# Do Accelerating Turing Machines Compute the Uncomputable?

B. Jack Copeland and Oron Shagrir

University of Canterbury, New Zealand
Hebrew University of Jerusalem, Israel

## ABSTRACT

Accelerating Turing machines have attracted much attention in the last decade or so. They have been described as "the work-horse of hypercomputation" (Potgieter and Rosinger 2009). But do they really compute beyond the "Turing limit"—e.g., compute the halting function? We argue that the answer depends on what you mean by an accelerating Turing machine, on what you mean by computation, and even on what you mean by a Turing machine. We show first that in the current literature the term "accelerating Turing machine" is used to refer to two very different species of accelerating machine, which we call *end-stage-in* and *end-stage-out* machines, respectively. We argue that end-stage-in accelerating machines are not Turing machines at all. We then present two differing conceptions of computation, the *internal* and the *external*, and introduce the notion of the *epistemic embedding* of a computation. We argue that no accelerating Turing machine computes the halting function in the internal sense. Finally, we distinguish between two very different conceptions of the Turing machine, the *purist* conception and the *realist* conception; and we argue that Turing himself was no subscriber to the purist conception. We conclude that under the realist conception, but not under the purist conception, an accelerating Turing machine is able to compute the halting function in the external sense. We adopt a relatively informal approach throughout, since we take the key issues to be philosophical rather than mathematical.

## KEYWORDS

Accelerating Turing machine; supertask; halting problem; ATM paradox; hypercomputation; external and internal computation; epistemic embedding; ontology of computing; Turing-machine purism; Turing-machine realism; Thompson lamp paradox

## *1. The Rough Guide to Accelerating Turing Machines*

In 1927, Hermann Weyl considered a machine (of unspecified architecture) that is capable of completing

> an infinite sequence of distinct acts of decision within a finite time; say, by supplying the first result after 1/2 minute, the second after another 1/4 minute, the third 1/8 minute later than the second, etc. In this way it would be possible ... to achieve a traversal of all natural numbers and thereby a sure yes-or-no decision regarding any existential question about natural numbers.
>
> (Weyl 1927: 34; English translation from Weyl 1949: 42)

It seems that this temporal patterning was first described by Bertrand Russell, in a discussion of Zeno's paradox of the race-course (during a lecture given in Boston in 1914). Russell said, "If half the course takes half a minute, and the next quarter takes a quarter of a minute, and so on, the whole course will take a minute" (Russell 1915: 172-3). Ralph Blake described the same idea in 1926: "If, e.g., the first act ... takes 1/2 second, the next 1/4 second, etc., the [process] will ... be accomplished in precisely one second" (1926: 651). Unlike Weyl, however, neither Russell nor Blake linked the idea specifically to the operation of a machine or to the mechanical solution of mathematical problems.

An *accelerating* Turing machine (Copeland 1998a, 1998b) is a Turing machine that operates in accordance with the Russell-Blake-Weyl temporal patterning. Accelerating Turing machines (ATMs) perform the second atomic operation called for by the program in half the time taken to perform the first, the third in half the time taken to perform the second, and so on. Let the time taken to perform the first atomic operation called for by the program be one "moment". Since

$$1/2 + 1/4 + 1/8 + ... + 1/2^n + 1/2^{n+1} + ...$$

is less than 1, an accelerating Turing machine can perform infinitely many atomic operations before two moments of operating time have elapsed.

In 1991 Stewart gave a cameo discussion of accelerating Turing machines, asking us to "imagine a Turing machine with a tape that accelerates so rapidly that it can complete an infinite number of operations in one second" (1991: 664). Stewart remarked that an ATM can decide the predicate calculus. The concept is also implicit in Boolos and Jeffrey (1980), who envisaged Zeus attacking problems in mathematical logic by enumerating infinite sets "in one second by writing out an infinite list faster and faster" (1980: 14). The relativistic machines of Hogarth (1992,

1994) are also related to ATMs.[i] The term "accelerating Turing machine" was introduced by Copeland in 1997.[ii]

The interest of accelerating Turing machines lies in their logical power. A well-known result of Turing's entails that no Turing machine can "solve the halting problem" in a finite number of steps.[iii] Yet it has been argued (Copeland 1998a, 1998b, 2002a) that an accelerating Turing machine solves the halting problem in a finite amount of time (notwithstanding the fact that an accelerating Turing machine *is* a Turing machine). In the remainder of this section, we illustrate this claim by detailing an accelerating Turing machine H that solves the halting problem. (H was first described in Copeland 1998a.) [Note to copy editor: please use cursive script for ' H'.]

Using a suitable encoding convention, the instructions of any given Turing machine can be represented by means of a single binary number; call this the *program number* of the machine. Before a Turing machine is set in motion, some sequence of binary digits may be inscribed on its tape (the initial data); call this binary number the machine's *data number*. (The data number of an initially blank tape is zero.) To be able to solve the halting problem is to be able to state—correctly—when presented with any program number and data number, whether or not the Turing machine with that program number will halt if set in motion with that initial data placed on its tape.

H is a form of universal Turing machine. Let t be any Turing machine. H, being a universal machine, will simulate t if given an appropriate description $D_t$ of t. $D_t$ is the number that is formed by writing out the digits of t's program number p followed by the digits of t's data number d. If H is set in motion with $D_t$ on its tape, H will simulate t, performing every operation that t does, in the same order as t (although interspersed with sequences of operations not performed by t). Since H is an accelerating machine, the simulation is completed before two moments of operating time have elapsed, even in the case where t never halts. To make it the case that H solves the halting problem, it suffices to provide H with a means of indicating whether or not t halted. A particular square of the tape, the "designated square", is used for this purpose.

The designated square is (for the sake of definiteness) the first square to the left of the block of digits comprising $D_t$. Once H is set in motion, its first action is to position the scanner over the designated square and print 0 (meaning "t does not

halt"—although H may subsequently revise this statement). Next H carries out its simulation of t. If H discovers that t halts, then H returns to the designated square and changes the "0" written there to "1" (meaning "t halts"). The design of the program ensures that this is the only circumstance in which H ever returns to the designated square. The contents of the designated square shift from "0" to "1" if and only if t halts.

*2. A Fork in the Road*

The type of accelerating machine described in Section 1 must be sharply distinguished from a second type of accelerating machine, also known in the literature as an "accelerating Turing machine". It is clearly a potential source of confusion that the same label is being used in the literature to refer to two very different kinds of accelerating machine. The difference between the two types of machine is simple to describe, and turns on whether an end-of-second-moment stage is or is not included in the specification of the machine.[iv] One might speak of "Type A" and "Type B" accelerating Turing machines. However, we prefer to reserve the label "accelerating Turing machine" for the type of machine described in Section 1, since, as we shall argue, machines of the second type are not Turing machines at all. Machines of the second type are accelerating *non*-Turing machines.

We call a machine whose specification includes an end-of-second-moment stage an *end-stage-in machine* and a machine whose specification does not include such a stage an *end-stage-out machine*. It is the end-stage-out form of accelerating machine that is presented in Section 1, and this is the form of accelerating machine that was presented in Copeland (1998a, 1998b, 2002a). According to the end-stage-out conception, the condition of the designated output square at the end of the second moment—whatever it is—is *not* part of the specification of the ATM. The end-stage-out ATM is a standard Turing machine that operates in accordance with a certain temporal patterning.

H, in particular, was constructed in such a way that the configuration of the machine is specified after one moment, after one-and-a-half moments, after one-and-three-quarter moments, and, in general, after $2 - \frac{1}{2}^{n-1}$ moments (where n is the $n^{th}$ atomic step). (The configuration at a given stage of the computation is a triple consisting of the state of the machine at that stage, the total contents of the tape at that

stage, and the content of the square under the scanner at that stage.[v]) Thus it is specified what the configuration of H will be at points during the semi-open time segment [0,2). But nothing above and beyond that is specified. This specification of what H is doing in the time segment [0,2) is *exhaustive*, in the sense that it constitutes a complete description of the machine. The specification simply does not refer to the configuration of H (if there is one at all) at the end of the second moment. In particular, the specification does not refer to the condition of the designated output square at that moment. In this respect H is analogous to Thomson's famous lamp (Thomson 1954). "The lamp is on at the end of the second moment" and "The lamp is off at the end of the second moment" are both logically consistent with the statement that the lamp has performed its super-task, namely to turn off after one moment, to turn on again after a further 1/2 moment, to turn off again after another 1/4 moment, and so on (Benacerraf 1962; Thomson 1970).[vi]

We are not of course denying that an accelerating machine might *reach* the end of the second moment. H's specification might be physically instantiated by an actual machine whose output square survives the acceleration; or by an actual machine whose output square simply vanishes at the end of the second moment. Both machines are instantiations of H in the sense that they act in accordance with the (exhaustive) specification of H's behaviour in the semi-open time segment [0,2). Whatever each one of these machines actually does at the end of the second moment is dictated by the laws of nature and the specific physical circumstances of the machine. Our point is just that the configuration of each instantiation of H at the end of the second moment—whatever it is—is neither part of, nor entailed by, H's specification. In particular, the appearance of any character in the output square at the end of the second moment ("0", "1", or anything else), or the appearance of no character at all, is consistent with the specification of H.

According to the end-stage-in conception, on the other hand, the value at the output square at the end of the second moment *is* a matter covered by the specification of the machine. This conception of an accelerating Turing machine is stated explicitly, perhaps for the first time, by Steinhart: "An ATM is a CTM [Classical Turing Machine] with an adjoined limit state" (Steinhart 2002: 272). This end-stage-in conception is also found in Calude and Staiger (2010), Fearnley (2009), Potgieter and Rosinger (2009), and Svozil (2009), each of whom provide a precise

definition of the output of the machine at the limit stage (see also Hamkins and Lewis 2000, and Steinhart 2002: 272-273).[vii]

We will use "ATM" to signify the original, end-stage-out conception of the accelerating Turing machine, and "ATM$^+$" to signify the more recent end-stage-in conception.

There are different ways of specifying the output of a machine of the ATM$^+$ type, but predominantly these resort—conceptually at least—to the idea of a limit state. Thus Hamkins, for example, suggests that when the machine is at a limit state, the value at the designated output square is the limit of previous values that this square has displayed during the computation. To make the discussion concrete, we introduce a machine H$^+$ that results from adding a limit state to the end-stage-out machine H. During the semi-open time segment [0,2) the machine operates in the classical way, in the sense that "the classical procedure determines the configuration of the machine … at any stage $\alpha + 1$, given the configuration at any stage $\alpha$" (Hamkins 2002: 526). At the end-of-second-moment stage, H$^+$ "is placed in the special *limit* state, just another of the finitely many states; and the values in the cells of the tapes are updated by computing a kind of limit of the previous values that cell has displayed" (ibid.). In particular, the value in the designated output square of H$^+$ is calculated as the limit of the previous values that the square has displayed. If the square displayed the value "0" in all the (infinitely many) stages that preceded the end-of-second-moment stage, then the value at the limit-state is set to "0".

To summarize: end-stage-out accelerating machines such as H are Turing machines, while end-stage-in accelerating machines such as H$^+$ are not, as we shall argue in the next section.

### *3. Why the end-stage-in accelerating machines are not Turing machines*

It is easy to see that H$^+$ computes the halting function. At the end of the second moment, the designated output square displays a representation of the halting state of the target Turing machine *t*. The value shown in the output square is "1" if, at some point before the end of the second moment, the "0" that was written in the square at the start of operations gets replaced by "1" (and then H$^+$ halts). The value in the output square at the end of the second moment is "0" just in case the original "0" never gets overwritten in the course of the computation.

$H^+$ is yet another important and interesting instance of a *hypercomputer*.[viii] However, neither $H^+$, nor (for the identical reason) any machine of the $ATM^+$ type, is a Turing machine. $H^+$ does not have the *computational structure* of a Turing machine. The end-of-second-moment limit stage is not and cannot be part of the specification of a Turing machine. What a Turing machine does at each stage is completely determined by the configuration at this stage (and, thus, the configuration of the machine at each stage $\alpha + 1$ is completely determined by the configuration of the machine at the preceding stage $\alpha$). Turing is explicit about this constraint.[ix]

> The possible behaviour of the machine at any moment is determined by the *m*-configuration $q_n$ and the scanned symbol $\xi(r)$. This pair $q_n$, $\xi(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine. … [A]t each stage the motion of a machine … is *completely* determined by the configuration. [Note to copy editor: please use upper-case Gothic script 'S' in place of 'ξ'.]
>
> (Turing 1936: 59-60)

Clearly, the behaviour of $H^+$ at the end-of-second-moment stage is not completely determined by the configuration. Nor is the configuration at the limit stage determined by the configuration at *the preceding stage*. This is because the end-of-second-moment stage has no unique predecessor. There are infinitely many stages between any given stage preceding the end of the second moment and the end-of-second-moment stage. At each stage other than the limit stage, the configuration of the machine is determined by the configuration at the preceding stage, but this is not so for the limit stage itself. Therefore $H^+$ is not a Turing machine.

This point about computational structure is important for another reason also. It shows that the (hyper-) computational power of $H^+$ essentially has *nothing to do with acceleration*. The end-stage-in maneuver leads to hypercomputational power and acceleration is unnecessary if end-stage-in machines are under consideration. Machines resulting from the addition of a limit state to a *non-accelerating* Turing machine have the same (hyper-) computational power as $H^+$. (Although acceleration may be useful for *epistemic* reasons: accelerating machines deliver their results in a finite amount of time, an essential feature for mortal users.) The so-called infinite-time Turing machines of Hamkins and Lewis are end-stage-in non-accelerating machines that, by operating in transfinite time, compute the halting function (Hamkins and Lewis 2000, Hamkins 2002). (We note, however, that infinite-time "Turing machines" are *not* Turing machines, for the same reason that $H^+$ is not a

Turing machine.) Moreover, a non-accelerating machine similar to $H^+$ can be described using only the idea of an ordinal limit stage (Cohen and Gold 1978); the description of this machine makes no reference to *duration* at all.

To summarize: machines of the $ATM^+$ type do not have the computational structure of a Turing machine; and their (hyper-) computational power does not derive from acceleration.[x] In the remainder of this paper we discuss end-stage-out machines.

*4. Turing machines cannot exceed the computational power of Turing machines!*

If, as we have argued, H is a Turing machine fair and square, then the claim that H computes the halting function is paradoxical.[xi] To express the paradox as starkly as possible: H computes the halting function, but H is a Turing machine, and there is a theorem (the "halting theorem") saying that no Turing machine computes the halting function. How is this paradox—which we call the *ATM paradox*—to be resolved? This section and the next canvass some different options.

First, one might deny that an ATM is a Turing machine. This does not seem promising. However, Fraser and Akl take this route. They say,

> [T]he accelerating machine cannot be considered a Turing machine, since the Turing machines operate on discrete time intervals. Accelerating machines do not.
>
> (Fraser and Akl 2008: 84)

We agree that the accelerating machine, operating as it does in rapidly decreasing time periods, differs from the examples of computing machines that Turing appears to have had in mind; but this does not show that an ATM is not a Turing machine. Moreover, the fact that the instruction time of an accelerating machine is of ever decreasing—but always discrete—duration clearly does not imply that an accelerating machine is not a discrete machine. As Turing emphasized, the leading characteristic of a discrete machine is that "it is natural to describe its possible states as a discrete set" (Turing 1948: 412). Accelerating Turing machines fall into the class of what Turing called discrete state machines—machines that "move by sudden jumps or clicks from one quite definite state to another" (Turing 1950: 446). In fact, the notion of a Turing machine is so useful precisely because the characterization of a

computation step does not refer to any specific temporal duration (more on this in Section 7).

A second route to a resolution of the ATM paradox is to argue that H does *not* compute the halting function, and that in general ATMs do not exhibit hypercomputational power.[xii] The argument is as follows. According to the textbook definition of Turing-machine computation, a Turing machine computes the (defined) values of a function *f* just in case whenever the machine starts with an input *x* it reaches an end-state ("halt state") with the value *f*(*x*) displayed in the output square(s).[xiii] If the machine does not reach an end-state then *f*(*x*) is undefined for the input *x*. In the case of the halting function, the Turing machine (starting with an input $D_t$) should reach an end-state and display "1" if the simulated machine t halts on input d, or reach an end-state and display "0" if t does not halt on input d. H, however, does not satisfy this definition.

This is because H computes *endlessly* in the case where t never halts, in the sense that each computation step is followed by infinitely many other computation steps. There is no moment during the computation at which H reaches an end-state and displays the value "0". It is true that, thanks to acceleration, the computation is not endless *in time*. H genuinely does perform a supertask: H completes the infinitely many computation steps within two moments of time. But the point to be emphasized, according to this approach to resolving the paradox, is that H's acceleration makes no *computational* difference. The only time at which we can appropriately look for the result "0" is at the end of the second moment and this lies outside the Turing machine's specification. Thus H performs exactly the task performed by its non-accelerating counterpart. H and its non-accelerating counterpart both return "1" if the simulated machine t halts and return no value if t never halts. The only difference is that H performs this task in a finite span of time.

To summarize: according to the second way of dealing with the paradox, H does *not* compute the halting function, although H does perform a supertask.[xiv]

### *5. The third way: internal versus external computation*

It may nevertheless seem that there *is* some *computationally* relevant difference between H and its non-accelerating counterpart. The third approach to resolving the ATM paradox highlights two differing conceptions of computation.[xv] Relative to the

first conception, there is no computational difference between H and its non-accelerating counterpart, but relative to the second, there is. These two conceptions of computation are termed *computation in the internal sense* and *computation in the external sense*, respectively.[xvi] It is the internal conception of computation that underlies the textbook definition of Turing-machine computability set out above. To put matters in a nutshell, the halting theorem states that no Turing machine is able to compute the halting function in the *internal* sense, whereas H computes the halting function in the *external* sense. No contradiction.

Generalizing the textbook definition set out above, a function *f* is computable by a machine in the *internal* sense just in case the machine is able to produce *f*(*n*) for any argument *n* in the domain, indicating that the value *f*(*n*) has been produced (e.g. has been printed in the output square) either by halting once the value is printed, or by some other means. For example, rather than entering an end-state, the machine may print a special symbol immediately following the square in which it prints the value *f*(*n*). This was in fact Turing's procedure in "On Computable Numbers": the Turing machine calculates *f*(1) and prints it, then prints the special symbol, then calculates *f*(2) and prints it, followed by the special symbol, and so on (1936: 79-80). In this case the Turing machine never reaches an end-state.

When computing in the internal sense, the only restriction on methods that may be used to indicate that the value has been produced is this: the method must involve no appeal to the behaviour of some device or system that is *external* to the specification of the machine—such as a clock.

When computing in the *external* sense, that restriction is not in play. A function *f* is computable by a machine in the external sense just in case the machine is able to produce *f*(*n*) (for any argument *n* in the domain) by performing, or failing to perform, some pre-specified action during a pre-specified time-interval (open or closed) that is delimited by reference to the activity of some entity external to the machine (i.e. lying outside the specification of the machine), typically but not necessarily a clock.[xvii] For example, the designer of a neural network may state: "The output arrangements are like this. If *n* is presented as input, *f*(*n*) = 1 if and only if *that* specific neuron—the output neuron—fires before *t* moments of time have elapsed; and otherwise *f*(*n*) = 0." This network computes *f* in the external sense. The network may never "halt" (a network "halts" or stabilizes if and only if there is eventually no

further change in the activity level of any of its units); and the activity of the output node at times outside the specified period may afford no clue to the value of *f*. Computation in the internal sense is quite different: the machine is required simply to produce *f*(*n*) and mark it, no matter *when* it does it, so long as it does it in a finite number of steps.

It is in the external sense that H computes the halting function. The value of the function is 1 if and only if there is a shift in the contents of the output square prior to the end of the second moment; and the value of the function is "0" if and only if there is no shift prior to the end of the second moment.[xviii] Appeal to an external device is essential in order to quantify and delimit the relevant duration.

*6. Epistemological considerations: accessing the result*

Either H's scanner replaces the "0" in the output square with "1" before the two moments have elapsed, or it does not. Given the specification of H, this suffices for the truth of the statement that H computes the function in the external sense. How to make the result of H's computation accessible to a user is another—epistemic—matter.

The same question arises even in the case of an ordinary Turing machine that computes the values of a function in the way set out in the textbook definition (Section 4).[xix] Normally we envisage no difficulty in introducing an observer or user who has access to the results of the computation. Perhaps the observer reads off the answer to the problem by pulling the tape out of the scanner once the Turing machine has halted, and identifying the digit printed on the square that was last in the scanner. But since the Turing machine's instruction table entails nothing about the state of the tape once the computation has halted, difficulties are not hard to imagine. *Anything* that happens once the Turing machine has halted is consistent with the Turing machine's specification. Let's suppose that in the very instant of the scanner's halting, all marks on the tape disappear, so frustrating an observer's attempt to read the output. Alternatively, we may imagine that the information does persist on the tape, but that the physics of the situation is such that any attempt to transmit the information to an observer will result in a blue shift that destroys the observer before the signal can be read.[xx]

The "epistemic embedding" of a computation—the embedding of the computation into a context in which an observer is able to access the result—will typically require some propositions to be true that are not entailed by the specification of the Turing machine (or other computing device) which is being embedded. We usually assume that the ink on the machine's tape persists, so that the observer can read the output once the scanner has halted; or we assume that the geometry of the scanner and the relevant physics of the situation are such that the observer can identify the output *while* it is being printed. But such assumptions lie beyond the specification of the Turing machine itself.

In the case of H, one way to supply an epistemic embedding is to require that whatever ink mark H leaves in the output square persists at the end of the second moment and beyond; and another is to require that the physics of the scenario (including the physics of the observer) is such that the unaided observer is able to tell, as the computation progresses, whether or not H's scanner replaces the "0" in the output square with "1". As in the case of the standard Turing machine, this is to require the truth of propositions that are not entailed by the specification of H.

Alternatively, the embedding may involve the introduction of some additional hardware, such as a hooter.[xxi] Logically, the hooter is not part of the Turing machine; like non-fading ink, it is something additional, used for the epistemic embedding of H. Wiring is arranged so that there is a hoot if and only if H returns to the output square and prints "1". (This might be done by using one of H's internal states to trip the hooter.) So if the observer hears a hoot before two moments have elapsed, the value of the function is 1, and if no hoot comes the value is 0.

To summarize: computation is one thing, the epistemic embedding of computation another. As with the standard Turing machine, there are different ways of embedding H into a context in which an observer is able to access the result of the computation.

## *7. Turing-machine purism*

The distinction between internal and external computation raises interesting questions about the nature of Turing machines. In the final two sections of this paper we present two very different answers to the question "What is a Turing machine?". This section presents what we call Turing-machine *purism* and the next what we call Turing-

machine *realism*. These different views of Turing machines have different implications in regard to the claim that H computes in the external sense. We will argue that, from the purist perspective, even if H computes the halting function in the external sense, it does not compute it *qua* Turing machine. According to purism, computation in the external sense is (like computation by end-stage-in machines) just another example of computation by *non*-Turing machines.

Turing-machine purism is widespread in theoretical computer science. According to purism, a Turing machine is a *purely mathematical object*: all its properties are abstract (including the tape, scanner, and tape contents).[xxii] Purism distinguishes sharply between the Turing machine, a mathematical object, and any physical realization of it. Naturally, a physical realization of a Turing machine has physical properties; and in fact a Turing machine can be realized by devices whose physical properties are very different from each other. But the point is that when we reason about the computational power of a Turing machine, then we must refer only to the abstract properties—which are the only kind of properties that the Turing machine has—and not to any physical properties belonging to the realization-level.

The purist has strong views about temporality in connection with Turing machines. The purist need not deny that a Turing machine can usefully be viewed as operating against some background temporal framework; this might be abstract time (as ultra-purists claim) or physical time. But the purist insists that the specification of a Turing machine is *time-neutral* in that the specification makes no essential (i.e. ineliminable) reference to any background temporal framework.

Purism may be regarded as an ontologically-driven programme in the philosophy of computing, like nominalism in the philosophy of mathematics and the Quinean "flight from intension" in the philosophy of language and philosophical logic.[xxiii] Consistently, the purist will strive to find time-neutral definitions of all forms of computing device: analogue computers, neural networks, digital logic circuits (the intuitive presentation of which is rich with references to the execution time of instructions, temporal delays, clocks, the simultaneous application of signals, etc), parallel computers (including synchronous and asynchronous networks of processors), and so on.

The purist emphasizes that the specification of a Turing machine is completely neutral about the *duration* of each computation step. (The purist points out that this is

perfectly consistent with the execution of the same computation step's taking longer when we change hardware—realization—and also with the fact that the execution of some atomic computation steps takes longer than the execution of others in the same hardware.) Consequently, all essential concepts associated with a Turing machine are time-neutral too. In particular, *halting* and *non-halting* are time-neutral: halting is the attainment of an end-state, and if the machine does not halt then the computation process consists of infinitely many steps. *How long* it takes to execute these infinitely many steps—a finite amount of time or an infinite amount—makes no difference at all to the specification of the Turing machine.

According to the purist, when we talk about a *moment* in connection with the activity of Turing machines (as in Turing's statement, quoted earlier, that the behaviour of the machine at every moment is determined by the *m*-configuration and the scanned symbol), we are really referring to a *stage* of the machine's behaviour (as Turing himself does later in the same quotation), and stages, the purist holds, are time-neutral. Moments in the purist's sense are not to be confused with *durations*: each moment (each stage) has a certain duration in any realization of the machine, and the purist may also allow that a moment (stage) has a duration against the background temporal framework, but the notion of duration is absent from the purist's specification of the Turing machine.

When it is said that the accelerating machine requires "at most two moments" to pass through the infinitely many stages of the computation, the term *moment* obviously means something very different. Here *moment* does refer to the background temporal framework (or to the physical time of the notional realizing hardware). Nevertheless, the only concept that matters when defining computation by a *Turing machine* is (for the purist) the number of atomic computational steps, not the number of moments in this temporal sense.

Having said all this, we can turn to the internal–external distinction. The purist argues that even if H computes the halting function in the external sense, it does not compute it *qua* Turing machine. The reason is now apparent. The purist thinks that the reference to an external clock, whether physical or abstract, is illicit. The system described is not a Turing machine; and so, a fortiori, no example has been given of a *Turing machine* computing a function in the external sense. No temporal reference is permitted in specifying *computes* as a predicate of Turing machines (nor of any other

computing device, according to the thoroughgoing purist). For the purist, all that is relevant to what is computed is the *number of atomic steps* of which the process consists, not the duration of time that it takes the machine to complete the process.

From the purist's point of view, the inclusion of a reference to a number of temporal moments (two in this case) is not much different from changing the computational structure by adding a limit-state. This is the dilemma posed by the purist: if H computes the halting function in the external sense, then H is not a Turing machine.

It is worth noting that the purist can introduce a *purist* version of the internal–external distinction. The purist is willing to agree that the Turing machine can proceed by means of displaying the value (of the function that is being computed) at a designated location *before t moments have elapsed*. The purist insists, however, that "before *t* moments have elapsed" just means "before *t* atomic steps have been carried out". For instance, the purist may agree that, in the foregoing example of a neural network, the network does compute *f* in the external sense, since $f(n) = 1$ if and only if the output neuron fires before *t* moments have elapsed (following the presentation of the input *n*); but the purist points out, again, that here "moment" means "stage", or more precisely "iteration" in the case of a neural network.

Does H compute the halting function in the *purist*-external sense? The answer is negative. This is because, from the purist point of view, external computation by a Turing machine is reducible to internal computation by a Turing machine: if a Turing machine computes a function in the purist-external sense, then there is a different Turing machine that computes exactly the same function in the internal sense. To see this, suppose that the Turing machine's output square (or the output neuron in the case of the network) is to be examined after *m* computation stages. There is another Turing machine that simulates the operations of the target machine (i.e. the network or the original Turing machine). This simulating machine uses a counter for tracking the number of simulated atomic operations (this counter is just a very simple sub-routine). The simulating machine also has a sub-routine that instructs it to halt when the counter reaches *m* (if it has not halted before). The simulating machine computes in the internal sense exactly what the target machine computes in the purist-external sense. The upshot is that H does not compute the halting function in the purist-

external sense; if it did, there would be another Turing machine that computes the halting function in the internal sense, in contradiction to the halting theorem.

To summarize: if Turing-machine purism is accepted, there is no Turing machine that computes the halting function in the external sense, either according to the original, temporally-laden, account of computation in the external sense, or according to the purist's ontologically reduced account of this concept.

*8. Turing-machine realism*

According to Turing-machine purism, Turing machines are to be located within the ontology of pure mathematics. Turing machines are *n-tuples* whose members are sets and functions. According to the well-known Lewis-Papadimitriou definition of a Turing machine, for example, a Turing machine is a quadruple $<K, s, \sum, \partial>$, where $K$ is a finite set of states, $s$ is a member of $K$ (the initial state), $\sum$ is a set of symbols, and $\partial$ is a function from $K \times \sum$ (the transition function), whose values are state-symbol pairs (including the special symbols "*L*" and "*R*").[xxiv] According to Turing-machine realism, on the other hand, Turing machines are—not denizens of set theory—but notional *machines*.

Turing-machine realism recognizes an ontological level lying between the realization (or physical-device) level and the level of pure-mathematical ontology. We term this the *level of notional machines*. At this level are to be found notional or idealized machines that are rich with spatio-temporality and causality. The types of component of the entities occupying this level are written about in engineering textbooks rather than in textbooks of pure mathematics—tape, read-write heads, clocks, counters, circuits (in the Turing-von Neumann sense), gears, levers, flip-flops, high-speed random-access memories, pipelines, and so on and so forth. Other forms of idealized computing machine, apart from the Turing machine, are to be found at this ontological level, including analogue computers, neural networks, quantum computers and hypercomputers. According to the realist's view of the Turing machine, the *n*-tuple formulation is a useful abstract *representation* of the Turing machine, but the representation must not be confused with the thing represented, in this case a notional machine.

On the basis of textual evidence, it seems transparent that Turing himself was not a Turing-machine purist. His "computing machines" of 1936 have a delicious

physicality to them. This is nowhere better highlighted than in the subroutines that Turing described.[xxv] In one (which is typical), the machine's scanner traverses the tape from right to left, scanning each successive square of tape, and when the scanner locates the left-most occurrence of the symbol it is searching for, the scanner erases it, then prints "1" in its place, and goes into m-configuration *A*; or, if the scanner strikes the special symbol marking the end of the tape without finding an occurrence of the symbol it is searching for, the scanner goes into m-configuration *B*. The positively industrial flavour of Turing's descriptions of the operation of the machine also extends to his explanation of the notion of an m-configuration: "The machine is susceptible to a number of m-configurations, $q_1$ ... $q_n$; that is to say the levers, wheels, et cetera can come to be arranged in a number of ways, called 'm-configurations'."[lxxvi]

Turing brought *machinery* into discussions of the foundations of mathematics. This is one of the features that make his approach so novel—even daring. In a biographical memoir of Turing for the Royal Society of London, Turing's friend and colleague Max Newman[xxvii] wrote: "It is difficult to-day to realize how bold an innovation it was to introduce talk about paper tapes and patterns punched in them, into discussions of the foundations of mathematics" (Newman 1955: 256). (Post, too, emphasized the physicality of his 1936 system—essentially similar to Turing's—which consisted of a "worker" moving in a "symbol space" composed of "boxes" each of which may be unmarked or marked with a single stroke. Post said: "the boxes are, conceptually at least, *physical entities*" (Post 1936: 105; our emphasis).) Turing-machine purism can be viewed as an ontological reaction to Turing's introduction of punched paper tape into discussions of the foundations of mathematics. Mathematics is re-purified by an ontological reduction of Turing machines.

Although Turing provides no explicit discussion of the temporal aspect of Turing machines, his descriptions of his machines nevertheless ooze temporality. He speaks of the machine being "set in motion" and of carrying out its operations "successively" (1936: 60, 61). Temporal succession is also indicated by the use of adverbs such as "previously", as in "the machine moves so that it scans the square immediately on the right of the one it was scanning previously". Turing's tables describing the behaviour of his machines are interpreted in terms of temporal succession ("the operations in the third column are carried out successively"); and it is temporal succession that enables the instructions "P1, R" and "R, P1" to be distinguished ("P1" means "print '1'" and "R" means "the machine moves so that it scans the square immediately on the right of the one it was scanning previously").

Turing's discussion of printing is also phrased in overtly temporal terms: he says that a computing machine "will write down successively" certain symbols (p.

74) and asks whether a machine "ever prints a given symbol" and whether a machine "prints 0 infinitely often" (p. 73). Terms that normally imply temporal duration, such as "motion" and "process", are used freely and without qualification. Printing is a "process" (p. 63), and other "processes" include "copying down sequences of symbols", "comparing sequences" symbol by symbol, and successively "erasing all symbols of a given form" (ibid.). As pointed out in Section 7, Turing's term "moment" may certainly be re-interpreted in a time-neutral way, by replacing "moment" by "stage"; but it seems clear that Turing himself was content to use the term in its natural, temporal sense—e.g. he uses this term in describing the behaviour of a human being, speaking of "the symbols which he is observing … at that moment" (p. 75)). In the later (1948), Turing uses the term "moment" in an *explicitly* temporal way:

> The times when these pulses arrive will be called "moments". Each unit is capable of having two states at each moment.
>
> (Turing 1948: 417)

The purist is, of course, correct that talk of temporality can be "paraphrased out" of Turing machine theory; and that, furthermore, all the necessary concepts can be re-expressed in language whose ontological commitments are restricted to those of pure mathematics. (In effect, the success of this paraphrasis is guaranteed by Turing's equivalence theorem that all λ-definable sequences are Turing-machine computable and vice versa (1936: 88).) Naturally, the realist acknowledges the value of paraphrasis and the utility—especially in proof theory—of the set-theoretic representation of the Turing machine. But the realist denies that the availability of these paraphrases has any ontological import. Similarly, a realist may agree that all statements about the natural numbers can be paraphrased in terms of the vocabulary of set theory (1 is $\{\varnothing\}$, 2 is $\{\varnothing, \{\varnothing\}\}$, and so on), and yet will deny that the natural numbers *are* sets. The natural numbers are *numbers*—and Turing machines are *machines*. The realist rejects the purist's attempted ontological reduction.

To return to the issue of external computation: for ontological reasons, the purist requires that no ineliminable temporal reference be permitted when specifying *computes* as a predicate of Turing machines (and so external computation in the purist's sense collapses to internal computation, as explained in Section 7). The Turing-machine realist, on the other hand, unlike the purist, lodges no such requirement. Defining *computes* via reference to an external clock involves an extension of the classical theory of Turing machines, but one that is fully consistent with the realist's view of the Turing machine.

## *9. Conclusion*

We have disambiguated the concept of an accelerating Turing machine, distinguishing between end-stage-in and end-stage-out machines and arguing that end-stage-in accelerating machines are not Turing machines. We have presented two differing conceptions of computation, the internal and the external, and two very different conceptions of the Turing machine. Which is correct, Turing-machine purism or Turing-machine realism? We leave the reader to decide! We end with a tabular summary of the possible positions regarding the question "Does an accelerating Turing machine compute the halting function?".

TABLE 1 ABOUT HERE

| computation type | Turing-machine purism | Turing-machine realism |
|---|---|---|
| internal | NO | NO |
| external | NO | YES |

Table I: Does an ATM compute the halting function?

REFERENCES

Andréka, H., Németi, I., Németi, P. (2009), 'General Relativistic Hypercomputing and Foundation of Mathematics', *Natural Computing*, 8, 499-516.

Barker-Plummer, D. (2004), 'Turing Machines', *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (ed.), <plato.stanford.edu/archives/spr2005/entries/turing-machine>.

Beggs, E.J., Tucker, J.V. (2006), 'Embedding Infinitely Parallel Computation in Newtonian Kinematics', *Applied Mathematics and Computation*, 178, 25-43.

Benacerraf, P. (1962), 'Tasks, Super-Tasks, and the Modern Eleatics', *Journal of Philosophy*, 59, 765-84.

Blake, R.M. (1926), 'The Paradox of Temporal Process', *Journal of Philosophy*, 23, 645-54.

Boolos, G.S., Jeffrey, R.C. (1980), *Computability and Logic*, 2nd edition, Cambridge: Cambridge University Press.

Calude, C.S., Staiger, L. (2010), 'A Note on Accelerated Turing Machines', *Mathematical Structures in Computer Science*, 20, 1011-1017. .

Cohen, R.S., Gold, A.Y. (1978), 'ω-computations on Turing Machines', *Theoretical Computer Science*, 6, 1-23.

Copeland, B.J. (1997), 'The Broad Conception of Computation', *American Behavioral Scientist*, 40, 690-716.

Copeland, B.J. (1998a), 'Even Turing Machines Can Compute Uncomputable Functions', In Calude, C.S., Casti, J., Dinneen, M.J. (eds) (1998), *Unconventional Models of Computation*, Singapore: Springer-Verlag: 150-164.

Copeland, B.J. (1998b), 'Super Turing-Machines', *Complexity*, 4: 30-32.

Copeland, B.J. (1998c), 'Turing's O-machines, Penrose, Searle, and the Brain', *Analysis*, 58, 128-138.

Copeland, B.J. (2000), 'Narrow Versus Wide Mechanism: Including a Re-Examination of Turing's Views on the Mind-Machine Issue ', *Journal of Philosophy*, 97, 5-32.

Copeland, B.J. (2002a), 'Accelerating Turing Machines', *Minds and Machines*, 12, 281-300.

Copeland, B.J. (2002b), 'Hypercomputation', in Copeland (2002-3), 461-502.

Copeland, B.J. (ed.) (2002-3), *Hypercomputation*, special issue of *Minds and Machines*, *12(4)*, *13(1)*.

Copeland, B.J. (ed.) (2004a), *The Essential Turing*, Oxford and New York: Oxford

University Press.

Copeland, B.J. (2004b), 'Colossus – Its Origins and Originators', *IEEE Annals of the History of Computing*, 26, 38-45.

Copeland, B.J. (2004c), 'Hypercomputation: Philosophical Issues', *Theoretical Computer Science*, 317, 251-267.

Copeland, B.J. (2005), 'Comments from the Chair: Hypercomputation and the Church-Turing Thesis', paper delivered at the American Philosophical Society Eastern Division Meeting, New York City.

Copeland, B.J. (2010), 'Colossus: Breaking the German "Tunny" Code at Bletchley Park. An Illustrated History.' *The Rutherford Journal: The New Zealand Journal for the History and Philosophy of Science and Technology*, 3, <www.rutherfordjournal.org>.

Copeland, B.J., Proudfoot, D. (1999), 'Alan Turing's Forgotten Ideas in Computer Science', *Scientific American*, 280 (April): 76-81.

Copeland, B.J. and Shagrir, O. (2007), 'Physical Computation: How General are Gandy's Principles for Mechanisms', *Minds and Machines*, 17, 217-231.

Copeland, B.J., Sylvan, R. (1999), 'Beyond the Universal Turing Machine', *Australasian Journal of Philosophy*, 77, 46-66.

Davies, B.E. (2001), 'Building Infinite Machines', *British Journal for the Philosophy of Science*, 52, 671-682.

Davis, M. (1958), *Computability and Unsolvability*, New York: McGraw-Hill.

Earman, J., Norton, J.D. (1993), 'Forever is a Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes', *Philosophy of Science*, 60, 22-42.

Earman, J., Norton, J.D. (1996), 'Infinite Pains: The Trouble with Supertasks', in Morton, A., Stich, S.P. (eds), *Benacerraf and his Critics*, Oxford: Blackwell, 231-261.

Fearnley, L.G. (2009), 'On Accelerated Turing Machines', Honours thesis in Computer Science, University of Auckland.

Fraser, R., Akl, S.G. (2008), 'Accelerating Machines: a Review', *International Journal of Parallel Emergent and Distributed Systems*, 23, 81-104.

Hamkins, J.D. (2002), 'Infinite Time Turing Machines', in Copeland (2002-3), 521-539.

Hamkins, J.D., Lewis, A. (2000), 'Infinite Time Turing Machines', *Journal of Symbolic Logic*, 65, 567-604.

Hogarth, M.L. (1992), 'Does General Relativity Allow an Observer to View an Eternity in a Finite Time?', *Foundations of Physics Letters*, 5, 173-181.

Hogarth, M.L. (1994), 'Non-Turing Computers and Non-Turing Computability', *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association*, 1, 126-138.

Hogarth, M.L. (2004), 'Deciding Arithmetic Using SAD Computers', *British Journal for the Philosophy of Science*, 55, 681–691.

Kripke, S.A. (1959), 'A Completeness Theorem in Modal Logic', *Journal of Symbolic Logic*, 24, 1-14.

Lewis, H.R., Papadimitriou, C.H. (1981), *Elements of the Theory of Computation*, Englewood Cliffs, NJ: Prentice-Hall.

Newman, M.H.A. (1955), 'Alan Mathison Turing, 1912-1954', *Biographical Memoirs of Fellows of the Royal Society*, 1, 253-263.

Pitowsky, I. (1990), 'The Physical Church Thesis and Physical Computational Complexity', *Iyyun*, 39, 81-99.

Post, E.L. (1936), 'Finite Combinatory Processes – Formulation 1', *Journal of Symbolic Logic*, 1, 103-5.

Potgieter, P.H., Rosinger, E.E. (2009), 'Output Concepts for Accelerated Turing Machines', *Centre for Discrete Mathematics and Theoretical Computer Science Research Reports*, http://hdl.handle.net/2292/3858.

Quine, W.V.O. (1960), *Word and Object*, Cambridge, MA: MIT Press.

Russell, B.A.W. (1915), *Our Knowledge of the External World as a Field for Scientific Method in Philosophy*, Chicago: Open Court.

Schaller, M., Svozil, K. (2009), 'Zeno Squeezing of Cellular Automata', *arXiv:0908.0835*.

Shagrir, O. (2004), 'Super-tasks, Accelerating Turing Machines and Uncomputability', *Theoretical Computer Science*, 317, 105-114.

Shagrir, O. (2010), 'Supertasks Do Not Increase Computational Power', under review.

Shagrir, O., Pitowsky, I. (2003), 'Physical Hypercomputation and the Church-Turing Thesis', in Copeland (2002-3), 87-101.

Steinhart, E. (2002), 'Logically Possible Machines', *Minds and Machines*, 12, 259-280.

Steinhart, E. (2003), 'The Physics of Information', in L. Floridi (ed.) *The Blackwell Guide to the Philosophy of Computing and Information* (Oxford: Blackwell), 178-185.

Stewart, I. (1991), 'Deciding the Undecidable', *Nature*, 352, 664-5.

Svozil, K. (1998), 'The Church-Turing Thesis as a Guiding Principle for Physics', in Calude, C.S., Casti, J., Dinneen, M.J. (eds),, *Unconventional Models of Computation*, London: Springer-Verlag, 371-385.

Svozil, K. (2009), 'On the Brightness of the Thomson Lamp. A Prolegomenon to Quantum Recursion Theory', in C.S. Calude, J.F. Costa, N. Dershowitz, E. Freire, and G. Rozenberg (eds), *Unconventional Computation. Lecture Notes in Computer Science, Vol. 5715*, Berlin: Springer-Verlag, 236-246.

Thomson, J.F. (1954), 'Tasks and Super-Tasks', *Analysis*, 15, 1-13.

Thomson, J.F. (1970), 'Comments on Professor Benacerraf's Paper'. In W.C. Salmon (ed.), *Zeno's Paradoxes*, Indianapolis: Bobbs-Merrill, 130-138.

Turing, A.M. (1936), 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society*, Series 2, 42, 230-265. In *The Essential Turing* (Copeland 2004a); page references are to the latter.

Turing, A.M. (1948), 'Intelligent Machinery'. National Physical Laboratory Report. In *The Essential Turing* (Copeland 2004a). A digital facsimile of the original document may be viewed in The Turing Archive for the History of Computing <http://www.AlanTuring.net/intelligent_machinery>.

Turing, A.M. (1950), 'Computing Machinery and Intelligence', *Mind*, 59,433-60. In *The Essential Turing* (Copeland 2004a); page references are to the latter.

Weyl, H. (1927), *Philosophie der Mathematik und Naturwissenschaft*, Munich: R. Oldenbourg.

Weyl, H. (1949), *Philosophy of Mathematics and Natural Science*, Princeton: Princeton University Press.

NOTES

[i] The idea of relativistic machines originated in Pitowsky (1990). For a discussion of the relations between ATMs and relativistic machines see Shagrir (2010); for a discussion of their physical possibility see Earman and Norton (1993, 1996), Copeland and Shagrir (2007), Andréka, Németi, and Németi (2009).

[ii] The variant term "accelerated Turing machine" (see e.g. Calude and Staiger (2009), Fearnley (2009), Potgieter and Rosinger (2009)) is from Copeland (1998a).

[iii] The "halting problem" was introduced and named by Davis (1958: 70) (not by Turing himself, contrary to popular belief: see Copeland (2004a: 40)).

[iv] Another way to put the difference is in terms of whether the specification of the machine refers to a last computation step (end-stage-in machine) or not (end-stage-out machine); see also note 7. In our terminology, a *step* of the computation consists in the execution of either an atomic operation or a subroutine, and is analogous to a step in the proof-theoretic sense, i.e. the application of a primitive or derived rule; a *stage* of the computing is analogous to a stage in the construction of a proof (see e.g. Kripke 1959). A *state* of the accelerating machine is either an *m*-configuration (in Turing's sense—see the quotations in Sections 3 and 8) or a limit-state (see below).

[v] See Boolos and Jeffrey (1980: 23), and Lewis and Papadimitriou (1981: 172). See also Turing (1936: 59), quoted in Section 3; and note 9, below.

[vi] See Copeland (2002a) and Shagrir (2004) for a discussion of the Thomson lamp in connection with accelerating Turing machines.

[vii] Following Benacerraf (1962: 772; and see also Fraser and Akl 2008) we can say that an end-stage-in machine performs a *super-duper task*, whereas an end-stage-out machine performs only a *supertask*. If performing a super-task includes a sequence of infinitely many atomic operations (of order type ω), then performing a super-duper task also includes another, *last*, operation at an extra limit stage.

[viii] The term "hypercomputer" was introduced by Copeland in *Scientific American* in 1999 (Copeland and Proudfoot 1999); see further Copeland (1997, 1998c, 2000, 2002b, 2004c), Copeland and Sylvan (1999).

[ix] Note that the configuration as defined by Turing in this quotation does not include every symbol contained on the tape at that stage (only the currently scanned symbol). Sometimes the term "total configuration" is used to distinguish the sense of "configuration" in which the total tape content (at that stage) is included.

[x] Shagrir (2010) argues that similar considerations apply to relativistic machines (e.g., Pitowsky 1990; Hogarth 1992, 1994, 2004; Shagrir and Pitowsky 2003) and to shrinking machines (e.g., Davies 2001; Beggs and Tucker 2006; Schaller and Svozil 2009). These machines also perform supertasks and exhibit hypercomputational power; yet, like $H^+$, they do not have the computational structure of a Turing machine. Their hypercomputational power is the result (at least partly) of a computational

structure that differs in crucial respects from the computational structure of a Turing machine.

[xi] Copeland (1998a, 1998b); Svozil (1998).

[xii] See Shagrir (2004, 2010). One could also challenge the physical feasibility of accelerating Turing machines (for discussion see Steinhart 2003, and Fearnley 2009), but the claim argued here is that H does not compute the halting function even if all problems of physical implementation are resolved.

[xiii] See Lewis and Papadimitriou (1981:170). A "halted configuration" is one whose state component is a halt state (p. 172). There may be other means (other than being in a "halt state") to indicate that the machine has halted, such as being in a halted configuration; see Boolos and Jeffrey (1980: 22-23), and Turing (1936).

[xiv] We use the halting function as an example of an uncomputable function; however, our arguments, here and in what follows, are general in nature and apply to uncomputable functions both below and beyond the halting function.

[xv] See Copeland (1998a, 2002a).

[xvi] The distinction and terminology were introduced in Copeland (1998a).

[xvii] It is important that the time-interval be a proper (i.e. delimited) interval and not simply the whole of (non-transfinite) time, since otherwise the reference to an external clock (or some other time-keeping arrangement) is rendered otiose. For example, if the time "interval" under consideration is the whole of time, then H's non-accelerating counterpart computes the halting function in the external sense: the value is 1 if only if the initial "0" is replaced by "1" at some time; and the value is 0 otherwise.

[xviii] As emphasized above, H's specification does not include a configuration of the machine *at* the end of the second moment.

[xix] Copeland (2005).

[xx] The situation described is similar to that with Hogarth's (1992) example of a Turing machine travelling through anti-de Sitter spacetime, which Earman and Norton (1993) showed to be subject to an observer-destroying blue shift.

[xxi] Copeland (1998a).

[xxii] See, for example, the definition of a Turing machine in Lewis and Papadimitriou (1981:170ff.; see also Section 8, below), and the entry "Turing machines" in the Stanford Encyclopedia of Philosophy—e.g. "Turing machines are not physical objects but mathematical ones" (Barker-Plummer 2004).

[xxiii] Quine (1960), ch. 6.

[xxiv] See Lewis and Papadimitriou (1981: 170-171) for the details.

[xxv] The subroutines are described on pp. 63-66 of Turing (1936).

[xxvi] Draft précis of "On Computable Numbers" (undated, 2 pp.; in the Turing Papers, Modern Archive Centre, King's College Library, Cambridge, catalogue reference K 4). In French; translation by Copeland.

[xxvii] For biographical information on Newman see Copeland (2004b, 2010).