happens next in a Turing machine or a polycellular automation has a strictly local character.

Now it is time for me to come down off the fence—on both sides.

1. I think it conceivable that in fifty or a hundred years time there will be machines of the tenth or twentieth generation which in some areas of mathematics will be regarded by first-rate mathematicians as valuable *colleagues* (not merely assistants). The machines will write papers, argue back when criticized, and will take part in discussion on how best to tackle a new problem. This is an updated version of the Turing Test. The conception raises an important point. In his book Penrose asserts forcefully that consciousness is essential to rational thought; I do not quite understand what he means by consciousness. But machines of the kind I am imagining would display the effects of consciousness—the ability to concentrate attention on a particular part or aspect of their input, the ability to reflect on and to alter the behaviour of subsystems in their hierarchic structure, the ability to produce a range of alternatives, and even, in a very restricted way, an ability to adapt their social behaviour.

I remember a long time ago asking Donald MacKay what sort of evidence or knowledge he thought would be required to attribute consciousness to a computer; he replied that it would not primarily be a matter of using evidence and knowledge, but a matter of having the nerve. If my fantasy becomes fact, people will have the nerve.

2. Now for the other side of the fence. Even if machines of that kind are built, discussions about them, explanations of their working, proposals for their improvement will (when not concerned only with the lowest levels of this hierarchical structure) be carried on not in terms of algorithms but in the same sort of terms as we use—as I used—in discussing human intelligence and rational thought. At a sophisticated level it is not practical, nor useful, nor sensible to discuss intelligent behaviour solely in terms of algorithms or machine programs; and this would remain true even if there were some master program producing the behaviour.

8

The Church–Turing Thesis: Its Nature and Status

ANTONY GALTON

1. The Church–Turing thesis (CT), as it is usually understood, asserts the identity of two classes of functions, the *effectively computable* functions on the one hand, and the *recursive* (or *Turing-machine computable*) functions on the other. In support of this thesis, it is customary to cite the circumstance that all serious attempts to characterize the notion of an effectively computable function in precise mathematical terms have ended up by defining the same class of functions, albeit in quite different ways. Thus CT is supported by a series of *theorems* to the effect that these various characterizations of effective computability (viz. Turing-machine computability, general recursiveness, λ-definability, Markov algorithm computability, and the rest) are extensionally equivalent.

Open any text on the theory of computing at the point where CT is discussed. You will find there a statement to the effect that CT is not a theorem, and thus cannot be proved, because one of the terms it contains, namely 'effectively computable', cannot be defined precisely, but rather refers to our vague, intuitive idea of what constitutes computability. This special character of CT as not susceptible to formal verification was already recognized by Church and Turing.[1] Thus Church (1936), in defining the notion

[1] Even before Church and Turing, Gödel had considered what may be thought of as a precursor to CT, that the class of functions computable by means of a finite procedure was coextensive with the class of recursive functions, where however Gödel does not (in contradistinction to Church) provide a precise definition of the latter class. Most pertinent for our present concern is Gödel's remark that this thesis 'cannot be proved, since the notion of finite computation is not defined, but it serves as a heuristic principle' (Gödel, 1934: p. 44).

of an *effectively calculable* function of positive integers, remarks that

> This definition is thought to be justified by the considerations which follow, so far as positive justification can ever be obtained for the selection of a formal definition to correspond to an intuitive notion.

Turing (1936), similarly, notes that

> [a]ll arguments which can be given [for CT] are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. (p. 249)

Despite this, it is almost universally admitted that the equivalence theorems mentioned above, together with various other considerations such as Turing's own account of his motivation for the detailed definition of his machines, constitutes *evidence* for CT. The suppositions that (*a*) CT does not admit rigorous proof, and (*b*) there exists evidence to support it, suggest that CT is more like a proposition of natural science than of mathematics, and thus properly to be regarded as an empirical hypothesis, even though its *subject-matter* appears to be mathematical in character.

2. Let us examine more closely the idea that certain considerations, both formal and informal, constitute evidence for CT. Evidence for a thesis means evidence that the thesis is true; so if it is to be meaningful to adduce evidence for CT, it must at least have a chance of being true. But has it? Doubt has been cast on this by Wang (1974):

> One often hears that in mathematical logic a sharp concept has been developed which corresponds exactly to our vague intuitive notion of computability. But how could a sharp notion correspond exactly to a vague notion?

Wang's own solution to this difficulty is to say that

> [a] closer look reveals that the sharp notion . . . is actually not as sharp as it appears at first sight. (p. 83)

In Wang's view, then, it would appear that the CT really asserts the identity not of a sharp notion with a vague one, but of two vague ones. This will hardly do, though, because, even if Wang is correct in saying that the formal notion is not as sharp as one would like, there can surely be no denying that it is *very much sharper* than our 'vague, intuitive' notion. And this leaves us with the problem of how two notions of very different degrees of imprecision can actually *be* equivalent.

If, as these last remarks suggest, CT is not even something that could possibly be true, then either we must modify it so that it no longer suffers from this defect, or we must revise our views as to its status.

3. Let us take it that the form of CT is aRb, where a is the vague, intuitive term ('effective computability'), b is the precise mathematical term (recursiveness, Turing-machine computability, or what have you), and R expresses the relation that is said to hold between them. In what I shall call the *classical formulation* of CT, a and b denote *classes*—the class of effectively computable partial functions from the natural numbers to the natural numbers, and the class of general recursive functions, respectively—and R is the identity relation. The thesis, on this interpretation, thus asserts that two classes of functions are identical. The unease we noted above arose from the fact that a is not well-defined, whereas b is, or at least much more nearly so, and hence cannot be meaningfully equated to a.

One obvious way in which we might try to alleviate this uneasiness is by redefining the role of the relational term R. Instead of regarding CT as an assertion *that* two classes are identical, we might regard it as having a *stipulative* force, in effect *defining* 'effective computability' to be the same as recursiveness.

This way of looking at the matter is suggested by some formulations of the thesis that one finds in the literature. Hermes (1965), for example, speaks of suggestions to 'make precise' the concept of algorithm and related concepts (including, by implication, effective computability),[2] while Minsky (1967) speaks of 'the thesis that Turing's notion of computability is an acceptable technical counterpart of our intuitive notion of effective computability'. This stipulative understanding of CT is also compatible with Church's original formulation, which uses the idea of 'correspondence'.

[2] Note, incidentally, that Hermes later on comes closer to what I called the classical formulation, when he speaks of the 'identification of the originally intuitively given concept of algorithm with a certain, exactly defined concept'. Likewise, Kleene (1952) explicitly countenances both interpretations when he says that CT 'may be considered a hypothesis about the intuitive notion of effective calculability or a mathematical definition of effective calculability' (pp. 318 f.).

Turing's formulation, on the other hand, is less readily interpretable in this way. His claim is that

> the [Turing machine] computable numbers *include* all numbers which could naturally be regarded as computable.[3] (p. 249)

The fact that Turing is here concerned with the computability of numbers rather than of functions need not worry us in this context: in Turing's treatment, a real number is in any case identified with the function which maps each positive integer n onto the nth place in the decimal expansion of that number. If we assume that Turing would have regarded all Turing-machine computable numbers (or functions) as computable in the intuitive ('natural') sense, it is hard to avoid reading Turing here as asserting the identity of two classes, in other words as stating the classical formulation of CT.

4. If we do opt for the weaker, stipulative interpretation, according to which recursiveness or an equivalent is proposed as a replacement for our ill-defined intuitive notion of computability, what follows? In effect, CT sets bounds on what we can reasonably set out to achieve computationally. It tells us not to waste our time looking for general solutions to the halting problem and other recursively unsolvable problems.

Now the trouble with taking CT purely stipulatively like this is that it does not supply us with a reason *why* we should regard certain prima-facie computable problems as off limits. The classical formulation of CT yields the same advice, and at the same time it *gives us a reason for following that advice*, namely it is a waste of time looking for general solutions to recursively unsolvable problems precisely *because no solutions exist*. This last move can only be made on the classical formulation, because only the classical formulation takes seriously the idea that there is such a thing as the class of all effectively computable functions, even if we cannot characterize it precisely independently of CT itself.

I take it that these considerations imply that the stipulative interpretation of CT, however attractive it may seem at first sight, cannot be seriously upheld. At best, it is arbitrary, and at worst incoherent. Let us then assume that the classical formulation is

[3] Turing (1936), my italics.

correct in so far as it asserts an identity between two classes of functions. By 'correct' here I mean that if there is to be a coherent statement of CT at all then it must be along these lines, however much modified in detail from the classical version. This still leaves us with the tension we noted earlier, arising from the impossibility we thought we discerned in the idea of equating a vague notion with a precise one.

The obvious remedy, if only it could be achieved, would be to try to sharpen the vague term (*a*) in the identity. But *this* seems impossible to do without begging the question: after all, as already remarked, every serious attempt to give a precise characterization of *a* has resulted in a *formal* notion of computability that is provably equivalent to recursiveness, the various proofs of equivalence being precisely what is held up as the main supporting evidence for the thesis. We thus seem to have the curious circumstance of a corpus of exact mathematical results being held up as evidence for a thesis which it is impossible to state clearly while maintaining its distinctness from those results.[4]

In the next two sections I consider what happens if we try to sharpen the intuitive notion *a* in the thesis *without* going so far as to define something that is already provably equivalent to the sharp term *b*. Note that there is an air of paradox about this plan: I am deliberately setting out to produce a formulation of CT that stands a chance of being true but which cannot be proved. The unprovability is necessary here, for an attempted formulation which *could* be proved would no longer be CT, but a piece of evidence for the 'real' CT—if such a thing exists. So my aim here is to see whether or not CT *can* be coherently formulated at all.

5. Gandy (1988) has argued that, contrary to received opinion, CT *is* a theorem, which he states in the following form:

> Turing's Theorem: Any function which is effectively calculable by an abstract human being following a fixed routine is effectively calculable by a Turing machine . . . and conversely. (p. 83)

[4] Kalmár (1959: 79) regards this as a reason for rejecting CT: 'There are pre-mathematical concepts which must remain pre-mathematical ones, for they cannot permit any restriction imposed by an exact mathematical definition. Among these belong . . . such concepts as that of effective calculability, or of solvability, or of provability by arbitrary correct means, the extension of which cannot cease to change during the development of Mathematics.'

But in order for this to be a theorem in the mathematical sense, a determinate proposition capable of being definitively proved, the terms occurring in it must all be definable in a precise way; in particular we need a precise definition of the term 'abstract human being'. If this is defined by means of a purely mathematical construction, the question arises how the computational power of this construction is related to that of real human beings, or real computing machines: in effect the core issue addressed by CT is pushed back, with Gandy's *a* term ('abstract human-being computability') becoming the new *b* term, the new *a* term reverting once more to the original imprecisely formulated notion of effective computability.

If, on the other hand, the notion of an abstract human being is not defined in such a way as to allow 'Turing's theorem', as stated by Gandy, to be proved, then it does not seem correct to call this formulation a theorem. If an abstract human being is obtained from a real human being by abstracting away 'the—very important—practical limitations on time and space' (Gandy, 1988: 83), then Gandy's version of CT begins to look very much like an empirical hypothesis about the computational powers of human beings. More often, though, CT has been understood as making a broader claim, about what *any* entity, real or abstract, can compute, not specifically about human beings. Thus while Gandy's formulation, in which the *a* term is made more precise without however becoming provably equivalent to the *b* term, is a step in the direction we have set ourselves to explore, it is arguably not general enough to play the role traditionally assigned to CT, that of defining the limits to computation *in general*.

If now we consider the various formal notions that have been used as the second, precise term of CT, i.e. recursiveness, Turing-machine computability, λ-definability, etc., we note that each of them gives us a *computationally specific* way of defining functions. By this I mean that each of them characterizes a class of computable functions by giving us a framework within which we can actually compute them. The Turing machine model of computability, for example, provides a precise specification of the class of all Turing machines. Each Turing machine does the job of computing the values for a particular function. Likewise, the scheme of primitive recursion plus the μ-operator shows us how to construct individual function definitions which we can use to compute values of the functions thereby defined.

My point here is that each of the formal notions provides a *realization* or *implementation* of the general notion of a computable function. Now the complement, as it were, of implementation is *specification*. A specification is precise about *what* is wanted but is silent on *how* it is to be achieved; an implementation does not in itself tell us *what* it gives us, but it is explicit about *how* it does it. And the relation between specification and implementation is, ideally, that a specification is *correctly realized* by an implementation.

If, then, we regard the specific models of computation which belong on the right-hand side of CT as implementations, then what goes on the left-hand side must be a specification. But so far, it is only a vague specification—that is the root of our problem. What we need to do is to put down the specification precisely: after all, a specification ought to be no less precise, in its own way, than an implementation, the difference being that, as already indicated, what they are precise about are quite different things.

Our task, then, is to find a precise term which can stand in relation to recursiveness and its formal equivalents as specification to implementation. We require a notion of computation which is precise as to what sort of thing results from a computation but is vague as to how it is achieved. To this end I propose to examine the following notion of *black-box computability*:

> A partial function f from the natural numbers to the natural numbers is *black-box computable* if there could exist a black box with input and output ports and a representation scheme for natural numbers, such that for each natural number n, when the representation of n is fed into the input port of the black-box then, if $f(n)$ is defined, its representation will eventually emerge from the output port, whereas if $f(n)$ is not defined, either nothing will ever emerge, or what emerges is not the representation of a natural number using the specified scheme.

This definition says nothing at all about the nature of the mechanisms inside the black box: anything will do, so long as the outward behaviour of the box is as specified. Thus black-box computability is defined in a very different way from Turing-machine computability, for which the details of the mechanism are crucial (even though to some extent arbitrary in the sense that they can be changed without affecting the notion of computability thereby defined).

6. The 'black-box formulation' of CT is that the class of black-box computable functions is equal to the class of recursive functions. In effect, this version of the thesis states that whatever mechanism a particular black box contains, it can be replaced by a Turing machine without altering its functional specification.

Is the black-box formulation of CT precise enough to avoid the incoherence we imputed to the classical formulation? It is surely not yet *perfectly* precise. A conspicuous source of vagueness lies in the use of the words 'there could exist'. In order to make precise what is meant here, we should need to specify exactly what space of possibilities is intended.

To appreciate the problems involved in doing this, consider the following black box. It is simply a form of universal Turing machine. But it has a special property: in any computation that it performs, the first step takes one second, and each subsequent step takes half the time of the one that precedes it. When a numerical encoding of any given Turing-machine/tape pair (M, T) is presented to the input port of the black box, the black box immediately proceeds to simulate the computation that would occur if the Turing machine M were to be activated with initial tape-configuration T. If the simulation reaches a 'halt' state after a finite number of steps, the black box halts and delivers the output 1 (meaning that the computation specified by the input terminates). But if after two seconds the simulated computation has not reached a 'halt' state, then the black box halts and delivers the output 0 (meaning that the computation specified by the input does not terminate).

Clearly, this black box solves the halting problem for Turing machines. Does this mean that the halting problem (or rather, the corresponding numerical function) is black-box computable? It all depends on the space of possibilities underlying the requirement that an appropriate black box 'could exist'. The black box I have described could not exist in physical form in our universe, although one can, perhaps, conceive of alternative possible universes in which it could: it is, after all, mathematically consistent. When we describe an abstract model of computation, it is customary to insist that it produce its result after only finitely many steps, because only in this case can the abstract process be realized physically: ultimately what interests us is the actual physical computations.

It therefore seems reasonable to introduce constraints into our understanding of what is possible so that the black boxes referred to in the definition of black-box computability could exist physically. If we do this, though, do we have any reason to suppose that there will be *any* correspondence between black-box computability and Turing-machine computability? For on the one hand, to quote Deutsch (1985: 101), 'there is no *a priori* reason why physical laws should respect the limitations of the mathematical processes we call "algorithms" ', which suggests that the introduction of physical considerations, so far from being a *limitation*, might allow us to go beyond the bounds of Turing-machine computability. And on the other hand, to quote Deutsch again,

> [n]or, conversely, is it obvious *a priori* that any of the familiar recursive functions is in physical reality computable. The reason why we find it possible to construct, say, electronic calculators, and indeed why we can perform mental arithmetic, cannot be found in mathematics or logic. *The reason is that the laws of physics 'happen to' permit the existence of physical models for the operations of arithmetic* such as addition, subtraction and multiplication. If they did not, these familiar operations would be non-computable functions. We might still know *of* them and invoke them in mathematical proofs (which would presumably be called 'non-constructive') but we could not perform them. (p. 101)

Even if we suppose that there is sufficient harmony between the physical world and the abstract world of mathematical logic for arbitrary Turing machines or their equivalents to be constructible in principle, it will still be the case that many, indeed most, Turing-machine computable functions will fail to be black-box computable, because the resources they require exceed the capacity of the physical world to supply them. After all, even so simple a function as addition can only be computed physically for arguments up to a certain size, and there must certainly exist infinitely many Turing-machine computable functions for which the computation for *any argument(s)*, using Turing machines or their equivalent, exceeds the resource bounds imposed by the physical world. If CT is right, these functions cannot be computed by any physical means at all.

Unfortunately, it seems impossible to specify the physical constraints on our black box in a non-arbitrary way. What we say here must clearly be influenced by the current state of our knowledge of the extent of the physical world, both in the large and in

the small, and also by our current technological capacities. As Enderton (1977) puts it,

> A person with a digital computing machine may regard a function f as being computable only when $f(x)$ is computable on his machine in a reasonable length of time. Of course, the matter of what is reasonable may change from day to day. And next year he hopes to get a faster machine with more memory space and tape drives. At that time, his idea of what is computable in a practical sense will be extended considerably.
>
> The class of effectively computable functions is obtained in the ideal case where all of the practical restrictions on running time and memory space are removed. Thus the class is a theoretical upper bound on what can ever in any century be considered computable. (pp. 529 f.)

Enderton here endorses the traditional view that purely physical considerations are not generally considered to be a part of the theory of computability.

But this has the result that CT, as ordinarily formulated, tends to be seen as irrelevant to the issue of defining what can or cannot be computed in practice. To quote von Neumann (1961),

> Throughout all modern logic, the only thing that is important is whether a result can be achieved in a finite number of elementary steps or not. The size of the number of steps which are required, on the other hand, is hardly ever a concern of formal logic. Any finite sequence of correct steps is, as a matter of principle, as good as any other. . . . In dealing with automata, this statement must be significantly modified. In the case of an automaton the thing which matters is not only whether it can reach a certain result in a finite number of steps at all but also how many such steps are needed. (p. 303)

We thus find, in conventional computability theory, on the one hand the requirement that a computation be completed in finitely many steps, and on the other hand the lack of any constraint on the complexity of a computation. The justification for the former requirement is that infinite computations cannot actually be implemented physically; the justification for the latter non-requirement is that our theory of computation is purely abstract and hence should not be constrained by physically motivated considerations. There is a tension here which needs to be resolved if CT is to be seen as having direct relevence to practical computational issues. What is required here is an answer to what is perhaps the central question in all this, namely: when is a computation procedure *effective*?

Even if we do not insist that the black box should be able to exist and perform its computations physically, one might suppose, *pace* Deutsch, that this condition should be sufficient in the sense that the possibility of physical realization should qualify a black box for the role required of it in the black-box formulation of CT. But as J. R. Lucas pointed out in discussion (see this volume, p. 104), there would be no reason, on this supposition, to disqualify a black-box which contained a human being who is responsible for deciding what output to give in response to each input. If we accept this (in effect allowing that the black box might *be* a person), then the black-box formulation of CT becomes identical with the ('strong AI') thesis that people too are subject to all the limitations of Turing machines, a position which Lucas himself has consistently argued against.

A more modest formulation of black-box computability seems to be in order. What further constraints should we add? One possibility is to insist that we can, in principle, *construct* the black box. This amounts to a substantial limitation inasmuch as we have no idea how to set about constructing a black box that is in any relevant sense equivalent to a person. But in 'we can construct' there appears once again a 'can', and we have to ask more closely just what sort of possibility or potentiality we require here.

One obvious answer is that there should be an effective procedure for constructing the black box. But if we insist on this then the black-box formulation of CT becomes circular: a function is effectively computable if and only if there is an effective procedure for constructing a black box which will compute it (cf. Péter's argument against the classical formulation of CT, discussed below). In order to break out of the circle we need to define what we mean by *effective*, and this is just what we were originally trying to use CT to tell us.

Suppose now we try to pin down what we mean by calling a black box effectively constructible by specifying what sorts of operations are to be allowed in the construction of it. At this point we would be well on the way to converting what we intended as a completely general specification of a computational model into a specific implementation. In short, we are following the same path as Turing when he originally invented Turing machines.

We thus find that while there is something very attractive about the notion of a black box standing in relation to a class of

particular models of computation as specification to implementations, it is extremely hard to make the specification precise without converting it into one of those implementations. This is a situation curiously reminiscent of that prevailing in artificial intelligence (AI). It has often been claimed that the classical 'specify-and-verify' methodology advocated in software engineering is inappropriate to AI precisely because it is not possible in AI to specify in advance the behaviour (essentially some aspect(s) of human behaviour) that one is attempting to realize computationally. Instead, proponents of AI tend to favour a methodology of 'exploratory programming', in which the programming process is guided by the programmers' confidence that although they cannot specify in advance the behaviour they are aiming for, they will at any rate recognize it when they see it. In a similar way, one might say of CT that even though one cannot give a general characterization of what sort of thing is to count as a computation, one might be able to recognize, informally, that such-and-such a specific formal model of computation succeeds in capturing that notion precisely.

7. In order to clarify the status of CT, a good plan would be to consider in what ways the thesis might come to be discredited. We may discount here the remote possibility that the proofs of equivalence for the various formal notions of computability might turn out to be flawed, thereby undermining the chief mathematical grounds for accepting CT. If this were to happen, CT would in effect be broken up into a collection of *rival* theses, each stating that a function is computable if and only if it has a certain formal property, viz. Turing-machine computability, recursiveness, computability by a normal Markov algorithm, and so on. The big question then would be which, if any, of these rival theses was correct.

We assume in what follows that we do have a robust formal notion of computability, and the question at issue concerns the relationship between this formal notion and our intuitive notion of computability. Here we are primarily concerned with the possibility that CT may come to require *revision*. We can envisage two kinds of revision, which differ according to whether it is the '*a*' term in the '*aRb*' presentation of the thesis that is changed, or the '*b*' term. Revision of the former kind amounts to *conceptual revision*, because it involves a shift in what we want to understand by the notion of effective computability. To advocate conceptual revision of CT is not necessarily to stigmatize the older version of the thesis as false, but only to recommend the replacement as in some way more appropriate or relevant to one's needs. Revisions of the second kind, however, are *substantial*, since they do directly contradict the version that is replaced: whereas the older version identifies the effectively computable functions with the recursive functions, the replacement identifies the same set of effectively computable functions with some other class.

We can also classify possible revisions of CT according to whether they postulate that the class of computable functions is more or less inclusive than in the classical version. We may speak of *upward* and *downward* revisions respectively.

8. Substantial revision of CT would be necessitated if someone were either to discover an effective method for computing some function that is demonstrably not recursive (upward substantial revision), or if it were to be shown that not all recursive functions were effectively computable after all (downward substantial revision).

Downward substantial revision hardly seems possible, since we already know how to specify a black box to compute any given recursive function: it is called a Turing machine. We must not be too hasty, however. To be sure, if we *construct* a Turing machine, then it is reasonable to claim that the function it computes is effectively computable. To extrapolate from this observation to the notion that an arbitrary function can be regarded as effectively computable if and only if there exists a Turing machine which can compute it, however, is not necessarily legitimate. The problem, as was pointed out by Péter (1959), is that the notion of existence represented by the phrase 'there is' has to be defined constructively in order for it to play the role required of it in this formulation of CT: we need to insist, in other words, that we can effectively construct a Turing machine to do the job. But we cannot insist on this without circularity, since the whole purpose of CT is to provide a *definition* of the notion of effectiveness.

In order to avoid this circularity, the best we can do is to restrict our notion of effective computability to cover those classes of functions for whose Turing machines we have up to now discovered

effective constructions. But to do this is, in effect, to propose a substantial downward revision of CT, albeit one for which no universality can be claimed, since it is not possible to predict what advances may be made in our ability to specify effective methods for constructing Turing machines.

If we reject this argument (and it is surely not totally compelling), then the only other way in which downward revision of CT could occur would be if we were to argue that some Turing-machine computations, for example those which require exponential time, do not really count as computations. But this would be a conceptual revision of CT, not a substantial one. We shall consider this case in its proper place.

Upward revision of the thesis, on the other hand, could be genuinely substantial, involving the discovery of a new technique. Of course, we cannot say what such a technique would be, though we might suspect that the appropriate direction to look would be in the area of either analogue computation or connectionism. We do not yet know enough about the formal relations between these two kinds of computation and ordinary digital computation for us to be able to rule such a discovery out of court. A few general observations are possible, however.

Suppose someone presents us with a black box which he claims can solve the halting problem for Turing machines. If this claim is true, then the black box can compute a function which no Turing machine can, a non-recursive function. But if the black box can compute it, then it must be computable, in our informal intuitive sense. Hence CT is false.

That is the claim made by the person who made the box. But why should we accept it? How do we *know* that the black box really does solve the halting problem? We might try it out on a large number of examples for which we already know the answer, and find that in every case the correct result appears. But this does not *prove* that the machine has the capacity claimed for it. For what is claimed is that the black box will correctly determine termination or non-termination for *every possible* machine/tape pair. Nothing short of this will do: after all, we already have respectable recursive techniques for testing large classes of *algorithms* for termination, and there is no reason to suppose that such techniques could not be adapted to Turing machines instead. What we lack, and this black box purports to provide, is a *general* technique.

How could the owner of the black box convince us that it does what he claims it does? Only by revealing to us its *mechanism*. Unless he can prove that what goes on inside the box is always such as to lead from an input machine/tape pair to a correct determination of whether or not the corresponding computation terminates, we have no reason whatever for accepting his claim.

In saying this, I am in no way prejudging the issue as to what kind of thing is going on inside the box. However, since both the input and output are in discrete, digital form, it is natural to expect that the inner workings of the box will have this form too. In that case what goes on in the box must be a sequence of operations (or perhaps a set of such sequences in parallel). This sequence can, presumably, be broken down into a set of primitive operations together with a set of controlling principles which govern the way in which the primitive operations are combined to yield the completed sequence. Some of these operations and controlling principles will doubtless be things that can be performed by a Turing machine, but *at least one* of them must be beyond the capability of such a machine.

Now one is inclined to think that a primitive operation must be rather simple. Turing's original construction of his machine was motivated by a close analysis of all the possible processes which might form part of any computational technique. Here we are postulating the existence of some simple operation that Turing missed; something sufficiently unobvious for all subsequent researchers to have missed too. It does not seem very likely, but on the other hand we cannot entirely rule out the possibility. In this connection, Webb (1983, p. 337) has pointed out that the only plausible candidate for an effective operation that might be expected to take us beyond the realm of the recursive, namely diagonalization, actually fails to do so, inasmuch as the class of general recursive functions is closed under this operation.

But perhaps we are begging the question in assuming that the mechanism inside the box has to be analysable into a sequence of discrete steps. Perhaps it is an essentially continuous process which cannot be redescribed in discrete terms, even though its input and output are both discrete. We enter here the realm of speculation and it is very difficult at this stage to say much about what may or may not be possible.

But *if* such a discovery were to be made, there would be a need

for us, as upholders of CT, to respond in some way to it. The more conservative approach would be to insist that the new discovery just doesn't count as a case of black-box computability, for example because the relevant internal mechanism of the box is not discrete. This would amount to a conceptual revision of CT, and properly belongs in the next section. It would enable us to retain our faith in CT, but is somewhat desperate, and tantamount to a rejection of the black-box formulation of the thesis, since it replaces black-box computability by *discrete* black-box computability.

A more radical response would be to try to revise CT to accommodate the new technique. This would require us to try to extend the formal term, i.e. to look for a new formal account of computation which coincides in extension with the enhanced notion of computability provided by the new computation techniques. Once again, it is impossible for us to say, at our present state of knowledge, what such a formal account would look like.

9. We now turn to conceptual revisions of CT. As I remarked in the previous section, downward conceptual revision of CT would occur if, as is perfectly possible, we came to embrace a notion of effective procedure that incorporated a resource bound, such as polynomial time, as an intrinsic part.

To some extent, this has already happened, in that it is common to regard as *feasible* only those computations which can be performed in polynomial time. So an exponential algorithm, on this view, is not an effective procedure. We could justify adopting this line on the grounds that feasibility, as here defined, is, like recursiveness, a *robust* notion, that is, it remains constant over a wide range of possible modifications of technique and representation (cf. Harel, 1987).

Suppose then that computer scientists very generally came to accept that a function is effectively computable if and only if it can be computed by a black box in polynomial time. Let us call this *polynomial black-box computability*. If we simply substitute polynomial black-box computability for black-box computability *tout court* in our black-box formulation of CT, we would obtain the thesis that polynomial black-box computability is equivalent to Turing-machine computability. But this is highly implausible: it amounts to the claim that any Turing machine, even

if its computations exceed polynomial time, is equivalent to a black box whose computations only require polynomial time.

For that reason, the natural response to the feasibility requirement is to revise CT downwards so that it equates effective computability with polynomial Turing-machine computability. But the black-box formulation of this revised CT comes out looking suspiciously like a special case of the original: it says that polynomial black-box computability equals polynomial Turing-machine computability. In view of the invariance of the major complexity classes across different models of computation, this formulation of CT does not look all that different from the original black-box formulation. And as I shall indicate, the new formulation gives rise to a problem that we do not find in the original.

For on the one hand it is quite easy to design a standard language for the description of Turing machines, in such a way that there is an effective decision procedure for determining whether an expression in this language is the description of a Turing machine or not. Similar remarks apply to recursive function expressions, λ-expressions, normal Markov algorithms, and so on. In other words, the class of entities which CT, in any of its original forms, associates with the intuitive notion of effective computability, is itself effectively computable.

On the other hand, by contrast, there does not appear to be *any* effective way of identifying those Turing machines which perform all their non-terminating computations in polynomial time. In other words, the formal term in the correspondence posited by the downward-revised CT is itself not amenable to effective determination. This makes things very awkward from the point of view of *applying* the thesis. Currently, if one holds the original form of the thesis, it is enough to find a Turing machine which computes a given function for one to be satisfied that that function is computable; there need never be any doubt that what is offered *is* a Turing machine, though admittedly proving that it does what is claimed for it can be tricky, and is in general recursively unsolvable. But with the downward-revised version of the thesis, things are much worse, for now it is, in general, impossible to be sure whether what is offered is a polynomial-time Turing machine, let alone that it computes the function in question.

Because of this difficulty, the downward-revised CT which identifies effective computability with computability by Turing machine

in polynomial time has much less to commend it than the original. The original CT, if we accept it, in effect succeeds in identifying an effective implementation of black-box computability. The revised version, on the other hand, does not. Instead, it merely replaces one specification with another: for it insists that a Turing machine have polynomial complexity, without providing any implementational details as to how this can be achieved.

For the downward-revised CT to have more 'punch', it would be necessary that we discover some *new* construction, which can be specified and recognized effectively, and which is either exactly equivalent to the polynomial-time Turing machine, or sufficiently nearly so to be a satisfactory replacement for it in the revised CT: at all events, we require some implementational schema which is guaranteed to produce polynomial computations. As far as I am aware, no such construction has ever been found, nor is it at all clear how one might set about looking for one (cf. Gurevich, 1988). As Minsky (1967, §8.5) says,

> Those who object that Church's (or Turing's) thesis . . . allows too much, usually do so on the grounds that the Turing machine formulation of computability allows computations whose lengths cannot be bounded in advance in any reasonable way. The impossibility of computing bounds . . . is one of the obstacles that seems to stand in the way of finding a formulation of computability which is weaker yet not completely trivial. (p. 153)[5]

An alternative here would be to shift the focus from computational *mechanisms* to computational *problems*. Our concern, after all, is to characterize what is or is not computable; traditionally, we have done this by defining an abstract model of computation and then saying that a function is computable if and only if it can be computed using that model. This is fine so long as we are not interested in resource bounds; but as we have seen, it is very difficult to modify a given abstract computational model in such a way that the computations it defines have some complexity bounds specifiable in advance.

The work reported in Stewart (1996) gives ground for hope that the problem-oriented approach suggested here might meet with more success than the traditional, mechanism-oriented approach. This work shows that one can characterize the complexity of problems

[5] Cf. Rogers (1967, §1.1).

in terms of the logical resources needed to specify them. A given logic $\mathcal{L}$ might correspond to a complexity class of computational problems $\mathcal{C}$ in the sense that a problem belongs to $\mathcal{C}$ if and only if it can be specified using the logic $\mathcal{L}$. A resource-bounded formulation of CT might then run something like this: A function is effectively computable if and only if it can be defined in the logic $\mathcal{L}$ (where the particular logic $\mathcal{L}$ chosen will depend on what complexity of computations one is prepared to countenance as effective).

10. What about upward conceptual revision? An argument of Kalmár (1959) appears to fit into this category. Kalmár considers functions of the form

$$\psi(x) \equiv \begin{cases} y & \text{if } y \text{ is the least natural number such that } \phi(x, y) = 0 \\ 0 & \text{if } \phi(x, y) \neq 0 \text{ for every natural number } y, \end{cases}$$

where ϕ is general recursive. Kalmár claims that any function of this form is effectively computable by means of the following procedure:

> Calculate in succession the values $\phi(p, 0)$, $\phi(p, 1)$, $\phi(p, 2)$, . . . and simultaneously try to prove by all correct means that none of them equals 0, until we find either a (least) natural number q for which $\phi(p, q) = 0$ or a proof of the proposition stating that no natural number y with $\phi(p, y) = 0$ exists; and consider in the first case q, in the second case 0 as result of the calculation.

Yet Kleene (1936) had shown that some functions ψ of this form are not general recursive. It follows that, if CT is correct, then for suitable ϕ and p, none of the numbers $\phi(p, 0)$, $\phi(p, 1)$, $\phi(p, 2)$, . . . is zero and yet this fact cannot be proved by any correct means. Rather than accept this 'very strange consequence', Kalmár concludes that the totality of 'correct means' of proof goes beyond the realm of the general recursive, thus falsifying CT.

Kalmár is not here postulating the existence of some specific technique for computing non-general-recursive functions. Rather, by allowing 'any correct means' of proof, he would appear to be expressing the belief that no particular collection of techniques can ever be regarded as exhaustive. Only by laying down in advance what techniques of proof and calculation are allowable does it appear plausible that the realm of the effectively calculable is limited to the general recursive functions. There is no reason to suppose

that, if arbitrary computational procedures are allowed, this limitation cannot be transcended.

There is certainly something quite attractive about Kálmar's view, but it has not been widely adopted. It is, in some ways, the opposite of the position held by Péter, which we discussed above. Péter felt that the notion of 'arbitrary recursive function' was too general, and that for the computation of a function to be effective in a way that does not simply beg the question (because of circularity) it must involve some more specific form of recursion; Kalmár, on the other hand, wants to allow 'arbitrary correct means' of computation, which he believes can take us beyond the realm of the general recursive.

Another kind of upward conceptual revision might occur if it could be shown that the notion of black-box computability we have been using is too narrow to characterize everything that we should be prepared to count as a computation. For according to this notion, a computation takes a single input and delivers a single output, and moreover, both the space of possible inputs and the space of possible outputs must be discrete. Only if these conditions are met can we legitimately think of a computation as computing a function from the natural numbers to the natural numbers.

Now there are at least two ways in which these conditions might be regarded as inadequate. On the one hand, one might seek to relax the requirement that what is computed is a relation between an initially given input and a final output. On the other hand, one might want to consider 'computations' in which the input space or output space (or both) is continuous rather than discrete. And of course, one may well wish to make both of these moves at the same time. In the next section we will consider the implications for CT of the first possibility, and in the section after that we shall consider the implications of the second.

11. Pnueli (1985) distinguishes two views of computational systems:

> The first view regards programs as *functions* from an initial state to a final state . . . This view is particularly appropriate for programs that accept all of their inputs at the beginning of their operation and yield their outputs at termination. We call such programs *transformational* . . . On the other hand, there are systems that cannot be covered by the transformational view. Some systems, such as operating systems, process control programs, seat reservation systems, etc., ideally never terminate. Moreover, the purpose for which they are run is not to obtain a final result, but rather to maintain some interaction with their environment. We refer to such systems as *reactive systems*. (pp. 510f.)

Pnueli indicates that the techniques used for reasoning about transformational programs, e.g. to prove their correctness relative to some specification, cannot be applied to reactive programs, and goes on to advocate temporal logic as a suitable tool for reasoning about the more complex correctness requirements of the latter class.

From our point of view, it is necessary to determine whether reactive programs involve a notion of computation that goes beyond the kind of computation considered in CT, the kind of computation which, according to CT, can always be performed on a Turing machine. To this end, let us define a *reactive black box*. This does not simply accept an input and, after executing some processes hidden from our view, deliver an output; instead, it generates a (possibly infinite) output stream $O_1O_2O_3\ldots$ in response to a (possibly infinite) input stream $I_1I_2I_3\ldots$, with no requirement that the input be completed before the output begins.

Can we replace a reactive black box by an ordinary (transformational) black box with the same computational characteristics? In fact, we can do so quite easily. What we need to do is to see the reactive black box as computing a function from natural numbers to natural numbers. Now the class of all possible infinite input streams is non-denumerable and hence such streams cannot be replaced by natural numbers in a one-to-one fashion. Instead we exploit the following *monotonicity* property of reactive systems: if the finite input stream I yields the finite output stream O, then any extension of I must yield an extension of O. This property is a necessary consequence of the reactive nature of the system, which cannot know, once it has received input stream I, whether or not further inputs will arrive, and hence its response must be such as to be appropriate in either case.

Hence if input stream $I_1I_2I_3\ldots$ yields output stream $O_1O_2O_3\ldots$ then the finite input streams I_1, I_1I_2, $I_1I_2I_3$, . . . must yield the finite output streams O_1, O_1O_2, $O_1O_2O_3$, . . . , respectively, and moreover, this denumerably infinite set of finite input–output pairs uniquely determines the single infinite input–output pair we began with. Moreover, the set of *all* finite input–output pairs for the system

is denumerable and yet uniquely determines the non-denumerable set of all infinite input–output pairs. It follows that the complete behaviour of a reactive system can be specified by a denumerable set of input–output pairs, and hence can be regarded as computing a function from the natural numbers to the natural numbers.

A natural objection to this analysis is that a reactive system will not always respond identically to identical inputs, and hence the system cannot be considered to be computing a function. Thus in a certain well-known operating system the output corresponding to the input 'ls' will differ according to the contents of the current directory. But this is to miss the point. Let us call the state of the system when it is first installed, before it has performed any computations at all, the 'zero state'. The argument above, which showed that the reactive black box could be replaced by an equivalent transformational one, must be taken as referring to the zero state of the system. After performing some computations, the reactive system will, in general, be in a different state from the zero state. There will, correspondingly, be a different transformational black box corresponding to the system in this state. That is why identical inputs received by a reactive system at different times may yield different outputs: the equivalent transformational systems at the two times may be different. But this in no way lessens the force of the original argument.

To conclude this section then, it seems clear that, while reactive systems are such that they require very different techniques from a practical point of view, from the point of view of computability theory they still fall within the terms of reference of what I have called black-box computability, and hence CT, if it is true, must apply to reactive systems just as much as to transformational ones.

12. The second extension to the notion of computability that we envisaged was to allow the input and output spaces to be continuous. A possible motivation for this would be that such an extension might allow us to regard many physical processes as computations which would otherwise be hard to describe in this way.

A major problem facing any attempt to define computations over continuous spaces arises from the finiteness requirement discussed in section 6. An effective computation must be completable in a finite amount of time. We must include in this the time taken to

enter the input into the black box and to retrieve the output. If the input and output spaces are to be truly continuous, it is not possible for the data to be presented digitally, i.e. as strings of symbols taken from a finite alphabet, for the only way a truly continuous space can be represented in this way is for us to allow expressions containing an infinite number of digits (cf. the real numbers).

But if the data are not presented digitally, they must be presented in an 'analogue' way, i.e. by means of some continuously variable physical quantity. The process of retrieving the output then becomes one of *measurement*, and the limitations of this kind of computation can only be determined by a close consideration of the physical nature of measurement. Fields (1989, and see p. 168 below) argues that if we accept the nonclassical (quantum mechanical) conception of measurement then even continuous systems cannot compute anything that a Turing machine cannot. This suggests that inclusion of continuous input and output spaces does not, in fact, take us beyond the class of Turing-machine computable functions. This conclusion is not contradicted by that of Deutsch (1985), who claims that his Universal Quantum Computer has 'remarkable powers not reproducible by any Turing machine'—which do not, however, include the computation of non-recursive functions.

But even apart from this issue, there are reasons to treat cautiously any proposal to extend the notion of computation to cover continuous processes. It seems plausible that *any* physical system can be regarded in indefinitely many different ways as a black box which takes input from a continuous space of possibilities and delivers output from a continuous space of possibilities. On the other hand, to extend our notion of computation so that anything that happens is a computation reduces the whole enterprise to absurdity. It is therefore necessary, if we are to extend CT so as to encompass computations with continuous input and output, to find a way of specifying what properties a process must have in order to count as a computation in this sense. A preliminary attempt to do this (Sloman, 1988, and see also his paper in this volume, pp. 179–219) suggests that when we start looking at things in this way, what we will find is not just a single replacement of CT but a whole battery of replacements, corresponding to different views, all equally legitimate, of what constitutes the essence of computationality.

13. I conclude with some remarks concerning the relevance of CT to artificial intelligence. In view of the difficulty of finding a satisfactory formulation of CT that encompasses systems whose input and output form continuous spaces, I shall restrict the discussion to the discrete input–output case. This is not a restriction from the point of view of conventional AI, since the tool of this trade is the digital computer, which does indeed satisfy the discreteness condition. Some, but possibly not all, forms of connectionism can probably be accommodated under this rubric too. So our terms of reference are certainly broad enough to be getting along with, even if we should eventually want to broaden them.

Depending on one's viewpoint, CT might be used either in support of the thesis that all human cognitive processes can be simulated on a machine, or to argue against it. The two arguments, both of which are valid, run as follows:

1. A function is black-box computable iff it is Turing-machine computable.
   A human being is a black box.
   Therefore, a human being is equivalent to a Turing machine.
2. A function is black-box computable iff it is Turing-machine computable.
   A human being is not a black box.
   Therefore, a human being is not equivalent to a Turing machine.

In both cases, the claim rests on two premisses. The first premiss in each case is simply CT. The second premiss is, in the first case, that human beings can be regarded as black boxes in the sense required by CT; and in the second case, that they cannot. The crucial question is therefore whether or not we can legitimately regard a human being as a black box in the sense required by the black-box formulation of CT.

It need not matter here that human cognition appears to be separable into a collection of processes each enjoying a certain autonomy. This does not mean that we cannot consider the aggregate of such functions as a single superprocess representing the total cognitive activity of a human individual. A collection of black boxes packed into one big black box can itself be regarded as a black box. The input to the big black box is in turn fed into one or

more of the internal black boxes; the eventual output is determined in some way, either systematic or random, from the output(s) produced by the internal black boxes.

Nor, as we have seen, is it a serious consideration that people resemble reactive systems rather than transformational ones.

The crucial issue has rather to do with the functionality of a human being *qua* black box. For CT to get any purchase, we must be dealing with inputs and outputs taken from some denumerable space of possibilities. Now it is not entirely clear that, for the human, this is the case. *Sensory* input, to be sure, may well be in the last analysis digital, since it is limited by thresholds as regards both the extent of its range and its resolution power. But cognition is not a function of sensory input alone. One's thinking is deeply affected by one moods and feelings, and these depend, in part, on the state of one's body chemistry, e.g. the concentrations of various chemicals in the bloodstream, which in turn are influenced by external non-sensory inputs such as food and air.

A computing mechanism is not specified until its behaviour over the whole range of predetermined possible inputs is specified. Now, for a human, *what counts as an input, and what counts as an output*? Is the cheese sandwich I had for lunch an input? I don't just mean the taste and feel of the sandwich: being sensory these are inputs in the relevant sense, sure enough; I mean the sandwich itself. It went into my stomach, some of its molecules are by now doubtless coursing through my brain: so the food I eat cannot be wholly separated from my cognitive activities. As Dennett (1978, ch. 13) puts it,

> we have to decide which of the impingements on the animal count as input and which as interference, and it is not at all clear what criteria we should use in deciding this. (p. 260)

Later, Dennett draws the even more despairing conclusion that

> By suitable gerrymandering . . . it ought to be possible to interpret any man as any Turing machine—indeed as all Turing machines at the same time. (p. 262)

which recalls my earlier worries about the infinity of ways of describing a given physical system as a continuous black box.

Now it *may* be that these factors can be dismissed as irrelevant. One might, for example, adopt the stance that human cognitive

behaviour as we actually observe it must be conceived as a *performance* which only imperfectly realizes an underlying cognitive *competence*, and that it is the latter which is the true object of study in AI.[6] The imperfect transition from competence to performance can be explained, on this view, as arising from just such messy biochemical considerations as we have already considered. The cheese sandwich (as opposed to its sensory qualities) is not a cognitively relevant input, since its effect on my cognitive performance is incidental, accomplished through biochemical pathways that are quite disjoint from the mechanisms of cognitive competence.

The opposing view would be that it simply is not possible to make a clean separation between cognitive and non-cognitive aspects of human behaviour; this is argued for fairly persuasively by French (1990). It would appear to follow from this (though French himself does not so argue) that the attempt to account for human cognition in terms that allow CT to be applied is fundamentally misconceived. Thus viewed, human behaviour is simply the wrong kind of thing to be regarded as computational, not because it lacks a physical basis, but because its physical basis resists description in the discrete terms required of CT.

This would be the case if, for example, Deutsch's notion (mentioned above in section 6) that there need be no exact correspondence between what is computable in the mathematical sense and what is computable in the physical sense, were correct. In that case, the physical nature of human behaviour and cognition would tell *against* the possibility of simulating it on a machine whose design is guided by the mathematical notion of computation; but it would still leave open the possibility of simulating human behaviour on a radically different kind of machine, for example a neural network, or the Universal Quantum Computer discussed by Deutsch.

All this is relevant to the 'strong' AI thesis that the human brain is a computer, and human cognitive processes are its computations. It is not very relevant to the business of a more modest brand of AI, which merely seeks computational ways of performing certain tasks or solving certain problems which humans are rather good at but which machines, so far, have done rather badly, if at all. No doubt it is the former variety of AI, rather than the latter, that is more in harmony with Turing's own convictions.

[6] For the distinction between competence and performance, originally introduced in relation to linguistic behaviour, see Chomsky, 1965.

REFERENCES

Chomsky, N. (1965), *Aspects of the Theory of Syntax*, Cambridge, Mass.: MIT Press.

Church, A. (1936), 'An Unsolvable Problem of Elementary Number Theory', *American Journal of Mathematics*, 58: 345–63.

Dennett, D. C. (1978), *Brainstorms: Philosophical Essays on Mind and Psychology*, Hassocks: Harvester Press.

Deutsch, D. (1985), 'Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer', *Proceedings of the Royal Society, London A*, 400: 97–117.

Enderton, H. B. (1977), 'Elements of Recursion Theory', in J. Barwise (ed.), *Handbook of Mathematical Logic*, Amsterdam: North-Holland, 527–66.

Fields, C. (1989), 'Consequences of Nonclassical Measurement for the Algorithmic Description of Continuous Dynamical Systems', *Journal of Experimental and Theoretical AI*, 1: 171–8.

French, R. M. (1990), 'Subcognition and the Limits of the Turing Test', *Mind*, 99: 53–65, and this volume, pp. 11–26.

Gandy, R. (1988), 'The Confluence of Ideas in 1936', in R. Herken (ed.), *The Universal Turing Machine*, Oxford: Oxford University Press, 55–111.

Gödel, K. (1934), 'On Undecidable Propositions of Formal Mathematical Systems', in M. Davis (ed.), *The Undecidable*, New York: Raven Press, 1965: 41–74.

Gurevich, Y. (1988), 'Algorithms in the world of bounded resources', in R. Herken (ed.), *The Universal Turing Machine*, Oxford: Oxford University Press, 407–16.

Harel, D. (1987), *Algorithmics: the Spirit of Computing*, Reading, Mass.: Addison-Wesley.

Hermes, H. (1965), *Enumerability, Decidability, Computability*, Berlin: Springer Verlay.

Kalmár, L. (1959), 'An Argument against the Plausibility of Church's Thesis', in A. Heyting (ed.), *Constructivity in Mathematics*, Amsterdam: North-Holland, 72–80.

Kleene, S. C. (1936), 'General Recursive Functions of Natural Numbers', *Mathematische Annalen*, 112: 727–42.
—— (1952), *Introduction to Metamathematics*, Amsterdam: North-Holland.
—— (1967), *Mathematical Logic*, New York: Wiley.
Minsky, M. L. (1967), *Computation: Finite and Infinite Machines*, Englewood Cliffs, NJ: Prentice-Hall.
Péter, R. (1959), 'Rekursivität und Konstruktivität', in A. Heyting (ed.), *Constructivity in Mathematics*, Amsterdam: North-Holland, 226–33.
Pnueli, A. (1985), 'Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends', in J. W. de Bakker, W.-P. de Roever, and G. Rozenberg (eds.), *Current Trends in Concurrency*, Springer: Lecture Notes in Computer Science 224.
Rogers, H. (1967), *Theory of Recursive Functions and Effective Computability*, New York: McGraw-Hill.
Sloman, A. (1988), 'What isn't Computation?', in *Proceedings of the 8th European Conference on Artificial Intelligence*, 728–30.
Stewart, I. (1996), 'The Demise of the Turing Machine in Complexity Theory', this volume, pp. 221–32.
Turing, A. (1936), 'On Computable Numbers, with an Application to the Entscheidungs-problem', *Proceedings of the London Mathematical Society*, Series 2, 42: 230–65.
von Neumann, J. (1948), 'General and Logical Theory of Automata', in *Cerebral Mechanisms in Behaviour—the Hixon Symposium*, New York: John Wiley.
Wang, H. (1974), *From Mathematics to Philosophy*, London: Routledge & Kegan Paul.
Webb, J. C. (1983), 'Gödel's Theorem and Church's Thesis: A Prologue to Mechanism', in R. S. Cohen and M. W. Wartofsky (eds.), *Language, Logic, and Method*, Dordrecht: Reidel, 309–53.

9
Measurement and Computational Description

CHRIS FIELDS

INTRODUCTION

The mathematical theory of computation does not directly address the questions of whether, and if so, how physical processes might be characterized as computations, or physical systems characterized as computers. These questions lie at the heart, however, of the technology of computer design, and of empirical investigations in cognitive science, computational neuroscience, and other sciences concerned with the processing of information. Addressing these questions requires a theory not of computation *per se*, but of the computational description of physical systems. My purposes in the present paper are, first, to outline a theory of computational description that is based explicitly on the theory of physical measurement, and, second, to examine some consequences of this theory for some questions in the philosophy of computer science and artificial intelligence.

The ubiquity and utility of abstract computational descriptions in cognitive science have been pointed out by a number of authors, writing from a number of theoretical perspectives (e.g. Fodor, 1974; Marr, 1982; Cummins, 1983; Pylyshyn, 1984; Ullman, 1986; Sejnowski *et al.*, 1988). The notion of a virtual machine developed in computer science (e.g. Tanenbaum, 1976, ch. 1) typically serves as the starting-point for accounts of computational description as it is applied to natural systems. Marr (1982, ch. 1),

I thank Eric Dietrich, Stephanie Forrest, and Aaron Sloman for comments on an earlier draft of this paper. This work was partially supported by NASA Innovative Research Program grant NAGW-1592 to the author and J. Barnden.