



The concept of computability

Carol E. Cleland

*Department of Philosophy & Institute for Cognitive Science, University of Colorado,
Boulder, CO 8009, USA*

Received 8 August 2003; received in revised form 1 November 2003

Abstract

I explore the conceptual foundations of Alan Turing's analysis of computability, which still dominates thinking about computability today. I argue that Turing's account represents a last vestige of a famous but unsuccessful program in *pure* mathematics, viz., Hilbert's formalist program. It is my contention that the plausibility of Turing's account as an analysis of the computational capacities of *physical* machines rests upon a number of highly problematic assumptions whose plausibility in turn is grounded in the formalist stance towards mathematics. More specifically, the Turing account conflates concepts that are crucial for understanding the computational capacities of physical machines. These concepts include the idea of an "operation" or "action" that is "formal," "mechanical," "well-defined," and "precisely described," and the idea of a "symbol" that is "formal," "uninterpreted," and "shaped". When these concepts are disentangled, the intuitive appeal of Turing's account is significantly undermined. This opens the way for exploring models of hypercomputability that are fundamentally different from those currently entertained in the literature.

© 2003 Elsevier B.V. All rights reserved.

Keywords: *Entscheidungsproblem*; Hilbert; Turing machine; Computable

1. Introduction

Alan Turing's analysis still dominates thought about the nature of computability and the capacities of physical computing machines.¹ This is somewhat ironic because Turing explicitly designed it to solve a problem arising out of an unsuccessful early 20th century program in the foundations of *pure* mathematics. The program, known as

E-mail address: cleland@colorado.edu (C.E. Cleland).

¹ There is disagreement about whether the received view on Turing's analysis is historically correct—accurately represents Turing's actual views (e.g., Copeland [6]). This is an interesting and historically important debate. Nevertheless I am going to ignore it in this paper because my target is not the historical Turing but the contemporary view of computability, a view that is (correctly or not) commonly attributed to Turing.

formalism, originated with the work of David Hilbert. Hilbert was responding to the discovery of paradoxes in Georg Cantor's remarkably fruitful new set theory. In this paper I trace the development of the formalist school of thought in mathematics and analyze its impact on current conceptions of computability. I argue that the plausibility of Turing's account as an analysis of the computational capacities of *physical* machines rests upon a number of highly problematic tacit assumptions. When these concepts are disentangled, the intuitive appeal of Turing's account (as providing an analysis of the general concept of computability) is significantly undermined. I conclude with a discussion of the consequences of my findings for the possibilities for hypercomputation (computation beyond the so-called "Turing limit").

2. Paradise lost

The discovery of paradoxes (e.g., Bertrand Russell's set of all sets that are not members of themselves) in Cantor's wonderful new set theory plunged mathematics into crisis; for a more extensive discussion of the history, see [12,15]. Designed to clarify the foundations of mathematical analysis, Cantor's set theory sanctioned the full strength of the controversial real number system with its mysterious irrational numbers. His predecessors had found irrational numbers problematic because they seem to involve *actual* infinities, e.g., Leibniz's infinitesimals. The actually infinite transcends all possible experience. It is perceptually indistinguishable from the unending and the indefinitely large or small. Indeed, before Cantor, most mathematicians doubted the intelligibility (let alone the existence) of the concept of actual infinity. Limit processes were introduced by Weierstrass to circumvent the actual infinite by systematizing a notion of the merely potentially infinite, potential infinity being characterized as an incomplete entity that is indefinitely increasable or decreasable. Irrational numbers were defined (by Weierstrass and follow travelers) in terms of the limits of sequences or series of rational numbers. By the late 18th century, however, it was clear that limit processes could not keep the actual infinite at bay; understanding the irrational numbers required a coherent concept of completed infinities. Cantor cut the Gordian knot (so-to-speak) by developing a revolutionary set theory that explicitly introduced hierarchies of completed infinite sets into mathematics. Classical arithmetic, with its problematic irrationals, was reconstructed in terms of these hierarchies. The importance of Cantor's work to mathematics cannot be overstated. Without analysis much of modern mathematics (including the calculus, geometry, abstract algebra, and most of applied mathematics) would disappear. The discovery of paradoxes in Cantor's set theoretic paradise was thus a serious blow to mathematicians.

David Hilbert hoped to save Cantor's set theory from the ravages of the paradoxes by radically reconceptualizing mathematics.² Mathematics is not the study of abstract

² Hilbert was the originator of formalism but there is significant disagreement about his actual views. It is generally conceded that his version of formalism is weaker than the position now associated with formalism. Indeed, he has been criticized for his fence-straddling realism about mathematics; he identified the content of mathematics with physical marks and their manipulations. My concern in this paper, however, is not with the history of Hilbert's ideas so much as the contemporary understanding of the formalist position. For a detail discussion of Hilbert's early views and their relation to those of later formalists, see [12,15].

entities (e.g., the number 2) and their interrelations (e.g., less than). Mathematics is the study of formal systems. The terms and operations of formal systems consist of finite numbers of primitive symbols (marks or “strokes”) and finite numbers of “purely mechanical” operations on these symbols. Considered as part of a formal system, the symbols are meaningless. They may stand for anything whatsoever or nothing at all. All that matters is their structure—their proverbial “shape”. Any structural feature may play the role of symbol so long as different primitive symbols are represented by distinct structures. The operations of a formal system are sensitive only to the structure (vs. meaning) of a symbol. Their purpose is to construct strings of symbols (formulae) and to transform them into other strings of symbols in a step-by-step fashion (i.e., to construct proofs).

Hilbert’s plan was to formalize enough of classical arithmetic for doing analysis while avoiding the paradoxes. To accomplish this he needed the right kind of formalism, namely, a formalism whose *formal* consistency corresponds to the *logical* consistency of the relevant portion of classical arithmetic. The basic idea was to start with an axiomatization of classical arithmetic (of which a number were already available, including Whitehead and Russell’s, in *Principia Mathematica*), and then formalize it. In the process of formalization, the traditional content of mathematics is stripped away. The features that remain are purely formal. The task of the formalist was to show that the resultant formalism provides a consistent and complete formal theory of classical arithmetic. Proofs of consistency and completeness could not, however, employ methods resting upon suspect transfinite ideas; this would defeat the whole purpose of the formalist program. The powerful existence proofs of classical mathematics were thus unavailable. As a consequence, one of the central problems facing the formalist was that of finding a *definite finitary formal* procedure that could be used to unequivocally decide the provability of *any* claim in formalized mathematics. This decision problem became known as Hilbert’s *Entscheidungsproblem*.³

The similarity between Turing’s account of computability and the formalist account of mathematics is obvious. The symbols of Turing machines are “uninterpreted” (meaningless) structures that are manipulated solely on the basis of their “shapes”. Indeed, despite the popular conception of Turing machines as ultimately simple, abstract mechanisms, mathematicians and theoretical computer scientists analyze Turing machines as formal mathematical structures. This is hardly an accident. Turing did not invent his “machines” to provide an analysis of the computational capacities of physical machines. He designed them to solve the *Entscheidungsproblem*, a problem specific to the formalist program in the foundations of pure mathematics. Not all programs in the foundations of mathematics would find a formalist proof convincing. Indeed, the two other major schools of thought, intuitionism and Platonism, reject the idea that everything important to classical arithmetic can be captured in a formalization; for more on this, see [12]. Something meaningful and unformalizable that renders arithmetic logically (vs. merely formally) inconsistent or consistent could have been left

³ Hilbert officially introduced the *Entscheidungsproblem* in 1928 in a textbook on logic co-authored with his student Ackermann.

out of the formalism. The point is Turing's account of computability is founded upon an idiosyncratic philosophical view about the nature of mathematics.

Nevertheless, it did not take long for Turing's analysis to be extended to the computational capacities of physical machines, a position which (as the title of Martin Davis's recent book, *Engines of Logic*, illustrates) still dominates in the fields of computer science and mathematics. If the formalist program had been successful this would make good theoretical sense. Classical arithmetic would have been fully captured in a formal system. As a consequence, the formal possibilities for computing functions would have exhausted *all* the possibilities, setting an absolute logical limit on what functions are computable. But as is well known, Kurt Gödel effectively demolished the formalist program with the discovery of his famous incompleteness theorems. In a nutshell, Gödel proved that a formal system rich enough to encapsulate elementary arithmetic (let alone classical arithmetic) could not be both consistent and complete; if arithmetic is consistent, arithmetical truth goes beyond what can be proven in any (recursively axiomatisable) formal system. This means that formalism cannot provide us with a satisfactory account of the nature of arithmetic.

In light of the failure of the formalist program, the claim that Turing machines capture the computable functions is highly suspect, ultimately resting upon non-mathematical suppositions about the nature of computation and physical processes. Turing embraced a highly anthropomorphic view of computation. He unselfconsciously introduced his "machines" in the context of a person plugging away at arithmetical calculations with pencil and paper [18, p. 135]. Given that physical devices can do many things (e.g., escape the pull of Earth's gravitational field, withstand tremendous pressures at great ocean depths) that cannot be done by unaided human beings, why would any one be convinced that human beings provide a good model for the computational capacities of physical machines? Furthermore, it is clear that the capacities of nature are not just a matter of having the right formal structure. Build the walls of a deep sea submersible out of the wrong material and it will quickly implode. It is not enough for defenders of the Turing view to retort that Turing machines represent *idealized* human beings, and thus are not subject to human frailties. For this merely side steps the central question, which is why model computability on the capacities of limited creatures like human beings in the first place!

Besides, there are many different ways in which one could idealize the computational activity of human beings. Turing opted for setting no upper bound on the number of actions his "machines" could perform; he allowed them unlimited space and time in which to operate. But he could have idealized human action in other ways. He could have set no upper bound on the speed with which they perform their actions, for instance. What justifies selecting one idealization of human action over another? An obvious retort is that there is an upper limit to the speed of physical processes in our universe. Einstein's theory of relativity tells us that no real-valued mass can travel faster than light. But this is an empirical consideration. Why not invoke empirical considerations in the case of space and time too? Entropy, for example, sets limits to the capacities of physical objects to perform actions. No physical device can go on and on, performing actions forever. What justification can there be for preferring one empirically unrealistic idealization of human action to another?

Just because humans who are consciously performing calculations do it in a step-by-step (sequential and discrete) fashion, does not mean that all calculation must be done in this manner, i.e., must actually be constituted by a sequence of discrete primitive sub calculations. Insofar as functions are identified with sets of ordered pairs, it seems that any process that achieves the requisite pairings ought to count as a computation. Why take human computational behavior as the paradigm for all computational processes?

The familiar retort to those who raise such concerns is that Turing's account says nothing about *how* physical processes actually compute functions. It merely says that (however they do it) no physical processes can compute something that a Turing machine could not compute. But given the failure of the formalist program in mathematics, what could motivate such a claim? Again, there is a familiar response. Turing's analysis initially faced two competitors, Herbrand-Gödel general recursiveness and Church's lambda-calculus. Turing demonstrated that all three accounts are extensionally equivalent vis-à-vis the computation of the number-theoretic functions. There is thus strong inductive support for the claim that mathematicians have captured the decidable (a.k.a. computable) number-theoretic functions. But this response ignores the fact that all three accounts are grounded in the formalist perspective on mathematics. Indeed, each proposal was explicitly designed to solve Hilbert's *Entscheidungsproblem*. Viewed in this light, it is not so surprising that they end up classifying the same functions as computable; they are built on the same conceptual bedrock. Finally, the oft-repeated allegation that no one has been able to come up with a function that cannot be computed by a Turing machine but can nonetheless be computed by some other means provides little support for the official Turing view. For it studiously ignores the growing body of literature on inductive machines (e.g., Burgin [1], Gold [11] and Putnam [13]), analogue chaotic neural nets (e.g., Siegelmann [16]), and accelerating (Zeus) machines (e.g., Copeland [7] and Steinhart [17]) claiming to have done just this! Defenders of the received view on computability ignore this literature because they are not willing to countenance an activity that fails to conform to the strictures of the official Turing line as a computation. There is more than a whiff of circularity here.

It is high time that the conceptual foundations of the official Turing account be laid bare and subject to critical scrutiny. For we cannot reach an informed decision about its relevance to the capacities of physical machines unless we understand the assumptions upon which it rests. The remainder of this paper is devoted to this task. It reviews work that has been presented elsewhere (Cleland [2–5]). I argue that the plausibility of the received view of computability rests upon some highly problematic concepts, more specifically, the idea of an “operation” or “action” that is “formal,” “mechanical,” “well-defined,” and “precisely described,” and the idea of a “symbol” that is “formal,” “uninterpreted,” and “shaped”. These concepts play pivotal roles in motivating the claim that Turing's account captures the general intuitive concept of computability. Turing machines are typically characterized as providing paradigms of definite methods in virtue of (1) the “precision” with which their instructions specify the actions that they prescribe and (2) the “purely mechanical” nature of those actions. But as I show, a close look at these claims reveals that Turing machines cannot live up

to their vaunted reputation of providing perfectly precise specifications of action and the claim that Turing machine operations are “purely mechanical” confuses different meanings of the word “mechanical”. Similarly, the source of the great generality of Turing machines is reputed to lie in the “purely formal” character of their symbols and operations. But the level of generality is too high. One cannot get bona fide actions out of purely formal symbols and operations. I conclude by defending the radical claim that Turing machines do not provide us with genuine procedures. They provide us with mere procedure schemas. When these schemas are filled in with genuine action, we get authentic procedures. As I argue in the final sections, understanding the capacities of physical machines to compute functions requires more than a theory of procedural schemas. It requires a theory of authentic procedures, something that the official Turing account (lashed tightly to the conceptual framework of the formalist account of mathematics) is intrinsically incapable of providing.

3. The received view on computability

We begin our discussion with a brief review of the structure and function of the peculiar objects at the heart of the received view on computability, namely, Turing machines. I shall restrict my discussion to the simplest Turing machines, namely, the deterministic, sequential “machines” of introductory textbooks; everything that I say applies fairly straightforwardly to the more complex non-deterministic and multidimensional Turing machines.⁴

Turing machines are characterized in the literature in two quite different ways, namely, as abstract mechanisms and as mathematical structures. The former characterization is much more machine-like than the latter. A Turing machine is described as consisting of a “mechanism,” known as a “finite state machine,” coupled to an external storage medium, known as the “tape,” through an abstract device known as the “head.” In the standard scenario, each square of the tape is occupied by one of two different symbols (S_0 and S_1), and the head, which is always positioned over a square of the tape, performs one of a small set of extremely simple basic operations; the head can “erase” or “write” a symbol, “move” to the left one square, or “halt.” What the head does to the tape is determined by the instruction that it is currently carrying out. Turing machines compute arithmetical functions by implementing lists of conditional instructions called “programs.” The arguments and values of the function are “coded” on the tape as strings of S_0 s and S_1 s.

The use of common English imperatives, such as “write” and “move,” to describe the basic operations of Turing machines underscores their anthropomorphic character. Turing machines do the sorts of things that human beings do when consciously performing calculations with pencil and paper. But, as I have argued [4,5], the idea that

⁴ The core of the following argument first appeared in [4], where I compared and contrasted the “precision” of the instructions of numerical algorithms, “quotidian” (ordinary, everyday) procedures, and Turing machine instructions in detail.

a Turing machine can perform an action (however broadly or abstractly construed) is fundamentally mistaken.

Action requires something to manipulate; if nothing is manipulated no action is performed. The entities reputedly manipulated (“written” and “erased”) by Turing machines are symbols. In keeping with the formalist stance on mathematics, Turing machine symbols are purely formal entities. Their meaning plays no role in what is done to them. They are manipulated solely on the basis of their proverbial shapes. It follows that the shapes of Turing machine symbols are crucial to the identity of the actions prescribed by Turing machines.

Unfortunately, however, the idea that Turing machine symbols have distinguishing shapes is fraught with difficulties. One might try identifying the shape of a Turing machine symbol with some geometrical feature that all of its instances have in common. But the very same symbol may be instantiated by objects having incommensurable shapes, e.g., numerals, pen strokes, pebbles on squares of toilet paper, and nails in tin cans. Moreover, non-geometrical characteristics such as weights, colors, durations of a sound, or intensities of light may also be used to instantiate Turing machine symbols. The problem is that the differing symbols of a Turing machine may be instantiated by any definite but distinct physical entities. The only physical constraints on the symbols of a Turing machine hold within (vs. across) its physical realizations. Within a particular realization, every token of the same symbol must share some (it does not matter what) physical property and all tokens of different symbols must differ in some (it does not matter what) physical property. Considered independently of a particular realization, however, the most that may be said is that different symbols have different but not any definite distinguishing physical properties. The idea of a mere difference in some completely indeterminate physical property is not enough to secure the idea that Turing machine symbols have distinguishing shapes.

As one might expect, the situation only gets worse on the formal mathematical view of Turing machines. Turing machines are identified with mathematical structures, which 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 “usual” (prototypical) 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.⁵ The symbols are typically represented as “0” and “1,” which suggests that they are numerals. But numerals have definite geometrical shapes. If they are numerals, it makes no sense to talk about instantiating the same Turing machine in a physical system whose symbols consist of pebbles or flashes of light, for instance. That is to say, identifying “0” and “1” as numerals amounts to conflating a specific abstract mathematical structure (the usual structure for a Turing machine) with one of its concrete instantiations. On the other hand, if they are not numerals, what can they be? Integers? Integers (qua abstract mathematical objects) have no physical features whatsoever. If they are integers we lose even the minimalist idea that distinct Turing machine symbols must differ in at least *some* physical property. We are left with a relation of bare numerical

⁵ The usual structure for a Turing machine is usually articulated in terms of the universal Turing machine, but this fact does not affect the point I am making.

difference among symbols. Bare numerical difference is incompatible with the idea that uninstantiated Turing machine symbols have distinguishing “shapes,” however broadly or indefinitely construed. To appreciate this, one need only entertain the theoretical possibility of authentic twins, e.g., two leaves or snow flakes that are literally identical. Although numerically distinct, such objects would be exactly alike in all their physical characteristics.

This brings us to what is perhaps the most problematic aspect of the idea that Turing machine symbols have distinguishing “shapes.” In mathematics, Turing machines are identified only up to isomorphism; they are not actually identified with their usual structures. More specifically, on the formal mathematical account, a Turing machine is a class of isomorphic structures (abstract and concrete), one of which (the usual structure) happens to be picked out as representative of the class as a whole. From a logical point of view, a *class* of Turing machine instantiations is no more a Turing machine than the class of all red objects is a red object. In other words, considered independently of a particular instantiation, the symbols of a “Turing machine” amount to nothing more than logical roles in a second order structure that (strictly speaking) is *not* a Turing machine. Rather than being symbols of some extraordinarily abstract and refined sort, Turing machine “symbols” are just placeholders for symbols. It is not until we descend to the level of the individual structures in the equivalence class defining a Turing machine, that we get authentic symbols. It follows that we can not 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 the real McCoy, however, we still could not get precise specifications of action out of Turing machine instructions. Although action presupposes something to manipulate, having something to manipulate is not sufficient to pick out an action. Specifying a knife and a carrot in a recipe, for example, does not fix what is to be done to the carrot by the knife. The possibilities are wide open. They include dicing it, slicing it, shaving it, stabbing it, stroking it with the blade, and pounding it with the handle, to mention just a few. In other words, despite the use of familiar English expressions for action, Turing machine instructions do not specify what is to be done once their symbol-placeholders are filled by genuine symbols. “Erasing” a pebble could be realized by activities as diverse as painting it, pulverizing it, flipping it over, or removing it from a tin can.

One of the great virtues of Turing’s account is supposedly the “purely mechanical” character of Turing machine operations. As earlier, the use of this expression is misleading. When physical scientists speak of mechanical actions they have in mind the proverbial pushes and pulls of Newtonian mechanics. The instructions of Turing machines are not limited to actions of this sort, however. They may be satisfied by activities as diverse as action-at-a-distance (which requires no intervening causal chain) and angels creating and annihilating pebbles. It does not matter whether action-at-a-distance or angels really exist. All that matters for our purposes is that they violate no laws of logic and could be used to instantiate a Turing machine. This underscores a crucial point. In keeping with the formalist framework for mathematics, the constraints imposed upon Turing machine “actions” are purely structural: They must be distinct from one another (discrete) and they must occur in a time-ordered sequence.

These are exceedingly minimal requirements. The physical scientist's notion of mechanical encompasses much more. In order to qualify as mechanical, an activity must have the right kind of intrinsic causal character. The upshot is that Turing machine actions cannot be said to be "mechanical" in anything like the physical scientist's sense.

It is possible, of course, that Turing and his defenders had some other notion of mechanical in mind. Indeed, Turing sometimes used the word "automatic" in place of "mechanical" [18, p. 118] and his student Robin Gandy [10, p. 80] spoke of Turing machines performing actions without "thought or volition." Unfortunately this use of "mechanical" does not capture the idea of a finite constructive process, a crucial ingredient in *any* notion of mechanism (Newtonian or otherwise). As an illustration, one can coherently speak of bored angels "automatically" performing baffling miracles. In short, the terminology may be suggestive but one should not be fooled by it into thinking that Turing machines are mechanisms in anything like the sense in which a physical machine is said to be a "mechanism". There is even some question as to whether Turing had in mind physical machines when he used expressions like "mechanical" and "mechanism." According to Gandy [10], Turing's target was human calculability and he never intended his analysis to apply to physical machines. Gandy attempted to extend Turing's analysis to physical machines in light of very general considerations from contemporary physical theory [9,10]. Oron Shagrir argues [14], however, that Gandy's account does not encompass all instances of finite machine computation, suggesting perhaps that Turing's analysis cannot be extended to physical computation after all.

As is hardly surprising, the formal mathematical account does not offer us any help in making sense of the idea that Turing machine instructions prescribe genuine actions. The basic Turing machine operations ("erase," "write," "move") are defined in terms of ordered n -tuples (mathematical functions). Ordered n -tuples do not require change (let alone change that qualifies as action) for their realization. Thus the structures in the equivalence class defining a Turing machine need not be dynamic, let alone mechanical.⁶ They may be instantiated by spatially ordered structures such as mineral crystals. In moving from the informal to the formal account, we have lost the most essential feature of action, namely, dynamic change.

Let us pull this all together. The instructions of Turing machines (considered as uninstantiated, multiply realizable abstract entities) do not prescribe actions of any sort. If they do not prescribe actions, they can hardly be said to precisely describe them. The vaunted precision of Turing machine instructions thus turns out to be a myth. This has serious consequences for the received view on computability. At the heart of the received view is the notion of an effective procedure: Turing machines are said to be paragons of effective procedure. For a procedure to be effective, the actions it prescribes must be precisely specified (i.e., "precisely described" or "well defined"). So if Turing machine programs do not precisely specify actions, they cannot be said to supply us with effective procedures. Indeed, because they do not prescribe actions

⁶ Gandy [9] also makes this point; he attempts to incorporate the idea of time and mechanism into Turing's account.

of any sort, they cannot even be said to supply us with authentic procedures. At best, Turing machines may be said to provide procedure schemas, i.e., time-ordered, skeletal frameworks for procedures. To get authentic procedures, these frameworks must be filled in with specifications of action.

4. Causation, effective procedures, and physical computability

Understanding the computational capacities of physical machines requires a theory of authentic (vs. schematic) procedures. We need to understand the general conditions under which an authentic procedure is effective, i.e., reliably produces a certain result (e.g., the decimal expansion of π) when correctly followed. The best examples of authentic procedures do not come from mathematics, however, but from ordinary, everyday life. A recipe for a cake and instructions for assembling a child's tricycle provide salient examples. In previous work [4,5], I dubbed such procedures “quotidian” in order to distinguish them from the schemas provided by Turing machine theory.

Unlike Turing machines, it is clear that quotidian procedures prescribe bona fide actions, e.g., whip the egg yolks until they form a yellow ribbon and bolt the handlebars to the frame. Admittedly, they do not provide perfectly precise specifications of action. But as I have argued [4,5], the perfect precision of Turing machine instructions is a myth. Turing machines are literally *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 “write” an S_0 and yet fail to “place” an S_0 on the tape. In contrast, failure is always a very real possibility for a follower of a quotidian procedure. No instruction prescribing an authentic action can preclude the possibility of being misunderstood or misapplied. The success of human beings in following imprecisely described instructions is a product of training coupled with a shared repertoire of basic bodily actions such as moving a finger or rotating a wrist.⁷

From an intuitive standpoint, many quotidian procedures are *effective* in the sense that they *reliably* produce specific outcomes when correctly followed. Unlike Turing machine procedures, however, the reliability of a quotidian procedure does not depend just upon the identity (and time-order) of the actions it prescribes (however imprecisely). It also depends upon the causal consequences of performing the actions. Nevertheless, these consequences are not themselves specified by the procedure; the procedure specifies only the actions that produce them. To appreciate this, consider a

⁷ Basic bodily actions are specified in terms of their direct effects on the body, namely, basic bodily motions. The instruction to move one's finger, for example, does not describe how to do it; it presupposes that you already know how to move your finger. No instruction can tell one how to perform a basic bodily actions—either you are physiologically intact and know how to do it, or you can not do it because of some disability or restraint. Anyone who knows how to perform basic bodily actions can be trained to apply instructions prescribing them, and hence to follow procedures specifying complex spatio-temporal arrangements of basic bodily actions. Children learn to follow such procedures, and as they become increasingly sophisticated, they can follow procedures (e.g., a recipe for Hollandaise sauce) prescribing complex spatio-temporal arrangements of intricate actions.

recipe for Hollandaise sauce. One is instructed to pour melted butter by droplets into warm egg yolks while continuously beating them with a wire whisk. What produces the Hollandaise sauce, however, is not the mere performance of these actions but the physical processes that they generate and sustain. If performing these actions reliably caused egg yolks to become granular and float in butter (instead of causing them to absorb butter and hold it in creamy suspension) the recipe would not be reliable for making Hollandaise sauce. But although they are crucial to the effectiveness of the recipe, the causal consequences of these actions are not themselves specified by the recipe. To see this, imagine a possible world, W_1 , in which the laws of chemistry differ from those in the actual world, W_a , in such a way that whisking droplets of butter into warm egg yolks causes them to turn into a smooth elastic mass.⁸ Now suppose that someone in W_1 correctly follows the recipe for Hollandaise sauce. The result is nothing like a Hollandaise sauce, and this difference in outcome does not correspond to a difference in recipe. In both worlds, people follow the same instructions, performing the same (types of) actions in the same order in time. But they get different results. Performing the same actions (in the same order in time and space) causes very different things to happen in W_a and W_1 . In other words, the *same* recipe may be effective in one world and ineffective 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. The *identity* of the recipe *qua* procedure does not.

This is in stark contrast to Turing machines, whose “outcomes” are completely determined in advance by their instruction sets (and their initial configurations). The appearance of an S_0 on a Turing machine’s tape does not result in any further changes to the tape until another instruction, containing an S_0 in its antecedent, is implemented. This is not to deny that if the S_0 were interpreted as the integer 1 it would have arithmetical consequences. But in this scenario one is no longer dealing with a Turing machine simpliciter. One is dealing with one of its interpretations. Assigning different integers to S_0 yields different interpretations, with correspondingly different *arithmetical* consequences. Thus the same Turing machines can compute different arithmetical functions. Similarly, if one encodes the S_0 by placing a stone in a cup of water, there will be *causal* consequences such as a rise in the level of the water. But, again, one is not dealing with a Turing machine simpliciter. One is dealing with one of its instantiations. Considered just in itself—independently of any of its instantiations or interpretations—the sequence of “actions” specified by a Turing machine is the only process that the machine engages in. Put another way, unlike the case with quotidian procedures, there is no distinction between the sequence of actions performed by the machine and the process that generates the result with respect to which it is said to be reliable (a.k.a. effective). Viewed in this light, it is easy to see why everything that a Turing machine “does” is characterized as *completely* determined in advance by the procedure (given the initial configuration of the machine), and hence why the effectiveness of a Turing

⁸ I am not committing myself to a particular view about the status of possible worlds. For my purposes, it does not matter whether they are metaphysically real in David Lewis’ robust sense or (as seems more plausible) merely linguistic/conceptual devices for entertaining hypothetical alternatives to the actual world.

machine procedure for a particular outcome is absolute, i.e., does not change from one possible world to the next. It is also easy to see why every Turing machine is characterized as qualifying as an effective procedure. But of course these characterizations of Turing machines are misleading. They presuppose that Turing machines provide us with bona fide procedures. As we have seen, they do not.

Their dependence upon the causal character of the world in which they are implemented is the source of the great power of quotidian procedures. The actions specified by quotidian procedures represent adroit causal interventions in the course of nature, interventions which give rise to things that nature would rarely (if ever) produce on her own. We owe most of the comforts of civilization to the design of quotidian procedures for forcing nature along improbable causal pathways. The limits to what can be achieved by means of quotidian procedures are set by the causal possibilities for intervening in physical processes. These are not the same from one possible world to another. In order to support quotidian procedures, a world must have localized causal openings. Localized causal openings are nature's ready-made switches. They are local physical factors that may be modified independently of other physical factors with reliable causal consequences. A quotidian procedure may thus be viewed as a temporally and spatially organized arrangement of selective causal interventions. These manipulations of nature serve to redirect and reshape natural processes into the improbable outcomes with respect to which the procedure is reliable.

But as I have argued [5], what happens as a result of these highly selective interventions is not specified by the procedure. There is a causal gap between the actions specified by a quotidian procedure and the physical processes that make it reliable. One might suspect that this gap is a disadvantage since, unlike the case with Turing machines, it means that the reliability of the procedure can not be attributed to the procedure per se. Indeed, it explains why the reliability of some familiar quotidian procedures (e.g., recipes) varies under different physical circumstances (altitude) within the same world. But further reflection reveals that the gap between action and process is actually a perk. Because of it, one can make a great Hollandaise sauce without knowing any chemistry. One need only know how to perform the actions specified by the recipe. Nature takes care of the rest. It also means that people can invent procedures for getting things done without knowing very much about the physical and chemical processes involved. The chef who invented Hollandaise sauce almost certainly knew little chemical theory but this did not prevent him or her from finding a reliable method for creating a delectable sauce. In other words, quotidian procedures insulate us from the messy causal details of the world while at the same time allowing us to effectively intervene in those details so as to reliably produce advantageous outcomes.

With an understanding of quotidian procedures firmly in hand, we can make better sense of the schematic character of Turing machines. As we have seen, quotidian procedures represent systematic ways of causally intervening in the world. In contrast, Turing machines (with their placeholders for action) are best viewed as logical possibilities for systematic causal intervention. This explains why their reputed effectiveness does not vary from one possible world to another, namely, every possible world contains at least the bare logical possibility of reliable procedures. In this context, it is

important to keep in mind the anthropomorphic character of Turing machines; indeed, as discussed earlier, they provide us with a special idealization of human (procedure-following) behavior. For this reason, they cannot be construed as exhausting all the logical possibilities for procedure.

Whether a logical possibility for procedure is physically realizable, however, depends upon the causal character of the world in which it is implemented. Worlds lacking dependable localized causal openings will not support reliable quotidian procedures. Moreover, physical processes within the same world may vary greatly in the frequency and accessibility of their causal openings. Exploiting a causal opening may require physical circumstances that are very difficult to bring about. Some physical processes will thus be easier to harness by means of procedures than others. All other things being equal, those worlds whose physical processes contain the largest number of easily accessible, localized causal openings will support the richest technologies.

Our discussion underscores an important point. In keeping with the formalist account of mathematics, Turing's analysis treats effectiveness as a purely formal property of procedures. But as we have seen, it is not. Turing machine programs cannot be said to be effective because they are not authentic procedures; they are mere schemas for procedure. Quotidian procedures, on the other hand, are genuine procedures. But their reliability for a given outcome does not depend upon their formal (schematic) properties. It depends upon a complex causal relation standing between the actions they prescribe and suitable causal openings in the right kinds of physical process.

5. Hypercomputation

What are the implications of these findings for the computational capacities of physical machines? There is little reason to suppose that Turing machines tell us much about the capacities of physical machines to achieve reliable outcomes, and this includes computing arithmetical functions as well as fabricating things like synthetic fibers and wonder drugs. The key to computing an arithmetical function is finding a physical process that can be harnessed by a procedure in such a way as to produce something that may be *interpreted* as pairing the right numbers, viz., the arguments and values of the function. Mere logical possibilities for arranging completely indeterminate actions in prearranged orders in time cannot shed light upon what functions are computable any more than they can shed light upon the possibilities for synthesizing synthetic fabrics. For in both cases, it is the *effects* of performing *authentic* actions that ultimately produce the outcomes of interest, and these causal consequences lie outside the scope of the specifications of a Turing machine.

Nevertheless most models of hypercomputation are closely wedded to the Turing model. All of the models mentioned earlier (viz., inductive machines, analogue chaotic neural nets, and accelerating machines) closely resemble traditional Turing machines. Indeed, they differ from them only insofar as they lift various restrictions on their structure. In other words, most models of hypercomputation presuppose that the key to

computation beyond the so-called Turing limit is to be found in the structural features of procedures.

Accelerating machines [7,8] provide a salient example. Traditional Turing machines idealize human computational abilities to unlimited space and time. Accelerating machines add yet another layer of idealization, permitting traditional Turing machines to work indefinitely fast. By performing each Turing machine operation in a fraction (e.g., half) of the time that it takes to perform its predecessor, an accelerating machine can complete an infinite number of steps in a finite amount of time, thus computing functions (e.g., the halting function [7]) that the universal Turing machine cannot compute. The important point for our discussion, however, is that, except for their ability to work faster and faster, accelerating machines are just like traditional Turing machines. As a consequence, they can not be said to prescribe authentic actions any more than traditional Turing machines can be said to prescribe authentic actions; they specify only that completely indefinite “actions” be performed at faster and faster rates. The same point holds for inductive machines and analogue chaotic neural nets, which also achieve hypercomputation by eliminating restrictions on the structure of traditional Turing machines. In the case of inductive machines, the requirement that a Turing machine produce a result only when it halts is eliminated [1,11,13]. Analogue chaotic neural nets [16], on the other hand, eliminate the requirement that the input and output of a Turing machine be strictly finite. The most that may be said about inductive machines and analogue chaotic neural nets is that, like accelerating machines, they provide us with new varieties of procedural schema.

In order to get a physical device to realize these alternative procedural schemas, however, they must be filled in with authentic actions, and this brings us to a problem. Given our current understanding of physics, these schemas cannot be realized. Although one can readily imagine worlds in which, for example, a physical device behaves like an accelerating machine and performs an infinite number of structurally distinct actions in a finite amount of time, to the best of our knowledge, our world is not among them. But it does not follow from this that our world can not support hypercomputational physical devices. For as I have argued, procedural schemas are not the place to search for physically powerful models of hypercomputation. What we really need to look for are physical processes that can be causally harnessed by selective physical interventions (authentic actions) in such a way as to produce physical effects that may be interpreted as Turing uncomputable real numbers.⁹ There is no reason to suppose that achieving such outcomes requires physical machines that perform distinct actions at speeds well in excess of that of light—indeed, at speeds approaching infinity as a limit! Investigation of the purely structural features of procedures cannot, however, be expected to reveal how physical processes could be reshaped by selective causal interventions to yield an outcome of the right sort.

⁹ (Turing) uncomputable real numbers are linked to uncomputable number-theoretic functions in the following fashion: A real number whose decimal expansion is $d_1, d_2, \dots, d_n, \dots$ cannot be computed by the universal Turing machine IFF there is no function $f(n) = d_n$ on the positive integers that is computable by the machine.

Jack Copeland's coupled Turing machines [8] provide us with another example of the untoward influence of Turing's model on contemporary conceptions of hypercomputation. Coupled Turing machines achieve hypercomputation by relaxing yet another structural feature of the traditional Turing model. Glossing over the details, the simplest version of a coupled machine consists of a Turing machine linked to its environment by means of a single input channel. Unlike a traditional Turing machine, whose input must be inscribed on the tape in advance (before the machine starts), the input channel supplies a stream of digits to the machine's tape as it operates. To get a coupled Turing machine to compute a Turing uncomputable function, however, more is required. The input coming over the channel from the environment must have the right character. It must consist of an infinite sequence of binary digits $d_1, d_2, \dots, d_n, \dots$ that cannot be generated by the universal Turing machine. Put another way, the sequence must consist of the digits of the decimal expansion of some Turing uncomputable real number. Upon receipt of each digit, a hypercomputational coupled Turing machine performs some simple arithmetical operation such as multiplying it by 2. This results in a sequence $(2xd_1, 2xd_2, \dots, 2xd_n, \dots)$ that cannot be produced by the universal Turing machine. In other words, assuming that the machine does not halt, it "computes" a function $f(n) = 2xd_n$ that cannot be computed by the universal Turing machine; if it halts, the number of digits supplied by the input channel is finite, and the universal machine will be able to compute the resultant function. But this is hardly surprising. The interesting work has already been done by whatever process supplied the input to the coupled machine in the first place. It is clear that the universal Turing machine cannot simulate this process.

Similar considerations apply to Turing's famous O-machines [8], which achieve hypercomputation by means of traditional Turing machines that (unlike coupled machines) feed the arguments of uncomputable number-theoretic functions to mysterious "oracles" (proverbial black boxes), which (like the environment of a coupled machine) produce the values of these functions; the real hypercomputational work is done by the oracle [4,5]. The point is the hypercomputational capacities of O-machines and coupled Turing machines do not derive from fiddling with the structure of a traditional Turing machine. They derive from coupling a Turing machine to enigmatic processes, of which the most that can be said is that they are utterly unlike a Turing machine! Rather than focusing upon enriching or eliminating various structural requirements on traditional Turing machines, computer scientists would be better off focusing on how a non-Turing machine-like process could produce output of the kind required to turn a coupled Turing machine or an O-machine into a hypercomputational device.

One of the most common objections to the possibility of physical hypercomputation concerns the difficulty of verifying that a physical device has computed a Turing uncomputable function. But as I have argued elsewhere [3,5], the verification problem is not unique to hypercomputational devices. It afflicts all physical devices for computing functions. As an example, it is impossible to conclusively verify that my hand calculator computes basic arithmetical functions like addition. For my hand calculator will break down long before it finishes computing a *total* function like addition. If I am lucky and no electronic glitches have occurred during its remarkably short life,

my calculator will compute a partial function that is consistent with its computing addition. Yet this partial function is also consistent with my calculator's computing any one of an uncountably infinite number of other total functions that are not addition. On the other hand, I am often unlucky (particularly with cheap calculators!) since all physical devices are subject to physical perturbations that may cause them to malfunction. This underscores a frequently overlooked point. The claim that a physical machine (or, for that matter, human) computes a given function is an empirical hypothesis. Its plausibility ultimately depends upon physical considerations, both empirical and theoretical (e.g., probabilistic causal relations, counterfactual suppositions grounded in physical law), as well as mathematical considerations (e.g., identity relations among different arithmetical operations). Thus, for example, we have good empirical and theoretical reasons (based on the commonly accepted interpretation of quantum theory¹⁰) for believing that some discrete physical processes (radioactive decay) are truly random despite the fact that the universal Turing machine can not simulate them; for this reason, radioactive processes are sometimes used in place of pseudo-random algorithms, whose outputs are not genuinely random. The upshot is that we cannot dismiss the possibility that we will someday have reasons for believing that some physical device computes a Turing uncomputable function that are just as good as the reasons that we currently have for believing that our hand calculators compute addition.

In conclusion, computer scientists are more likely to succeed in designing physical devices that actually compute Turing uncomputable functions if they focus on physics rather than on fiddling with the structure of traditional Turing machines. For if I am right, the most promising possibilities for computing the Turing uncomputable depend upon finding appropriate localized causal openings in the right sorts of physical processes. Whether such physical processes exist is of course an open question. Moreover, even supposing that they do exist, it may be technologically difficult to procedurally harness them in such a way as to reliably produce an outcome that can be unambiguously *interpreted* as representing a Turing uncomputable function. But none of this should be very surprising. Despite its historical connections with the formalist program in mathematics, computer science is not a branch of mathematics or logic. It is a branch of applied science.

References

- [1] M.S. Burgin, Inductive turing machines, *Sov. Math. Dok* (1983) 1289–1293.
- [2] C.E. Cleland, Is the Church–turing thesis true? *Minds Mach.* 3 (1993) 282–312.
- [3] C.E. Cleland, Effective procedures and computable functions, *Minds Mach.* 5 (1995) 9–23.
- [4] C.E. Cleland, Recipes, algorithms, and programs, *Minds Mach.* 11 (2001) 219–237.
- [5] C.E. Cleland, On effective procedures, *Minds Mach.* 12 (2002) 159–179.
- [6] J. Copeland, The broad conception of computation, *Amer. Behavioral Sci.* 40 (1997) 690–716.

¹⁰ Under some interpretations of quantum mechanics, radioactive decay is a Turing computable process. The hidden variable interpretation provides a salient example. It holds that there are empirically inaccessible variables that (unbeknownst to us) render what seems to be a fundamentally probabilistic process completely deterministic.

- [7] J. Copeland, Accelerating Turing machines, *Minds Mach.* 12 (2002) 281–301.
- [8] J. Copeland, Hypercomputation: philosophical issues, *Theoret. Comput. Sci.*, this Vol. (2004).
- [9] R. Gandy, Church's thesis and principles for mechanism, in: J. Barwise, H.J. Keisler, K. Kunen (Eds.), *The Kleene Symposium*, North-Holland, Amsterdam, 1980, pp. 123–148.
- [10] R. Gandy, The confluence of ideas in 1936, in: R. Herkin (Ed.), *The Universal Turing Machine: A Half-Century Survey*, Oxford University Press, Oxford, 1988, pp. 55–112.
- [11] M.E. Gold, Limiting recursion, *J. Symbolic Logic* 30 (1964) 28–48.
- [12] S. Körner, *The Philosophy of Mathematics*, Dover, New York, 1960.
- [13] H. Putnam, Trial and error predicates and the solution to a problem of Mostowski, *J. Symbolic Logic* 30 (1965) 49–57.
- [14] O. Shagrir, Effective computation by humans and machines, *Minds Mach.* 12 (2002) 221–240.
- [15] W. Sieg, Hilbert's programs: 1917–1922, *Bull. Symbolic Logic* 5 (1999) 1–44.
- [16] H.T. Siegelmann, Computation beyond the Turing limit, *Science* 268 (1995) 545–548.
- [17] E. Steinhart, Logically possible machines, *Minds Mach.* 12 (2002) 259–280.
- [18] A. Turing, A, On computable numbers with an application to the *Entscheidungsproblem*, in: M. Davis (Ed.), *The Undecidable*, Raven Press, New York, 1965, pp. 116–151.