion to the magnitude of the property that is being represented.

As the case of the architect's model makes plain, analogue representation can be *discrete* in nature: there is no such thing as a fractional number of windows. Among computer scientists the term 'analogue' is sometimes used more narrowly, to indicate representation of one continuously-valued quantity by another (e.g. speed by voltage). As Smith remarks:

> 'Analogue' should ... be a predicate on a representation whose structure corresponds to that of which it represents ... That continuous representations should historically have come to be called analogue presumably betrays the recognition that, at the levels at which it matters to us, the world is more foundationally continuous than it is discrete. (1991: 271.)

The two computers that I am about to describe, M1 and M2, are analogue machines in both the narrow and the 'proper' sense.

Let me stress at the outset that these machines are idealisations - as are Turing machines, with their indefinitely long, and hence, presumably, indefinitely massive tapes, and their components that function reliably through all eternity. Let me stress also that M1 and M2 are intended as nothing more than in-principle examples of the sort that are philosophers' stock in trade. If these simple examples succeed in illustrating how there could possibly be mechanical procedures that cannot be carried out, nor simulated, by a Turing machine, then they will have served their purpose.

According to some classical physical theories, the world contains continuously-valued physical magnitudes (for example, the theory of continuous elastic solids in Euclidean space). Let us suppose, for the purpose of describing these notional machines, that there are such magnitudes. For the sake of vividness I will refer to one such as 'charge'. Any real number can be represented by some quantity of charge. Quantities of charge can be stored in a device called an accumulator. Figure 2 depicts such a device.

FIGURE 2 ABOUT HERE ('An Accumulator')

If 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. M1 consists of this device embedded in a programmable control structure. (The details of the control structure, which are pedestrian, need not detain us.) The example requires that the control have the ability to clear line i_2 and apply the charge present on either the output line or line i_1 to i_2. Here is the algorithm governing the operation of M1. (':=' may be pronounced 'becomes'.)

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

When the representation of any real number x is presented as input, M1 delivers a representation of 3x as output. Since x may be either a Computable number or an Uncomputable number, M1 computes an Uncomputable function.

Charge may be negative in value. This means that an accumulator can be set to the task of subtracting one real number from another. A representation of x–y becomes available on the output line if a representation of x is applied to one input line and of –y - that is to say, a negative charge - to the other input line. Adding a little more hardware to M1 produces a machine M2 capable of executing the following algorithm. Given representations of any two real numbers x and y as input, M2 prints '1' if and only if x=y and prints '0' if and only if x≠y. If the inputs into M2 are restricted to Computable numbers then M2 computes the function E(x,y), which as already mentioned is Uncomputable.

```
BEGIN
INPUT TO $i_1$          (A charge representing x is received from an external device and placed on
                    line $i_1$.)
INPUT TO $i_2$          (Ditto for y and $i_2$.)
OPSIGN $i_2$            (The sign of the charge on $i_2$ is changed.)
ADD ($i_1$, $i_2$)
IF $o=0$ PRINT '1'
IF $o \neq 0$ PRINT '0'
```

HALT

(A subdevice of M2 can test whether or not the charge on the output line of an accumulator is positive, negative or null. OPSIGN does its work by entering the charge on i_2 into an accumulator then clearing i_2 and applying to i_2 whatever charge it determines must be entered into this accumulator in order to produce null charge on the output line.)

Notice that each of these nonclassical algorithms evidently meets the Kleene-Hofstadter requirement that an algorithm be capable of being 'communicated reliably from one sentient being to another of reasonable mathematical aptitude by means of language' (Kleene 1987: 493-4, Hofstadter 1980: 562). (To anticipate a little, this remark applies also to algorithms executed by the nonclassical digital machines introduced in section 3.)

The action of M1 can be *approximated* by Turing machine, in the sense that if, for any real number x and for any integer k, some Turing machine is given the first k places of a decimal representation of x, it will compute at least the first k places of 3x. The belief is often expressed that the action of *any* continuous system can be approximated by a Turing machine to any required degree of fineness. Dreyfus, for example, claims that 'even an analogue computer, provided that the relation of its input to its output can be described by a precise mathematical function, can be simulated on a [classical] digital machine' (1992: 72). This belief is false. Let us use as the measure of the degree of fineness of the computations the maximum number of digits d in the decimal strings employed to approximate whatever real numbers are in question (assuming for simplicity that these real numbers are always less than one). Clearly there is no d such that, for any Computable numbers x and y, a Turing machine which is given the first d digits of x's decimal representation and the first d digits of y's decimal representation will print '1' if and only if M2 prints '1'. Thus the action of M2 cannot be approximated by a Turing machine (even though the relation of M2's input to its output 'can be described by a precise mathematical function').

In conversation one sometimes encounters references to a mysterious theorem which allegedly states that the action of any analogue computer can be approximated by a Turing machine to any required degree of precision. In fact there is no such theorem in the literature. What is to be found is a theorem due to Pour-El and Shannon stating that this is so for a *particular* type of analogue computer, the GPAC or general-purpose analogue computer. The GPAC is an idealisation of a differential analyser (much as a Turing machine is an idealisation of a general-purpose digital computer). The first differential analyser, which was mechanical, was built in 1931 by Vannevar Bush, working at MIT (Bush 1931, 1936). In subsequent versions the wheel-and-disc integrators and other mechanical components were replaced by electromechanical, and finally by electronic, devices (Bush and Caldwell 1945). A differential analyser may be conceptualised as a collection of black boxes connected together in such a way as to allow considerable feedback. Each box performs a primitive operation. Among these primitives are addition, multiplication, integration and an equivalent of the OPSIGN operation discussed in connection with M2. Programming the machine consists of wiring together boxes in such a way that the desired sequence of primitive operations is executed. (Since the boxes can work in parallel, an electronic differential analyser solves equations very fast. Against this must be set the time-consuming process of massaging the problem that is to be solved into the exact form demanded by the analogue hardware.)

Shannon (1941) gave a mathematical treatment of an idealisation of the differential analyser. His idealised machine became known subsequently as the GPAC. Building on Shannon's work, Pour-El proved a number of theorems concerning the limitations of the GPAC. In brief summary her results are: (1) there are Turing-machine-computable functions that cannot be computed by the GPAC; and (2) the action of the GPAC can always be approximated by a Turing machine (Pour-El 1974, theorems 3 and 7). There was a crucial lacuna in the proof of Shannon's principal theorem and Pour-El is able to obtain her results only because of a significant modification she makes to Shannon's idealised machine (Pour-El 1974, notes 4 and 12).[5] In her terminology the modification is that there must exist a 'domain of generation' of the initial conditions of the machine (1974: 8, 12, and note 4). In essence what this amounts to is that, within a given limit, small alterations in the input to, or the settings of, the machine produce no change in the output. As she says, 'In practice the operator is allowed to vary the initial conditions on an analog computer slightly' (1974: 12). I will call her modification to Shannon's idealisation the requirement of *limited precision*. When ε is the smallest number such that a given machine is able to distinguish a pair of numerical inputs that differ by that amount, ε is called the *precision constant* of the machine.

It is certainly the case that if an analogue computer is to be operable by a human being then it must conform to the requirement of limited precision. However, this requirement has no place in a treatment of the issue of whether there are computable functions that are not Turing-machine-computable: computability is not an epistemic notion. Moreover, the 'insensitive dependence on initial conditions' demanded by Pour-El renders the GPAC unable to distinguish between inputs that a Turing machine *can* distinguish. Let x and y be distinct numbers with finite decimal representations and which differ by less than the precision constant of the GPAC. Then, tautologically, the GPAC will be unable to distinguish an input of x from an input of y. However, a Turing machine will be able to so distinguish, for if two finite strings differ a Turing machine will be able to detect the difference. The only bound placed on the length of strings input into a Turing machine is that they be finite. In other words, a Turing machine is permitted the luxury of *unlimited* precision in the representation of the inputs x and y (subject always to the proviso that x and y are Computable). To insist that the GPAC have a non-zero precision constant is to give it an unfair handicap in a contest with a Turing machine. In view of this asymmetry, result (2), above, is weaker than it may at first appear. (It is because of the absence of a precision requirement, of course, that M2 is able to compute a function E(x,y) that can neither be computed nor approximated by a Turing machine, in apparent contradiction to Pour-El's result.)

The Shannon Pour-El theorem is of no more help than a crystal ball to someone who would like to know whether or not analogue machines that can compute Uncomputable functions may some day be built, or whether the human brain may contain such mechanisms. For the GPAC is an idealisation of a museum piece. Its horizons were set by the state of the art of electromechanical engineering at the time at which Shannon wrote. If new black boxes are added to perform primitive operations foreign to the differential analyser then by definition the resulting machine is not a GPAC and so falls outside the ambit of the theorem.

One area of computer science in which the idea of computing with real numbers is sometimes taken seriously is connectionism (for example, Garzon and Franklin 1989, MacLennan 1994, Rumelhart and McClelland 1986: 45-49, Siegelmann and Sontag 1994, Smolensky 1988, section 8, Wolpert and MacLennan 1993). Korb raises a doubt: to suppose 'neural networks to be computationally superior [to Turing machines] ... is to suppose that the analog device is sensitive to the *exact* value of the real-valued weight, *beyond any finite enumeration of its digits whatsoever*' (1996: 250). Chalmers (1996) runs the following argument:

> The presence of background noise ... implies that no process can depend on requiring more than a certain amount of precision. Beyond a certain key point (say, the 10^{-10} level on an appropriate scale), uncontrollable fluctuations in background noise will wash out any further precision. This means that if [by means of a Turing machine] we approximate the state of the system to this level of precision (perhaps a little further to be on the safe side - to the 10^{-20} level, for example), then we will be doing as well as the system itself can reliably do. (1996: 331.)

This argument is undermined by the fact that Siegelmann and Sontag (op. cit.) have provided a mathematical treatment of neural networks with continuously-valued weights and limited precision which can generate the values of functions that are not Turing-machine-computable. In any case, Chalmers' assumption that, in a world of finite precision, there is always a 'key point' beyond which no further increase whatsoever in precision can occur - an assumption for which he offers no supporting argument - is much stronger than the claim that in the real world precision is never infinite (which is itself a claim of unknown truth-value). It is easy to show that if this strong assumption is not made then a continuous system operating under noisy conditions may nevertheless exhibit Uncomputable behaviour. For even granting that, for any given degree of precision, there is a Turing machine that simulates the continuous system operating in noisy conditions in which that degree of precision is the greatest attainable, it by no means follows that there is an effective procedure for obtaining, for each degree, a Turing machine program that simulates the continuous system operating in conditions affording that degree of precision. Analogously, there is no effective procedure for determining successive approximations to an Uncomputable real number. (If there were, then this procedure could be used by a Turing machine to generate sequentially each digit of the number, which is impossible.) For example, having established that a certain Turing machine simulates the continuous system when the latter is operating in conditions affording a degree of precision k, there may be no routine way of moving on to a description of a Turing machine that performs the simulation when the degree of precision is 2k. Each step up through the degrees may require fresh mathematical insights.[6] In the absence of such an effective procedure, there can be no single Turing machine that simulates the continuous system's behaviour through time as the level of noise fluctuates (even were there to be an effective procedure for simulating the fluctuations in the noise process itself). In other words, the continuous system behaves in accordance with a function that is Uncomputable.

Chalmers claims, without offering a supporting argument, that no discrete system can calculate the values of the halting function (1996: 331). (The halting function is described below.) This claim is shown false by Gandy (1980) (see also Copeland 1997c). Chalmers' brief discussion of Uncomputability entirely overlooks the possibility, explored in the next section, of *digital* systems that compute Uncomputable functions.

3. Nonclassical Digital Computing Machines

I shall be relying on the fact that there are Uncomputable functions all of whose values and all of whose arguments are integers. That some such functions must be Uncomputable follows immediately from the fact that there are uncountably many functions on the integers (as is shown by a straightforward Cantorian argument), whereas there are only countably many Turing machines. That the Turing machines are countable is a consequence of the fact that one Turing machine is distinguished from another only by its finite program. Indeed, each machine may be taken to be represented - or 'labelled' - by its program. As we have seen, each program P can be expressed as a finite string of binary symbols - the string which, if written on the tape of a given universal Turing machine, U, will cause U to behave as if it were a machine whose head has been programmed in accordance with P. Each of these strings is the binary representation of an integer; so there are no more programs than there are integers; so the Turing machines are countable. Given some standard choice of universal machine, each Turing machine can systematically be labelled with an integer - an integer which, if inscribed on the tape of this universal machine, would cause it to behave as if it were the machine in question. Turing called such an integer a *description number* of the machine.

Turing established the existence of Uncomputable functions on the integers by as direct a method as possible: he produced an example of one, the famous halting function H(x,y) (1936, section 8). The first argument, x, is a description number, the second argument, y, is any integer, and the value of the function is either 0 or 1. The second argument represents an input to the machine with the given description number. The function is defined as follows: H(x,y)=1 if and only if the Turing machine of which x is a description number eventually halts if set in motion with y inscribed on its tape; and H(x,y)=0 if and only if either x is not a description number of a Turing machine or the Turing machine of which x is a description number never halts if set in motion with y inscribed on its tape. (An accessible proof that no Turing machine can compute the halting function may be found in Minsky (1967: 148-9).)

Turing himself was the first to consider nonclassical digital computing machines, in his Ph.D. dissertation (published as Turing 1939). This work seems to be little known among cognitive scientists. Turing called his machines O-machines, or 'oracle' machines (1939: 172ff). An O-machine consists of an ordinary Turing machine augmented with a primitive operation that returns the values of an Uncomputable function on the integers (or several such primitive operations). This primitive operation is made available by a subdevice of the machine that Turing refers to as an 'oracle'. He remarks that an oracle works by 'unspecified means' and says that we need 'not go any further into the nature of [these] oracle[s]' (1939: 173). According to Turing's specification each oracle returns the values of an Uncomputable function that is, like the halting function H(x,y), two-valued. Calls to the oracle are effected by means of a special state χ, the call state, and a special symbol μ, the marker symbol. (Where an O-machine has more than one nonclassical primitive, a corresponding number of call states is required.) The machine inscribes the symbols that are to form the input to the oracle on any convenient block of squares on its tape, using occurrences of the marker symbol to indicate the beginning and the end of the input string. As soon as an instruction in the machine's program puts the machine into state χ, the input is delivered to the oracle and the oracle returns the value of the function. On Turing's way of handling matters the value is not written on the tape. Rather a pair of states, the 1-state and the 0-state, is employed in order to record values of the function. A call to the oracle ends with the oracle placing the machine in one or other of these two states according to whether the value of the function is 1 or 0.

One particular O-machine, $\mathbb{H}$, has as its classical part a universal Turing machine and as its nonclassical part a mechanism for returning values of the halting function H(x,y). $\mathbb{H}$ can compute many Uncomputable functions. This is because of the familiar phenomenon of the *reducibility* of one function to others (for example, multiplication is reducible to addition or to the various primitive functions made available by an ordinary Turing machine). Many Uncomputable functions are reducible to the halting function: the problem of determining the values of the function in question reduces to the problem of determining whether or not a certain Turing machine halts when given certain integers as input. All functions reducible to the halting function and/or the primitives of an ordinary Turing machine are computable by $\mathbb{H}$.

An oracle is a black box. So too are the subdevices that make available the primitive operations of a Turing machine. In neither case (1936, 1939) does Turing offer a discussion of what mechanisms might appropriately occupy these black boxes. In both papers, the use made of these notional computing machines is to delineate classes of mathematical problems, to which the issue of the possible physical implementation of the machine's logical components is not relevant. In 1936 it was, in fact, far from clear whether the construction of a universal Turing machine lay within the bounds of physical possibility. Once asked whether he thought that a universal Turing machine could actually be constructed, Turing replied dismissively that the machine would need to be as big as the Albert Hall.[7] It was not until Turing became familiar with electronic technology developed for other purposes (principally through his work on speech encipherment, from 1943 onwards) that he realised that the notional machines of his 1936 paper could actually be built. Perhaps his O-machines, too, will one day become an engineering reality. A necessary condition is the existence of physical processes that cannot be simulated by a Turing machine, and that can be harnessed to provide mechanisms filling in the black boxes. Speculation as to whether there may actually be Uncomputable physical processes stretches back over at least four decades (see, for example, Da Costa and Doria 1991, Doyle 1982, Kreisel 1967, 1974, Pour-El 1974, Pour-El and Richards 1979, 1981, Penrose 1989, 1994, Scarpellini 1963, Stannett 1990, Vergis et al. 1986; Copeland and Sylvan 1997 is a survey).

A notional oracle can certainly be implemented by means of a machine similar to M1. For example, relative to some given ordering of the arguments of the halting function, the successive values of the function form a certain infinite sequence of binary digits. Let this sequence be preceded by a decimal point. Some quantity of charge physically instantiates the real number so formed. Given some subdevices for performing appropriate arithmetical operations, values of the halting function can be extracted on demand from an accumulator storing exactly that quantity of charge.

The mechanism implementing an oracle need not employ a continuously-valued physical magnitude. It may be *discrete*, in the sense that the action of the mechanism can always be described by enumerating some sequence S_1, S_2, ... of states of the mechanism, where each of the totality of possible states of the mechanism is uniquely labelled by an integer. (The action of M1, for example, cannot be so described, for M1 has more possible states than there are integers.) Gandy (1980) has proved that any discrete mechanism satisfying certain rather general 'principles' can be perfectly simulated by a Turing machine. So if an oracle is to be implemented by a discrete mechanism, it must be a mechanism that contravenes one or more of these principles. As Gandy points out (1980: 145), there are notional machines obeying Newtonian mechanics that contravene the principles, for example ones employing instantaneous action at a distance, or signals travelling with arbitrarily large velocities, so that the distance a signal can travel during a single step of the machine's operation is unbounded. Such processes are ways of contravening what Gandy calls the 'principle of local causality' (1980: 141ff). It is an open question whether there are processes contravening this principle in the actual world. (The growth of quasicrystals is sometimes offered as a possible example of a process involving non-local causality (Penrose 1989, ch. 10, Stephens and Goldman 1991).)

If the number of possible states that can be occupied by a discrete machine is both finite and bounded then the machine can be perfectly simulated by a Turing machine. For the possible behaviours of the machine can in principle be represented by means of a finite 'lookup table', and a Turing machine with this table inscribed on its tape can effect the simulation simply by searching the table. So the following objection might be put: In the real world, assumed to be bounded in appropriate respects, there can be no discrete mechanism that implements an oracle. However, this statement conflates two different issues, that of the in-principle simulability by Turing machine of a discrete machine situated among bounded resources (time, energy, tape, etc.), and that of whether or not the machine in question is best understood as being one that acts in accordance with a Computable function. For instance, there might be no way of calculating the entries in the lookup table for the machine other than by employing the machine itself to compute them, for there might be no effective procedure other than the lookup table for obtaining values of the function that the machine computes. Although such a machine is always simulable in principle when situated among bounded resources, the architecture of the machine may nevertheless be best understood as being one that, under the idealisation of unbounded resources, produces the values of an Uncomputable function.

There are functions on the integers that no O-machine can compute, for Turing placed certain restrictions on the nonclassical primitive operations of O-machines (1939, sections 3 and 4). However, these restrictions are in a way arbitrary, and the notion of an O-machine can readily be generalised so that no function on the integers remains uncomputable. This gives rise to an obvious objection, namely that the attempt to broaden the concept of computability has ended with the concept becoming trivialised, for if each function on the integers is computable by some machine or other, the expression 'computable function on the integers' becomes synonymous with 'any function on the integers whatsoever'. The reply to this objection is straightforward. Computability is a relative notion, not an absolute one. All computation, classical or otherwise, takes place relative to some set or other of primitive capabilities. The primitives specified by Turing in 1936 occupy no privileged position. One may ask whether a function is computable relative to these primitives or to some superset of them. Answers will come in the form of theorems, anything but trivially won, to the effect that such-and-such functions are, or are not, computable relative to such-and-such primitive capabilities.

In this relativist landscape two sets of functions are of special - although certainly not exclusive - interest. These are, first, the set of functions that are in principle computable in the real world, which is to say, can be generated, under the idealisation of unbounded resources, by computing machines that physically could exist; and, second, the set of functions that are in principle computable by human beings working effectively with pencil and paper and unaided by machinery, again under the idealisation of unbounded resources. The extent of the first set is an open empirical question. The extent of the second is the topic of the Church-Turing thesis.

4. The Church-Turing Thesis

A myth has arisen concerning Turing's paper of 1936, namely that Turing established a fundamental result concerning the limits of what can be computed by machine - a myth that has passed into the philosophy of mind, to wide and pernicious effect. The 'Oxford Companion to the Mind' states: 'Turing showed that his very simple machine ... can specify the steps required for the solution of any problem that can be solved by instructions, explicitly stated rules, or procedures' (Gregory 1987: 784). Dennett maintains that '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' (1991: 215). In similar vein Sterelny asserts 'Astonishingly, Turing was able to show that any procedure that can be computed at all can be computed by a Turing machine ... Despite their simple organisation, Turing machines are, in principle, as powerful as any other mode of organizing computing systems' (1990: 37, 238). Paul and Patricia Churchland maintain that 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. That is, it can display any systematic pattern of responses to the environment whatsoever' (1990: 26). The entry on Turing in the recent 'A Companion to the Philosophy of Mind' contains the following claims:

'we can depend on there being a Turing machine that captures the functional relations of the brain', for so long as 'these relations between input and output are functionally well-behaved enough to be describable by ... mathematical relationships ... we know that some specific version of a Turing machine will be able to mimic them' (Guttenplan 1994: 595). These various quotations are typical of current writing on the foundations of the computational theory of mind.

Turing had no result entailing that a Turing machine can 'display any systematic pattern of responses to the environment whatsoever'. Indeed, he had a result entailing the opposite. His theorem that no Turing machine can compute the halting function entails that there are possible patterns of responses to the environment, perfectly systematic patterns, that no Turing machine can display. It is false that Turing proved (or, for that matter, attempted to prove) that a universal Turing machine 'can compute any function that any computer, with any architecture, can compute'. What he did prove is that his universal machine can compute any function that any *Turing machine* can compute. He also put forward, and advanced philosophical arguments in support of, the conjecture that (in modern terminology) every effective procedure can be carried out by a universal Turing machine. As I remarked in section 1, this is a thesis concerning the limits of what a human mathematician can compute, not a thesis concerning the limits of machine computation. Church put forward an equivalent form of the same thesis (Church 1936). Since Church published first, the thesis is usually called 'Church's thesis'. The term 'Church-Turing thesis' seems to have been first introduced by Kleene (with a small flourish of bias in favour of Church):

> So Turing's and Church's theses are equivalent. We shall usually refer to them both as *Church's thesis*, or in connection with that one of its ... versions which deals with 'Turing machines' as *the Church-Turing thesis*. (Kleene 1967: 232; see also Kleene 1952.)

In mathematical logic this is the standard use of the term 'Church-Turing thesis'. However, it is nowadays not uncommon for a very different proposition, that whatever can be calculated by any machine can be computed by a universal Turing machine, to be called 'the Church-Turing thesis' or

'Church's thesis'. I will refer to this stronger proposition as *the Church-Turing thesis improperly so-called*. Neither Church nor Turing endorsed it. (See further my 1997a.)

The Church-Turing thesis improperly so-called pervades the literature on the computational theory of mind (as the above quotations illustrate). Some writers - Dennett and Sterelny, for example - think themselves entitled to affirm this proposition because they believe that, somehow, Turing proved it. Other writers argue for it on the spurious ground that various prima facie very different attempts (by Turing, Church, Post, Kleene, Markov, and others) to characterise the informal notion of an effective procedure in precise terms have turned out to be equivalent to one another - spurious because this is, of course, evidence concerning the correct extent of the notion of an effective procedure, and not evidence concerning the extent of what can be computed by machine. In what follows I review some examples of the penetration into philosophy of mind, Artificial Intelligence, and computational psychology, of the myth that the Church-Turing thesis improperly so called is a secure claim.

Searle (1992, 1997) advances the following argument for the simulationist position (2) described at the end of section 1.

> Can the operations of the brain be simulated on a [classical] digital computer? ... The answer seems to me ... demonstrably 'Yes' ... That is, naturally interpreted, the question means: Is there some description of the brain such that under that description you could do a computational simulation of the operations of the brain. But given Church's thesis that anything that can be given a precise enough characterization as a set of steps can be simulated on a digital computer, it follows trivially that the question has an affirmative answer. (1992: 200.)

The program for the machine M1 displayed in section 2 forms a straightforward counterexample to the proposition that Searle (improperly) calls 'Church's thesis'. In his argument Searle simply overlooks the possibility that the provision of a precise mathematical description of the dynamics of the brain may exceed the resources of Computable mathematics. No doubt the same error underlies the claims by Guttenplan quoted earlier in this section. In similar fashion Leiber writes: 'If any formal system can be explicitly mechanized as a Turing Machine, so can any actual machine, nervous system, natural language, or mind in so far as these are determinate structures' (1991: 57). Even Dreyfus, who has exploded many myths concerning digital computers, acquiesces in the dogma that '*any* process which can be formalised so that it can be represented as a series of instructions for the manipulation of discrete elements, can, at least in principle, be reproduced by [a universal Turing machine]' (1992: 72). The machine table of any O-machine serves as a counterexample to this claim. Dreyfus goes on to state that human behaviour 'understood as motion' can 'in principle be reproduced to any degree of accuracy' on a Turing machine (1992: 195-6). (This is a surprisingly generous view of the power of effective procedures to find in a book entitled 'What Computers Can't Do'.) In effect Dreyfus is forced into this view by his mistaken belief that it is a 'fundamental truth that every form of "information processing" (even those which *in practice* can only be carried out on an "analogue computer") must *in principle* be simulable on a [Turing machine]' and that 'all physical processes can be described in a mathematical formalism which can in turn be manipulated by a [Turing machine]' (1992: 195).

It is ironic that prominent critics of AI like Searle and Dreyfus should share in what is possibly the most easily questioned assumption of arguments fielded by leading advocates of AI. Newell, for example, appeals to the Church-Turing thesis improperly so called in the course of his famous argument for the crucial sufficiency-part of his and Simon's physical symbol system hypothesis (Newell 1980). A physical symbol system is a universal Turing machine, or any equivalent system, situated in the physical - as opposed to the conceptual - world (1980: 147-50, 154-55, 161). (The tape of the universal machine is finite: Newell requires that the storage capacity of the machine be unlimited in the practical sense of finite yet not 'small enough to force concern' (1980: 161).) The sufficiency-part of the physical symbol system hypothesis is the claim that 'any physical symbol system can be organised further to exhibit general intelligent action', where '*general intelligent action* means ... that in real situations behavior appropriate to the ends of the system and adaptive to the demands of the environment can occur, within some physical limits' (1980: 170). To establish the sufficiency-part of the hypothesis would be to establish that the project of programming von Neumann computers to exhibit general intelligent action is viable in principle (if not in practice). Newell thinks the claim is easily shown: 'A universal system always contains the potential for being any other system, if so instructed. Thus, a universal system can become a generally intelligent system' (ibid.).[8] The second occurrence of 'system' in the first of these statements must, of course, be taken to mean 'system whose input-output function can be computed by a Turing machine' (else the statement is false). So if Newell's brief argument is deductively valid then it has an additional, unstated premiss, namely that a generally intelligent system does not work by means of computing Uncomputable functions (nor work by physical means other than computation that can be computationally simulated only by a machine that computes such functions). Without this additional premiss there is no valid inference from the premiss that Newell does state to the conclusion. Newell believes this additional premiss because he believes, on the basis (as he says) of the work of Turing and Church, that 'there exists a most general formulation of machine', which is to say, of 'the notion of ... determinate physical mechanism', and this general formulation 'leads to a unique set of input-output functions' (1980: 150). This 'maximal set of functions' is, he says, the set of Turing-machine-computable functions; the members of this set are 'the functions that can be attained by ... machines' or determinate physical systems (ibid.). His allusion to the work of Turing and Church is all that Newell provides by way of support for these claims. But, again, the mathematical treatment by Turing and Church of the notion of an effective procedure affords no clue as to the truth-value of the empirical conjecture that there are no naturally occurring systems whose 'input-output functions' cannot be simulated by any Turing machine (and nor do Turing and Church suggest that it does).

Dennett endorses a related argument (1978: 82-3). His conclusion is that AI (conceived of as a project involving only classical computation) is 'the study of all possible modes of intelligence' (1978: 83). The argument is, he says, an argument from 'some version of Church's Thesis (e.g. anything computable is Turing-machine computable)' (ibid.).

> [A] non-question-begging psychology will be a psychology that makes no ultimate appeals to unexplained intelligence, and that condition can be reformulated as the condition that whatever functional parts a psychology breaks its subjects into, the smallest, or most fundamental, or least sophisticated parts must not be supposed to perform tasks or follow procedures requiring intelligence. That condition in turn is surely

> strong enough to ensure that any procedure admissible as an 'ultimate' procedure in a psychological theory falls well within the intuitive boundaries of the 'computable' or 'effective' as these terms are ... used in Church's Thesis ... [A]ny psychology that stipulated atomic tasks that were 'too difficult' to fall under Church's Thesis would be a theory with undischarged homunculi. (1978: 83.)

Dennett offers no grounds for believing his claim that the condition he states - which is no doubt a reasonable one, if somewhat vague - is 'surely strong enough' to exclude from psychology any primitive operations that cannot be simulated by a Turing machine. The addition operation of machine M1 (section 2) is completely mechanical: the account of its realisation by means of an accumulator makes no 'appeal to unexplained intelligence'. Yet this operation is an atomic task '"too difficult" to fall under Church's Thesis'. Likewise a black box that returns values of an Uncomputable function on discrete symbols need not be occupied by an undischarged homunculus, but by brute physical mechanism, no more intelligent than the internal mechanism of the feedback amplifiers that return values of the integrals of given functions in Bush's differential analyser.

Fodor writes:

> Although the elementary operations of the Turing machine are restricted, iterations of the operations enable the machine to carry out any well-defined computation on discrete symbols. ... If a mental process can be functionally defined as an operation on symbols, there is a Turing machine capable of carrying out the computation ... The "black boxes" that are common in flow charts drawn by psychologists often serve to indicate postulated mental processes for which Turing reductions are wanting. Even so, the possibility in principle of such reductions serves as a methodological constraint on psychological theorizing by determining what functional definitions are to be allowed. (1981: 130; see also 1983: 38-39.)

The claim made in the first part of this quotation is false. For example, the halting-function machine $\mathbb{H}$ (section 3) carries out well-defined operations on discrete symbols. The algorithms that $\mathbb{H}$ and other O-machines follow are not essentially different from those followed by standard Turing machines: in all cases each line of the machine table is a clear and unambiguous specification of which primitive operations are to be performed when the machine is in such-and-such state and has such-and-such symbol beneath the head. Nor is there anything ill-defined about ℍ's nonclassical primitive operation: no one can complain that Turing's definition of the halting function is somehow improper. It is clearly possible that there are mental processes that can be functionally defined as operations on symbols but not as operations on symbols that can be carried out by Turing machine. Thus the methodological constraint that Fodor offers is a bad one, for it is a constraint that forecloses on this empirical possibility.

Pylyshyn writes:

> What is remarkable ... is that the range of actual computer architectures, or even the theoretical proposals for new architectures, available for our consideration [in developing models of cognition] is extremely narrow compared with what, in principle, is possible ... Because our experience has been with a rather narrow range of architectures, we tend to associate the notion of computation, and hence of algorithms, with the particular class of algorithms that can be executed by architectures in this limited class. (1984: 96-7.)

This is a sound observation. However, later on the very same page is a remark certain to encourage the very tendency that Pylyshyn is complaining about:

> The most primitive machine architecture is, no doubt, the original binary-coded Turing machine introduced by Alan Turing ... Turing's machine is 'universal' in the sense that it can be programmed to compute any computable function.[9] (1984: 97.)

In similar fashion Boden unwittingly sets the terms of computational psychology in such a way as to exclude any consideration of the possibility of studying architectures that are not Turing-machine-equivalent. In computational psychology, she tells her readers, 'the mind is conceived of in terms of the computational properties of universal Turing machines' (1988: 5). Then later: 'If a psychological science is possible at all, it must be capable of being expressed in [classical] computational terms' (1988: 259). She offers this claim as a consequence of Turing's having '*proved* that a language capable of defining "effective procedures" suffices, in principle, to solve any computable problem' (ibid.). Turing proved no such thing.

The Church-Turing thesis improperly so called, which is a dogma rooted in a misconstrual of Turing's work, is an unexamined assumption of much contemporary writing on the mind. Its promulgation by leading theoreticians has assisted in blotting out a significant category of possible models of human cognition. Until some models in this category are actually constructed and studied we have no way of knowing how fruitful such research may be.[10]

NOTES

1. For ease I have omitted to describe how the decimal point is to be dealt with. If the decimal representation of y, but not of x, can be calculated by a program that terminates, then once all significant digits of y's representation are exhausted, the machine proceeds to add 0 to each remaining digit of x's representation.

2. 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.

3. This appears to be Penrose's position. He claims that the brain is not equivalent to any Turing machine nor 'to *any* specific oracle machine' (1994: 381). (The term 'oracle machine' is explained in section 3, below.) I discuss Penrose's view in (1997b).

4. In 1993a and 1993b I explain why the Chinese room argument enjoys no success against position (1).

5. In fact Pour-El's own proof is also incomplete. The situation has now been improved by Lipshitz and Rubel (Lipshitz and Rubel 1987, Rubel 1988, 1989).

6. Geroch and Hartle (1986) make a similar point in connection with deriving predictions to within given accuracies from physical theories some of whose constants are Uncomputable numbers.

7. This was related to me by Robin Gandy.

8. This argument is widely endorsed. For example, Chalmers writes 'the (Church-Turing) *universality* of computation ... provides perhaps the best reason for believing in the (functional) AI thesis to begin with' (1996: 390).

9. In point of fact the alphabet of the machines of Turing 1936 was not binary.

10. With thanks to Ned Block, Sean Broadley, Martin Davis, Dan Dennett, Jon Doyle, Bert Dreyfus, Robin Gandy, Allen Hazen, Andrew Hodges, Kevin Korb, Donald Michie, Chris Mortensen, Roger Penrose, Diane Proudfoot, Bob Solovay, and Richard Sylvan for helpful comments and discussion, to Darren Walton for the reference to Fodor 1981, and to Emma Velde for the illustrations.

REFERENCES

Aberth, O. 1968. 'Analysis in the Computable Number Field'. *Journal of the Association of Computing Machinery*, 15, pp.275-99.

Boden, M.A. 1988. *Computer Models of Mind*. Cambridge: Cambridge University Press.

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

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

Bush, V. 1936. 'Instrumental Analysis'. *Bulletin of the American Mathematical Society*, 42, pp.649-69.

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

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

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

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

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

Copeland, B.J. 1993b. 'The Curious Case of the Chinese Gym'. *Synthese*, 95, pp. 173-86.

Copeland, B.J. 1996. 'What is Computation?'. *Synthese*, 108, pp.335-359.

Copeland, B.J. 1997a. 'What is the Church-Turing Thesis?'. Forthcoming.

Copeland, B.J. 1997b. 'Turing's O-machines, Searle, Penrose, and the Brain'. Forthcoming.

Copeland, B.J. 1997c. 'Accelerated Turing Machines'. Forthcoming.

Copeland, B.J., Proudfoot, D. 1996. 'On Alan Turing's Anticipation of Connectionism'. *Synthese*; 108, pp.361-377.

Copeland, B.J., Sylvan, R. 1997. 'Computability: A Heretical Approach'. Forthcoming.

da Costa, N.C.A., Doria, F.A. 1991. 'Classical Physics and Penrose's Thesis'. *Foundations of Physics Letters*, 4, pp.363-73.

Dennett, D.C. 1978. *Brainstorms: Philosophical Essays on Mind and Psychology*. Brighton: Harvester.

Dennett, D.C. 1991. *Consciousness Explained*. Boston: Little, Brown.

Deutsch, D. 1985. 'Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer'. *Proceedings of the Royal Society*, Series A, 400, pp.97-117.

Doyle, J. 1982. 'What is Church's Thesis? An Outline.' Laboratory for Computer Science, MIT.

Dreyfus, H.L. 1992. *What Computers Still Can't Do: A Critique of Artificial Reason.* Cambridge, Mass.: MIT Press.

Fodor, J.A. 1981. 'The Mind-Body Problem'. *Scientific American*, 244 (Jan.), pp.124-32.

Fodor, J.A. 1983. *The Modularity of Mind*. Cambridge, Mass.: MIT Press.

Gandy, R. 1980. 'Church's Thesis and Principles for Mechanisms'. In Barwise, J., Keisler, H.J., Kunen, K. (eds) 1980. *The Kleene Symposium.* Amsterdam: North-Holland, pp.123-48.

Gandy, R. 1988. 'The Confluence of Ideas in 1936'. In Herken, R. (ed.) 1988. *The Universal Turing Machine: A Half-Century Survey*. Oxford: Oxford University Press.

Garzon, M., Franklin, S. 1989. 'Neural Computability II'. Abstract. *Proceedings (vol. I), IJCNN International Joint Conference on Neural Networks*, 631-637.

Geroch, R., Hartle, J.B. 1986. 'Computability and Physical Theories'. *Foundations of Physics*, 16, 533-550.

Gregory, R.L. 1987. *The Oxford Companion to the Mind*. Oxford: Oxford University Press.

Guttenplan, S. 1994. *A Companion to the Philosophy of Mind*. Oxford: Blackwell.

Haugeland, J. 1985. *Artificial Intelligence: the Very Idea.* Cambridge, Mass.: MIT Press.

Hofstadter, D.R. 1980. *Gödel, Escher, Bach: an Eternal Golden Braid*. London: Penguin.

Johnson-Laird, P. 1987. 'How Could Consciousness Arise from the Computations of the Brain?'. In Blakemore, C., Greenfield, S. (eds) 1987. *Mindwaves*. Oxford: Basil Blackwell, pp.247-257.

Kleene, S.C. 1952. *Introduction to Metamathematics*. Amsterdam: North-Holland.

Kleene, S.C. 1967. *Mathematical Logic*. New York: Wiley.

Kleene, S.C. 1987. 'Reflections on Church's Thesis'. *Notre Dame Journal of Formal Logic*, 28, pp.490-98.

Korb, K.B. 1996. 'Symbolicism and Connectionism: AI Back at a Join Point.' In Dowe, D.L., Korb, K.B., Oliver, J.J. (eds) 1996. *Information, Statistics and Induction in Science*. Singapore: World Scientific.

Kreisel, G. 1967. 'Mathematical Logic: What Has it Done For the Philosophy of Mathematics?'. In R. Schoenman (ed.) 1967, *Bertrand Russell: Philosopher of the Century*, London: George Allen and Unwin.

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

Leiber, J. 1991. *An Invitation to Cognitive Science*. Oxford: Basil Blackwell.

Lipshitz, L., Rubel, L.A. 1987. 'A Differentially Algebraic Replacement Theorem, and Analog Computability'. *Proceedings of the American Mathematical Society,* 99, pp.367-72.

MacLennan, B.J. 1994. 'Continuous Symbol Systems: The Logic of Connectionism'. In D.S. Levine, M. Aparicio IV (eds) 1994, *Neural Networks for Knowledge Representation and Inference,* Hillsdale, N.J.: Erlbaum.

Minsky, M.L. 1967. *Computation: Finite and Infinite Machines*. Englewood Cliffs, N.J.: Prentice-Hall.

Newell, A. 1980. 'Physical Symbol Systems'. *Cognitive Science,* 4, pp.135-183.

Penrose, R. 1989. *The Emperor's New Mind*. Oxford: Oxford University Press.

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

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

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

Pour-El, M.B., Richards, I. 1981. ‘The Wave Equation with Computable Initial Data such that its Unique Solution is not Computable’. *Advances in Mathematics*, 39, 215-239.

Pylyshyn, Z.W. 1984. *Computation and Cognition: Towards a Foundation for Cognitive Science.* Cambridge, Mass.: MIT Press.

Rice, H.G. 1954. 'Recursive Real Numbers'. *American Mathematical Society Proceedings,* 5, pp.784-91.

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

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

Rumelhart, D.E., McClelland,J.L., and the PDP Research Group 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Vol.1: *Foundations*. Cambridge, Mass.: MIT Press.

Scarpellini, B. 1963. ‘Zwei Unentscheitbare Probleme der Analysis’, *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 9, 265-289.

Searle, J. 1980. 'Minds, Brains, and Programs'. *Behavioural and Brain Sciences,* 3, pp.417-424.

Searle, J. 1989. *Minds, Brains and Science: the 1984 Reith Lectures*. London: Penguin.

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

Searle, J. 1992. *The Rediscovery of the Mind.* Cambridge, Mass.: MIT Press.

Searle, J. 1997. *The Mystery of Consciousness*. Forthcoming.

Shannon, C.E. 1941. 'Mathematical Theory of the Differential Analyser'. *Journal of Mathematics and Physics of the Massachusetts Institute of Technology*, 20, pp.337-54.

Siegelmann, H.T., Sontag, E.D. 1994. 'Analog Computation via Neural Networks'. *Theoretical Computer Science*, 131, 331-360.

Smith, B.C. 1991. 'The Owl and the Electric Encyclopaedia'. *Artificial Intelligence*, 47, pp.251-288.

Smolensky, P. 1988. 'On the Proper Treatment of Connectionism'. *Behavioural and Brain Sciences*, 11, pp.1-23.

Solovay, R., Yao, A. 1996. 'Quantum Circuit Complexity and Universal Quantum Turing Machines'. Forthcoming.

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

Stephens, P.W., Goldman, A.I. 1991. 'The Structure of Quasicrystals'. *Scientific American*, 264 (Apr.), pp.24-31.

Sterelny, K. 1990. *The Representational Theory of Mind*. Oxford: Basil Blackwell.

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. 1939. 'Systems of Logic based on Ordinals'. *Proceeding of the London Mathematical Society*, 45, pp.161-228.

Turing, A.M. 1948. 'Intelligent Machinery'. National Physical Laboratory Report. In Meltzer, B., Michie, D. (eds) 1969. *Machine Intelligence 5*. Edinburgh: Edinburgh University Press, pp.3-23.

Vergis, A., Steiglitz, K., Dickinson, B. 1986. 'The Complexity of Analog Computation'. *Mathematics and Computers in Simulation*, 28, pp.91-113.

Wittgenstein, L. 1980. *Remarks on the Philosophy of Psychology*. Vol.1. Oxford: Blackwell.

Wolpert, D.H., MacLennan, B.J. 1993. 'A Computationally Universal Field Computer That is Purely Linear'. Santa Fe Institute Technical Report 93-09-056.