Is the Church-Turing Thesis True?

CAROL E. CLELAND
Department of Philosophy and Institute of Cognitive Science, University of Colorado, Boulder, CO, U.S.A.

Abstract. The Church-Turing thesis makes a bold claim about the theoretical limits to computation. It is based upon independent analyses of the general notion of an effective procedure proposed by Alan Turing and Alonzo Church in the 1930's. As originally construed, the thesis applied only to the number theoretic functions; it amounted to the claim that there were no number theoretic functions which couldn't be computed by a Turing machine but could be computed by means of some other kind of effective procedure. Since that time, however, other interpretations of the thesis have appeared in the literature. In this paper I identify three domains of application which have been claimed for the thesis: (1) the number theoretic functions; (2) all functions; (3) mental and/or physical phenomena. Subsequently, I provide an analysis of our intuitive concept of a procedure which, unlike Turing's, is based upon ordinary, everyday procedures such as recipes, directions and methods; I call them "mundane procedures." I argue that mundane procedures can be said to be effective in the same sense in which Turing machine procedures can be said to be effective. I also argue that mundane procedures differ from Turing machine procedures in a fundamental way, viz., the former, but not the latter, generate causal processes. I apply my analysis to all three of the above mentioned interpretations of the Church-Turing thesis, arguing that the thesis is (i) clearly false under interpretation (3), (ii) false in at least some possible worlds (perhaps even in the actual world) under interpretation (2), and (iii) very much open to question under interpretation (1).

Key words. Church-Turing thesis, Turing machine, effective procedure, causal process, analog process

I

The Church-Turing thesis is founded upon two independent analyses of the concept of an effective procedure. The first was based upon the lambda-calculus, a logical system developed by Alonzo Church (1934) in the mid-1930's as a foundation for standard mathematics. Church argued that λ-definability represented an intuitively plausible notion of what it is for a number theoretic function (a function defined on the natural numbers) to be effectively computable ("calculable"). When it was subsequently demonstrated (Kleene, 1936) that λ-definability was extensionally equivalent (vis-à-vis the computation of number theoretic functions) to the other well known mathematical notion of effective computability, viz., Herbrand-Godel general recursiveness, Church postulated (1965) that λ-definability represented an ultimate limit to the mathematical possibilities for computing a number theoretic function. That is, he proposed that there weren't any number theoretic functions which were effectively computable but not λ-definable. This proposal became known as "Church's thesis."

A few years later Alan Turing (1965), unaware of Church's work, proposed yet another analysis of the concept of effective computability. Turing based his analysis on the concept of a very simple abstract mechanism now known as a

Minds and Machines **3**: 283–312, 1993.

"Turing machine." Although his analysis was, like Church's, grounded in mathematical considerations, Turing argued that his analysis could be extended to any procedure which could be followed by a human being, as opposed to just those followed by an idealized mathematician. Turing later demonstrated (1937) that Turing machine computability was extensionally equivalent (vis-à-vis the computation of the number theoretic functions) to λ-definability and, since Turing machine computability was supposed to represent the more general analysis, hypothesized that there were no effectively computable number theoretic functions which weren't computable by a Turing machine. This supposition became known in mathematical circles as the "Church-Turing thesis;" it is also sometimes called "Turing's thesis" or "Church's thesis."

Although the Church-Turing thesis was originally construed as applying only to the computation of number theoretic functions, at least two bolder interpretations appear in the literature.

The thesis is frequently taken by computer scientists to represent an ultimate limit to *all* computation and, hence, has been construed as applying to functions in general, as opposed to just the number theoretic functions. In his now classic work, Marvin Minsky (1967, pp. 132–137), for example, introduces the Church-Turing thesis in the context of the general notion of a function; he does not, as do most mathematicians, explicitly restrict it to the number theoretic functions. Moreover, the thesis has been invoked against the claims of some computer scientists that connectionist networks (which are analog and parallel) represent potentially more powerful computational devices than digital computers.[1] Indeed, Lee A. Rubel (1989) has speculated that the thesis applies to analog (continuous) computation in general.[2]

It has also been maintained that the domain of application of the Church-Turing thesis extends beyond the computation of functions to the production of physical and/or mental phenomena. Insofar as Turing's analysis was based upon the notion of mechanism and physical and mental processes are often characterized as mechanistic, it is hardly surprising that such an interpretation would arise. In fact its first defender was Turing, himself! In "Computing Machinery and Intelligence" (1964) he explicitly extended the thesis to mental phenomena (consciousness, thought, etc.), arguing that mental processes are procedural (in the Turing machine sense) and, hence, that anything which could be produced by a mental process could also be produced by a Turing machine procedure. Turing concluded that a properly programmed electronic digital computer would be conscious and think. When one considers that Turing was an avowed materialist, it seems reasonable to conclude that he intended the thesis to apply to brain processes. In any case, however, the view that the thesis applies to brain processes has become fairly popular in recent years, being defended by the likes of Douglas Hofstadter (1980, p. 572) and Paul and Patricia Churchland;[3] in the words of the latter (1990, p. 301), "what the mind-brain does is [Turing] computable."

Finally, it has been maintained that the Church-Turing thesis applies to physical processes other than brain processes. Shortly after Turing argued for its extension to mental processes, von Neumann argued (1966) for its extension to self-reproduction and, very recently, a number of scientists, e.g., Finkelstein (1988), Bennett (1988), Geroch and Hartle (1986), have contended that it applies to non-biological physical processes as well. In this context, the question has been raised (Rosen, 1988, p. 525): "Is Church's thesis a fundamental restriction on material nature (akin to the exclusion of the *perpetuum mobile* by the laws of thermodynamics), or not?"

In short, there is a remarkable lack of agreement as to the proper domain of application of the Church-Turing thesis. It has been maintained that it applies to: (1) the number theoretic functions; (2) all functions; (3) mental and/or physical phenomena.[4] As a consequence, any evaluation of the truth of the Church-Turing thesis should be explicitly relativized to a particular domain of application; otherwise there is the danger that it will be dismissed as irrelevant. Ideally, of course, an evaluation of the Church-Turing thesis would address all three interpretations.

Regardless of one's views about its proper domain of application, it is important to appreciate that the Church-Turing thesis is not provable. For its truth depends upon whether a particular analysis of our intuitive concept of an effective procedure is correct, and that is not something which can be established by logic or mathematics alone. The Church-Turing thesis is, however, falsifiable. What is required to falsify it is a procedure which can be shown to be (1) effective in some intuitive sense, but not in Turing's sense, and (2) "more powerful" (vis-a-vis a particular domain of application) than any Turing machine procedure. Although a number of philosophers (e.g., Shapiro, 1981) and scientists (e.g., Penrose, 1988, Smolensky, 1988) have expressed misgivings about the thesis (as it applies to one or more domain of application), there currently do not exist any generally accepted counter-examples to it. This could, of course, be because the thesis is true (in, perhaps, the broadest sense). However, it could also be because we have no independent analysis of our general concept of an effective procedure to compete with Turing's. Attempts to solve the halting problem, for example, seem futile, since the halting problem is always formulated in terms of the very notions which must be rejected if the Church-Turing thesis is to be shown to be false. On the other hand, proposed counter-examples which do not conform to Turing's account are subject to the circular complaint that they do not represent *bona fide* effective procedures, since they are not Turing machine procedures. In other words, a genuinely convincing counter-example to the Church-Turing thesis presupposes that we have at hand an alternative analysis of the notion of an effective procedure, as opposed to, for example, the mere suspicion that some exotic physical process could out-perform any Turing machine.[5] For it is only in the context of a viable alternative that a proposed counter-example can come to be viewed as a *genuine* counter-example.

In the next three sections of this paper I develop an alternative analysis of our general concept of a procedure. Unlike Turing's and Church's, mine is grounded in ordinary, everyday procedures such as recipes, directions and methods, i.e., procedures which can only be implemented on concrete physical systems. I call them "mundane procedures." I argue that many mundane procedures can be said to be effective in the same sense that Turing machine procedures are said to be effective. Nevertheless, it is my contention that effective mundane procedures differ from Turing machine procedures in an important way: Effective mundane procedures, but not Turing machine procedures, generate causal processes when followed. Subsequently I turn to an evaluation of the Church-Turing thesis in light of my analysis of the concept of an effective mundane procedure.

I consider the broadest version of the thesis first, arguing, in Section V, that much of its plausibility derives from a failure to carefully differentiate Turing machine procedures from their physical realizations ("embodiments"). I admit that embodiments of Turing machine procedures generate causal processes. I argue, however, that the embodiments do not do so solely in virtue of following a Turing machine procedure. Indeed, it is my contention that while no Turing machine, qua abstract mechanism, can "perfectly simulate" (duplicate) the causal activity of a concrete physical system there are no formal activities of a Turing machine that couldn't, at least in principle, be perfectly simulated by a physical system. The upshot is that there are things which can be done by following an effective mundane procedure which cannot be done by following any Turing machine procedure, but not vice versa. In short, it is my contention that the broadest version of the Church-Turing thesis is false.

In Section VI, I address the first two interpretations of the Church-Turing thesis. I start with the second interpretation, which claims that the Church-Turing thesis represents a fundamental limit upon the possibilities of computation in general. I describe a hypothetical physical system which, I argue, could compute a real-valued function that couldn't be computed by any Turing machine. My argument crucially depends upon the claim, which I defend, that causal processes can be interpreted as computing functions, e.g., an object moving at unit speed in a world where space and time are genuinely continuous can be interpreted as computing the identity function for the real numbers. It is important to keep in mind that I do not establish that the second version of the thesis is *in fact* false; my purpose is only to show how it *could be* false. In either case, however, my conclusion stands and that is that the limits to computation are not to be found in abstract Turing machine theory; for the limits to computation are causal, as opposed to logical.

In this light, I briefly take under consideration the most narrow interpretation of the Church-Turing thesis. I argue that there is little conceptual basis for the supposition that an effective mundane procedure couldn't compute a so-called uncomputable number theoretic function. For as established in my evaluation of the second interpretation of the thesis, we have good reason to suppose that the

computational capacities of causal processes exceed those of Turing machines and we have no idea as to the ultimate capacities of causal processes to compute functions. My goal here is the very modest one of shifting the burden of proof onto those who would maintain that the Church-Turing thesis is true under at least its original interpretation. The challenge, of course, remains and that is to actually describe a physical system which could compute an "uncomputable" number theoretic function.

II

Procedures include recipes, methods, geographical directions, Turing machine programs and mathematical algorithms. What they all seem to have in common is the specification of something to be followed. This is frequently made explicit in theoretical computer science: In Richard Hamlet's words (1974, p. 2), "a 'procedure' is intuitively something to follow." It is also true of mathematical algorithms. Consider, for example, the standard algorithm for computing averages: One is instructed to add all the numbers, count the number of numbers and, then, divide the sum of the numbers by the number of numbers. Moreover, it is obvious that mundane (ordinary, everyday) procedures such as recipes and methods are no exception. The question is: What is it that one follows when one follows a procedure?

Let us begin our search for an answer to this question by investigating the nature of mundane procedures; subsequently, we will compare them to Turing machine procedures. In this context, consider the following examples of mundane procedures:

(A) A recipe for Petit Mont Blanc:

> Preheat the oven to 350°F. Cream the butter; add brown sugar and flour, blending until mixture resembles cornmeal. Stir in nuts . . . Bake 10 minutes. . . .[6]

(B) A method, for building a fire:

> . . . Turn the spindle with long, steady strokes of the bow. Keep going until heavy smoke pours from the notch. Lift the fireboard and tinder together and blow on the ember in the notch until it ignites the tinder.[7]

As the examples illustrate, mundane procedures usually come in the form of lists of instructions; indeed, any mundane procedure which does not have this form can always be put in this form. Each instruction expression (e.g., "stir in nuts," "turn the spindle with long, steady strokes of the bow") contains reference to an occurrence which is to be brought about, i.e., an action, as opposed to something

which merely happens or is undergone.[8] The instructions are expressed by imperatives, indicating that the action being referred to is to be performed by the follower of the instruction. In other words, an instruction expression not only refers to an action, it literally orders its performance; I will reserve the terms "indicate" and "specify" to distinguish this special referential relation, between an instruction expression and an action, from referential relations in general. Thus, we now have the beginning of an answer to the question with which we started: What one follows, when one follows a mundane procedure, are instruction-expressions, and one follows an instruction-expression by performing the action it indicates.

The action indicated by an instruction-expression is abstract and general, as opposed to particular. Different people can follow the same instruction-expression. In following the same instruction-expression, they perform the same *kind* of action, even though their individual actions are distinct.[9] That is to say, the actions specified by instruction-expressions are not action-tokens; they are action-kinds. Furthermore, the order of the performance of the action-kinds is not arbitrary. In the recipe for Petit Mont Blanc, for example, one is supposed to cream the butter *before* adding the brown sugar and flour. The order in which the action-kinds are to be performed is specified externally by the order of presentation of the instruction-expressions. Often, however, the order of the action-kinds is specified internally through the use of conditional instructions, e.g., "keep going *until* heavy smoke pours from the notch." But in either case, performing the specified action-kinds in the wrong order in time constitutes just as much a failure to follow the procedure as not performing them at all.

Finally, it is important to keep in mind that it is the action-kinds, as opposed to the instruction-expressions which indicate them, which are crucial to the identity of a mundane procedure. For the same mundane procedure can be expressed in different languages and in different terminology in the same language. Let us, therefore, draw a distinction between an instruction-expression and an instruction. More specifically, we shall say that different instruction-expressions express the same instruction if and only if they indicate the same action-kind or, if the instruction is conditional in form, the same action-kind and the same condition (generic state of affairs) for its occurrence.[10]

In summary, what one follows, when one follows a mundane procedure, is a list of instructions indicating that certain kinds of action are to be performed in a given order in time. One follows the instructions by performing the indicated action-kinds in the order specified.

Do Turing machine procedures differ from mundane procedures? A Turing machine is a very simple, *abstract* machine. It consists of a mechanism, known as a "finite state machine," coupled to an external storage medium, known as "the tape." The tape, which could be represented by anything (from a bunch of tin cans to graph paper), is divided into squares. At any given moment, the tape has only finitely many squares; however, an indefinite number of additional squares

can be added to either end of the tape, making it effectively infinite. Each square of the tape is occupied by at most one of a finite number of distinct symbols; the set of symbols, usually represented by $\{S_0, \ldots, S_n\}$, is called "the alphabet" of the machine. The finite state machine is linked to the tape through a head which is always positioned over one of the squares. At any given moment, the machine is characterized as being in one of a finite number of internal states, $q_l, \ldots, q_m$. As is the case with a concrete machine, these states completely define the Turing machine's mechanism, i.e., the finite state machine. However, unlike the internal states of a concrete machine, the internal states of a Turing machine are not physical structures. Rather they can be explicitly identified with instructions. A Turing machine is said to be in a particular internal state q_i only if it is about to carry out a particular instruction i. It doesn't matter how q_i is physically realized. All that matters is that the machine is about to carry out instruction i.

Turing machine procedures are commonly identified with Turing machine programs. A Turing machine program is extensionally defined by its instruction set (the set of all of its internal states). The instructions of a Turing machine program have the form of conditional rules which specify that certain, very simple, things be done (such as move to the right one square or erase the symbol currently being scanned and print another symbol in its place). There are a number of different ways to represent a Turing machine instruction set. One way is by means of a set of quadruples (see Fig. 1).

$$\{q_1S_0S_1q_1, q_1S_1Lq_2, q_2S_0S_1q_2, q_2S_1Lq_3, q_3S_0S_1q_3\}$$

Fig. 1.

The first two symbols of each quadruple stand for (respectively) the current state of the machine (instruction being followed) and the current symbol being scanned by the head. The second two symbols represent, in the order presented, the action to be performed and the next state of the machine. Accordingly, the first quadruple can be read as follows: if the machine is in state q_l (following instruction 1) and the head is scanning an S_0, erase the S_0, print an S_1 (in its place), and go into (in this case, stay in) state q_1. Similarly, the second quadruple says to move the head to the left one square and go into state q_2, if the machine is in state q_1 and the head is scanning an S_1. One can see how the machine operates by looking at the sequence of its moment by moment configurations (Fig. 2).

0000000	0000100	0000100	0001100	0001100	0011100
↑	↑	↑	↑	↑	↑
q1	q1	q2	q2	q3	q3

Fig. 2.

Each sequence of numbers represents the contents of the machine's tape during one of its configurations; the sequence to the far left representing the initial configuration (blank) and the sequence to the far right representing the final configuration (three consecutive S_1's, represented by 1's, on an otherwise blank tape). The q_i, which appear below ech sequence, represent the state of the machine during that particular configuration; each of the q_i are positioned directly below the symbol (in bold type) which is being scanned by the head at the moment in question. Thus, one can readily verify that the set of quadruples in Fig. 1 represents a Turing machine program which prints three consecutive S_1's, when started out on a blank tape in state q_1.

The same Turing machine program can also be represented by means of a machine table (Fig. 3). The entry 'S_1q_1' in row 'q_1' and column 'S_0' of Fig. 3 has the same meaning as the expression '$q_1S_0S_1q_1$' in Fig. 1. Machine tables have the advantage of more perspicuously revealing the complex conditional structure of individual Turing machine instructions. As an example, the first row in the above table represents a single instruction which has two possible outcomes, viz., (a) print an S_1 and go into state q_1, or (b) move the head to the left one square and go into state q_2, depending upon whether the symbol being scanned by the head is an S_0 (a) or an S_1 (b).

We are now in a position to explicitly compare Turing machine programs to mundane procedures. Like the instructions of a mundane procedure, the instructions of a Turing machine program specify the performance of generalized (repeatable) action-like occurrences; they do not specify something which merely happens to the machine or, for that matter, something which is undergone by the machine but, rather, something which the machine is to do (e.g., *print* an S_1, *move* to the left one square). Moreover, the instructions of a Turing machine program specify that the indicated action-kinds be performed in a certain order in time. The order is secured by the complex conditional nature of the individual

	S_0	S_1
q1	S1q1	Lq2
q2	S1q2	Lq3
q3	S1q3	S1q3

Fig. 3.

instructions in the program. More specifically, what kind of action is performed at any given moment by the machine depends upon its immediately preceding action; for its immediately preceding action determines the current state of the machine and the symbol being scanned by the head. But this, in turn, depends upon the action which immediately preceded that action, etc. The regress ends with the initial configuration of the machine. By definition, the machine is started out in some specific initial configuration, so the time-ordered arrangement of the specified action-kinds is completely determined by internal (vs. external) relations among the instructions in its instruction set. This explains why the instructions of a Turing machine program can be represented in the form of a set (which, of course, presupposes the specification of an initial configuration, in order to fix the actual sequence of action-kinds to be performed) rather than a list.[11] In short, Turing machine programs and mundane procedures do not differ, in any fundamental way, at the level of their instructions: Both specify the performance of time-ordered arrangements of action-kinds. This brings us to the concept of the effectiveness of a procedure.

III

A mundane procedure is said to be "effective" if following it always results in a certain kind of outcome, e.g., pastry, fire. In other words, our ordinary, everyday concept of an effective procedure is that of a procedure which consistently terminates in an outcome of a given kind. Effectiveness is not, however, characterized this way in Turing machine theory.[12] For as Minsky explains (1967, pp..105–106), the implementation of some mathematical and abstract mechanical procedures requires an infinite number of steps (actions). Such procedures go on forever. Hence, it makes little sense to speak of them as having a final outcome. Nevertheless, there may be a definite, predetermined outcome at each and every stage of their implementation. Accordingly, there is a sense in which they can be said to be effective despite the fact that they do not *terminate* in a final outcome. In Minsky's words (p. 105), "our concern here is not with the question of whether a process terminates with a correct answer, or even ever stops. Our concern is whether the next step is always clearly determined in advance." In this context Minsky proposes (p. 106) that we characterize effectiveness in terms of a set of rules which are "precisely described", where a set of rules is said to be "precisely described" if it "... tell[s] us, from moment to moment, precisely how to behave."

Any *finite* procedure which is effective in Minsky's sense will also be effective in the ordinary, everyday sense, since any finite sequence of predetermined actions terminates in a final action. Thus, every finite Turing machine program (i.e., every Turing machine program that always terminates, for any input, after a finite number of operations) will qualify as effective in both Minsky's sense and the ordinary, everyday sense; each step will be determined in advance and the

procedure will always terminate in a certain kind of outcome. But it is not obvious that every effective mundane procedure will qualify as effective in Minsky's sense. Whether an effective mundane procedure qualifies as effective in Minsky's sense depends upon whether its instructions (rules) can be "precisely described."

It has been maintained that the instructions specified by most effective mundane procedures cannot be precisely described. But this is hardly obvious. Our recipe for Petit Mont Blanc, for example, instructs us to "preheat the oven." What is so imprecise about this? Certainly one has to know how to "preheat" the oven in order to follow the instruction. Similarly, a Turing machine has to know (to speak very loosely) how to "print" symbols in order to "print an S_0." The only obvious difference between the two cases seems to be that we are expected to simply assume that the Turing machine instruction-expression, but not the recipe instruction-expression, tells us precisely what to do. But why would anyone suppose that this is so? Is it because cooks sometimes fail to preheat ovens when instructed to preheat them? But concrete machines sometimes fail to print symbols when instructed to print symbols! In neither case does the source of the problem seem to be a vaguely described instruction; rather, the problem seems to lie in an error on the part of the follower of the instruction.

Now, admittedly, no Turing machine which is instructed to "print an S_0" can fail to print an S_0, but this is because Turing machines are, by definition, idealized followers of Turing machine instructions. In this light, it seems that one could equally well postulate an ideal chef who never makes mistakes when following recipes. Such a chef would not only know how to preheat the oven, she would also know how to stir in the nuts and cream the butter. This is not to deny that creaming the butter is harder to do than preheating the oven. But there is little reason to suppose that the difficulty in question derives from an imprecision in the description of the instruction. The fact is that any moderately experience chef knows precisely what to do when asked to "cream the butter." So it does not take any more of a stretch of the imagination to imagine an ideal chef who never fails to cream the butter when instructed to do so than it does to imagine an ideal machine that never fails to print an S_0 when instructed to do so. This does not, of course, prove that our recipe for Petit Mont Blanc is precisely described. It does, however, suggest that there is little evidence for the claim that it is less precisely described than a Turing machine program.

In this context, one might be tempted to conclude that *no* instruction (mundane or Turing machine) is precisely describable, since no mere description of what to do can (just by itself) exclude the possibility of error. But even if this were so, it would still be possible to make sense of the claim that an instruction is *well defined*; for an instruction which is not precisely described (in the sense just mentioned) could still tell an ideal follower precisely what to do. Insofar as the latter is really all that Minsky is interested in, we could redefine effectiveness in terms of a set of rules which are well defined, i.e., a set of rules which tells an ideal follower ". . . from moment to moment precisely how to behave." The trick,

of course, is to make sense of an idealized follower. While we cannot do this for every mundane instruction (e.g., throw a double six on two fair dice), we can do it for many mundane instructions, particularly those in which there is a class of specialists who almost invariably get the same result when following the instruction. Insofar as such specialists exist, there is little reason to suppose that mundane instructions are less well defined than Turing machine instructions. Moreover in the case of mundane instructions for which there do not exist specialists, there is little reason to suppose that the procedures they constitute are effective even in the ordinary, everyday sense. In short, there is little evidence for the claim that so-called effective mundane procedures are not effective in Minsky's sense.

Insofar as all finite effective procedures terminate in definite kinds of outcome, they can be characterized as *effective for* given kinds of outcome. The Turing machine program specified by the machine table in Fig. 3, for example, is an effective method for printing three consecutive S_1's on an otherwise blank tape, but not an effective method for printing the sequence of symbols $S_1S_0S_1S_0$ on an otherwise blank tape. Similarly, mundane procedure B (the method for starting a fire by friction) is an effective procedure for building a fire; when it is followed correctly, a fire always results. But it is not an effective method for making chocolate icing or, for that matter, printing the sequence of symbols $S_1S_0S_1S_0$ on an otherwise blank tape. That is to say, finite procedures are not effective for most kinds of outcome. However, those generic outcomes for which a finite procedure is effective are outcomes which are invariably realized when the procedure is (correctly) followed. Indeed, we are primarily interested in procedures insofar as they are effective for particular kinds of outcome, i.e., insofar as they can be used to achieve various desirable ends. In this context, we can view an infinite effective procedure as effective for an infinite number of generic outcomes, e.g., in the case of a procedure for computing a function having an infinite number of arguments, the outcomes are the pairings of the appropriate values to the arguments. In any case, however, the question arises how does the mere following of an effective procedure, Turing machine or mundane, result in an outcome of the relevant kind?

Let us begin by looking at the role of the action-kinds specified by the instruction list of an effective mundane procedure.[13] Consider, for example, Julia Child's "traditional recipe" for making Hollandaise sauce (Child 1961, pp. 79–80); let us make the, admittedly, bold assumption that it really is an effective method for making Hollandaise sauce. The recipe instructs one to, among other things, pour on melted butter by droplets while continuously beating warm egg yolks with a wire whisk. What makes her recipe effective for making Hollandaise sauce is not, however, the mere performance of the action-kinds concerned (in the specified order in time). Rather it is something which is caused by their performance, namely, a process of making the egg yolks absorb butter and hold it in creamy suspension. If actions of the kind in question had a different effect

(they caused the egg yolks to become granular and float in the butter, for instance), the recipe would not represent an effective method for making Hollandaise sauce. In other words, the effectiveness of the recipe for Hollandaise sauce is dependent upon the causal capacities of actions of the kinds specified by the recipe to initiate and sustain a certain kind of causal process. Similarly, the effectiveness of mundane procedure B for building a fire is contingent upon the causal consequences of actions such as turning a spindle, set in a fireboard, with long steady strokes of a bow. In general, the effectiveness of a mundane procedure for a given end depends upon what kinds of causal processes would be generated, were the procedure to be followed.

The causal capacities of the action-kinds indicated by a mundane procedure are not, however, specified by the procedure's instruction list. All that is specified by the instruction list is the performance of a time-ordered arrangement of action-kinds. To fully appreciate this, imagine a possible world, W_i, which differs from the actual world, W_a, in that friction does not cause heat;[14] let us suppose that objects which are rubbed together become cooler. Now suppose that someone in W_i follows the instruction list for mundane procedure B. The result will not be a fire, as would be the case if someone in W_a were following it. (Indeed, campers in W_i find procedure B valuable only insofar as it provides them with a means for chilling their beer!) This difference in outcome is not a result of a difference in instruction list; in both worlds, people follow the same instruction list and perform the same kinds of action in the same relative temporal order. Rather, the difference is due solely to a difference in the causal makeup of the worlds concerned. Performing the very same kinds of action causes very different sorts of things to happen in W_a and W_i. In brief, the very same instruction list can be effective in one world and ineffective in another world for the very same kind of outcome.

In contrast, the effectiveness of a Turing machine program for a given kind of outcome does not vary from possible world to possible world. The action-kinds specified by a Turing machine program are causally inert. In following an instruction to print an S_0, for example, a Turing machine places an S_0 in the designated square on the tape; failure to do so would represent failure to perform an action of the designated kind. however, unlike the case with a mundane procedure, the appearance of an S_0 in the designated square does not have any causal consequences. This is not to deny that an "embodiment" of a Turing machine in a concrete machine would generate a causal process (or processes). But such a process is not a consequence of the activity of the Turing machine per se. Rather it is a consequence of the physical activity of the concrete machine, and the latter is, strictly speaking, not the activity of a Turing machine. Indeed, different concrete machines, generating very different kinds of causal process, can embody the very same abstract Turing machine. Finally, it is important to keep in mind that the symbols printed by a Turing machine are purely formal; they are uninterpreted. Accordingly, the effectiveness of a Turing machine program for a particular outcome (sequence of symbols on its tape) is independent of how the outcome is interpreted (what the symbols are taken to stand for) in a particular world. In other words, what a Turing machine does *qua* Turing machine (i.e., independently of its embodiment and the interpretation of its symbols) is the same in all possible worlds.

Accordingly, there does seem to be a difference between effective mundane procedures and Turing machine procedures (which are all effective, though not always for the same outcomes). In the case of an effective mundane procedure, the mere performance of the action-kinds specified by its instruction list does not directly produce the outcome; rather, the outcome is produced by a causal process which is generated by the performance of the action-kinds concerned. This is in stark contrast to the case of Turing machine procedures, where nothing causal stands between the sequence of actions performed by the machine and its outcome; the sequence of actions performed by the machine literally is what produces the outcome. Nevertheless, such a distinction between effective mundane procedures and Turing machine procedures will be interesting only if there is a fundamental difference between causal processes and sequences of actions (instances of procedures); otherwise, an effective mundane procedure will not differ in any interesting way from a complex Turing machine program in which one program "drives" another program (as opposed to generating a nonprocedural causal process).

One way in which causal processes seem to differ from instances of procedures is in the interrelation of their subparts. To see this, consider, again, the sequence of Turing machine configurations depicted in Fig. 2. Each configuration represents an action on the part of the machine. Insofar as no two actions overlap in time, each action can be said to both be separate (discrete) and distinct (different) from every other.[15] Moreover, no action is the cause of any other action. Admittedly, the next action on the part of the machine depends counterfactually on the immediately preceding action, since the latter supplies the antecedent condition for the next instruction to be executed. But this is a formal, not a causal, consequence of the preceding action. The succeeding action (e.g., movement of the head to the left one square) can no more be said to be caused by the preceding action (printing an S_0) than my daughter's reaching the age of 13 in 1992 can be said to be caused by her being born in 1979, despite the fact that it is true that if she had not been born in 1979, she would not have reached the age of 13 in 1992. For as Jaegwon Kim has noted (1980), a number of non-causal dependencies can also be expressed by counterfactual conditionals. Mere counterfactual dependency may be necessary for causation, but it is not sufficient. In short, the actions which constitute an instance of a Turing machine program are causally, as well as temporally, discontinuous.

The actions which constitute an instance of a mundane procedure are also causally discontinuous. My preheating the oven, while following the recipe for

Petit Mont Blanc, does not cause the butter to become creamed, nor does my creaming the butter cause the addition of brown sugar and flour. At best, the actions concerned can be said to represent "conjunctive forks,"[16] i.e., successive occurrences which are correlated in virtue of being the separate effects of a common cause (the person following the recipe). However, unlike the case with Turing machine procedures, the actions one performs when following a mundane procedure are not always temporally discontinuous. Julia Child's recipe for Hollandaise sauce, for example, requires that one slowly pour on the melted butter *while* beating the egg yolks. But, of course, it doesn't follow from this that the actions in question aren't distinct; for it is possible that there exists some non-temporal difference between them. In contrast, causal processes, at least as they are ordinarily construed, are neither temporally nor causally discontinuous. Causal processes are described as "continuously evolving." Their constituent parts (subprocesses) are not, as in the case of Turing machines actions, thought to be temporally separate. Causal processes are also said to be "self-generating" and "self-determining." Their subprocesses are ordinarily taken to be the direct causes of those subprocesses which immediately succeed them and the direct effects of those subprocesses which immediately precede them, as opposed to being merely correlated with them. In other words, our concept of a process is the concept of a continuously evolving causal chain of occurrences, whereas our concept of an instance of a procedure, whether Turing machine or mundane, is the concept of an arrangement of occurrences which are, at least, causally discontinuous. Causal processes just do not seem very procedure-like.

But, surely, such a claim is incompatible with a Humean analysis of causation. According to Hume (1975, p. 161), our concept of causation does not include the concept of some mysterious connection between immediately neighboring events; rather, our concept of causation is the concept of a sequence of generic events whose instances regularly succeed and precede each other. If Hume is right about this, and some scholars believe that he is, then there isn't, conceptually speaking, a fundamental difference between a causal chain of events and a sequence of merely correlated events, which, in turn, suggests that there isn't a fundamental difference between our concept of a causal process and our concept of an instance of a procedure. Viewed from this perspective, my account of the distinction between causal processes and procedures seems to rest upon a particular, and, perhaps, erroneous, theory of causation.

This would pose a bothersome objection if it weren't for the fact that we frequently do make distinctions between correlation (conjunctive forks, accidental relations, etc.) and causation. Hume and his defenders owe us an explanation of this distinction, albeit one which does not involve any reference to primitive causal connections (e.g., powers and liabilities); otherwise, he has simply failed to provide us with an adequate analysis of our concept of causation. There have, of course, been numerous suggestions as to how to do this in a way which does not violate the spirit of Hume's original account.[17] Although none of these neo-

Humean accounts have been very successful,[18] they all draw some sort of distinction between causation and other kinds of regularity, which means that they are all compatible with my claim that the way in which the subprocesses of a causal process are related to each other is significantly different from the way in which the actions in an instance of a procedure are related to each other. That is to say, even on a neo-Humean account of causation, causal processes are not procedure-like. This suggests that we cannot construe effective mundane procedures as merely complex effective Turing machine procedures, and, hence, that there is, at least potentially, an important difference between effective mundane procedures and effective Turing machine procedures.

So why didn't Turing, who argued that his analysis applied to *any* procedure which could be followed by a human being, notice this difference between effective mundane procedures and Turing machine procedures? I can only speculate. Turing based his analysis upon mathematical decision procedures. The kinds of action specified by a mathematical decision procedure are mathematical operations; mathematical decision procedures represent courses of action for manipulating mathematical entities such as numbers. Mathematical operations have only formal consequences; their results depend only upon the laws of logic and mathematics. But as I have just argued, not all of the procedures we follow represent purely formal ways of manipulating the world. Many of the ways we manipulate the world are causal, and such ways of proceeding are every bit as procedural as the former. In other words, in grounding his analysis of our general notion of an effective procedure on mathematical decision procedures, Turing seems to have blinded himself to the fact that some procedures are explicitly designed to exploit the causal structure of reality. Indeed, the difference between an effective mundane procedure and a Turing machine procedure is just the difference between a causal and a formal way of manipulating the world.

IV

Before we turn to an evaluation of the Church-Turing thesis, it is important that we become clear about the conditions under which different descriptions of procedures can be said to specify the same procedure; as will become apparent in Section V, this is necessary if we are to avoid certain, very natural, confusions between Turing machine programs and their physical realizations. As discussed in Section II, our general concept of a procedure is the concept of something to follow, and what one follows, when one follows a procedure, are instructions specifying that certain kinds of action be performed in a certain order in time. This suggests that identity conditions for both Turing machine procedures and mundane procedures can be formulated in terms of sameness of instructions, or sameness of time-ordered arrangement of action-kinds, since, as discussed in Section II, instructions are the same just in case they indicate the same kinds of action. Accordingly, Turing machine procedures and effective mundane proce-

dures do not seem to differ qua procedures, despite the fact that they represent different ways (formal vs. causal) of bringing things about.

Unfortunately, however, a mere difference in instructions is not always taken as sufficient for a difference in procedure. This is particularly evident in computer science. Consider, for example, a *C* program and a Fortran program for computing averages. Let us assume that the *C* program passes its scalar arguments by value, whereas the Fortran program passes, as do all Fortran programs, its scalar arguments by reference. Accordingly, the programs specify different sorts of action. Yet no legal expert would be willing to patent them as different procedures (algorithms) on these grounds alone. Moreover, the legal debate over the identity of procedures is not confined simply to questions about computer programs. It includes questions about the identity of chemical syntheses for making drugs, metallurgical methods for extracting precious metals from ore, etc. In all such cases, the procedure is construed as being in some definite, but unspecified, sense more coarsely grained than the instruction list.

Such procedures are to be contrasted with Turing machine procedures, where a difference in program always guarantees a difference in procedure. As we have seen, a Turing machine does no more nor no less than what is explicitly specified by its machine table. Thus, a Turing machine cannot be characterized as following anything other than what is specified by its program. As a consequence, no distinction can be drawn between procedure and program. In contrast, a concrete computer does much more than what is explicitly specified by, say, a program in Fortran. The computer first decodes the program, via an interpreter or compiler, into a sequence of basic machine operations. Depending upon their physical structure, different kinds of machines (e.g., SUN workstations as opposed to IBM mainframes) have different basic machine operations. Hence, the same program may be decoded by different machines into different kinds of machine action. Furthermore, it is possible for different programs (e.g., programs for computing averages written in different computer languages) to be decoded by the very same machine into the same sequence of basic machine operations. Thus, two computers executing the same program can be performing different kinds of action and the same computer executing different programs can be performing the same kinds of action. In such cases, it seems wrong to simply identify the procedure with the program being implemented.

The problem is that the action-kinds specified by a high-level program must be realized in physical actions, if a concrete machine is to be able to execute the program. Insofar as the high level program does not, itself, specify the machine operations into which it is decoded, a computer, which is executing a high level program, can be characterized as following, at least, two different procedures, the one specified by the high level program and the one specified by its machine language translation. Nevertheless, both procedures are instanced by the same sequence of individual actions, viz., the sequence of physical actions actually performed by the machine. For just as a physical object can be of different kinds

(e.g., red and round), so an individual, physical, action can be of different kinds. In contrast, the individual actions performed by a Turing machine are of only one kind, viz., the kind explicitly specified by the instruction it is following. Failure to keep this difference between Turing machines and concrete machines in mind leads to puzzles about the identity of computer algorithms: If what the machine does is qualitatively the same, how can the procedures it implements be said to be different? The answer is that qualitatively identical, but numerically distinct, sequences of physical actions can instance different procedures, as many different procedures as the physical actions concerned are of different kinds. Indeed, the very same sequence of physical actions can realize both a formal mathematical procedure and a mundane (causal) procedure; I will have more to say about this later. Moreover, the same formal mathematical procedure (or, for that matter, mundane procedure) can be instanced by dissimilar sequences of physical actions; for actions which fail to resemble each other in some ways may, nevertheless, resemble each other in other ways. Hence, it does not follow from the fact that a computer does something similar (at different times) that the high level programs it executes specify the same procedure, nor does it follow from the fact that different machines do different things, that the high level programs they execute specify different procedures. In short, many of the concerns of our patent rights experts about the identity of computer algorithms are based upon a failure to distinguish procedures, which are multiply instantiatable, from the sequences of physical actions which instance them.

A similar analysis can be given many other ostensible exceptions to the proposal that identity conditions for procedures be formulated in terms of sameness of instructions. However, some mundane procedures, such as recipes and methods, pose a special problem. As an example, consider two different methods for starting a fire. One, procedure *B* in Section II, instructs one to turn a spindle in a fireboard. The other instructs one to rub two sticks together, back and forth. Clearly, the specified action-kinds differ. Yet both methods utilize the same generic causal process (friction) to achieve the same kind of end (combustion). Accordingly, there is a sense in which what one does, when following either procedure, is qualitatively the same, independently of questions about whether the individual actions being performed are similar in ways not explicitly specified by the procedures.

Nevertheless, it would be a mistake to conclude from this that the procedures are the same and, hence, that identity conditions for procedures cannot be formulated in terms of sameness of instructions. As discussed in Section I, our concept of a procedure is the concept of something to follow, and what one follows when one follows a procedure are instructions, as opposed to a causal process. Indeed, it is possible to follow correctly an instruction without knowing anything whatsoever about the causal consequences of the action-kind it specifies, since, as I have argued, the causal consequences in question are not, themselves, specified by the instruction. This is not to deny that there is a sense in which

someone following procedure *B* can be said to be "doing the same thing" as someone rubbing sticks together. It is, however, to deny that what is qualitatively the same is procedural; rather, it is causal, having to do with a similarity in the causal consequences of the actions being performed, as opposed to a similarity in what is being followed. What is being followed is, strictly speaking, explicitly specified by the instructions, and the instructions of the procedures specify different kinds of actions, despite the fact that the causal processes being generated are of the same kind and, for that matter, the actions actually being performed have some properties, qua individuals, in common. Viewed from this perspective, the concerns of patent rights experts are misdirected (when, of course, not just reflecting a confusion between the action-kinds specified by a procedure and the individual actions which instance it). They are not so much concerns about the identity of procedures as concerns about the identity of generic causal processes. These concerns may be legitimate and important, particularly if one is interested in new ways of exploiting the causal structure of the world. But they are not concerns about the identity of procedures.

In this context, it is, perhaps, worth noting that requiring sameness of generic causal process for the identity of mundane procedures would not make it any easier to decide practical questions, such as those posed by patent right experts, about the identity of procedures. This is readily appreciated by considering the question of whether Julia Child's two recipes for Hollandaise sauce, the "easy blender method" and the "traditional method," generate, when followed, the same or different kinds of causal process. The recipes differ in a number of ways. For example, the easy blender recipe specifies that one add melted butter to cold egg yolks, whereas the traditional recipe requires that one add melted butter to warm egg yolks. The question is do these differences correspond to a relevant difference in causal process? In order to even begin to answer this question one would have to have a detailed understanding of the chemistry of cooking. Such an understanding is certainly beyond the capacities of most chefs and, I suspect, most chemists too. This example is typical: In the case of most of the procedures we follow, we know how to perform the indicated action-kinds but we have no idea how performing the indicated action-kinds produces the kinds of outcome with respect to which they are said to be effective.

In conclusion, there seem to be no compelling reasons for supposing that identity conditions for mundane procedures cannot be formulated in the same way as identity conditions for Turing machine procedures, namely, in terms of sameness of instructions. Most, if not all, puzzles about the identity of mundane procedures are rooted in a failure to distinguish the procedure being followed from either its causal consequences or the sequence of individual actions which instances it. Such problems do not arise in the case of Turing machine procedures because there is no sense in which a Turing machine can be said to be doing something other than what is explicitly specified by its machine table. This is a consequence of the fact that individual Turing machine actions, unlike physical actions, lack causal consequences and are never of more than one kind. As I argue below, the two broader versions of the Church-Turing thesis lose much of their cogency when these important differences between physical actions and Turing machine actions are taken into account.

V

We are, at last, in a position to begin our evaluation of the Church-Turing thesis in light of the analysis I have given of the general notion of an effective procedure. The thesis is commonly characterized as the claim that there are no effective procedures more powerful than Turing machine procedures. As discussed in Section I, at least three different interpretations of "more powerful than" can be identified in the literature. More specifically, the thesis is sometimes construed as claiming that: (1) there are no number theoretic functions which couldn't be computed by means of a Turing machine program but could be computed by means of some other kind of effective procedure; (2) there are no functions (number theoretic or otherwise) which couldn't be computed by means of a Turing machine program but could be computed by means of some other kind of effective procedure; (3) there is nothing that couldn't be done (as opposed to computed) by a Turing machine program that could be done by means of some other kind of effective procedure.

Let us begin with the broadest interpretation of the thesis, (3). As suggested in Section I, this interpretation is rejected by most mathematicians. Nevertheless, it has a growing following among some philosophers and scientists. For this reason alone it is important that we give it serious consideration.

On my account of the nature of effective mundane procedures, interpretation (3) is clearly false. Julia Child's easy blender method for making Hollandaise sauce provides one, among many, example of an effective mundane procedure which is "more powerful" than (can do things that cannot be done by) any Turing machine program. What is crucial to the production of Hollandaise sauce is a causal process which forces the egg yolks to absorb butter and hold it in creamy suspension; in the absense of a process of this kind, one will not get a Hollandaise sauce. Insofar as the action-kinds specified by a Turing machine program are purely formal, they have no causal consequences and, hence, cannot, when performed, generate a causal process. The upshot is that there are no Turing machines which are effective for making Hollandaise sauce. The only reason that Julia's recipe is effective vis-a-vis Hollandaise sauce is that it specifies action-kinds having the right kinds of causal consequences. In brief, there are things which can be done by following an effective mundane procedure which can't be done by implementing a Turing machine program on a Turing machine.

This is not to deny that a Turing machine program can be "embodied" in a physical system. Indeed, one can use just about anything to embody a Turing machine program. Joseph Weizenbaum's favorite example (see his 1976, pp. 51–

52) is a system involving a roll of toilet paper and some stones. All that is required is that the physical system have the right kind of formal structure. More specifically, the physical system must have parts which can be placed into one-to-one correspondence with the basic elements (e.g., symbols, actions) of a Turing machine in such a way as to preserve the formal interrelationships specified by the Turing machine program (machine table). In the case of the toilet paper and stones system, for example, one could let the stones stand for the symbols, the physical action of placing a stone on a square of toilet paper for the basic Turing machine action of placing a symbol on a square of tape, etc., and, then, construct a set of conditional instructions specifying a sequence of physical actions which are analogous to those specified by the Turing machine program. The resultant system would physically embody the Turing machine program.

Nevertheless, the procedure which is followed by the toilet paper and stones system is not the same as that which is followed by the Turing machine. There are substantial differences between the kinds of action specified by the Turing machine program and the kinds of action specified by its physical analogue. As an example, the Turing machine program instructs one to place a symbol on a square of tape, whereas its physical analogue instructs one to place a stone on a piece of toilet paper. Considered just in themselves (i.e., independently of any externally established correspondences), these are different action-kinds. Accordingly, the procedures which specify them cannot be the same, since, as I argued in Section IV, procedures differ if they specify different kinds of action. In other words, physical analogues of Turing machine programs are, strictly speaking, not Turing machine programs. Hence, the fact that Turing machine programs can be embodied in physical systems cannot be taken as evidence for the claim that there is nothing that can be done by means of an effective mundane procedure that couldn't be done by means of a Turing machine program.

To this I can imagine the following retort (since I have gotten it on a number of occasions from computer scientists): Any physical system can be "simulated to any desired degree of accuracy whatsoever" by a Turing machine. As stated, this claim is ambiguous. It could amount to the claim that any physical system can, at least in principle, literally be *duplicated* by a Turing machine or it could amount to nothing more than the claim that physical systems have no formal properties which couldn't be realized by a Turing machine. The first interpretation is, presumably, what Douglas Hofstadter had in mind (at least for brain processes) when he claimed (1981, pp. 439-446) that it was possible to construct a simulation of Einstein's brain which would literally *be* Einstein (have Einstein's consciousness, thoughts, desires, hopes, fears, etc.). The second interpretation may be what Bennett had in mind (1988) when he talked about simulating physical processes with Turing machines, since his major concern seems to have been with physical complexity. In any case, however, the latter interpretation is far too weak to secure the conclusion under consideration, which is that there is *nothing* that could be done by a physical system which couldn't be done by a Turing machine. This brings us back to the first interpretation. The question is do Turing machine simulations of physical systems leave anything out? If they do, then, regardless of how "accurate" they may be said to be, they are not capable of duplicating physical systems.

As we have seen, a Turing machine can do no more than what is explicitly specified by its program. A Turing machine program specifies a sequence of purely formal action-kinds. Thus, the activity of a Turing machine consists in nothing more than the performance of a sequence of inert actions. In contrast, a physical system can initiate and sustain causal processes. Moreover, as I argued in Section III, the behavior of a causal process is not at all procedure-like. Unlike a sequence of Turing machine actions, a relation of causation stands between immediately succeeding and preceding subprocesses of a causal process. No Turing machine simulation of a causal process can capture this relation. Now, admittedly, the nature of causation is not well understood, but this doesn't change the fact that it amounts to more than correlation. Insofar as the latter is all that can ever be simulated by a Turing machine, it follows that there are limits to the capacities of Turing machines to simulate causal processes. How serious these limits actually are depends, of course, upon the nature of causation. If, as a number of philosophers have recently argued (e.g., Harre, 1970, Anscombe, 1980, Cleland, 1985), causation is a "physically real" relation, then they may be very serious indeed.

In summary, the actions performed by a physical system are different from those performed by a Turing machine. Unlike physical actions, Turing machine actions are purely formal and have no causal consequences. As a consequence, no Turing machine can realize the causal activity of a physical system. But some of the things that are done by physical systems require causal activity. To paraphrase John Searle (1985, p. 305), no formal simulation of lactation can produce milk and no formal simulation of photosynthesis can produce sugar. Moreover, as Searle argues, there are good reasons for believing that at least some mental states (e.g., understanding) are also of this nature; after all, human brains are causally active entities. In short, there is very good evidence for the claim that physical systems can do things that no Turing machine could do. To repeat, this is not to deny that it is at least possible for a Turing machine program to be embodied in a physical system which produced milk or sugar or understanding, when the physical analogue of the program was implemented. But in such a case it would be a mistake to infer that what produced the milk or the sugar or the understanding was the sequence of action-kinds specified by the Turing machine program. Insofar as Turing machine actions lack causal consequences, they can't produce anything. Their physical analogues can, but the latter are, strictly speaking, not Turing machine actions. In other words, the physical embodiment produces the milk or the sugar or the understanding qua causally efficacious, concrete machine, as opposed to purely formal Turing machine. Physical embodiments of Turing machines are not Turing machines; they are concrete machines

which just happen to have the same formal structure as a Turing machine. In this context, it is important to recall (Section IV) that the same concrete machine can simultaneously implement different procedures; for the same concrete action can be of more than one kind. As a consequence, it is possible for a concrete machine to be simultaneously computing a function *and* producing sugar. But, to repeat, it doesn't follow from this that the procedure which produces the sugar is the same as the procedure which computes the function.

Although no Turing machine can realize the causal activity of a concrete machine, there is no reason to suppose that Turing machines have any formal properties which couldn't, at least in principle, be realized by a concrete machine. All of the basic Turing machine action-kinds (e.g., move to the left one square, erase an S_0 and print an S_1) can be performed by concrete machines. There are thus no number theoretic functions which can be computed by a Turing machine which couldn't, ignoring physical limitations on time and space, be computed by a concrete machine. Nevertheless, concrete machines can do more; for they can perform causally efficacious actions such as beating egg yolks and creaming butter. The upshot is that anything which can be done by a Turing machine program can, at least in theory, be done by an effective mundane procedure, but not everything which can be done by an effective mundane procedure can be done by a Turing machine program. The conclusion seems inescapable: Under the third interpretation, the Church-Turing thesis is false.

VI

This brings us to the second interpretation of the Church-Turing thesis, which says that there are no functions (number theoretic or otherwise) which couldn't be computed by means of a Turing machine program but could be computed by means of some other kind of effective procedure. As noted in Section I, this interpretation is of particular interest to computer scientists insofar as they are interested in computation in general and it postulates that Turing machine programs represent just such a limit. Most mathematicians, however, construe the Church-Turing thesis more narrowly, viz., as applying only to the number theoretic functions. In this context, it is interesting to note that there is some evidence that Turing held that the thesis applied to the real-valued (as well as number theoretic) functions, since he explicitly argued (1964, p. 22) that Turing machines could imitate the behavior of continuous machines. In any case, however, the second interpretation represents a bold and widely accepted interpretation of the Church-Turing thesis. As I argue below, there are, nonetheless, good reasons to suppose that it too is false.

In order to fully understand the second interpretation of the Church-Turing thesis, we must first become clear about the notion of computing a function. From a mathematical standpoint, a function is an assignment of values to arguments, or a set of ordered pairs. The set of all those arguments to which the function assigns values is called the domain and the set of all those values which the function assigns to its arguments is called the range. As an example, the range of the function $f(n) = 2n$, whose domain is the set of all positive integers $\{1,2,3,\ldots\}$, is the set of all even positive integers $\{2,4,6,\ldots\}$. The function maps each positive integer n to the even positive integer $2n$. Hence, the function can be regarded as having an infinite number of elements of the form $< n,2n >$.

So what is it to "compute" a function? Stephen Kleene has suggested (1988, pp. 17-19) that it requires an actual pairing of distinct values to distinct arguments. His proposal reflects the intuitively compelling notion that *each* argument in the function's domain must become matched with its assigned value in the function's range. Moreover, in order for a function to be actually computed *every* argument in its domain must be paired with the appropriate value in its range; otherwise, one has only succeeded in approximating the function through a proper subset of the set of all its ordered pairs. In other words, an actual computation of a function must mirror the detailed formal structure of the function: Each and every element in the function's domain must become matched to the appropriate element in its range.

So how do Turing machines compute functions? In order to be able to compute a function, a Turing machine must be able to represent each individual element in its domain and range by a different string of symbols; for that is the only way in which a Turing machine can represent something as a distinct individual. The Turing machine computes the function by replacing each and every string of symbols representing an argument by the string of symbols representing its assigned value, i.e., it achieves the requisite assignment of values to arguments by transforming symbolic representations of arguments into symbolic representations of values. However, the sense in which a Turing machine can be said to "transform" a string of symbols is, at best, metaphorical; being an abstract mechanism, it really can't *do* anything, i.e., perform any actions. Viewed from this perspective, Turing machines and functions are much alike: Both represent merely abstract assignments of values to arguments. Indeed, as Minsky puts it (1967, p. 132), "we can think of a Turing machine as *defining*, or *computing*, or even as *being a function*." Nevertheless, it would be a mistake to conclude that a Turing machine *is* a function, since, as even Minsky later admits (p. 134), different Turing machine can compute the same function; for a function is *only* an assignment of values to arguments, whereas a Turing machine is a *way* of assigning values to arguments, and there are different ways (qua different sequences of Turing machine actions) of assigning values to arguments.

It follows from the above account that no Turing machine can compute a function whose domain and range are the set of all real numbers, since the set of real numbers is uncountably infinite and the set of all finite strings of symbols (Turing machine tapes) is countably infinite.[19] In other words, there aren't enough strings of symbols to represent all of the real numbers as distinct individuals, let alone pairs of them. But this is what is required in order to actually compute the

function. For it is not enough to merely match a proper subset of the domain to the appropriate subset of the range. All of the values in the domain must be matched to their assigned values in the range. Insofar as no Turing machine can do this, no Turing machine is capable of precisely mirroring the detailed formal structure of a function from the reals onto the reals. As a consequence, no Turing machine can, strictly speaking, compute even the identity function $i(r)$ which assigns to each real number r the value r itself. This is not to deny that there may be other ways of representing a real number, for example, as a Dedekind cut of all the rational numbers into two classes or the limit of a convergent infinite (Cauchy) sequence of rational numbers. However, such representations cannot be used by a Turing machine to compute $i(r)$, since they cannot be translated into distinct strings of symbols. The question is: Could a physical system following an effective mundane procedure carry out such a computation?

Let us imagine a possible world in which space and time are both genuinely continuous, i.e., a world in which any interval of space and time, no matter how small, has an uncountably infinite number of places and times; it is worth noting that many people believe that the actual world is such a world. Let us further suppose that motion is a continuous process, more specifically, a process of occupying different places at different times in such a way that every intervening place along the path of motion is (at least) passed through; in other words, moving objects do not jump over (to speak metaphorically) intervening places. Thus, a moving object can be thought of as pairing places with times. But the times and places concerned are, by assumption, uncountably infinite in number. As we have seen, no Turing machine could mirror such a pairing. This certainly suggests the possibility that there are functions which can be computed by physical systems which can't be computed by Turing machines.

Now, admittedly, the elements of a continuum (whether places, times, or real numbers) are not separate from one another as are the elements of a discrete set (such as the integers). In a continuum, there are an uncountably infinite number of elements between any two elements, no matter how close in magnitude (value, size, extent) they are. As a consequence, given a particular element, it makes no sense to speak of the very *next* element; each element is merged with the elements in its "immediate" neighborhood.[20] Nevertheless, the elements in a neighborhood are distinct from one another; immediately neighboring real numbers, for example, are not exactly alike. Thus, it is possible to assign the individual elements in one continuous set to those in another. The identity function $i(r)$ for the reals is one example of such an assignment. Another is a function which maps the positive reals onto the negative reals. In short, as mathematicians have long realized, one can make good sense of the notion of a function whose domain and range contain an uncountably infinite number of elements. What is required for the computation of such a function is a way of achieving the requisite assignment of values to arguments, and such a way would be provided by a process of motion such as that described above. Indeed, if we let the places and times concerned

stand for the appropriate real numbers, then an object moving at unit speed can be interpreted as computing $i(r)$.

So far I have spoken only of processes, not procedures. But the Church-Turing thesis is ordinarily characterized in terms of procedures. We need to make sense of the claim that it is by *following an effective mundane procedure* that a physical system is able to compute a function which couldn't be computed by a Turing machine.

As discussed in Section III, the role of the action-kinds specified by an effective mundane procedure is to initiate and sustain a causal process. It is the process which actually produces the outcome with respect to which the procedure is said to be effective. The procedure is important only insofar as it represents a method for bringing about the right kind of causal process; in the case of $i(r)$, one which actually pairs an uncountably infinite number of distinct elements to an uncountably infinite number of distinct elements. In other words, the follower of the procedure does not have to match up the elements concerned; the causal process she generates can do that. In this light, consider the following procedure: 'Walk half way across the room and, then, walk the remaining distance.' This procedure specifies an effective method for getting across a room, without requiring that the person following it perform an infinite number of separate transitions in place. It is to be contrasted with the procedure given by Zeno in his infamous bi-section paradox. Zeno asks us to traverse a distance by first walking half the distance, then, half the remaining distance (3/4 of the way), then, half of that remaining distance (7/8 of the way), etc. That is to say, he specifies a linear sequence of an infinite number of separate actions, which no one can perform. Zeno concludes that motion is impossible. But he should have concluded only that his procedure was ineffective for traversing distances. For there is no reason to suppose that the process of moving is constituted by a linear sequence of separate transitions in place, just because our procedures are constituted by linear sequences of separate actions. As discussed in Section III, processes are, conceptually speaking, not procedure-like. Moreover, as I have argued elsewhere (1990, p. 227–278) there is good, reason to believe that motion is a process whereby one *passes through*, without being wholly *in*, each of an uncountably infinite number of distinct places. In other words, Zeno's bi-section paradox represents a confusion between a procedure and a process.

It should now be clear why mundane procedures are at least potentially more powerful than Turing machine programs with respect to the computation of functions. Computing a function requires that each and every argument of the function be matched to its assigned value. A Turing machine accomplishes this by transforming strings of symbols. The only way in which a Turing machine can transform strings of symbols is by "performing" linear sequences of separate actions. That is to say, a Turing machine can compute a function only if it can implement a procedure which precisely mirrors the structure of the function. No Turing machine can compute a function having an uncountably infinite number of

values and arguments, since that would require a *linear* sequence of an uncountably infinite number of *separate* actions, which is impossible. In contrast, the procedures implemented by physical systems need not, themselves, mirror the structures of the functions they compute. For an effective mundane procedure can achieve the requisite assignment of values to arguments by specifying separate action-kinds which generate the right kind of causal process; i.e., the causal process, as opposed to the procedure, can provide for the mapping of values to arguments. In other words, the limits to the computational capacities of mundane procedures are not, as in the case of Turing machine programs, set by the formal possibilities for arranging separate (discrete) actions in a linear sequence. Insofar as it is possible for an effective mundane procedure to initiate a genuinely continuous causal process, it is possible for an effective mundane procedure to compute a function that could be computed by no Turing machine.

In this context, it is worth noting that Turing considered and rejected (1965, p. 136) the significance of analog (continuous) computation on the grounds that there is no way to discriminate between arbitrarily close, but distinct, elements. This objection is commonly raised against any suggestion to the effect that analog processes might provide us with new computational insights. It is important to see that this objection is misguided. First, it is irrelevant to the issue at hand. The question is can physical processes *compute* continuous functions, not can we *tell* whether they have? If physical processes having the right kind of formal structure exist, then the Church-Turing thesis is false; it doesn't matter whether we can ever know that it is. Second, the objection tacitly assumes that the distinctness of elements can only be preserved by separate (discrete) representations. Since it is not possible to provide such representations for each and every element in a continuum, it is concluded that it is not possible to represent the elements of a continuum as distinct. Hence, it is not possible to discriminate between them. But this argument is flawed. The mathematical theory of the continuum tells us that distinct elements can fail to be separate from one another. So why should we assume that it is impossible to represent them as distinct elements? Rather than settling for mere discrete approximations of continua, what we really need to do is explore the representational capacities of continua. In other words, we need to develop modes of representation other than those provided by strings of discrete symbols. Perhaps such modes of representation are to be found in connectionist dynamical systems, whose states are defined by continuous numerical values (connection strengths, activations) which change continuously in time. In any case, however, my point remains: It is not at all obvious that we require discrete representations in order to preserve the distinctness of elements. Hence, there is no a priori reason to suppose that analog processes couldn't provide us with new insights into the nature of computation, including, perhaps, a definitive refutation of the second interpretation of the Church-Turing thesis.

So what are the limits of computation? Insofar as functions represent purely formal assignments of values to arguments and Turing machines do not pose limits to the causal possibilities of actually assigning values to arguments, the fact that a particular function cannot be computed by a Turing machine has no bearing

upon whether or not the function is computable. This is just as true for the so-called uncomputable number-theoretic functions as it is for real-valued functions such as $i(r)$. Intuitively speaking, a function is computable if it is *possible* to actually match each and every element in its domain to the assigned element in its range, i.e., if there is some possible world in which it is computed. The mere existence of the function establishes the bare possibility of a matching relation; for that is exactly what a purely formal assignment of values to arguments represents! Whether a particular function can be in fact computed depends, however, upon the detailed causal structure of our world, i.e., upon whether there exist causal processes having the requisite formal structure. It is very likely that not all functions can be computed in our world. It is also possible that some functions that cannot be computed in our world can, nonetheless, be computed in other causally possible worlds, i.e., worlds whose causal structure differs in important ways from our world. Such functions could be said to be in theory, but not in fact, computable. Moreover, it is possible that there exist functions (e.g., the uncomputable number-theoretic functions) that cannot be computed in any causally possible world and, hence, are not even in theory computable. But in the absence of any idea as to the theoretical and/or factual capacities of causal processes to mirror formal structures, we have no reasons for believing that this is so. Indeed, to the extent that we can readily imagine a scenario in which an effective mundane procedure could be used to compute a function that could not be computed by any Turing machine program (e.g., $i(r)$), we have every reason to suppose that the formal structure of causal processes is potentially richer than the formal structure of Turing machine programs, and, hence, good reason for withholding judgement on even the narrowest version of the Church-Turing thesis. In short, the limits to computation are not to be found in abstract Turing machine theory. They are to be discovered through empirical research and philosophical analysis. For the limits to computation are not logical; they are causal.

In summary, three different interpretations of the Church-Turing thesis appear in the literature. I have shown that the broadest interpretation of the thesis (which holds that no effective procedure could do something that couldn't be done by means of a Turing machine program) *is* false. There are things which can be done by following an effective mundane procedure that cannot be done by following a Turing machine program, e.g., make Hollandaise sauce. This is not to deny that a physical embodiment of a Turing machine could make Hollandaise sauce. It is, however, to deny that Hollandaise sauce could be produced by the following of a mere Turing machine program; for Hollandaise sauce is a product of a causal process and, strictly speaking, the action-kinds specified by a Turing machine program have no causal consequences whatsoever. What is required to produce a Hollandaise sauce is a physical analogue of a Turing machine program and, as I have argued, a physical analogue of a Turing machine program is not, itself, a Turing machine program; rather, it is a mundane procedure.

I have also shown how the second interpretation of the Church-Turing thesis *could be* false. Whether it is in fact false depends upon the causal structure of the world. If, as many believe, there exist genuinely continuous causal processes, then the capacities of actual physical systems to compute (i.e., mirror the structure of) functions exceeds that of Turing machines. Finally, I have argued that there is at present no conceptual basis for even the modest supposition that no effective mundane procedure could compute a so called uncomputable number-theoretic function. This is not to claim that an effective mundane procedure could compute such a function or even to show how it would be possible for an effective mundane procedure to compute such a function. It is to shift the burden of proof onto defenders of the first interpretation of the Church-Turing thesis. For when all is said and done we have good reason to suppose that the computational capacities of causal processes exceed those of Turing machine programs and we have no idea as to the ultimate limits of causal processes to mirror precisely the structure of functions.[21]

Notes

[1] See, for example, R. J. Nelson's commentary (1988) on Paul Smolensky's account (1988) of connectionism.

[2] Such claims are also frequently made in informal discussions over electronic mail.

[3] Again, electronic mail is a good source for such claims.

[4] These domains of application are not mutually exclusive. Most people who take the thesis to apply to physical and/or mental phenomena also take it to apply to functions in general. But it is not clear that everyone does. As Bill Reinhardt has pointed out to me, Turing may not have held that it applies to the real-valued functions, since he never explicitly suggested in any of his writings that the thesis applied to functions other than the number theoretic functions. On the other hand, Turing considers and rejects (1964) an objection to his account of mental processes based upon the continuity of the nervous system, which certainly suggests that he may have taken the Church-Turing thesis as applying to the computation of real-valued functions as well as number theoretic functions.

[5] Roger Penrose (1990, pp. 400–404), for example, has speculated that quantum systems could provide an exception to the stronger versions of the Church-Turing thesis.

[6] This recipe is from my personal file. It is from an old (and, alas, long gone) issue of *Bon Appetit*.

[7] from a recent *Boy Scout Manual*.

[8] I am assuming here that the performance of an action does not, as many philosophers have maintained, presuppose intentionality. On my view, natural phenomena (e.g., hurricanes, floods) and machines can "perform" actions too. For a defense of this position, see my paper "Events, Actions and Mere Happenings" (forthcoming).

[9] Like Goldman (1970, p. 10), I identify the performance of an action-kind with the exemplification of a special kind of dynamic property (an "act-property"). In "Events, Actions and Mere Happenings" (forthcoming), I argue that action-kinds are "purely causal properties," i.e., properties whose application presupposes only that a certain kind of effect occur. The notion that an action-kind is a purely causal property helps us to make sense of the notion that an action is a *doing*, as opposed to a mere occurrence. As an example, *any* occurrence which causes a light to come on instances the action-kind of turning on a light. Such occurrences include toggles being placed in upright positions, buttons being depressed and hands being waved in rooms equipped with high tech motion activated lighting systems. Moreover, any occurrence which does not cause a light to come on cannot be said to instance the turning on of a light, no matter how similar it is (in non-causal properties) to other occurrences which have, in the past, caused a light to come on. Thus, if today the switch on my office wall is broken, my flipping it will not qualify as an action of turning on a light, despite the fact that it did yesterday. This is, I argue, in direct contrast to those generic events which are not action-kinds (e.g., the collapsing of a bridge), whose application does not presuppose the occurrence of any kind of effect.

[10] On my analysis of action this amounts to presupposing the same kinds of effects.

[11] In contrast, mundane procedures, whose instructions are frequently non-conditional, are most often represented by lists. Nevertheless, this difference in mode of representation does not represent a fundamental difference between Turing machine programs and mundane procedures. For any non-conditional instruction can be transformed into a conditional instruction by making the performance of the instruction conditional upon the performance of the preceding instruction. As an example, in our recipe for Petite Mont Blanc one could secure the temporal order of the specified action-kinds by using conditional instructions such as: "beat butter until it becomes creamy;" "when butter has become creamy, add brown sugar and flour."

[12] I am indebted to Paul Smolensky for mercilessly pressing me on this point.

[13] In this paper, all the mundane procedures I discuss are finite. It is worth noting, however, that some mundane procedures can be construed as infinite insofar as they require the indefinite repetition of a finite number of distinct actions; an example, which I owe to Clayton Lewis, is crop rotation.

[14] I am not committed here to any particular conception of possible worlds.

[15] As is illustrated by the points on a real line, elements can be *different* from one another without being *separate* from one another; this is an important point which I will develop in Section VI.

[16] This expression was first coined by Hans Reichenbach (1956, p. 159).

[17] For an excellent survey of the major attempts (many of which have been based upon a reputed syntactic and/or semantic distinction between law statements and non-law statements), see the introduction to Myles Brand's *The Nature of Causation* (1976).

[18] As Roderick Chisholm pointed out some time ago (1955), it does not seem possible to draw the distinction between law statements and non-law statements without employing modal terms such as "causal dependency," "physical possibility," "necessary connection" and the like.

[19] Of course, the uncountability of the reals is not crucial to the point I am making here; for the number of functions from the natural numbers into the natural numbers is also uncountably infinite. Hence, there must be number-theoretic functions which are not computable by any Turing machine. I chose functions from the reals onto the reals for the reason that it is easier to see why they are not computable by any Turing machine, namely, it is obvious that there aren't enough strings of Turing machine symbols to represent each and every one of their uncountably infinite number of arguments; in contrast, the number of elements in the domain of any function from the natural numbers into the natural numbers is not uncountably infinite.

[20] In mathematical terminology, its ε neighborhood, or the set of all elements whose distance from the element in question is less than some vanishingly small distance ε.

[21] I am indebted to a number of people for comments on earlier drafts of this paper but I would particularly like to single out my CU colleagues Paul Smolensky, Computer Science Department, Bill Reinhardt, Department of Mathematics, and Michael Tooley, Department of Philosophy, for extensive comments and very useful follow up discussions. I am also indebted to the Center for the Study of Language and Information (CSLI) at Stanford University for providing me with the opportunity to spend several weeks discussing these issues with institute members during June of 1991 and to the Graduate Committee on the Arts and Humanities at the University of Colorado (Boulder) for funding my visit to Stanford.

References

Bennett, Charles (1988), 'Logical Dept and Physical Complexity', in R. Herkin, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford: Oxford University Press, pp. 227–258.

Brand, Myles (1976), *The Nature of Causation*, Chicago: University of Illinois Press.

Child, Julia (1961), *Mastering the Art of French Cooking*, New York: Knopf.

Chisholm, Roderick (1955), 'Law Statements and Counterfactual Inference', *Analysis* 15, pp. 97–105.

Church, Alonzo (1934), 'The Richard Paradox', *American Mathematical Monthly* 41, pp. 356–361.

Church, Alonzo (1965), 'An Unsolvable Problem of Elementary Number Theory', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 88–107.
Churchland, Paul and Patricia (1990), 'Stalking the Wild Epistemic Engine', in W. Lycan, ed., *Mind and Cognition*, Cambridge: Basil Blackwell.
Cleland, Carol (1985), 'Causality, Chance and Weak Non-Supervenience', *American Philosophical Quarterly* 22, pp. 287–298.
Cleland, Carol (1990), 'The Difference Between Real Change and *Mere* Cambridge Change', *Philosophical Studies* 60, pp. 247–278.
Cleland, Carol (forthcoming), 'Events, Actions and Mere Happenings'.
Finkelstein, David (1988), 'Finite Physics', in R. Herkin, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford: Oxford University Press, pp. 349–376.
Geroch, Robert & Hartle, James (1986), 'Computability and Physical Theories', *Foundations of Physics* 16, pp. 533–548.
Goldman, Alvin (1970), *A Theory of Human Action*, Princeton: Princeton University Press.
Hamlet, Richard (1974), *Introduction to Computing Theory*, New York: Intext.
Harre, Rom (1970), *Principles of Scientific Thinking*, Chicago: UNiversity of Chicago Press.
Hofstadter, Douglas (1980), *Gödel, Escher, Bach*, New York: Random House.
Hofstadter, Douglas (1981), 'A Conversation with Einstein's Brain', in D. Hofstadter & D. Dennett, eds., *The Kind's I*, New York: Basic Books, pp. 439–446.
Hume, David (1975), "Treatise of Human Nature (Bk. I, Pt. III, Sec. XIV)', in Selby-Bigge, ed., *Hume's Treatise of Human Nature*, Oxford: Oxford University Press, pp. 155–172.
Kim, Jaegwon (1980), 'Causes and Counterfactuals', in E. Sosa, *Causation and Conditionals*, Oxford: Oxford University Press, pp. 192–194.
Kleene, Steven (1936), 'Definability and Recursiveness', *Duke Mathematical Journal* 2, pp. 340–353.
Kleene, Steven (1988), 'Turing's Analysis of Computability, and Major Applications of It', in R. Herkin, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford: Oxford University Press, pp. 17–54.
Minsky, Marvin (1967), *Computation: Finite and Infinite Machines*, Englewood Cliffs: Prentice-Hall.
Nelson, R. J. (1988), 'Connections Among Connections', *Behavioral and Brain Sciences* 11, pp. 45–46.
Penrose, Roger (198), 'On the Physics and Mathematics of Thought', in R. Herkin, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford: Oxford University Press, pp. 491–522.
Penrose, Roger (1990), *The Emperor's New Mind*, Oxford: Oxford University Press.
Reichenbach, Hans (1956), *The Direction of Time*, Berkeley: University of California Press.
Rosen, Robert (1988), 'Effective Processes and Natural Law', in R. Herkin, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford: Oxford University Press, pp. 523–537.
Rubel, Lee A. (1989), 'Digital Simulation of Analog Computation and Church's Thesis', *Journal of Symbolic Logic* 54, pp. 1011–1017.
Searle, John (1985), 'Minds, Brains and Programs', in Haugeland, ed., *Mind Design*, Cambridge: The MIT Press.
Shapiro, Stewart (1981), 'Understanding Church's Thesis', *Journal of Philosophical Logic* 10, pp. 352–365.
Smolensky, Paul (1988), 'The Proper Treatment of Connectionism', *Behavioral and Brain Sciences* 11, pp. 1–23.
Turing, Alan, (1937), 'Computability and λ-definability', *Journal of Symbolic Logic* 2, pp. 153–163.
Turing, Alan (1964), 'Computing Machinery and Intelligence', in A. Anderson, ed., *Minds and Machines*, New Jersey: Prentice Hall, pp. 4–30.
Turing, Alan (1965), 'On Computable Numbers, With an Application to the *Entscheidungs* Problem', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 116–154.
von Neumann, J. (1966), *Theory of Self-Reproducing Automata*, Urbana: University of Illinois Press.
Weizenbaum, Joseph (1986), *Computer Power and Human Reason*, San Francisco: Freedman & Co.