The Church/Turing Thesis
Procedures, Processes and Causation

Carol E. Cleland

Department of Philosophy
and
Institute of Cognitive Science
University of Colorado
Boulder, CO 80309

Technical Report 92-04

The Church/Turing thesis makes a bold claim about the theoretical limits to computation. It is based upon an analysis of the general notion of an effective procedure ("definite method") proposed by computer scientist Alan Turing (1965). Turing based his analysis on the concept of a very simple abstract mechanism now known as a "Turing machine." Other analyses of the concept of an effective procedure had previously been presented, e.g., Church-Kleene λ-definability (Kleene, 1965), Herbrand-Godel general recursiveness (Godel, 1965), but unlike Turing's, these were designed only for the analysis of mathematical decision procedures. Like these analyses, Turing's was grounded in mathematical considerations. However, Turing argued (1964, p. 9; 1965, p. 117) that his analysis could be extended to any effective procedure whatsoever, whether mathematical or non-mathematical. Turing computability was later shown to be equivalent to the mathematical notions and, since Turing computability was supposed to represent an analysis of our general, intuitive concept of an effective procedure, it was concluded that there could be no effective procedures more powerful than Turing machine procedures. This result has since come to be known as the Church/Turing thesis.

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 than any Turing machine procedure. Although a number of philosophers and scientists (e.g., Penrose, 1988; Shapiro, 1981; Smolenky, 1988) have expressed misgivings about the applicability of the thesis to actual physical systems, there currently do not exist any generally accepted counter-examples to it. This could, of course, be because the thesis is true. 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 the mere suspicion that some exotic physical system (e.g., a quantum system) could out perform any Turing machine. 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 Section I-III of this paper I develop an alternative analysis of the general concept of an effective procedure. Unlike Turing's account, my account 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 maintain that effective mundane procedures differ from effective Turing machine procedures in representing causal, as opposed to formal, ways of manipulating the world. In Sections IV-V I argue that much of the plausibility of the Church/Turing thesis and many of the concerns of patent rights experts about the identity of procedures derive from a failure to carefully differentiate Turing machine procedures from their physical embodiments. Indeed, it is my contention that while no Turing machine can adequately simulate the causal activity of a physical system, a physical system can always simulate the formal activity of a Turing machine. The upshot is that there are things which can be done by following a mundane procedure which cannot be done by following any Turing machine procedure. Subsequently, I explicitly address the concerns of mathematicians, arguing that it is at least possible for a physical system to compute a function which could not be computed by any Turing machine. I conclude that the truth of the Church/Turing thesis is a

wholly contingent matter, dependent upon the causal structure of our world as opposed to the merely formal possibilities posed by Turing machines.

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), "a 'procedure' is intuitively something to follow" (p.2). 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. ...[1]

B) A method for building a fire by friction:

> ... 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.[2]

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 a deed which is to be done. The deed is always an action; it represents an occurrence (event) which is to be brought about, as opposed to something which merely happens or is undergone.[3] Moreover, instructions are ordinarily expressed by imperatives. The function of the imperative form of the verb is to indicate that the action being referred to is *to be* performed--that it is not an action which will be performed by someone or something else. 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.[4] 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.

(Indeed, as any moderately experienced cook knows, failure to do so is an invitation to culinary disaster!) In this case, the order in which the action-kinds are to be performed is specified externally by the order of presentation of the instruction-expressions: The instruction to cream the butter immediately precedes the instruction to add the brown sugar and flour. 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. In short, mundane procedures specify not only that certain action-kinds be performed but, also, that they be performed in a certain relative order in time.

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.[5]

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 $\{S_0, ..., S_n\}$, 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, $q_1, ..., q_m$. 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 are 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:

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

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 in the set above can be read

as follows: if the machine is in state q_1 (following instruction 1) and the head is scanning an S_0, erase the So, 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:

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

Fig. 2

Each of the above six sequences 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 each 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 the following machine table:

	S_0	S_1
q_1	S_1q_1	Lq_2
q_2	S_1q_2	Lq_3
q_3	S_1q_3	

Fig. 3

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., (1) print an S_1 and go into state q_1, or (2) 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 or an S_1. That the machine table in question defines the same instruction set (and, hence, Turing machine program) as the set of quadruples in Fig. 1 can be verified by writing down the sequence of machine configurations it specifies for any initial machine configuration (i.e., initial arrangement of symbols on its tape, initial position of head over tape and initial state of the machine) and, then, comparing them with the sequence of machine configurations specified by the set of quadruples for the same initial machine configuration; they will always be the same.

We are now in a position to compare explicitly 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 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. So given that the machine is started out in some specific initial configuration (which, by definition, it always is), 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.[6] 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. That is to say, our general concept of an effective procedure is the concept of a procedure which consistently terminates in the same generic outcome, when implemented. Effectiveness is not, however, characterized this way in Turing machine theory.[7] For as Marvin Minsky explains, in his canonical work *Computation: Finite and Infinite Machines* (1967, pp. 105-106) the implementation of some mathematical and abstract mechanical procedures require 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. Yet each and every step may be completely determined in advance. Thus, it is possible for a procedure which has no final outcome to, nevertheless, have a predetermined outcome at each and every stage of its implementation. In order to accommodate such procedures, effectiveness is defined in Turing machine theory in terms of the individual steps of the procedure, as opposed to the outcome of the procedure as a whole. More specifically, a procedure is said to be effective if it is specified with sufficient completeness and precision such that "... the next step is always clearly determined in advance" (Minsky, 1967, p. 105).

Any finite procedure which is effective in Minsky's sense ("effective$_2$") will also be effective in the ordinary, everyday sense ("effective$_1$"), since any finite sequence of predetermined actions terminates in a predetermined action. As we have seen, every finite Turing machine program completely specifies the performance of a finite sequence of definite action-kinds. Hence, all finite Turing machine programs will qualify as effective$_1$ procedures. But it is not obvious that all effective$_1$ mundane procedures will qualify as effective$_2$ procedures. Whether an effective$_1$ mundane procedure qualifies as an effective$_2$ procedure depends upon whether the action-kinds specified by its instruction set have been "well defined." Is it possible for a mundane procedure to be effective in the ordinary, everyday sense and, yet, not well defined?

It has been claimed that many effective$_1$ mundane procedures are not effective$_2$ on the grounds that their instructions do not describe the action-kinds they specify with adequate precision. But this is hardly obvious. Our recipe for Petite Mont Blanc, for example, tells us to cream the butter. What is so imprecise about this? Certainly one has to know how to "cream" butter. Similarly, a Turing machine has to know (to speak very loosely) how to "print" symbols in order to print an S_0. The only difference I can see between the two cases is that we are expected to simply assume that the Turing machine instruction, but not the recipe instruction, precisely describes the action-kind it specifies.

But why would anyone suppose that this is so? Just as cooks can fail to cream butter, concrete machines can fail to print symbols. In neither case is the source of the problem obvious. It could be due to an inherent vagueness in the instruction or a simple failure (through ignorance or mechanical error) to correctly follow the instruction. Now, admittedly, no Turing machine which is instructed to print an S_0 can fail to print an S_0, but this has more to do with the hypothetical nature of a Turing machine than the precision with which the action-kind concerned is described. By definition, Turing machines are ideal mechanisms which are not subject to error. In this light, it seems that one could equally well postulate an ideal chef who never makes mistakes when following recipes. This does not, of course, prove that our recipe for Petit Mont Blanc is precisely described. But it does demonstrate that there is little evidence for the claim that it is less precisely described than a Turing machine program.

But even supposing that our recipe for Petit Mont Blanc were inherently vague, it would be a mistake to conclude that it was ipso facto ineffective. For the Turing machine notion of effectiveness was originally designed as an extension (as opposed to revision) of our common sense notion of effectiveness. To turn around and claim that our common sense notion is somehow defective because it does not conform to conditions which were provided for the explicit purpose of accommodating some questionable cases of effectiveness is circular. Indeed, it seems much more reasonable to deny the effectiveness of infinite procedures (procedures which never terminate) than to deny the effectiveness of tried and true recipes and methods. In other words, the notion of terminating in a definite outcome is at the very core of our concept of an effective procedure, whether Turing machine or mundane. Viewed from this perspective, it is a mistake to speak as if there were two different notions of effectiveness. There is only one general notion of effectiveness, namely, the notion of a procedure which always terminates (when implemented) in a definite generic outcome. This notion can be extended, in certain exceptional cases, to encompass procedures which do not terminate, but this has no bearing upon the status of

those that do. In short, I propose that we reject, as confused, the claim that it is possible for a procedure to always terminate (when implemented) in the very same kind of outcome and, yet, fail to be genuinely effective.

Effectiveness is not an absolute property of a procedure. 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, effectiveness is a relational property whose relata are a procedure and a generic outcome (or, in the case of an infinite procedure, an infinite sequence of generic outcomes). Effective procedures are ineffective vis-a-vis most kinds of outcomes. However, those generic outcomes with respect to which a procedure is effective are outcomes which are always realized when the procedure is followed. The question is how is such an outcome produced by the mere following of a procedure?

Let us begin by looking at the role of the action-kinds specified by the instruction list of an effective mundane procedure. 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 wisk. What makes her recipe effective vis-a-vis 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 Julia's recipe is dependent upon the causal capacities of actions of the kinds specified by her recipe to initiate and sustain a certain kind of causal process. Similarly, the effectiveness of mundane procedure B 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 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 its 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;[8] 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 both effective and ineffective vis-a-vis the same kind of outcome, depending upon the causal structure of the world in which it happens to be implemented.

In contrast, the effectiveness of a Turing machine program (vis-a-vis a certain kind of result) does not vary from possible world to possible world. For 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 represents failure to perform an action of the designated kind.[9] 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 (so-to-speak) of a Turing machine in a concrete machine can generate a causal process. 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 processes, can embody the very same abstract Turing machine. In other words, what a Turing machine does *qua* Turing machine is the same in all possible worlds.

Accordingly, there does seem to be a difference between effective mundane procedures and effective Turing machine procedures. 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. However, such a distinction between effective mundane procedures and effective Turing machine procedures will be interesting only if there is a fundamental difference between causal processes and sequences of actions (instances of procedures); otherwise, a mundane procedure will not differ in any interesting way from a complex Turing machine program in which one program (the process) is "driven" by another (the procedure).

One way in which causal processes seem to differ from mere instances of procedures is in the interrelation of their subparts, actions in the case of procedures and subprocesses in the case of causal processes. 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.[10] 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, vs. 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 Kim (1974, pp. 41-52; 1980, pp. 192-194) has noted, 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,"[11] 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'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 machine procedures, 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 arrangment 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.[12] Although none of these neo-Humean accounts have been very successful,[13] 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.

But why didn't Turing notice this difference between effective mundane procedures and effective Turing machine procedures? 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. However, as we have seen, not all of the procedures we follow represent 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 of the ways in which we manipulate the world exploit the causal structure of reality. Indeed, the difference between an effective mundane procedure and an effective Turing machine procedure is just the difference between a causal and a formal way of manipulating the world.

IV

As discussed in Section II, our 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. That is, it suggests that effective Turing machine procedures and effective mundane procedures do not 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 actions. 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 automatically 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 abstract entities, 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, 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 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 simply do not know how the kinds of actions they specify are capable of producing 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 we shall soon see, the Church/Turing thesis loses much of its 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. The Church/Turing thesis is often characterized as the claim that there are no effective procedures more powerful than effective Turing machine programs. But what, exactly, does this mean?

An extreme view is that it amounts to the claim that anything which can be done by following a mundane procedure could be done by following a Turing machine program. Although many mathematicians and computer scientists reject this interpretation, it has a number of well known advocates. Paul and Patricia Churchland (1990), for example, have claimed that "what the mind-brain does is [Turing] computable" (p. 301). Similarly, Douglas Hofstadter (1980) has maintained that "the behavior of any component (typically assumed to be a cell) can be calculated ... to any desired degree of accuracy, given a sufficiently precise description of the component's internal state and local environment" (p. 572). It has even been suggested, e.g., by Charles Bennett (1988, pp. 227-257), that the thesis can be extended to physical processes in general, as opposed to just biological processes.

Granting, for the sake of argument, that this interpretation is correct, it follows, from my account, that the the Church/Turing thesis is demonstrably false. Julia Child's easy blender method for making Hollandaise sauce provides one, among many, examples of a mundane procedure which is "more powerful" than 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 absence 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 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 a 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 is a system involving a roll of toilet paper and some stones.[14] 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. For 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 kinds of actions. Accordingly, the procedures which specify them cannot be the same, since, as I have argued, procedures differ if the kinds of action they specify differ. In other words, physical analogues of Turing machine programs are, strictly speaking, not Turing machine programs.

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 by a Turing machine. Hence, there can't be a fundamental difference between a physical system and a Turing machine. Simply put, this objection amounts to the claim that the activity of a physical system is, for all intents and purposes,

just like the activity of a Turing machine. This is, presumably, what Douglas Hofstadter (1981, pp. 439-446) had in mind when he claimed 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.).

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 Turing machine actions. Thus, the activity of a Turing machine consists just in the performance of a sequence of actions. In contrast, a physical system can initiate and sustain causal processes. 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 causal 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 many philosophers have come to believe,[15] causation is a "physically real" relation, then they may be very serious indeed.

As I have argued, 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 duplicate the causal activity of a physical system, i.e., no Turing machine can generate a causal process. It is for this reason that, as John Searle notes (1985, p. 305), no formal simulation of lactation can produce milk and no formal simulation of photosynthesis can produce sugar. This is not, of course, to deny that it is at least possible for a Turing machine program to be embodied in a physical system which produces milk or sugar when the physical analogue of the program is implemented. But in such a case it would be a

mistake to infer that what produced the milk or the sugar was the mere performance of a sequence of actions of the kind specified by the Turing machine program. Insofar as Turing machine actions have no causal consequences, they can't produce anything. However, their physical analogues can. That is to say, it is the physical analogues of the Turing machine actions which are responsible for the production of the milk or the sugar. As I have argued, physical analogues of Turing machine actions are not Turing machine actions, even supposing that they share the same formal properties. Hence, the fact that it is possible for a Turing machine program to be embodied in a physical system which produces milk or sugar does not establish that a Turing machine, qua purely formal system, could produce milk or sugar. In short, no Turing machine can adequately simulate the causal activity of a physical system.

On the other hand, it seems that there are no activities of Turing machines which couldn't, at least in principle, be adequately simulated by a physical system! For although Turing machine actions lack the causal properties of physical actions, there are no formal properties of Turing machine actions which couldn't be had by a physical action. It is for this reason that physical simulations of addition can produce sums and physical simulations of division can produce quotients. The upshot is that anything which can be done by a Turing machine program can, at least in principle, be done by a mundane procedure, but not everything which can be done by a mundane procedure can be done by a Turing machine program. The conclusion seems inescapable: Under the proposed interpretation, the Church/Turing thesis is false.

Many mathematicians and scientists, however, accept a much weaker version of the Church/Turing thesis. They do not accept the claim that a Turing machine can do *anything* which can be done by a physical system, but the much more modest claim that any function which can be computed by a physical system can also be computed by a Turing machine. In other words, they accept the "... computational equivalence of real computers and

Turing machines" (Malitz, 1979, p. 90). In order to understand fully this claim we need to become clearer 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,...}, is the set of all even positive integers {2,4,6,...}. 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 (1988, pp. 17-19) has suggested that it requires an actual pairing of values and arguments. His proposal reflects the intuitively compelling notion that the computation of a function must mirror the detailed formal structure of the function--that *each* argument in the function's domain must become matched with its assigned value in the function's range. Thus, in order to compute a function, a Turing machine must be able to represent each individual element in the function's 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 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 (1967) puts it, "we can think of a Turing machine as *defining*, or *computing*, or even as *being a function.*" (p.

132). 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 machines 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.[16] 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. As a consequence, no Turing machine is capable of precisely mirroring the detailed formal structure of a function from the reals onto the reals. This means that 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 a mundane procedure carry out such a computation?

Let us imagine a possible world in which space and time are both 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 merges with its immediate neighbors. Nevertheless, the elements of a neighborhood are distinct from one another; no two immediately neighboring real numbers, for example, are 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* a 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 a mundane procedure is to initiate and sustain a 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 process; in the case at hand, one which 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 process she generates can do that. Thus, the procedure 'walk half way across the room and, then, walk the remaining distance' specifies an effective method for getting across the room, without requiring that the person following it perform an uncountably infinite number of separate transitions in place. This procedure is to be contrasted with the one 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 for traversing distances is ineffective. 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, pp. 274-278), 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 even with respect to the computation of functions. Computing a function requires that each 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 mirror the structures of the functions they compute. For a mundane procedure can achieve the requisite assignment of values to arguments by specifying action-kinds which generate the right kind of causal process; i.e., the causal process, as opposed to the procedure, can provide for the actual mapping of values to arguments. Hence, the limits to the computational capacities of mundane procedures are not, as Turing believed, set by the computational capacities of linear sequences of separate (discrete) actions.

So what are the limits of computation? Insofar as functions represent formal assignments of values to arguments and Turing machines pose no a priori limits to the 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 it is in principle or in fact computable. A function is in principle computable, if it is possible to match each element in its domain to the assigned element in its range. The mere existence of the function establishes the bare possibility of the matching relation; for that is exactly what a purely formal assignment of values to arguments represents! Whether a particular function is in fact computable depends upon the detailed causal structure of our world, i.e., upon whether there exist causal processes having the requisite formal structure. Not all functions are computable in our world, however, many functions which are not computable in our world are computable in other possible worlds. Whether all functions are at least in principle computable depends upon the general nature of causation--upon whether there are any inherent limits to the capacities of causal processes to precisely mirror the structures of functions. In short, the place to look for the limits to computation is not abstract Turing machine theory. For the limits to computation are not logical. They are causal.

In the space remaining, I would like to consider very briefly an objection to this line of argument. In his pioneering paper (1965, p. 136), Turing rejected 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 insights into the nature of computation. It is my contention 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 the concept of a continuous process, cycling through an uncountably infinite number of states. 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.[17]

NOTES

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

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

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

[4] Like Goldman, *A Theory of Human Action*, (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 "Actions and Events" (forthcoming), I argue that action-kinds are "purely causal properties," i.e., properties whose application presupposes only that a certain kind of effect occur. 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.

5 on my view (see "Actions and Events" (forthcoming)) this amounts to presupposing the same kinds of effect.

6 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."

7 I am indebted to Paul Smolensky for pressing me on this point.

8 I am not committed here to any particular conception of possible worlds.

9 Any occurrence which causes an S_0 to be printed on a tape is an action of printing an S_0 on a tape and any occurrence which doesn't result in an S_0 being printed on a tape is not (no matter how similar it is to other actions which have caused an S_0 to appear on a tape) an action of printing an S_0 on a tape; see my "Actions and Events" (forthcoming).

10 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 V.

11 This expression was first coined by Hans Reichenbach (1956, p. 159).

12 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).

13 As Roderick Chisholm pointed out some time ago, 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; see his "Law Statements and Counterfactual Inference" (1955).

14 See his *Computer Power and Human Reason* (1976), pp. 51-52.

15 See, for example, my "Causality, Chance and Weak Non-Supervenience" (1985) and Rom Harre's *Principles of Scientific Thinking* (1970).

16 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 onto 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 onto the natural numbers is not uncountably infinite.

17 I would like to thank Paul Smolensky, Chris Shields and Paul Saalbach for helpful comments on the first draft of this paper. I am also indebted to members of the Center for the Study of Language and Information (CSLI) at Stanford University for useful discussions last summer and 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 Depth and Physical Complexity" in Herkin (ed.) *The Universal Turing Machine: A Half-Century Survey*, Oxford: The University Press.

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, 97-105.

Churchland, Paul & Patricia

1990 "Stalking the Wild Epistemic Engine" in Lycan (ed.) *Mind and Cognition*, Cambridge: Basil Blackwell.

Cleland, Carol

1985 "Causality, Chance and Weak Non-Supervenience," *American Philosophical Quarterly* 22, 287-298.

1990 "The Difference Between Real Change and Mere Cambridge Change," *Philosophical Studies* 60, 257-280.

Godel, Kurt

1965 "On Undecidable Propositions of Formal Mathematical Systems" in Davis (ed.) *The Undecidable*, New York: Raven Press.

Goldman, Alan

1970 *A Theory of Human Action*, Princeton: The University Press.

Hamlet, Richard

1974 *Introduction to Computing Theory*, New York: Intext.

Harre, Rom

1970 *Principles of Scientific Thinking*, Chicago: University Press.

Hofstadter, Douglas

1980 *Godel, Escher, Bach*, New York: Random House.

1981 *The Mind's I*, New York: Basic Books.

Hume, David

1975 "Treatise of Human Nature (Bk. I, Pt III, Sec. XIV)" in Selby-Bigge (ed.) *Hume's Treatise of Human Nature*, Oxford: The University Press.

Kim, Jaegwon

1974 "Noncausal Connections," *NOUS* 8, 41-52.

1980 "Causes and counterfactuals" in Sosa (ed.) *Causation and Conditionals*, Oxford: University Press.

Kleene, Stephen

1965 "General Recursive Functions of Natural Numbers" in Davis (ed.) *The Undecidable*, New York: Raven Press.

Malitz, Jerome

1979 *Introduction to Mathematical Logic*, New York: Springer-Verlag.

Penrose, Roger

1988 "On the Physics and Mathematics of Thought" in Herkin (ed.) *The Universal Turing Machine: A Half-Century Survey*, Oxford: The University Press.

Reichenbach, Hans

1956 *The Direction of Time*, Berkeley: University of California Press.

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, 352-365.

Smolensky, Paul

1988 "On the Proper Treatment of Connectionism" *in Behavioral and Brain Sciences* 11, 1-23.

Turing, Alan

1964 "Computing Machinery and Intelligence" in Anderson (ed.) *Minds and Machines*, New Jersey: Prentice-Hall.

1965 "On Computable Numbers, With an Application to the *Entscheidungs* problem" in Davis (ed.) *The Undecidable*, New York: Raven Press.

Weizenbaum, Joseph

1976 *Computer Power and Human Reason*, San Francisco: Freedom & Co.