Effective Procedures and Computable Functions

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

Abstract. Horsten and Roelants have raised a number of important questions about my analysis of effective procedures and my evaluation of the Church–Turing thesis. They suggest that, on my account, effective procedures cannot enter the mathematical world because they have a built-in component of causality, and, hence, that my arguments against the Church–Turing thesis miss the mark. Unfortunately, however, their reasoning is based upon a number of misunderstandings. Effective mundane procedures do not, on my view, provide an analysis of our *general* concept of an effective procedure; mundane procedures and Turing machine procedures are different kinds of procedure. Moreover, the same sequence of *particular* physical actions can realize both a mundane procedure and a Turing machine procedure; it is sequences of particular physical actions, not mundane procedures, which "enter the world of mathematics." I conclude by discussing whether genuinely continuous physical processes can "enter" the world of real numbers and compute real-valued functions. I argue that the same kind of correspondence assumptions that are made between non-numerical structures and the natural numbers, in the case of Turing machines and personal computers, can be made in the case of genuinely continuous, physical processes and the real numbers.

Key words. Church–Turing thesis, effective procedure, mundane procedure, Turing machine procedure, isomorphism, compute, mirror, follow.

Horsten and Roelants (1995) raise some important questions about my analysis and evaluation of the Church–Turing thesis (Cleland, 1993). In order to bring these questions into sharp focus, however, we need to clear up some basic misunderstandings. These misunderstandings can be encapsulated as follows: (1) none of the versions of the Church–Turing thesis that I address is *the* Church–Turing thesis; (2) the notion of mundane procedure provides a complete analysis of our general concept of procedure; (3) my concept of effectiveness has a built-in component of causality. Once my position on these issues has been clarified, we can focus on the central question, which is whether a physical device, exploiting genuinely continuous, physical quantities and processes, could be said to "compute" a non-algorithmic function in the same sense in which a Turing machine or my personal computer can be said to "compute" an algorithmic function.[1] I contend that the answer is yes and that this is enough to cast the truth of all but the most narrow interpretation of the Church–Turing thesis into serious doubt.

I

Horsten and Roelants intimate that neither of the three versions of the Church–Turing thesis that I discuss is *the* Church–Turing thesis.[2] As I discuss in the introduction to my paper (pp. 283–285), there are a large number of variants of the Church–Turing thesis. In its original form, the thesis was very narrow, being concerned only with determining the capacities of idealized humans to consciously

Minds and Machines **5**: 9–23, 1995.

compute number-theoretic functions. It amounted to the claim that any number-theoretic function that a human being, given an unlimited amount of time and memory, could compute in a step by step fashion could be computed by a Turing machine, and vice versa. In other words, the original version of the thesis did not address computation in general. It had nothing to say about the capacities of continuous devices to compute functions or the computation of functions defined on sets such as the real numbers. Thus, while the original version of the Church–Turing thesis is probably true, its interest is limited. It didn't take long, however, for broader interpretations of the Church–Turing thesis to be proposed. Indeed, Turing conjectured, fairly early on, that the thesis could be extended to machines in general and mental phenomena in general (1964, pp. 20 and 22), and many contemporary thinkers have followed him in construing it as making a very general claim about computation.[3]

In my paper, I discuss three different theses which have been identified with the Church–Turing thesis and are more general than the original thesis:

CT_1 Every effectively computable number-theoretic function is Turing-computable.
CT_2 Every effectively computable function is Turing-computable.
CT_3 Every effective procedure is Turing-computable.

While I sympathize with those who feel that CT_3 ought not to be considered a bona fide version of the Church–Turing thesis, on the grounds that it really isn't a thesis about computation,[4] I do not sympathize with those who would reject CT_1 and CT_2 on the grounds that they are not the original thesis. In the first place, CT_1 and CT_2 are both theses about computation, theses which include the original thesis as a special case. Moreover, Turing, himself, considered something stronger than CT_1, but, perhaps, weaker than CT_2, to be a natural extension of his original thesis; he did not discuss functions in general. Most importantly, however, theses CT_1 and CT_2 reflect a view that many scientists and scholars suspect is true, viz., that Turing machines provide some kind of ultimate limit to the possibilities for computing functions. The interesting question is whether they are right about this, not whether there are historical reasons for thinking that it is a mistake for them to call their conjecture the "Church–Turing thesis." From the point of view of computer science, for example, the really interesting question is whether there are functions, number-theoretic or otherwise, which could be computed by physical devices but not by Turing machines. Similarly, neuroscientists and cognitive scientists wonder whether the computational capacities of biological brains (vs. conscious minds) might not exceed those of Turing machines.[5] In short, to dismiss CT_1 and CT_2 on the grounds that they are not the original version of the Church–Turing thesis is to miss the point and engage in a mere verbal disagreement.

This brings us to the second and third misunderstandings, which are interrelated. Despite what Horsten and Roelants say, I do not consider mundane procedures to provide a more general or complete analysis of our concept of procedure than do Turing machine procedures. Mundane procedures are ordinary, everyday procedures such as recipes for making Hollandaise sauce and methods for starting camp fires; they are methods for manipulating physical things such as eggs and pieces of wood. Turing machine procedures, on the other hand, are methods for "manipulating" abstract symbols. Neither kind of procedure provides a more prototypical example of procedure than the other. For, on my analysis, "... our general concept of procedure is the concept of something to follow, and what one follows, when one follows a procedure, are instructions specifying that certain *kinds* of action be performed in a certain order in time" (p. 297). Both mundane procedures and Turing machine procedures fit this characterization of procedure equally well, which is why I consider them both to be equally good examples of procedure.

Mundane procedures differ, however, from Turing machine procedures in that the action-kinds they specify have causal but not formal consequences; they have causal consequences insofar as the objects manipulated are physical things. In contrast, the actions specified by a Turing machine procedure have formal but not causal consequences, since the objects manipulated are abstract symbols, as opposed to physical things. As a result, no Turing machine procedure can qualify as a mundane procedure, or vice versa. In brief, on my analysis, mundane procedures don't include Turing machine procedures as a special case but, rather, represent a completely different kind of procedure.

Nevertheless, it is possible for a sequence of *concrete* (particular, physical) actions to realize both a Turing machine procedure and a mundane procedure. For a concrete action may involve the manipulation of something which is both a physical object and a symbol, and, hence, may have both causal and formal consequences (pp. 298–299). In other words, it is important to distinguish procedures, which are general and repeatable, from the sequences of concrete actions which instantiate them. Physical devices, whose actions are always concrete, can realize both Turing machine and mundane procedures, but Turing machines (whose actions are not concrete) cannot realize mundane procedures. This is why physical machines can enter the world of mathematics, but Turing machines cannot enter the physical world.

Where the consequences of the action-kinds specified by a procedure become important is in the notion of an effective procedure. Intuitively speaking, a procedure is said to be "effective" if following it consistently results in a certain sort of outcome. The outcome (with respect to which a given procedure is said to be effective) may be either a causal or a formal consequence of the action-kinds specified by the procedure. In the case of an effective mundane procedure such as the recipe for Petit Mont Blanc, the outcome (a pastry) is a causal consequence of the activity of a cook on various physical substances, e.g., sugar, butter, flour. The outcome of a Turing machine procedure, on the other hand, is a formal

consequence of the manipulation of strings of symbols; it is only in virtue of the fact that the strings of symbols on the machine's tape are taken to stand for certain numbers that a Turing machine program for computing averages, for example, can be said to yield an average. That is to say, on my account, a procedure is effective in virtue of invariably having certain consequences (when followed), and these consequences need not be causal. Hence, Horsten and Roelants are wrong in concluding that my notion of effectiveness has a built-in component of causality. The source of their misunderstanding of my position can be traced to either a failure to distinguish a procedure from its implementations (in sequences of concrete actions which have both causal and formal consequences) or a failure to recognize that Turing machine procedures and mundane procedures are fundamentally different kinds of procedure.

One might suspect, from the above characterization of effectiveness, that no procedure which failed to terminate in a final outcome (e.g., a pastry, a number) could qualify as effective. If this were the case, my account would be inadequate, since it is clear that implementations of many Turing machine procedures require the performance of an infinite number of actions, which is impossible, and some mundane procedures, e.g., crop rotation, are non-terminating. As I suggested in my paper, however, it is possible to accommodate such procedures by focusing on the individual steps of a procedure, independently of whether the procedure ever terminates when followed.

My proposal is closely related to Minsky's suggestion that we classify a procedure as effective if "... the next step is always clearly determined in advance" (1967, p. 105). To appreciate this, it is important to distinguish steps from instructions and consequences. As mentioned earlier, a procedure can be viewed as a set of instructions specifying that certain kinds of action be performed in a given order in time. A step is not an instruction. It is a stage in the implementation of a procedure. More specifically, a step is an action-kind (which is specified by an instruction) having a particular position in the sequence of action-kinds specified by a procedure's instruction set. The performance of an action-kind has consequences insofar as it does something to something, e.g., erases a symbol, changes a physical object. In other words, steps are not consequences; they have consequences (when executed).

A step can be said to be clearly determined in advance if the identity and position of its constituent action-kind, in the sequence of action-kinds, is determined in advance of its performance. In the case of Turing machine procedures, this is achieved by using a set of conditional instructions, whereas in the case of mundane procedures, this is often achieved by lists of unconditional instructions. In either case, however, the position of the next step is determined in advance of its performance by the instruction currently being followed, since that instruction specifies, either in virtue of its content or its position, the next instruction to be followed. The next instruction to be followed will secure the identity of the action-kind if it "precisely describes" it.[6] In short, procedures which are "effective" in Minsky's sense predetermine time ordered sequences of

action-kinds, independently of whether the total number of actions is finite, infinite or indefinite.

On my view, however, this isn't enough to get us effectiveness. In addition, each action-kind must consistently have a certain kind of causal or formal consequence (when performed). That is to say, on my proposal, a procedure, whether Turing machine or mundane, can be classified as effective if and only if (i) it meets Minsky's condition and (ii) each of the action-kinds it specifies invariably has (under normal conditions) a certain kind of consequence. Corresponding steps in different implementations of such a procedure will invariably have the same result, regardless of whether the number of steps is finite, infinite or indefinite. Thus, a procedure for computing $f(x) = x^2$, defined over the set of all positive integers, can be said to be effective, even though the computation can never be completed. For at each and every stage of its implementation, the procedure invariably "pairs" those strings of symbols corresponding to the appropriate integers in the domain and range of the function. On the other hand, if the number of steps specified by the procedure is finite, its implementations will terminate in a final outcome, an outcome which may be defined in terms of either the consequence of the final step (e.g., the final string of symbols on the tape, the final chemical reaction as the batter bakes) or a series of consequences, each corresponding to one (or more) of the finitely many steps (e.g., the range of $f(x) = x^2$ restricted to the set of integers $\{1,2,\ldots,100\}$). In this context, it is important to keep in mind that a procedure may be said to be effective for a number of different outcomes (see Cleland, 1993, p. 293). As an example, the same Turing machine procedure may be effective for computing the integer 7 and for printing three consecutive S_1's on an otherwise blank tape.

In summary, on my account, effective mundane procedures and Turing machine procedures both qualify as bona fide effective procedures, and neither is more prototypical of effective procedures than the other; they are fundamentally different in kind. It follows that my notions of effectiveness and procedure are not the same as those analyzed by Church, who was interested only in mathematical procedures. But it is not obvious to me that the concepts I have analyzed are different from those Turing purported to be analyzing. For Turing claimed to be analyzing our general concept of an effective procedure, and that is exactly what I claim to be doing. Indeed, it is my contention that Turing's analysis failed to capture the general notion of an effective procedure, because he ignored the existence of effective mundane procedures. Insofar as my analysis encompasses both Turing machine procedures and effective mundane procedures, I contend that my analysis is a significant improvement over Turing's.

II

We are now in a position to begin addressing the question of whether continuous physical systems (e.g., an object moving at unit speed in space and time) can be interpreted as computing real-valued functions (the identity function on the

reals). As Horsten and Roelants note, Turing machines and electronic, digital computers are said to compute number-theoretic functions in virtue of certain assumptions about their relations to the natural numbers. According to Horsten and Roelants, the assumptions in question are "isomorphism assumptions"; Turing machines and personal computers are characterized as being, in some sense, isomorphic to "the natural number structure". They contend that one cannot make these assumptions in the case of continuous physical systems and the real numbers, and, therefore, that continuous physical systems cannot be interpreted as computing real-valued functions. But is it really the case that isomorphism to the natural number structure is required for interpreting Turing machines and electronic, digital computers as computing number-theoretic functions? In this section, I argue that the answer is no. Moreover, I show that the correspondence assumptions that do seem to be required for interpreting Turing machines and electronic, digital computers as computing number-theoretic functions can be made in the case of continuous physical processes and the real numbers.

Let us begin with the question of whether isomorphism assumptions are required for interpreting Turing machines as computing number-theoretic functions. Isomorphism is a technical mathematical concept which is designed to capture the notion of structural identity. Two mathematical structures are said to be isomorphic if and only if there exists a one-to-one correspondence among their elements which preserves all functions, relations and constants.[8] Unfortunately, however, the "usual" mathematical structure for the Universal Turing Machine is not isomorphic, in the technical mathematical sense, to the usual structure for arithmetic (the positive integers with addition and multiplication). For the usual structure for the Universal Turing Machine does not have the same similarity type as the usual structure for arithmetic, e.g., the former includes three binary functions (the "next place" function, the "next symbol" function, the "next state" function) whereas the latter includes only two binary functions (addition and multiplication).

Perhaps there is an informal, intuitive sense in which the usual structure for the Universal Turing Machine can be said to be isomorphic to the usual structure for arithmetic. After all, one can define on the usual structure for the Universal Turing Machine all the algorithmic, number-theoretic functions that one can define on the usual structure for arithmetic. Nevertheless, there are number-theoretic functions that can be defined on the usual structure for arithmetic which cannot be defined on the usual structure for the Universal Turing Machine, since, as is well known, there are non-algorithmic, number-theoretic functions. Moreover, to any given algorithmic, number-theoretic function, there corresponds more than one Turing machine function; for the Universal Turing Machine can achieve the requisite pairings of finite strings of Turing machine symbols (designating the appropriate pairs of positive integers) by different routes, viz., by erasing and printing symbols in different orders. In other words, despite the fact

that one can define all the algorithmic, number-theoretic functions on it, the usual mathematical structure for the Universal Turing machine is quite different from the usual structure for arithmetic, and, hence, cannot be said to be isomorphic, in even a non-technical sense, to the usual structure for arithmetic. If Turing machines can be said to compute number-theoretic functions, it is not in virtue of isomorphism assumptions about their relations to the usual structure for arithmetic.

This brings us to the correspondence assumptions made in interpreting electronic, digital computers as computing number-theoretic functions. From what has just been said, it should be clear that they don't *have* to include being isomorphic to the structure of the positive integers. Indeed, the relation of electronic, digital computers to the positive integers is typically handled indirectly, through the medium of Turing machines and their presumed relation to the positive integers. As Horsten and Roelants note, an electronic, digital computer is commonly said to *be* a Turing machine. If this is true and if, as everyone admits, Turing machines can compute number-theoretic functions, then electronic, digital computers can compute number-theoretic functions. The question is in what sense can an electronic, digital computer be said to *be* a Turing machine? Subsequently, we will return to the correspondence assumptions required in order to interpret a Turing machine as computing a number-theoretic function.

Turing machines are frequently characterized as abstract mechanisms. They are mechanism-like to the extent that the instructions in their instruction sets are normally expressed by conditional statements, i.e., statements having the same structure as statements expressing causal relations. They are abstract in the sense that structure is all that they have in common with singular causal occurrences. In other words, Turing machines represent formal processes. It is for this reason that Turing machines can be treated as purely mathematical structures, i.e., as entities consisting of only sets, functions, relations, and constants. Viewed from this perspective, it is clear that no physical device can be said to be identical to a Turing machine, and vice versa, since physical machines have both structural and non-structural (physical) properties and Turing machines do not, and, by Leibniz's law, two things can be the same thing only if they have all the same properties in common.

But this doesn't mean that there couldn't be something about a physical device, such as an electronic, digital computer, and a Turing machine which is the same. Indeed, although Turing machines cannot have any of the non-structural properties of physical devices, it seems possible, at least in principle, for a physical device to have the same structural properties as a Turing machine. And this brings us back to the concept of isomorphism. If a physical device had the same structural properties as a Turing machine, it could be construed as isomorphic to a Turing machine. To repeat, this wouldn't be enough to secure the identity of the physical device and the Turing machine, since the physical machine would still have properties which the Turing machine lacked. On the other hand, there

wouldn't be any properties had by the Turing machine (except, perhaps, abstractness) which the physical device lacked. So there would be a sense in which the physical machine could be said to be a Turing machine, but not vice versa. Viewed from this perspective, the "is" involved in the claim that a personal computer is a Turing machine is the "is" of predication, as opposed to the "is" of identity, and, quite frankly, I think that this is the right way to view such claims. Turing machines ought to be construed as very complex, structural properties which could be, at least in principle, instantiated by a physical device. Any physical device which instantiated such a property would be isomorphic to a Turing machine of that sort.

Nevertheless, there are good reasons for supposing that being isomorphic to a Turing machine is not required for the computation of number-theoretic functions. For there are physical machines which are widely acknowledged to compute number-theoretic functions and, yet, are not isomorphic to any Turing machine, namely, analog computers. In other words, although electronic, digital computers may be construed as being isomorphic to Turing machines, it is not necessary that they be so construed in order to make sense of the claim that they compute number-theoretic functions. Moreover, as we saw earlier, isomorphism to the usual structure for arithmetic isn't a necessary condition for interpreting a device as computing a number-theoretic function either; otherwise, Turing machines couldn't be said to compute number-theoretic functions. So what kind of correspondence assumptions *are* necessary for interpreting a physical device or a Turing machine as computing a number-theoretic function?

Adders are computational devices which compute only a single function, viz., addition; they are not interpreted as computing any other function on the positive integers. This suggests that the computation of a number-theoretic function requires only that the device doing the computing be in correspondence with the function concerned. The device need not, as Horsten and Roelants seem to think, be in correspondence with more complex structures, of which the function is a part. The correspondence relation between the device and the function, however, must be very fine-grained, since the device must *uniquely* represent any function that it can be said to compute; no device can be said to compute a function which it can't differentiate from other functions. In this context, I propose that the correspondence condition which a device must meet in order to be interpreted as computing a number-theoretic function is that it *mirror* (represent the detailed structure of) the function. On the set theoretic notion of a function (as a set of ordered pairs), this amounts to claiming that (i) the device includes a set of distinct objects which are in one-to-one correspondence with the set of positive integers in the field (domain and range) of the function and (ii) the device pairs each and every object corresponding to an integer in the domain of the function with the object corresponding to the appropriate integer in the range of the function.[10] It is my contention that this – mirroring the function – is the only correspondence assumption that we need to make in order to interpret a device

(whether a Turing machine, an electronic, digital computer, or an analog computer) as computing a number-theoretic function.

It is clear that Turing machines mirror the number-theoretic functions they are said to compute. For any Turing machine which computes a function is such that there exists (i) a one-to-one correspondence between a set of distinct strings of Turing machine symbols and the set of positive integers in the field of the function and (ii) a sequence of Turing machine operations which transforms (pairs) each and every string of Turing machine symbols, corresponding to an integer in the domain of the function, into the string of Turing machine symbols corresponding to the appropriate integer in the range of the function. It is not so obvious, however, that electronic, digital computers mirror the functions they are said to compute. The problem is that, unlike Turing machines, physical devices eventually break down or wear out. As a consequence, no electronic, digital computer can ever achieve all the pairings of physical structures required in order to mirror a total function on the positive integers.[11] Thus, electronic, digital computers seem to be limited, on the proposed analysis, to the computation of partial functions. But this is contrary to what we ordinarily say about electronic, digital computers. We claim that they can add and subtract, and, technically speaking, addition and subtraction are total functions on the positive integers.

There is more to the structure of a physical device, however, than its actual behavior. To appreciate this, suppose that my Texas Instrument TI-85 calculator is destroyed. Now consider some partial function on the positive integers that it never computed; there will be many, but let's choose one having a very small domain, viz., the set of integers from 0 to 30. Despite the fact that my calculator never computed this function, it remains true that it *could* have computed it. Why? Because the actual behavior of my calculator does not exhaust its capacity for behavior. The behavior of my calculator comes under causal laws which have counterfactual force. It is part of the nature of my calculator that it *would have* achieved all of the requisite pairings of physical structures *if* it had received certain commands (which it did not). The same can be said for total functions on the positive integers. If my calculator never broke down or wore out (and had unlimited memory) it would achieve all of the countably infinite pairings of physical structures needed in order to explicitly mirror the function of addition. In other words, the physical structure of my calculator includes counterfactual, as well as factual, pairings of physical structures. Accordingly, it seems that my calculator can be said to mirror total functions on the positive integers after all.

In conclusion, the only correspondence assumption that seems to be required for interpreting a device, whether concrete or abstract, as computing a number-theoretic function is that the device mirror the function. Moreover, I see no reason to suppose that this result can't be extended to functions other than the number-theoretic functions. The question thus becomes are there devices which can mirror non-algorithmic functions (functions which cannot be computed by any Turing machine)? In the next section, I will argue that there are such devices,

viz., genuinely continuous devices. Whether or not there exist any genuinely continuous, *physical* devices is, of course, an open question. But the mere possibility of such devices is enough to make my point, viz., that it is possible for a device to mirror a non-algorithmic function. Subsequently, we will address the crucial remaining question, namely, is mirroring a function a sufficient condition for computing it?

III

Consider an object moving at unit speed in space and time. Suppose that space and time are both genuinely continuous, then every interval of space and time, no matter how small, consists of an uncountably infinite number of distinct places and times. Further suppose that motion is a genuinely continuous process – a process of occupying different places at different times in such a way that every intervening place and time along the path of motion is, at least, passed through. If these suppositions are true (and they may not be), a moving object will associate an uncountably infinite number of distinct places with an uncountably infinite number of distinct times.

Any object which pairs places and times in the above manner mirrors a function. Admittedly, the function is not a numerical function, but, from a set theoretical point of view, it is still a bona fide function. Moreover, the function in question cannot be computed by any Turing machine, since a set consisting of an uncountably infinite number of places or times can't be placed in one-to-one correspondence with the set of all distinct Turing machine symbols; there aren't enough strings of symbols. In short, regardless of whether it can be said to mirror a real-valued function, the moving object can be said to mirror a non-algorithmic function.

In order for the moving object to mirror a real-valued function, however, there needs to be a one-to-one correspondence between the set of real numbers in the field of the function and the sets of, respectively, places and times "occupied" by the object during its motion. Let us take as our function the identity function on the positive reals. Granting that the object "occupies" an uncountably infinite number of places and times, there will be a one-to-one correspondence between the real numbers in the field of the function and the set of places/times "occupied" by the object. In this context, it is important to keep in mind that it doesn't matter whether any human being could actually match each and every positive real number to a distinct time or place, or, for that matter, given any particular time or place, pick out the corresponding real number. From a mathematical point of view, a one-to-one correspondence is nothing more than an abstract relation among the elements of two sets. In other words, mirroring, as I have defined it, is an abstract, but, nevertheless, objective, relation among structures. It is simply a fact about a physical system, which we may or may not discover, that it mirrors some function. As a consequence, all that is required for a moving object to mirror the identity function on the positive reals is that the above physical assumptions be true, i.e., that space, time and motion be genuinely continuous. Whether or not we can describe the moving object as *computing* the identity function on the positive reals, however, is another question – a question which we shall soon take up.

Horsten and Roelants, however, would object to this conclusion on the grounds that the function being mirrored isn't "uniquely determined."[12] There are many one-to-one correspondences between the set of positive real numbers and a set consisting of an uncountably infinite number of places or times. Depending upon which positive real numbers are paired with which places and times, the moving object will mirror different real-valued functions. But as discussed earlier, no function can be said to mirror (represent the detailed structure of) a function which it doesn't represent as distinct from other functions. Accordingly, it seems that a moving object cannot be said to mirror (let alone compute) the identity function on the positive reals after all.

Insofar as there exist many different one-to-one correspondence relations between the set of positive real numbers and the sets of places and times in question, the moving object mirrors many different real-valued functions. Nevertheless, each mirroring is different, since each one-to-one correspondence is different. Pick out a particular one-to-one correspondence and the moving object will represent a unique real-valued function. That is to say, it is only within the context of a definite one-to-one correspondence, that the moving object can be said to uniquely determine a real-valued function.

Admittedly, there aren't any obvious natural assignments of real numbers to places and times; space and time do not seem to have an absolute origin, for example. Indeed, ". . . we can associate real numbers with places and times in any way we like." But this shouldn't be viewed as posing a special problem for physical systems, since it is also true of Turing machines. Like physical devices, Turing machines do not operate directly on numbers; they transform strings of symbols. Insofar as there are different possible assignments of positive integers to distinct strings of Turing machine symbols, a given Turing machine mirrors different number-theoretic functions. In order to uniquely determine a function computed by a given Turing machine, one must select a particular one-to-one correspondence between the set of positive integers and the set of strings of Turing machine symbols. This is standardly done by convention. It is simply stipulated that the symbol S_0 stands for the integer 0, the symbol S_1 stands for the integer 1, and the right most symbol, in a string of symbols, is always taken as being in the "one's" place (i.e., the first power of the base 2). Similarly, one can use convention to associate real numbers with places and times, and, depending upon what convention one settles upon, the moving object will determine a unique function. In short, an object moving at unit speed in a continuous space and time can be said to uniquely determine the identity function on the positive reals in the same sense that a Turing machine can be said to uniquely determine

the function of addition on the positive integers, viz., in the context of a definite one-to-one correspondence.

It is important to keep in mind here that our inability to provide a unique, discrete (discontinuous) physical representation for each and every real number does not preclude the possibility of establishing a definite one-to-one correspondence between the set of real numbers and a set of continuous physical quantities. Moreover, there aren't any a priori reasons to suppose that there couldn't be a physical device which was sensitive to differences among distinct but continuous physical quantities, despite the fact that it couldn't produce a unique, discrete physical quantity for each real number. Ignoring, for the sake of argument, certain quantum mechanical considerations, one can imagine a continuous physical device which mirrored the characteristic function of the set of irrational numbers by printing a "1" every time it detected the physical quantity corresponding to an irrational number and a "0" every time it detected the physical quantity corresponding to a rational number. Although we couldn't use this machine to uniquely identify or count real numbers, we might, nonetheless, have good reasons (e.g., inference to the best explanation, simplicity) for believing that the machine was mirroring the characteristic function of the irrationals (which is, of course, a non-algorithmic function). Analogously, we have good, but not conclusive, reasons for believing that electronic, digital computers mirror total functions on the positive integers; for as discussed earlier, any physical device will break down or wear out long before it can achieve all of the requisite pairings of integers. In other words, our situation with respect to the question of whether a continuous physical device mirrors a non-algorithmic function is not much different from our situation with respect to the question of whether my TI-85 calculator really does mirror the detailed structure of addition: It depends upon whether the causal processes and physical quantities involved have the right structures.

So this brings us to the pivotal question: Is there a difference between mirroring a function and computing a function? From an intuitive standpoint, it seems that there is. Surely, falling rocks don't compute functions, even supposing that they mirror them. That is to say, there seems to be a difference between a mere representation of a function, no matter how detailed, and the computation of a function. But what could this difference amount to? A natural suggestion is that computation requires not only the mirroring of a function but, also, the *following* of a procedure; falling rocks don't compute functions because they don't follow procedures.

There is a major problem with this proposal, however, and that is that Turing machines are frequently construed as *purely* mathematical objects. They are defined in terms of the same kinds of basic entity (viz., sets, functions, relations and constants) as other mathematical structures. A Turing machine is said to *compute* a number-theoretic function if a function can be *defined* on its mathematical structure which has the same detailed structure as the number-theoretic

function concerned; there isn't a distinction, in Turing machine theory, between computing a function and defining a function. Indeed, Turing machines can be instantiated by static, physical systems, and such systems can no more be said to follow procedures than the structure of the positive integers can be said to follow a procedure. What a Turing machine really does is to represent the detailed structure of a number-theoretic function in a peculiar way, viz., as a sequence of separate transformations of strings of symbols. While such a representation mirrors a procedure (as well as a function), it does not amount to the *following* of a procedure, and that is the crux of our problem. If computing a function presupposes following a procedure, then neither Turing machines nor falling rocks can be said to compute functions.

In this light, one might wonder whether it is a mistake to construe Turing machines as purely mathematical objects. After all, they are characterized as (abstract) mechanisms too. Indeed, perhaps mechanisms, whether physical or abstract, have special representational features which are relevant to the computation of functions but which can't be captured in terms of the notion of a mathematical structure. But, again, what could these features be? They can't amount only to the presence of genuinely dynamic processes, since falling rocks are genuinely dynamic objects. Perhaps, as is often suggested, Turing machines represent a very special kind of mechanism, viz., an idealized human mind, and computing a function amounts to mirroring a function in the way that human beings mirror functions. On this proposal, a function would be computable only insofar as its detailed structure could be consciously reconstructed by an idealized person, viz., in a step-by-step fashion. Thus, a falling rock would fail to qualify as computing a real-valued function on the grounds that no human being could consciously reconstruct such a function in a step-by-step fashion. But why restrict the concept of computation to the impoverished representational capacities of conscious human minds, particularly given that we commonly characterize non-human mechanisms, such as personal computers, as computing functions? Moreover, as Jim Fetzer (1994, pp. 22–24) has recently pointed out, there are kinds of human reasoning (heuristics and asymmetrical decision procedures) which cannot be successfully analyzed as instances of following an algorithmic (Turing machine like) procedure but might, nevertheless, be analyzed as instantiating or as satisfying "procedures", understood a disciplined step satisfaction. Perhaps even human beings sometimes compute functions by performing mental operations that do not, strictly speaking, follow algorithmic procedures?

In conclusion, it isn't obvious that there is a significant distinction between mirroring a function and computing a function. If there is, then Turing machines either don't compute functions or are not purely mathematical structures. In either case, we need a new analysis of computation, and it isn't at all obvious what form it will take. On the other hand, if there isn't a distinction, then the limits to physical computation are set by the physical structure of the world, and either the first or the second version of the Church–Turing thesis (CT_1 or CT_2)

could turn out to be, in fact, false. For if there are genuinely continuous causal processes and physical quantities, then, Horsten's and Roelant's arguments notwithstanding, physical systems compute non-algorithmic, real-valued functions. Similarly, there may be physical systems which compute non-algorithmic, number-theoretic functions. Whether we could ever discover that such physical systems exist, let alone exploit them, is another question, a question which is, alas, beyond the scope of this paper. In any event, however, my point remains, and that is, barring a significant distinction between mirroring a function and computing a function, the truth of CT_1 and CT_2 depends upon the physical (vs. logical) structure of the world.[13]

Notes

[1] In order to avoid confusion over functions which are computable in Turing's sense and functions which are computable in some other (not Turing's) sense, I will reserve the term "algorithmic function" for a function which is computable in Turning's sense. Thus, I won't be caught in the seemingly contradictory position of speaking of computable, uncomputable functions.

[2] This objection has been advanced by a number of other people too, e.g., David Israel (in a personal communication).

[3] See the introduction to my paper (1993, pp. 283–285) for further details and references.

[4] Nevertheless, it has been advanced in the literature as a version of the Church–Turing thesis (see my 1993 paper, pp. 284–285, for details), which is why I take it under serious consideration. In this context, it is important to keep in mind that CT_3 is a thesis about procedures, albeit not computational procedures. (I should, perhaps, note here that Horsten and Roelants do not explicitly object to CT_3 being identified as a version of the Church–Turing thesis, although a number of other readers have done so.)

[5] In this context, it is worth noting that some neuroscientists and cognitive scientists have a tendency to confuse *computing* with *causally producing*, thus conflating CT_2 with CT_3. More specifically, they speak of the brain as computing sensations, consciousness, or intuitions, mental phenomena that seem very different from mathematical functions.

[6] As I discussed in my paper (1993, pp. 291–293), the notion of an instruction *precisely describing* a given action-kind is problematic, since no mere description of what to do can (just by itself) exclude the possibility of error in the case of either a human follower or a device. Indeed, as I argue, the notion of an instruction precisely describing an action-kind must be relativized to an idealized follower, such as a Turing machine, which, by definition, cannot fail to print an S_0 when asked to print an S_0, or an idealized chef, who by definition, understands precisely what to do when asked to cream the butter. The upshot is that there is no reason to think that the action-kinds specified by a mundane procedure are less precisely described than those specified by a Turing machine procedure; the instructions of both need to be relativized to idealized followers. The problem with mundane procedures is that we don't have a theory of idealized followers, e.g., chefs (in the case of recipes), mechanics (in the case of instructions for rebuilding an engine), travelers (in the case of geographic directions).

[7] Technically speaking, all Turing machine procedures are effective, but not, of course, for any old outcome. For each and every stage of implementation of a given Turing machine procedure will always result in the same string of symbols on the tape, and, hence, the procedure can be said to be effective for producing a certain series of strings of symbol.

[8] The correspondence is actually one-one and onto, but, for the sake of simplicity of exposition, I will follow mathematical tradition and, misleadingly, refer to the correspondence as a "one-to-one correspondence."

[9] I am assuming here that the structure on the natural numbers that Horsten and Roelants have in mind by "the natural number structure" is the usual structure for arithmetic. Also, it is important to

note that, technically speaking, there isn't a so-called "usual" structure for the Universal Turing Machine, as there is for arithmetic. Different mathematicians and computer scientists define the structure for the Universal Turing Machine in different ways, viz., in terms of different functions and relations, according to the problem at hand. Moreover, many mathematicians and computer scientists provide only a piece of the structure, assuming that the rest could be filled out in an obvious way. Almost all of the proposed structures for the Universal Turing Machine, however, have in common the assumption that they ultimately "include" the same set of functions and relations, i.e., the union of the set of functions and relations in terms of which the structure is defined and the set of functions and relations which can be defined on the structure are the same for different structures for the Universal Turing Machine. So, for simplicity of exposition, I have elected to call one particular structure for the Universal Turing Machine the "usual" structure, viz., the structure consisting of (in addition to a finite set of states, a finite set of symbols, and the initial state) three binary functions, (the "next-place" function, the "next symbol" function and the "next state" function). Some structures for Turing machines combine these functions into fewer functions; see, for example, Lewis & Papadimitriou (1981, p. 170), who define the structure for the Universal Turing Machine in terms of a single (next state-symbol-place) function.

[10] In conversation, Horsten has informed me that this is really all that he had in mind when he spoke of isomorphism, even though he referred to the "natural number structure." While I agree that any structure which mirrors a function can be construed as isomorphic to a structure consisting of the set of elements in the field of the function and the function itself, I prefer to use the term "mirroring", since the term "isomorphism" is usually reserved, in mathematics, for more complex structures.

[11] I owe this objection to mathematician Jerry Malitz, who adamantly insists that no physical device can be said to "compute," in the strict, mathematical sense of the term, *any* total function on the positive integers.

[12] Horsten and Roelants (1995) articulate this objection in the context of the claim that a moving object computes the identity function on the positive reals. But, on my view, what they were really objecting to is the claim that a moving object mirrors the function.

[13] I would like to than Jerry Malitz and Jan Mycielski, of CU's Mathematics Department, for a number of very enjoyable and helpful discussions. Any mistakes and all outrageous claims are, of course, my own!

References

Cleland, Carol E. (1993), 'Is the Church–Thesis True?', *Minds and Machines* 3, pp. 283–312.

Fetzer, James H. (1994), 'Mental Algorithms: Are Minds Computational Systems?', *Pragmatics & Cognition* 21(1), pp. 1–29.

Horsten, Leon & Roelants, Herman (1995), 'The Church–Turing Thesis and Effective Mundane Procedures', *Minds and Machines* 5, pp. 1–8 (this issue).

Lewis, Harry R. & Papadimitrious, Christos H. (1981), *Elements of the Theory of Computation*, Englewood Cliffs: Prentice-Hall.

Minsky, Marvin (1967), *Computation: Finite and Infinite Machines*, Englewood Cliffs: Prentice-Hall.

Turing, Alan (1964), 'Computing Machinery and Intelligence', in A. Anderson, ed., *Minds and Machines*, New Jersey: Prentice Hall, pp. 4–30.

Turning, Alan (1965), 'Systems of Logic Based on Ordinals', in M. Davis, ed., *The Undecidable*, New York: Raven Press, pp. 155–222.