

# On Effective Procedures

CAROL E. CLELAND

*Department of Philosophy & Institute for Cognitive Science, University of Colorado, Boulder, CO, USA*

**Abstract.** Since the mid-twentieth century, the concept of the Turing machine has dominated thought about effective procedures. This paper presents an alternative to Turing's analysis; it unifies, refines, and extends my earlier work on this topic. I show that Turing machines cannot live up to their billing as paragons of effective procedure; at best, they may be said to provide us with mere procedure schemas. I argue that the concept of an effective procedure crucially depends upon distinguishing procedures as definite courses of action(-types) from the particular courses of action(-tokens) that actually instantiate them and the causal processes and/or interpretations that ultimately make them effective. On my analysis, effectiveness is not just a matter of logical form; 'content' matters. The analysis I provide has the advantage of applying to ordinary, everyday procedures such as recipes and methods, as well as the more refined procedures of mathematics and computer science. It also has the virtue of making better sense of the physical possibilities for hypercomputation than the received view and its extensions, e.g. Turing's o-machines, accelerating machines.

**Key words:** causal process, effective procedure, hypercomputation, precisely described instruction, procedure schema, quotidian procedure, Turing machine

## 1. Introduction

Whether one is talking about conventional computation or theorizing about hypercomputation, Turing's analysis dominates thinking about effective procedures. This paper presents an alternative to Turing's analysis; it unifies, refines, and extends my earlier work on this topic (Cleland, 1993, 1995, 2001).

I begin by reviewing the history of the received view. In the technical literature of computer science, the concept of effective procedure is closely associated with the idea of a 'precisely described' or 'well-defined' action. As I show in Section 3, however, Turing machines cannot live up to their reputation of providing perfectly precise specifications of action. The claim that Turing machine operations are 'purely mechanical' provides the primary motivation for the historic claim that the concept of the Turing machine (unlike general recursiveness or lambda-definability) captures the intuitive idea of a definite method for calculating functions. But as I argue in Section 3, this claim confuses different meanings of the word 'mechanical'. The source of the great generality of Turing machines supposedly lies in the purely formal character of their symbols and operations; anything having the right structural features can instantiate them. Yet as I argue in Section 3, we cannot get bona fide actions out of purely formal symbols and operations. I conclude by defending the radical claim that Turing machines don't provide



*Minds and Machines* 12: 159–179, 2002.

© 2002 Kluwer Academic Publishers. Printed in the Netherlands.

us with bonafide procedures; they provide us with mere procedure schemas. When these schemas are filled in with genuine action, we get authentic procedures.

In Section 4, I return to the pre-analytic, intuitive notion of effectiveness, namely, the idea of a procedure that reliably produces a definite result when followed. I argue that one must carefully distinguish procedures (qua definite courses of action-types) from what makes them effective and the particular courses of action(-tokens) that instantiate them. With these crucial distinctions in hand, I develop an alternative account of effective procedure. I conclude with a discussion of some consequences of my analysis of effective procedure. If I am right, a significant number of popular views about the nature of minds and the possibilities for physical computation require rethinking.

## 2. The Received View on Effective Procedures

At the core of our most general concept of procedure is the idea of *something to follow*. What one follows are instructions prescribing that certain things be done in a prespecified order in time. In other words, a procedure is a prescription for a course of action. This is just as true of the more refined procedures of mathematics (e.g. the Euclidean algorithm) and computer science (e.g. a JAVA program for sorting words) as it is for quotidian (a.k.a. mundane<sup>1</sup>) procedures such as a recipe for Hollandaise sauce or the instructions for assembling a child's tricycle.

From a pre-analytic, intuitive standpoint, a procedure is effective if correctly following it reliably yields a definite outcome. Correctly applying the Euclidean algorithm to two integers invariably yields their greatest common divisor. Thus, it is said to be effective for computing the greatest common divisor of two integers. Similarly, a recipe for Hollandaise sauce may be characterized as effective in virtue of the fact that it reliably produces Hollandaise sauce when followed by a competent and attentive chef. Some procedures go on forever, however, and yet have a definite predetermined outcome at each and every stage of their implementation. Many mathematical algorithms fall into this category. The Wallis' algorithm for computing  $\pi$  (the constant ratio of the circumference of a circle to its diameter) provides a good example. It is non-terminating but nevertheless may be used to calculate successive and exact values of the decimal expansion of  $\pi$ , thus allowing one to ever more closely approach, without actually reaching, the full decimal expansion of  $\pi$ . In light of cases like this, the concept of effectiveness has come to be defined in the technical literature in terms of the outcomes of applying individual instructions, as opposed to the outcome of completing a procedure as a whole. As Marvin Minsky explains in his classic textbook on computer science, 'our concern here is not with the question of whether a process terminates with a correct answer, or even ever stops' (Minsky, 1976, p. 105).

Minsky doesn't stop, however, with correctly noting that our concept of effectiveness needs to include non-terminating procedures. He further stipulates that the effectiveness of a procedure be construed in terms of 'whether the next step is

always clearly determined in advance' (Minsky, 1976, p. 105) *by the procedure*. In other words, for Minsky, the effectiveness of a procedure depends upon how well its instructions *specify* the actions they prescribe. This amounts to a linguistic proposal for analyzing effectiveness, and, indeed, Minsky subsequently characterizes effective procedures in linguistic terms, explicitly referring to them as 'well defined' or 'precisely described' (Minsky, 1976, p. 106). It is important to keep in mind that Minsky does not view himself as introducing a novel notion of effectiveness. He sees himself as explicating the received view among computer scientists and mathematicians. In the words of J.B. Rosser, an 'effective procedure' is '... a method each step of which is *precisely* [my italics] predetermined...' (Rosser, 1936, p. 225). The basic idea is that the instructions of genuinely effective procedures are so clear that a follower can correctly apply them 'without the exercise of thought or volition' (Gandy, 1988, p. 80). Such procedures may be said to be 'perfectly precise' in the sense that they tell a follower *everything* she needs to know in order to successfully implement them.

Despite its widespread acceptance among mathematicians and computer scientists, Minsky's characterization of effectiveness represents a substantial and potentially problematic departure from the original intuitive concept. In the first place, it isn't at all obvious that applications of instructions providing vague specifications of action couldn't reliably yield definite outcomes. Moreover, as will shortly become clearer, the results with respect to which most procedures may be said to be effective are not themselves specified by their instructions, even supposing that the instructions provide precise specifications of action. The intuitive notion of effectiveness accommodates such procedures. Minsky's notion does not. We shall return to this important issue in Section 4. But first we need to place the received view on effective procedures in its widely accepted historical context.

At the turn of the twentieth century, mathematicians faced a growing epistemic crisis. In contrast to other intellectual disciplines, mathematics had traditionally been viewed as a repository of certain knowledge. The certainty of mathematics was reputedly grounded in mathematical practice, namely, the requirement that a mathematical conjecture be proven before it is accepted and the claim that, unlike other disciplines, mathematical proof is epistemically conclusive, based solely on indubitable axioms and the *a-priori* knowable laws of logic. Yet it was widely acknowledged that some of the greatest successes of mathematics (e.g. the calculus) incorporated mysterious notions such as limit, infinitesimal, and a hierarchy of actual infinities, and that the axioms and theorems in which these concepts figured were not self-evidently true. Indeed, some of them inhabited the shadowy borderland between intelligibility and unintelligibility; attempts to collect Georg Cantor's transfinite cardinal and ordinal numbers into single sets generated logical paradoxes, e.g., the cardinal number of the set of all cardinal numbers must be larger than any cardinal number.

Concerned mathematicians sought a remedy. As is well known, Gottlob Frege attempted to 'reduce' arithmetic (the *prima facie* secure foundation of all of math-

ematics) via the ostensibly more transparent and simpler concepts of logic and set theory; the integers were to be identified with sets, arithmetic operations (e.g., addition) with complexes of set operations such as union and intersection, and the rest of mathematics (the rationals, reals, and complex numbers) constructed from the integers by means of set theoretic operations. Bertrand Russell's set-theoretic paradox put an end to Frege's dream, demonstrating that the concept of a set was not nearly as innocuous as everyone had thought. Some mathematicians attempted to escape from the paradox by carefully placed patches (Zermelo–Franel set theory) or, alternatively, more general but nonetheless intuitively unmotivated means (Russell's theory of types). Concerned about the ad hoc nature of these efforts, David Hilbert proposed a different approach, an approach emphasizing the role played by logical proof in mathematical practice over the reconstruction of the problematic content of mathematics.

Hilbert conceptualized mathematics as a purely formal game played by manipulating finite numbers of symbols according to their external shapes via the application of finite numbers of well-defined rules; the meaning of the symbols had no affect on what went on in the game. Thus, on Hilbert's view, mathematics is not about numbers; it is completely devoid of content, containing only uninterpreted symbolic elements. Hilbert stipulated that formal mathematical systems meet three minimal conditions: Consistency, completeness, and decidability. Hilbert's program was effectively destroyed by Kurt Gödel's first incompleteness theorem, which demonstrated that a formal system rich enough to encapsulate elementary arithmetic could not be both consistent and complete. This left decidability – the question of whether there exists some definite finitary procedure that could in principle be applied to decide the truth of any claim in formal mathematics. The problem of finding such a method amounted to a search for a decision procedure for first order logic. It became known as Hilbert's '*Entscheidungsproblem*'.

It is in the context of the search for a definitive solution to the *Entscheidungsproblem* that we finally arrive at the received view on effective procedures. A solution to the *Entscheidungsproblem* required nothing less than the discovery of a uniform, formal decision procedure for all of mathematics; a collection of specialized decision procedures tailored to specific mathematical problem-types wouldn't count. Gödel's work strongly suggested a negative answer to the *Entscheidungsproblem*. The central difficulty in proving that the *Entscheidungsproblem* is unsolvable lay in providing an intuitively satisfying account of a definite method for solving the problems of formal mathematics. It was generally agreed that such a method must be constructive and finite; this rules out the powerful existence proofs of mathematics. By the mid-thirties, there were three proposals on the table, Alonzo Church's lambda-definability, Herbrand–Gödel general recursiveness, and Turing computability. All three methods were purely formal, specifying different finitary ways of combining and manipulating uninterpreted symbols. Alan Turing demonstrated that the three proposals were extensionally equivalent (*vis-à-vis* the computation of the number theoretic functions), which strongly suggested that

mathematicians had finally captured the decidable (a.k.a. computable) number theoretic functions. This conviction is encapsulated in the famous Church–Turing thesis.<sup>2</sup>

Although they compute the same functions, the three proposals nonetheless represent different methods. This is reflected in their differing utility. General recursiveness (particularly, Kleene’s refined version) is superior for the purpose of developing the abstract theory of computation in a system of formal logic. Turing computability, on the other hand, seems much closer to the intuitive idea of carrying out a computation. Indeed, Turing introduced it in the context of an idealized person doing calculations with pencil and paper. Moreover, he explicitly characterized these idealized humans as ‘machines’ and described their activity as ‘purely mechanical’. In Turing’s words, ‘A function is said to be ‘effectively calculable’ if its values can be found by some purely mechanical process. We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine’ (Turing, 1965a, p.160). To the extent that mechanical processes are commonly regarded as paradigms of constructive, finitary processes, it is hardly surprising that the received view on effective procedures is almost invariably couched in terms of Turing machines. As we shall see in the next section, however, the concept of a purely formal, mechanical process is highly problematic.

According to the received view, Turing machines provide us with paragons of Minsky’s ‘precisely described’ or ‘well defined’ instructions. Indeed, mathematical algorithms and computer programs are judged effective just insofar as they are thought to have Turing machine translations. Quotidian procedures, in contrast, are dismissed as non-effective on the ground that their instructions do not provide the requisitely ‘precise’ specifications of action; following them requires thought or expertise. The question is do Turing machines really provide us with ‘precise’ specifications of action in Minsky’s sense, i.e., do their instructions really unequivocally determine in advance the next step? To properly answer this question we need to look closely at the official line on Turing machines.

### 3. Problems with the Received View on Effective Procedures

Turing machines are sometimes characterized formally as mathematical structures and sometimes characterized informally as abstract mechanisms. Since their informal characterization provides the motivation for the received view on effective procedures, we will begin with it. Our discussion will be restricted to the simplest Turing machines, namely, the deterministic, sequential machines of introductory computer science and logic texts, since everything I have to say about these ‘machines’ applies straightforwardly to the more complex nondeterministic and multidimensional Turing machines.

Characterized informally, a Turing machine is said to consist of a ‘mechanism’, known as a ‘finite state machine’, coupled to an external storage medium, called the ‘tape’. The tape is divided into squares, and may be indefinitely extended in either

direction. In the standard set-up, each square is occupied by one of two distinct symbols,  $S_0$  and  $S_1$ . The finite state machine is linked to the tape through a ‘head’ which is described as ‘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’,  $q_1, \dots, q_m$ . A Turing machine is said to be in a specific state  $q_i$  only if it is about to carry out a specific instruction  $i$ .

Turing machine instructions are conditional in form. Depending upon which symbol is being scanned, they prescribe that certain, extremely simple, things be done by the head (print a symbol, erase a symbol, move left, move right, halt). As an example, a Turing machine instruction might read as follows: ‘If the symbol in the square being scanned is an  $S_1$ , move to the left one square; otherwise [the symbol is an  $S_0$ ] erase the symbol in the square and print an  $S_1$ ’. A Turing machine procedure consists of a set of instructions, called a ‘program’. The basic idea when computing a numerical function is to ‘code’ numbers in strings of  $S_0$ s and  $S_1$ s, perform a sequence of simple ‘mechanical’ operations (specified by the instructions) on the string, and then decode the resulting string of  $S_0$ s and  $S_1$ s as numbers.

The use of common English imperatives such as ‘write’ and ‘move’ to characterize Turing machine actions suggests that they are similar to ordinary, everyday human actions. Unfortunately, as we shall see below, this impression is grounded in a misleading metaphorical attribution of action to Turing machines.<sup>3</sup>

Action presupposes something to manipulate. Turing machine symbols, which are the entities reputedly manipulated (‘written’ and ‘erased’) by Turing machines, are traditionally characterized as ‘purely formal’. They are said to be purely formal in the sense that the ways in which they are manipulated depend only upon their ‘shape’. The shapes of Turing machine symbols are thus crucial to the identity of the action-types prescribed by Turing machine instructions. The question is, can we make good sense of the claim that Turing machine symbols have shape?

It might be thought that one could determine the shape of a Turing machine symbol by generalizing across different physical realizations of a Turing machine. The idea is to identify the shape of a symbol with whatever geometrical feature is had in common by all of its tokens. But this won’t work. Across different physical realizations of the same Turing machine, the same symbol may be instantiated by objects having incommensurable shapes, e.g., pencil squiggles, punched squares, pebbles in tin cans. Moreover, we needn’t use geometrical features to encode Turing machine symbols. We could use weights or colors, or even properties of events (e.g. the duration of a flash of light). The point is any definite but distinct physical properties may be used to encode and distinguish the symbols of a Turing machine. Indeed, the only physical constraints on tokens of Turing machine symbols hold within (vs. across) physical realizations of Turing machines, namely, all tokens of the same symbol must have in common some (it doesn’t matter what) physical property and all tokens of different symbols must differ in some physical property. Thus, although different code-types for Turing machine symbols (which are specific to particular realizations of Turing machines) may be distinguished in terms

of the physical properties of their tokens, different Turing machine symbols (qua constituents of a Turing machine considered independently of its instantiations) can not be so distinguished. For the most that may be said about Turing machine symbols per se is that tokens of different symbols have different but not any definite physical properties. At best, this gives us a relation of bare physical difference among symbols. Unfortunately, a relation of bare physical difference isn't enough to provide us with the concept of the shape (however broadly construed) of a Turing machine symbol since it is an external relation among symbols and external relations aren't capable of individuating their relata.<sup>4</sup>

In the context of the formal mathematical analysis of Turing machines, even this minimalist conception of the shape of a Turing machine symbol must be relinquished. Mathematicians analyze Turing machines in terms of mathematical structures. Mathematical structures consist of functions and relations (both of which are identified with sets of n-tuples, ordered in the case of the former) and constants. The prototypical ('usual') mathematical structure for a Turing machine includes three binary functions (the 'next place' function, the 'next symbol' function, and the 'next state' function), and two symbols, which are commonly characterized as 0 and 1.<sup>5</sup> It is important to appreciate that the latter can't be viewed as numerals (which have shape) since this would amount to conflating a specific abstract mathematical structure (viz., the usual structure for the Turing machine) with a concrete representation. So it seems that they must be bona fide integers. But integers (qua abstract mathematical objects) have no physical features. This suggests that the only thing that can serve to distinguish different Turing machine symbol-types is bare numerical difference, which completely precludes the possibility of making sense of the claim that they have unique distinguishing structures, however minimal. Finally, mathematicians do not identify Turing machine types with their prototypical mathematical structures. They identify them only up to isomorphism. Considered independently of a particular realization, a Turing machine is merely a class of isomorphic structures (both abstract and concrete). Viewed from this perspective, Turing machine symbols amount to nothing more than logical roles in a second order structure, which, considered in itself, is no more a Turing machine than the set of all red objects is a red object. Accordingly, Turing machine symbols are best thought of as conveniences of discourse, as opposed to symbols of some intangible, rarefied sort. Until we get to the level of the individual structures in the equivalence class defining a Turing machine we don't have bona fide symbols. We have placeholders (variables) for symbols. As a consequence, we can't be said to have specifications of action, however imprecise, at the level of a Turing machine considered independently of any of its instantiations.

Even supposing that Turing machine 'symbols' were genuine symbols, however, the instructions of Turing machine programs still couldn't be said to provide us with precise specifications of action. Although action presupposes things to manipulate, having something to manipulate isn't enough to constitute action. It wouldn't suffice to specify a knife and a carrot in a recipe. One also has to specify

what is to be done to the carrot by the knife, and there are a large number of possibilities, including dicing it, slicing it, shredding it, and shaving it. Despite the use of familiar English expressions for action, Turing machine instructions do not specify what is to be done once their symbol-variables are replaced by genuine symbols. What counts as ‘erasing’ and ‘writing’ a symbol is left completely open. If pebbles replaced the variables, ‘erasing’ a pebble could be realized by activities as diverse as pulverizing it, painting it, or removing it from a tin can.

Furthermore, although the claim that Turing machine operations are ‘purely mechanical’ is one of the great pillars of the received view, the actions performed by instantiations of Turing machines need not be ‘mechanical’ in the physical scientist’s sense. Forces that act over a distance (without any intervening causal chain) could satisfy the instructions of a Turing machine program just as well as the more familiar Newtonian ‘pushes’ and ‘pulls’.<sup>6</sup> Indeed, even magical actions could satisfy them. An accommodating deity could instantiate a Turing machine by creating and annihilating pebbles. The point is, the only constraints on Turing machine actions are purely structural: Turing machine actions must be (i) distinct from one another and (ii) occur in a time-ordered sequence. Insofar as these are external relations among actions, they are purely formal features of action. Whether or not an action is mechanical (in the physical scientist’s sense) depends upon its intrinsic character, and the intrinsic character of the actions prescribed by Turing machine instructions just doesn’t matter. The much-lauded mechanical character of Turing machine operations thus turns out to be a myth.

It is possible, however, that many proponents of Turing’s account have something else in mind when they use the word ‘mechanical’. Turing sometimes used the word ‘automatic’ in place of ‘mechanical’ (e.g., Turing, 1965b, p. 118). Similarly, as noted earlier, Gandy spoke of Turing machines performing actions without ‘thought or volition’. But while this is a perfectly legitimate use of the word ‘mechanical’, it is not a use that captures the idea of a finite, constructive process. Although a mechanism (in the technical scientific sense) may be said to perform its operations ‘automatically’, it doesn’t follow that something that performs its actions automatically (without thought or volition) is a mechanism. Indeed, one can coherently speak of God ‘automatically’ completing an infinite task or performing single actions that individually violate physical law. In other words, one shouldn’t be fooled by the suggestive terminology into thinking that Turing machines are mechanisms in anything like the sense in which physical machines are said to be ‘mechanisms’,<sup>7</sup> and this has nothing to do with their idealized or abstract character. It has to do with the fact that processes that do not meet the physical scientist’s standards for ‘mechanical’ can nonetheless realize them. I suspect that much of the popularity of the received view – the idea that with the concept of a Turing machine ‘. . . one has for the first time succeeded in giving an absolute definition of an interesting epistemological concept’ (Gödel, 1965, p. 84) – rests upon confusing these different meanings of ‘mechanical’.

As earlier, the formal mathematical account only exacerbates the difficulty. There is no requirement that the structures in the equivalence class defining a Turing machine be dynamic, let alone mechanical.<sup>8</sup> The basic Turing machine operations ('erase', 'write', 'move') are defined in terms of ordered n-tuples (mathematical functions), which do not require change (let alone change that qualifies as action) for their realization; they may be instantiated by spatially as well as temporally ordered structures. As a mathematician once cheerfully conceded to me, a Turing machine could be realized by a quartz crystal! In short, on the formal mathematical account, the operations specified by Turing machine instructions cannot be said to represent dynamic activity of any sort, mechanical or supernatural. It follows that even supposing that their 'symbols' qualified as genuine symbols, Turing machine instructions still could not be said to prescribe bona fide actions.

The conclusion seems inescapable: Considered independently of a particular instantiation, Turing machine instructions cannot be said to prescribe actions, let alone *precisely* describe them. If they do not prescribe actions, then Turing machines cannot be said to supply us with procedures either; for at the core of the concept of procedure is the idea of something to follow, and what one follows are instructions prescribing that certain things be *done* in a predetermined order in time. Finally, if they do not supply us with procedures, then they certainly can't be said to supply us with *effective* procedures. At best, Turing machines may be characterized as providing procedure schemas, i.e., temporally ordered frameworks for procedure. When these schemas are fleshed out with specifications of genuine action, we get bona fide procedures.

#### 4. The Crucial Role of Causation

In light of the failure of Turing machines to live up to their promise of providing precise specifications of action, let us return to the pre-analytic, intuitive concept, namely, the idea of a procedure that *reliably* produces a certain kind of outcome when followed. As mentioned earlier, the intuitive notion applies to quotidian procedures (from recipes to numerical algorithms) since many of them reliably produce definite outcomes when followed. It is also at least implicitly present in the minds of many who subscribe to the official Turing account. In the words of mathematician David Berlinski, 'an effective procedure [is] a way of getting *something done* [my italics] in a finite number of discrete steps' (Berlinski, 2000, p. xvi). The intuitive notion may be extended to non-terminating procedures by focusing on the *outcome* of applying an individual instruction (as opposed to the *precision* with which the instruction specifies an action). In this context, it is important to keep in mind that a procedure is reliable for certain outcomes but not for others. The Wallis' algorithm for computing  $\pi$  is reliable for producing successive values of the decimal expansion of  $\pi$  but not for computing the base of the natural logarithms. Similarly, a recipe for Hollandaise sauce is reliable for producing a creamy sauce but not for producing a loaf of bread, let alone computing the number  $\pi$ . In other

words, a procedure that is reliable for one outcome will not be reliable for most other outcomes. The question thus becomes, what is the relationship between the result(s) with respect to which a procedure is (intuitively speaking) effective and the specifications provided by its instructions?

Unlike Turing machines and their programs, quotidian procedures are bona fide procedures; their instructions prescribe genuine actions. Admittedly, they do not provide 'precise' specifications of action in Minsky's sense.<sup>9</sup> But such precision is a will-o'-the-wisp. No instructions, not even Turing machine instructions, can do this. Language is irremediably vague. The lauded precision of Turing machine instructions is secured by fiat. Turing machines are defined as machines that do exactly what their instructions tell them to do. It is thus not logically possible for a Turing machine to execute an instruction to 'print' an  $S_0$  and yet fail to 'place' an  $S_0$  on the tape. In the case of genuine procedures and their concrete followers, however, failure is always a very real possibility; no instruction, considered just in itself, can rule out the possibility of misunderstanding or misapplication. But it doesn't follow from this that quotidian procedures aren't reliable, and hence aren't effective. What is important to the effectiveness of quotidian procedures is not the precision of their specifications but, rather, the fact that people can be trained to accurately apply imprecise instructions.<sup>10</sup>

In light of the inherent imprecision of their specifications of action, one might suspect that the effectiveness of quotidian procedures depends only upon the identity and time-order of the actions they prescribe. Surprisingly, this is not the case. Consider, for example, Julia Child's 'traditional recipe' for Hollandaise sauce. At one stage, the recipe prescribes pouring melted butter by droplets into warm egg yolks while continuously beating them with a wire whisk. What makes the recipe effective for Hollandaise sauce is not the mere performance of these actions. It is what is *caused* by their performance, namely, a physical process in which egg yolks absorb butter and hold it in creamy suspension.<sup>11</sup> If the actions had a different effect – caused egg yolks to become granular and float in butter, for instance – the recipe would fail to qualify as an effective procedure for making Hollandaise sauce. In other words, the effectiveness of the recipe depends upon the causal consequences of performing actions of the designated types. If performing these action-types did not initiate, modulate, and sustain the right sorts of physical processes, following the recipe would not produce a Hollandaise sauce.

The causal consequences of performing the actions prescribed by a quotidian procedure are not, however, specified by the procedure. This is best appreciated by imagining a possible world,  $W_1$ , which differs from the actual world,  $W_a$ , in that whisking droplets of butter into warm egg yolks causes them to turn into a smooth, elastic purple mass;<sup>12</sup> clearly, the laws of chemistry in  $W_1$  differ from those in the actual world. Now suppose that someone in  $W_1$  correctly follows the instructions for Child's recipe. The result is nothing like a Hollandaise sauce; indeed, mothers in  $W_1$  use the recipe to produce the functional equivalent of our playdough for their children. This difference in outcome does not correspond to a difference in recipe.

In both  $W_1$  and  $W_a$ , people follow the same instructions and perform the same types of action in the same order in time. But they get different results because performing the same type of action causes very different things to happen in  $W_a$  and  $W_1$ . In other words, the very same recipe may be effective in one world and non-effective in another world for a given outcome. The *effectiveness* of the recipe for a given outcome depends upon the causal character of the world in which it is followed. In contrast, the *identity* of the recipe *qua* procedure (i.e., as something to be followed, or a course of action) does not; it depends only upon the identity and time-order of the action-types prescribed.<sup>13</sup>

It is instructive to compare the relationship between the actions prescribed by quotidian procedures and the outcomes with respect to which they are said to be effective with the relationship between the metaphorical actions prescribed by Turing machine programs and the outcomes with respect to which they are said to be effective. The ‘actions’ prescribed by Turing machine instructions have no consequences beyond their mere ‘performance’. In following an instruction to print an  $S_0$  a Turing machine ‘places’ an  $S_0$  in the designated square of the tape. But the ‘appearance’ of the  $S_0$  in the square does not have any consequences. If the  $S_0$  is *interpreted* as the integer 1, then what is on the tape has arithmetical consequences. But one is no longer dealing with a Turing machine simpliciter. One is dealing with a particular interpretation of a Turing machine. Assigning different integers to  $S_0$  yields different interpretations, with correspondingly different arithmetical consequences. If one *encodes* the  $S_0$  as a numeral or a pebble (as opposed to interpreting it as an integer) one is dealing with a concrete realization of a Turing machine, which has causal consequences (and may, of course, also be given an arithmetical interpretation). Considered just in itself (as an uninterpreted, abstract entity) a Turing machine provides us with no distinction between procedure and process; the following of the program is the only process involved. It is for this reason that, given its program and initial configuration, *everything* a Turing machine ‘does’ is completely determined in advance, and hence every Turing machine program seems to qualify as an effective procedure.

In contrast, even supposing that initial conditions are supplied, a quotidian procedure can’t be said to determine in advance the consequences with respect to which it is effective. What happens when one follows a quotidian procedure depends upon the causal character of the world in which it is implemented. The causal character of a world is determined by its natural laws, which may be very complex and probabilistic and/or deterministic. But although quotidian procedures depend upon natural laws for their effectiveness, they do not (in any sense) designate them; all they specify is time-ordered sequences of action (types). In other words, unlike the case with Turing machines, the outcome with respect to which a quotidian procedure may be said to be ‘effective’ is not determined in advance by the procedure.

The dependence of the effectiveness of quotidian procedures on the causal character of the world in which they are implemented is the source of their power.

The actions specified by quotidian procedures represent adroit causal interventions in the course of nature. These interventions give rise to things that nature, left to her own devices, would rarely or never produce. A good example is fire. The most common natural source of fire is lightning, and early humans exploited it by collecting smoldering embers from struck trees. The invention of procedures for starting fires freed our ancestors from their dependency upon electrical storms, allowing them to start fires when and where they were needed. Most of the comforts of contemporary civilization (bread, cloth, Tylenol, automobiles, and microwave ovens, to name just a few) would never be manufactured by the unaided operation of natural processes. They owe their existence to the design of specialized procedures for forcing nature along improbable causal pathways. In this context, the term 'effective' seems a highly appropriate appellation for reliable procedures since it makes implicit reference to causality (the concept of effect).

The limits to what can be produced by means of quotidian procedures are set by the physical possibilities for intervening in natural processes, and these are not the same from one possible world to another. Interventions in a world are possible only insofar as it has exploitable causal openings. A causal opening is a physical factor that may be modified independently of other physical factors with reliable causal consequences.<sup>14</sup> Quotidian procedures represent ways of combining and rearranging natural processes by exploiting localized causal openings (nature's ready-made switches); the basic idea is to strategically and surgically intervene in the normal course of nature. A quotidian procedure may thus be viewed as a temporally organized arrangement of causal interventions. These manipulations redirect and reshape natural processes, producing the uncommon outcomes with respect to which they may be characterized as 'effective'. But to recapitulate, the procedure does not itself specify what happens as a result of these interventions. If the causal character of the world is inappropriate, correctly following the procedure will not yield the targeted outcome.

The gap between what makes quotidian procedures effective and what is explicitly specified by their instructions is the source of their great utility. Following an effective quotidian procedure doesn't require knowledge of the physical processes responsible for producing the targeted outcome. A cook needn't know chemistry in order to make a great Hollandaise sauce. All she needs to know is how to follow the recipe, i.e., how to perform the prescribed actions in the designated order in time; nature takes care of the rest for her. Similarly, a master chef doesn't need to know chemical theory in order to invent a new recipe. In other words, quotidian procedures insulate us from the messy causal details of the world while at the same time allowing us to combine and rearrange those details in very specific and highly efficacious ways. We can shape the course of nature despite knowing very little about her inner workings.

In light of the above discussion, how should we understand Turing machines? If quotidian procedures can be understood as time-ordered arrangements of actions *qua* causal interventions in the course of nature, Turing machines and their pro-

grams may be understood as mere logical possibilities for such interventions. This interpretation of Turing machines is consistent with the idea that they are procedure schemas. It also explains why their reputed effectiveness does not vary from one possible world to another, namely, because every possible world contains *at least* the bare logical possibility of effective procedures.

Considered just in themselves, however, logical possibilities for procedure cannot be said to be effective. Effectiveness requires authentic procedures yielding definite outcomes when followed. Whether a logical possibility for procedure is physically realizable depends upon the causal structure of an individual world. Worlds lacking localized causal openings do not permit interventions and hence won't physically support the existence of procedures. The past provides us with a good model for a world that would not support procedures. We have no procedures for modifying the past because our present actions cannot affect the past; the past remains inexorably the same no matter what we do. The causal openings of the actual world are all future directed. Moreover, even supposing that a world does permit interventions, they may not be reliable. A world lacking even pseudo-deterministic physical processes might support procedures but they wouldn't be effective. In short, effective procedures are physically possible only in worlds in which (at least some) physical processes have a special kind of causal structure.<sup>15</sup> But it isn't enough for a physical process to have the right causal structure. In order to be exploited by an effective procedure, the causal openings provided by the process must causally intermesh in just the right way with the actions prescribed by the procedure. As an example, whisking droplets of water into warm vinegar will not produce a creamy sauce, but whisking droplets of melted butter into warm egg yolks will. Creatures with a different repertoire of basic actions might be able to exploit causal openings that are inaccessible to us and hence achieve things that are beyond human capabilities. And this brings us to my central point. Turing machine theory treats effectiveness as a purely formal property of procedures. But effectiveness is not a formal property of quotidian procedures such as recipes; it crucially depends upon a complex causal relation standing between the content of the procedure (the intrinsic character of the actions it prescribes) and the physical character of the world in which the procedure is implemented.

The case of numerical algorithms is less clear because the ontological status of mathematics is more controversial. Nevertheless, it is clear that traditional numerical algorithms do not represent empty logical possibilities for procedure. To more fully appreciate this point, consider the following traditional version of the Euclidean algorithm:

Let  $a$  and  $b$  be specific positive integers. Calculate the remainder of  $b$  with respect to  $a$ . Call this remainder  $r$ . If  $r = 0$ ,  $a$  is the g.c.d. of  $a$  and  $b$ . If  $r \neq 0$ , replace  $b$  by  $a$  and  $a$  by  $r$ , and repeat the first step.

One cannot follow these instructions without a good understanding of what an integer is, how to perform long division, etc. Moreover, this algorithm is not effective for computing just any old function; it is effective for computing a specific

arithmetic function, namely, the greatest common divisor of two positive integers. To insist, in the face of these crucial references to arithmetic content, that the Euclidean algorithm is nothing more than an uninterpreted formal schema (which, strictly speaking, is all that a Turing machine procedure amounts to) is simply to beg the question.

Admittedly, Hilbert's program maintains that mathematics is just a formal game played with meaningless symbols. But as discussed earlier, Hilbert's program was demolished by Kurt Gödel's first incompleteness theorem, which demonstrated that a formal system rich enough to encapsulate elementary arithmetic could not be both consistent and complete. Ironically, the received view on effective calculability is a remnant of Hilbert's program. It grew out of efforts to solve his *Entscheidungsproblem*. In establishing that there exist number-theoretic functions that can't be computed by any Turing machine, Turing proved that the *Entscheidungsproblem* is unsolvable. But Turing didn't reject the idea that effective calculability is a purely formal concept. If I am right about effective procedures, he should have. Effectiveness is not just a matter of logical form. To turn a Turing machine into a *bona fide* numerical algorithm, one must give it arithmetic content – make assignments to the placeholders for action provided by the schema. Depending upon one's views about the ultimate nature of mathematics, this may involve assigning numerals (physical inscriptions) and physical operations to the symbol and operation variables, or, alternatively, assigning abstract entities and operations ('real' numbers and arithmetic operations) to the symbol and operation variables. In either case, however, we end up with an *interpretation* of a Turing machine, and an interpretation of a Turing machine is more than a Turing machine.

Alternatively, one might reject the claim that Turing machines exhaust the logical possibilities for procedure. Gödel seems to have had something like this in mind. He believed that humans might be capable of using non-finitary reasoning to perform non-mechanical calculations and (by such means) mentally compute Turing uncomputable functions; see Wang (1974). If Gödel is correct, the possibilities for mental procedures outstrip the logical possibilities provided by traditional Turing machines. But this doesn't affect my central point. The effectiveness of a conventional numerical algorithm or a Gödelian super-computational mental procedure is just as dependent upon its mathematical content (whatever its ultimate nature) as the effectiveness of a recipe is dependent upon its empirical content. Considered just in itself, an uninterpreted logical structure can no more be said to be effective for computing the greatest common divisor of two integers than it can be said to be effective for making a Hollandaise sauce. In short, whether one is dealing with recipes, conventional numerical algorithms, or super-computational mental procedures, the content of the procedure is crucial to its effectiveness. In this context, it is important to keep in mind that, regardless of the nature of mathematics and the human mind, the possibilities for using physical artifacts for computing numerical functions ultimately depend upon the nature of the causal processes at our disposal. To this issue we now turn.

## 5. Applications to Minds and Machines

The implementation of a program on a concrete computer like my Macintosh PowerBook closely resembles a human following a quotidian procedure. Like chefs, programmers and users can specify in the context of their interests and purposes what is to be done without having to concern themselves with the messy physical details required to do it. A JAVA programmer need know nothing about how a computer works in order to program it to do something new, just as a chef need not know chemistry in order to invent a new recipe. The programming language isolates the programmer and user from the messy details involved in the physical operation of the computer. Indeed, different types of computer (having different physical components and architectures) can execute the same JAVA program and the same computer executing the same JAVA program at different times will (depending upon what else it happens to be doing) activate different circuits and chips.

It is important to keep in mind just how different this is from what a Turing machine (metaphorically) does when it ‘follows’ a program. A Turing machine program logically predetermines everything the machine ‘does’. Nothing is left open. As a consequence there isn’t a distinction between *what* the machine does and *how* it does it. In contrast, no computer program (considered just in itself) provides a complete specification of the behavior of the computer implementing it. Even the lowest level programs (machine language programs) depend upon a complicated encoding scheme called the ‘machine code format’ to fix the physical states the machine passes through. Moreover, the relation between the high level program and what goes on in the machine executing it is ultimately causal. Physical representations of the instructions in the machine language translation of the high level program cause changes in the electronic processes going on in the machine, and these changes are what make the program effective, whether for computing the greatest common divisor of two integers, sorting words, or assembling auto parts. In other words, we use the high level program to manipulate electronic processes in the machine just as we use a recipe to manipulate chemical processes in the kitchen, and like the recipe, it is the latter that ultimately secure the effectiveness of the program. The main difference between the recipe and the program is that we don’t perform the actions prescribed by the program; the computer does this for us. In other words, the utility and power of modern digital computers resides in our having succeeded in automating the following of a quotidian procedure, as opposed to our having succeeded in instantiating a Turing machine.

If I am right about the significant relation between concrete computers and their programs, some popular views about the nature of minds, machines, and computation need rethinking. What computers executing programs mimic is not the covert activity of human thought or brain processes but rather the overt behavior of humans following procedures. The overt behavior of a human following a procedure is quite different from the neurological processes underlying it, just as the overt

behavior of a concrete computer following a program is quite different from the continuous electronic activity occurring in its hardware. The former is procedural, the latter is a physical process, and as I have urged procedures are not physical processes. Procedures prescribe time-courses of action (qua localized causal interventions). The causal structure of a time-course of action is very different from an instance of a physical process. The subparts of an on-going physical process are directly and continuously connected in time whereas the actions that constitute a following of a procedure are distinct and independent, being related only indirectly as effects of a common cause (viz., the follower). These differences are objective to the extent that they can be captured in probabilistic and counterfactual relations of dependency and independence.<sup>16</sup>

Insofar as the causal structure of instantiations of procedure differs from that of instantiations of physical processes, there is no more reason to suppose that a computer simulation of Einstein's brain could think than there is to suppose that a computer simulation of photosynthesis could produce sugar.<sup>17</sup> Put another way, the difference between a simulation and the proverbial real McCoy is just the difference between a procedure and a physical process. Conversely, it makes little sense to speak of Searle's wall or Hinck's pail implementing every Turing machine.<sup>18</sup> For just as no procedure, considered in isolation from the physical processes it affects, constitutes a physical process, so no arbitrary assignment of symbols and operations to physical entities and their interrelations constitutes a procedure. To recapitulate, procedures are strategies for causally intervening in the course of nature to attain certain ends. The causal details are crucial to the outcome for which a procedure is said to be effective. Following a procedure will get the right results only if the causal circumstances are right. The same procedure will get different results in different possible worlds and also in different circumstances in the same world, as any chef who has cooked at low and high altitude will readily attest. This dependency of outcome on causal circumstances is obscured by the Turing machine model which, failing to distinguish procedures from processes, analyzes the mind (and just about everything else!) as wholly procedural. In short, on my account, the celebrated computer metaphor for the mind is fundamentally misguided.

This brings us to the computational capacities of concrete computers. The distinction I have drawn between procedure and process suggests that it is a mistake for theoretical computer scientists to focus on Turing machines as paragons for the design of concrete computers. For there is no more reason to require that the physical processes generating the input/output functions specified by programs have the logical structure of Turing machines than there is to suppose that the neurological or psychological processes of a chef following a recipe resemble her overt bodily behavior in the kitchen. Instead the focus should be on physics, more specifically, on identifying and exploiting localized causal openings in exotic physical processes for the purpose of reliably obtaining results that are currently unobtainable or could be obtained in a more rapid or efficient manner.

In his Ph.D. dissertation, Alan Turing (1965a, pp. 166–167) introduced the concept of an oracle to try to pin down what it would take for a physical machine to exceed the computational capacities of standard Turing machines. Oracles are logical black boxes for carrying out uncomputable tasks. They accept digital input and produce digital output, and hence can be integrated into standard Turing machines, allowing them to compute functions they otherwise couldn't compute; the composite machine is known as an 'o-machine'. The traditional worry about oracles has to do with the mystery of their inner workings; all that is said about them is that they aren't Turing machines. On my account, however, this worry disappears since the whole point of concrete computers is to provide us with the ability to specify tasks independently of the physical details of carrying them out. Put another way, it doesn't matter *how* an oracle does what it does; all that matters is *what* it does. Moreover, from the point of view of contemporary physical theory, there is little reason to suppose that every physical process is completely mechanistic in the traditional Newtonian sense of permitting localized causal interventions (throwing the proverbial wrench in the mechanism) at any time during its evolution. Some physical processes may have only a few points at which causal intervention is physically possible, thus allowing one to specify an input and output relation but not permitting a mechanistic account of what goes on between input and output. An oracle would be such a process. It is important to keep in mind, however, that being non-mechanical is not sufficient for being an oracle.

To better grasp how one might exploit an oracle to compute something Turing uncomputable consider the following procedure: 'Walk half way across the room and, then, walk the remaining distance'. This procedure specifies an effective method for getting from one place to another 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 distinct and independent actions, which no one can perform. Zeno concludes that motion is impossible. But he should have concluded only that his procedure was non-effective for traversing distances. For there is no reason to suppose that the physical process of moving must actually be constituted by a linear sequence of distinct and independent transitions in place just because our procedures are constituted by linear sequences of such actions. It is possible that the *prima facie* infinite divisibility of motion is, as Aristotle maintained (McKeon, 1970, pp. 335–340), merely potential – that motion is not constituted by an actual infinity of separate submotions. In contrast, following a procedure requires that each of the prescribed action-types be separately actualized in the specified order in time.

Viewed from this perspective, attempts (see Copeland's contribution to this volume) to flesh out Turing's mysterious oracles in terms of accelerating machines, which actually perform an infinite number of actions in a finite span of time, are

misguided. Procedures amount to nothing more than methods for exploiting the causal contingencies of the world. Considered in themselves, independently of the physical processes they exploit, they do not reveal subtle truths about the computational powers of concrete machines, or, for that matter, the physical nature of motion or mind. They reveal only the abstract structure of possibilities for localized intervention in the flow of nature. A traditional Turing machine cannot perform an uncountably infinite number of actions and neither can we. But this doesn't mean that it is impossible to physically compute Turing uncomputable functions (such as the halting function) any more than Zeno's paradox means that it is impossible to cross a room. Similarly, just as it doesn't follow from Zeno's paradox that motion requires the completion of an infinite sequence of distinct acts of moving in a finite amount of time, so it doesn't follow that the computation of Turing uncomputable functions requires super machines that actually perform an infinite number of distinct actions in a finite amount of time. In both cases, the source of the problem is the same—failure to distinguish procedures (courses of action) from the physical processes that make them effective.

In closing, I want to comment briefly on the oft expressed concern that oracles are of only theoretical interest because one could never *confirm* that a proposed physical candidate produced something genuinely Turing uncomputable. As I have argued elsewhere (Cleland, 1993, pp. 17–20), there is no more reason to insist that a physical device utilizing an oracle couldn't compute a Turing uncomputable function on the grounds that it is impossible to conclusively verify it than there is to insist that my hand calculator can't compute division on the grounds that it is impossible to conclusively verify it. Indeed, strictly speaking, my hand calculator doesn't compute division since it will eventually break down and thus only compute a partial function, which is consistent with its computing any one of an uncountably infinite number of total functions, including division. And this underscores a frequently overlooked point. Whether a machine or a human being determines the correct values of a definite numerical function when following a procedure expressly designed for this purpose is secured neither by the procedure itself nor by what the human or machine does when following it. It is an empirical hypothesis. At best, there may be good reasons for believing it to be true, reasons based on hypothetical physical considerations (probabilistic relations, counterfactual suppositions, etc.) and well entrenched, theoretical mathematical considerations (identity relations among functions, for instance). In other words, just as there may be good physical and mathematical reasons for thinking that my calculator computes division so there might be good physical and mathematical reasons for thinking that some non-mechanistic physical process could be harnessed (as an oracle) for computing a Turing uncomputable function. It is on the search for such physical processes – as opposed to purely formal extensions of Turing machines – that the development of hypercomputing machines ultimately rests.

## Notes

<sup>1</sup> In my 1993 and 1995 papers, I used the expression ‘mundane procedure’ but I have since come to prefer ‘quotidian procedure’ because it sounds less mundane!

<sup>2</sup> As I discuss in my earlier papers (1993, pp. 283–287, and 1995, pp. 9–10), there is significant disagreement about the proper domain of application of the Church-Turing thesis, some people claiming that it applies only to the number-theoretic functions and others going so far as to claim that it applies to mental or physical processes. In this paper, however, I am restricting it to the number-theoretic functions, which, historically speaking, were the primary concern of Church, Gödel, and Turing.

<sup>3</sup> The core of the following argument first appeared in ‘Recipes, Algorithms, and Programs’ (Cleland, 2001, pp. 228–230). In this article I also discuss in detail the ‘precision’ of the instructions of numerical algorithms (pp. 223–226) and quotidian procedures (pp. 220–223), and compare them to Turing machine instructions.

<sup>4</sup> The distinction between internal and external relations may be a bit obscure for non-philosophers. Internal relations depend upon the internal features of an entity. What makes an object a circle, for instance, is a relation among its geometrical subparts; namely, every point on its surface is equidistant from its center. In contrast, external relations are independent of the internal constitution of the objects they relate. A good geometrical example is the relation of contiguity. It doesn’t depend on the internal structure of the objects it relates; objects of any shape whatsoever may be contiguous.

<sup>5</sup> The ‘usual structure’ for a Turing machine is usually articulated in terms of the universal Turing machine, but this fact doesn’t affect the point I am making.

<sup>6</sup> I am not disputing Robin Gandy’s claim (1980, pp. 123–145) about the computational capacities of Newtonian machines exceeding those of Turing machines in the sense that for every number theoretic function (including the Turing uncomputable number-theoretic functions), there is a Newtonian machine that can compute it; the reason that Newtonian machines can do this is because a Newtonian universe can contain rigid rods of arbitrary length and signals traveling at superluminal speeds, and hence can support accelerating (Zeus) machines. My point here is only that the conventional ‘pushes’ and ‘pulls’ of Newtonian physics, which form the basis for the intuitive notion of mechanism, are not required to instantiate a Turing machine.

<sup>7</sup> Gandy argues (1980) that Turing’s account was based on human calculation, and that (despite the use of the words ‘mechanical’ and ‘mechanism’), Turing did not concern himself with physical machines. Gandy attempted to extend and generalize Turing’s work in light of the actual physics of concrete machines.

<sup>8</sup> Gandy (1980) also makes this point; he attempts to incorporate the idea of time and mechanism into Turing’s account.

<sup>9</sup> For a more detailed discussion of the ‘precision’ of quotidian procedures, see ‘Recipes, Algorithms, and Programs’ (Cleland, 2001, pp. 220–223).

<sup>10</sup> As I have argued elsewhere (Cleland, 2001, pp. 220–23), most procedures (with the possible exception of informal mathematical algorithms) are ultimately analyzable in terms of prescriptions of basic bodily action. Basic bodily actions are specified in terms of their direct effects on the body, namely, basic bodily motions. As an example, the instruction ‘move your finger’ specifies a motion of one’s finger. It does not tell one how to move one’s finger; it merely assumes that you already know how to do it. No instruction can tell one *how* to perform a basic bodily action – either you know how to do it or (because of some disability) you don’t. But someone who knows how to perform basic bodily actions can be trained to apply instructions prescribing them, and if one knows how to apply instructions prescribing basic bodily actions, then one can follow procedures prescribing them. Finally, if one knows how to implement procedures prescribing basic bodily actions, one can implement more complex procedures. Eventually one is able to follow a recipe for Hollandaise sauce, despite the fact that the instructions of the recipe do not provide precise descriptions of what is to be done.

<sup>11</sup>The following arguments fill out and extend material adumbrated in ‘Is the Church–Turing Thesis True?’ (Cleland, 1993, pp. 291–297), where I first argued that the effectiveness of quotidian (‘mundane’) procedures depends upon causal (vs. logical) considerations.

<sup>12</sup>I am not committing myself to a particular view about the status of possible world. For my purposes, it doesn’t matter whether they are merely linguistic/conceptual devices for entertaining alternatives to the actual or metaphysically real in David Lewis’ robust sense.

<sup>13</sup>As discussed in ‘Is the Church–Turing Thesis True?’ (Cleland, 1993, pp. 297–301), failure to carefully distinguish a procedure (qua course of action-*types*) from the individual courses of action (tokens) that instantiate it or the causal processes that make it effective is the source of many puzzles about when different descriptions of procedure may be said to specify the same (vs. different) procedures. These puzzles afflict not only questions about when computer programs specify the same procedure (e.g., Does a Fortran program for computing the Euclidean algorithm specify the same procedure as a C program for computing it?, Does a Fortran program for computing averages specify the same procedure when it is implemented on machines having very different architectures?) but also questions about the identity of quotidian procedures such as recipes and methods (e.g., Does Child’s ‘easy blender method’ for Hollandaise sauce specify the same procedure as the traditional method?, Under what conditions can two chemical syntheses for making a drug be said to specify the same procedure?)

<sup>14</sup>See Daniel Hausman (1998, pp. 55–106) for a detailed discussion of the metaphysics of causal openings and their connection to agency and intervention. It should be noted that my use of these ideas does not presuppose the correctness of his controversial theory of causation (in terms of ‘the independence condition’).

<sup>15</sup>Hausman (1998, pp. 55–106) argues that causation requires such opening – that a world without such openings would have physical processes but not causation. He also maintains that the direction of causal openings supplies the direction of time. I wish to remain agnostic on both these points.

<sup>16</sup>There is an extensive literature on the use of these relations to distinguish common causes from direct causes; for an excellent discussion and review of this literature see Judea Pearl (2000).

<sup>17</sup>These famous examples appear in Douglas Hofstadter’s ‘A Conversation with Einstein’s Brain’, and John Searle’s ‘Minds, Brains, and Programs’, both reprinted in Hofstadter and Dennett (1981).

<sup>18</sup>These well known examples (which are essentially the same) are supposed to be counterexamples to the received view on computation; they are supposed to show that (on the received view) every Turing computable function is computed by virtually any physical system. The wall example can be found in John Searle (1992, pp. 208–209). Ian Hinckfuss is the author of the legendary Hinck’s pail. He presented it as a problem case during a discussion on computation at the 1978 meeting of the Australasian Association of Philosophy.

## References

- McKeon, R. (1970), *The Basic Works of Aristotle* (Bk. VI, Ch.8, Sec., 9), New York: Random House.
- Berlinski, D. (2000), *The Advent of the Algorithm*, New York: Harcourt.
- Cleland, C.E. (1993), ‘Is the Church–Turing Thesis True?’, *Minds and Machines* 3, pp. 283–312.
- Cleland, C.E. (1995), ‘Effective Procedures and Computable Functions’, *Minds and Machines* 5, pp. 9–23.
- Cleland, C.E. (2001), ‘Recipes, Algorithms, and Programs’, *Minds and Machines* 11, pp. 219–237.
- Copeland, J. (1998), ‘Turing’s O-machines, Searle, Penrose, and the Brain’, *Analysis* 58, pp. 129–131.
- Gandy, R. (1988), ‘The confluence of Ideas in 1936’, in R. Herkin, ed., *The Universal Turing Machine: A Half-Century Survey*, Oxford: Oxford University Press, pp. 55–112.
- Gandy, R. (1980), ‘Church’s thesis and principles for mechanisms’, in J. Barwise, J.J. Keisler and K. Kunen, eds., *The Kleene Symposium*, Amsterdam: North Holland, pp. 227–258.

- Gödel, K. (1965), 'Remarks Before the Princeton Bicentennial Conference on Problems in Mathematics', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 84–88.
- Hausman, D.M. (1998), *Causal Asymmetries*, Cambridge: Cambridge University Press.
- Hofstadter, D. and Dennett, D. (1981), *The Mind's Eye*, New York: Basic Books.
- Kleene, S. (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, M. (1967), *Computation: Finite and Infinite Machines*, Englewood Cliffs: Prentice-Hall.
- Pearl, J. (2000), *Causality*, Cambridge: Cambridge University Press.
- Rosser, J.B. (1939), 'An Informal Exposition of Proofs of Godel's theorem and Church's Theorem', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 223–229.
- Turing, A. (1965a), 'Systems of Logic Based on Ordinals', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 155–222.
- Turing, A. (1965b), 'On computable Numbers with an Application to the *Entscheidungsproblem*', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 116–151.
- Searle, J. (1992), *The Rediscovery of the Mind*, Cambridge: MIT Press.
- Wang, H. (1974), *From Mathematics to Philosophy*, London: Routledge and Kegan Paul.