An Exercise in Unplugging the Church-Turing Thesis

Edgar G. Daylight

December 18, 2018

Abstract

Resorting solely to concepts from classical computability theory, I provide mathematical arguments to doubt — if not dismiss, on an objective basis — the Church-Turing Thesis. Defending the thesis amounts to believing that “any process which could naturally be called an effective procedure can be realized by a Turing machine” (Minsky, 1967). Specifically, I present a natural modification of the “Turing machine” model of computation, called the “Alternative Deterministic Machine” model (or ADM for short). The relevance of this new model hinges on the following observation: Turing machines have a lower model fidelity than ADM’s with regard to human (and electronic) computers. Both a Turing machine and an ADM model a human computer who contributes to her research community by publishing mathematical findings. But, in reality, humans publish in a piecemeal fashion, rather than all in one go. Turing machines, as formally defined by Hopcroft & Ullman (1979), do not capture this particular trait of human activity, while ADM’s do. To recapitulate in technical terms: a Turing machine provides meaningful output (for the outside world to see) instantly, after having halted, while an ADM provides meaningful output (to its environment) incrementally, before possibly halting. This distinction will allow me to (i) prove that Turing machines partially compute less functions on the naturals than ADM’s, and (ii) prescribe an ADM-based method to generate a subset of the naturals that is not computably enumerable. I shall furthermore discuss multiple ways to generalize the ADM model. This discussion will lead to a new incentive, based on a *mathematician-as-typewriter* metaphor, to embrace even more powerful models of computation — i.e., the so-called “eventually correct machines” in the literature — as natural formalizations of algorithms.

*** The author can be contacted via email (*egdaylight “at” dijkstrascry “dot” com*) and via postal mail (*Karel Van Oudheusden, School of Media and Information, Siegen University, Herrengarten 3, 57072 Siegen, Germany*).
*** The present document is catalogued by the Belgian Association of Authors, Composers and Publishers (*www.sabam.be/en*) and is a slightly modified version of an article submitted for peer review in a computer science journal on December 4, 2018.

1 Basic Ideas

In 1952, Stephen Kleene wrote about Turing’s Thesis:

> “What we must do is to convince ourselves that any acts a human computer could carry out are analyzable into successions of atomic acts of some Turing machine.” [33, p.377]

In the present article, I argue that there *are* relevant acts a human computer carries out which are *not* captured by Turing machines: a human computer can show some of her preliminary, yet

fixed, results (e.g., lemmas) to the outside world, while she is still deriving her major finding (e.g., a theorem). Another way to convey this basic idea is as follows:

> Any prefix of the final output of an algorithm is useful to have as soon as possible.

Turing machines, as formally defined in textbooks today, do not capture this particular aspect of algorithms.

The focus in the present paper lies on human computing, but, as an aside, it should also be mentioned that several algorithms implemented in electronic computers provide definitive output at various stages of computation, before possibly halting. An example is ordinary batch processing: the first sheet containing printed output can already be used, while the follow-up output (of the same batch) is still being processed. I will therefore introduce Alternative Deterministic Machines (ADM's) as a slight modification of Deterministic Turing Machines (DTM's).

To be more precise, I will build on the following observation:

> A deterministic Turing machine M, in the modern sense of Kleene, models a mathematician who is computing a function on the naturals (i.e., the natural numbers) from x to $f(x)$. Machine M works on string representations of naturals. Consider some standard encoding function *enc* from naturals to strings of, say, binary digits. Then we can say that Turing machine M takes $enc(x)$ as input and, if all goes well, produces a finite output, i.e. the string $enc(f(x))$. The crux is that a halting Turing machine M reveals its complete result, i.e. $enc(f(x))$, *all in one sweep*. An Alternative Deterministic Machine N, by contrast, will, on input $enc(x)$, provide the first bits of $enc(f(x))$ for the outside world to see as soon as they have been computed.

In all other respects, an Alternative Deterministic Machine is merely a Deterministic Turing Machine.[1] Nevertheless, ADM's will turn out to be computationally a bit stronger than DTM's, thereby apparently contradicting Turing's Thesis[2] and, by extension, Church's Thesis: "Every effectively calculable function (effectively decidable predicate) is general recursive" [33, p.300]. Indeed, Turing's Thesis and Church's Thesis are equivalent according to Kleene [33, p.376], and I will use the umbrella phrase "the Church-Turing Thesis" to capture the core notion under scrutiny in this paper.

The technical reason why ADM's can partially compute more functions than Turing machines is neither complicated, nor particularly novel. Consider, for instance, an ADM V that can simulate an arbitrary DTM M on arbitrary input w. Before starting the simulation, V prints the symbol 0 on its output tape. On the one hand, if the simulation does not terminate, then V has printed nonempty output (i.e., symbol 0 and nothing more), which differs from M's output (i.e., the empty string). On the other hand, if the simulation stops, then M can print one more output symbol (i.e., 0 or 1) such that its entire output differs from M's output. The net result is that V computes a function on the naturals that differs from every DTM-computable function.

Some experts (coming from outside of computer science proper) have orally expressed doubts about my contradistinction between Turing machines and ADM's. I now paraphrase their concerns:

> Why would "an ADM, providing meaningful output incrementally, before possibly halting" lead to more computable functions than Turing machines? Can't a Turing

[1] I explicitly use the adjective "deterministic" in "ADM" and in "DTM," in anticipation of a later publication about the contradistinction between deterministic and nondeterministic machines. All machines discussed in the present article are deterministic and the reader can therefore safely ignore the adjective at all times.

[2] Note that I am not defending the claim that Turing, himself, endorsed what we today call "Turing's thesis" (cf. Jack Copeland et al. [14, Sec.5]), nor am I questioning that claim.

machine reveal some of its output as it continues to compute? It is, after all, possible to program Turing machines in a way that they have meaningful partial outputs through a computation for certain problems.

My answer is two-fold. First, abiding by the formal setup of John Hopcroft & Jeffrey Ullman [32], a Turing machine only provides output (to the outside world) after halting, not before. The same remark holds for the writings of Kleene [33] and Martin Davis [16, 19]. Second, suppose that, contrary to Hopcroft & Ullman (and other modern textbooks), we *do* tolerate a more liberal connotation of a Turing machine; that is, we allow our liberal Turing machines to do precisely what ADM's do. Then, the results in the present paper will convey that liberal Turing machines with one output tape (cf. ADM's) are computationally inferior to liberal Turing machines with two output tapes (cf. BDM's), which, in turn, are inferior to liberal Turing machines with three output tapes (cf. CDM's). In other words, we would then obtain a non-standard result in computability theory, contradicting the following, well-known property of Turing machines:

> "One reason for the acceptance of the Turing machine as a general model of a computation is that the model with which we have been dealing is equivalent to many modified versions that would seem off-hand to have increased computing power." [32, p.159]

To recap, ADM's are not Turing machines.

The next question, then, is: Why would ADM's (which have one output tape) partially compute less functions than BDM's (which have two output tapes)? On the one hand, for any ADM M there clearly exists a BDM N that can simulate M. So ADM's are definitely not more powerful than BDM's. On the other hand, we can concoct a BDM N^* that diagonalizes out of the class of all ADM's. When N^* is asked to run on input $\langle M, w \rangle$, BDM N^* will not only simulate ADM M on input word w on its work tape. BDM N^* will also ensure that its own output is spread across its two output tapes, such that, at all times, the output (generated so far) by N^* differs from the output (generated so far) by M. Leaving the details for later, the implication is that N^* partially computes a function f^* that is provably not partially computable by any ADM. The crux will be that:

- The output tape of M and the two output tapes of N^* are, at all times, observable to the outside world.
- Once a symbol is printed on an output tape, it remains there forever.
- Output symbols are printed from left to right, for each output tape.

A similar construction leads to the conclusion that BDM's (which have two output tapes) are computationally inferior to CDM's (which have three output tapes). And so on.

To remain at an elementary level, however, I now present five introductory subsections. The last subsection outlines the rest of this article.

1.1 Syntax and Semantics

The critical reader will observe that I am not abandoning the input/output perspective which prevails in recursive function theory. Specifically, I stick to partial functions on the naturals ($\mathbb{N}$) and to the viewpoint that machines (e.g., DTM's and ADM's) serve the sole purpose of "mathematically implementing" such functions. Several paradigms that are orthogonal to classical computability theory, many of which are concurrency- and interaction-centric [26, 30], are only discussed in the present and in the final section, not in the main body of this article.

Moreover, when mathematically modeling a human computer, I will not resort to multiple forms of hypercomputation that can be found in the literature today [13, 23]. For example, I will not rely on infinitely fast computations, nor on storage of infinitely many bits in a finite space, etc.

With those topics put into context, note that I am nevertheless advocating a more general conception of the word "algorithm." On the one hand, I am following Alan Turing in 1936 with his computable real numbers in that an algorithm need not halt. (Turing even eschewed halting computations altogether.) On the other hand, both Turing's original 1936 machines [56] and the later re-cast "Turing machines" [16, 33] have in common that the complete result of an algorithm attains a meaning in the outside world *in one, full sweep* only. An ADM, by contrast, can provide finite, definitive output to the outside world at various stages. Even an ADM that prints one bit and immediately halts differs conceptually from a halting Turing machine that has the same output (for the same input). The Turing machine reveals its one-bit output to the outside world only after halting, not before. This observation clarifies my personal preference to write about *alternative* machines, not generalized Turing machines, although my inspiration definitely comes from computer science textbooks in which occurrences of the words "Turing machine" are abundant.[3]

To elaborate on the distinction between ADM's and DTM's, I distinguish between the syntax and the semantics of both kinds of machines:

- Syntactically, an ADM N *is* a 2-tape DTM. Specifically, N has one read-write tape, serving both as input and work tape, and it has a write-only output tape. Both tapes are one-way infinite with a leftmost cell and infinite progression to the right. ADM N can print out symbols α (with α in some finite alphabet, called Γ) on its output tape *from left to right only*, and as long as it has not (yet) printed the end marker \$, with $\$ \notin \Gamma$.[4]
- Semantically, however, an ADM N is slightly different from any conceivable DTM in that N's generated output, which at any moment is finite in length, *can be perceived by the outside world, also when N's computation has not (yet) terminated.*

The semantic part comes naturally (at least to the present author) when attempting to model the aforementioned mathematician: a solitary, smart, human computer who contributes to her research community by publishing research findings piecemeal instead of all in one go. The research output perceived by the outside world is at any time a prefix of the mathematician's complete output, which, in the limit, can be either finite or infinite in length. A finite output in the limit will be considered desirable in most of this article. (The exception is Section 10 where I zoom in on Turing's 1936 paper.) My objective, after all, is to scrutinize the Church-Turing Thesis by:

1. Seeking mathematical machines that have a better model fidelity with regard to human computers than Turing machines.
2. Deviating as little as possible from the modern "Turing machine" concept, as provided by Kleene [33], Davis [16, 19], and Hopcroft & Ullman [32].

It is Hopcroft & Ullman's definition of a "Turing machine" that I will mostly duplicate in Section 2 and tweak in Section 4, in order to provide novel theoretical content in Section 6 and onward.

[3] My observation pertains to the modern "Turing machine" concept [32]. For an epistemological build-up of that concept, see Liesbeth De Mol's account [45].

[4] It is of course possible to formalize the same idea by resorting to a 1-tape DTM or a 3-tape DTM, formalizations that lie outside the scope of the present article.

1.2 Nonhalting Computations

To recapitulate, apart from some input $w \in \Sigma^*$ provided at the beginning of the computation (where Σ is a finite alphabet), an ADM-modeled mathematician does not receive input from the outside world at all. The main difference between a DTM M and an ADM N is that one has to wait for M to come to a halt before any part of its output attains a meaning, while this constraint does not apply to N. From a computability perspective, the bottom line will turn out to be this:

> A nonhalting DTM doesn't mean anything (to Kleene et al. in the 1950s and later), but a nonhalting ADM can convey useful information, provided it prints from left to right a finite number n of output symbols, with $n > 0$.

The mathematical implication, as will be demonstrated, is the existence of a function on $\mathbb{N}$ partially computable by an ADM but by no DTM (Theorem 25). The Church-Turing Thesis is potentially in jeopardy (Corollary 26). Any definitive claim about the status of the Thesis lies, partially due to its sociological dimension, outside the scope of this article. The mathematical findings presented in the following sections do not immediately suggest any new, practical import.

Before raising objections concerning ADM's, as any critical reader will, let us first contemplate the following common remark pertaining to DTM's and the partial functions f on $\mathbb{N}$ that they implement. When the outside world relies on a DTM M computing on some input w, it generally does not know *a priori* when M will have spent enough computational steps on w to derive an output. Loosely speaking, we have the following common scenario:

> The practical user of M only has finite resources at his disposal and, therefore, might falsely conclude that M on w does not halt. That is, even though $f(x)$ is defined, with $w = enc(x)$, the user might incorrectly assign an "undefined" value, $\perp$, to $f(x)$.

Indeed, a Turing machine computation is, in general, only trustworthy *in the limit*; one can only correctly assert $f(x) = \perp$ provided that space and time constraints do not apply (and the computation in hand fails to terminate). Coming now to ADM's, no more than a similar critique holds. In the general case, and colloquially speaking, one cannot know *a priori* whether ADM N on some input w prints only a finite number n of symbols, with $n > 0$. Consider also the following feasible scenario:

> The practical user of N does know *a priori* that N will print at most two symbols. Unfortunately, due to finite resources, he falsely concludes that N on w outputs σ_1 while, in fact, it outputs $\sigma_1\sigma_2$ in the limit.

The crux is, that, in the general case, a computation of an ADM N is only fully reliable *in the limit*; one can only confidently make assertions about N's complete functional semantics, i.e., $f(x)$, when resource constraints do not apply.

The remarks just presented do not prohibit the reader from working with infinitely large subsets of DTM's and ADM's, which *are* guaranteed to map finite input to finite output, optionally with constraints on the output length.

1.3 ADM's in Context

I use the natural phrase "computing in the limit" since I am perusing computations that need not halt. My intended meaning of this phrase does not coincide with that of Stewart Shapiro [39], Mark Burgin [9], Peter Kugel, et al. Specifically, the following words from Kugel convey the crux of what are called *eventually correct machines* in the literature. His words do not capture the essence of ADM's.

> "When we use a computer to compute, we take its first output to be its result. But when we use it to compute in the limit, *we take its last output as its result* without requiring that it announce when an output is its last." [36, p.35, my emphasis]

In contrast to the previous passage, ADM's spit out symbols that are part of the *final* output. The symbols are produced from left to right only; that is, they cannot be retroactively modified. Again, ADM's are mathematical models of real mathematicians, who, using pencil and paper, are unable to provide all digits of their outcome instantly, only incrementally. Turing in 1936, and Kleene in the 1940s, did not model this particular trait of human computing — even though Turing, in his 1936 work, took the limitation of the human sensory apparatus into account.[5] In retrospect, perhaps Turing was modeling the solitary mathematician (like himself) who does not compute results for the outside world to see; cf. [47, p.97].

Kleene modeled Turing machine computation from a black-box perspective: input symbols go into the box, and, if all goes well, output symbols come out. Unlike Kleene, I adhere to a grey-box perspective: I distinguish between symbols that come out earlier than others. (Likewise, input symbols need not go inside the box in one sweep either. This characteristic of real input/output computation is, however, ignored in the present paper.) The conceptual discrepancy is perhaps best conveyed by referring, not to Kleene's original work on automata theory, but, rather, to Marvin Minsky's pedagogical — and historically influential — 1967 book *Computation: Finite and Infinite Machines [43].* Minsky's "finite-state machines" in his Chapter 2 provide output symbols to the outside world in a piecemeal fashion; that is, as the computation in hand proceeds. But the same cannot be said of his "Turing machines" in his Chapter 6. Again, the reader might ask, why can't Turing machines produce definitive output symbols (for the outside world to see) at interval stages; that is, in a piecemeal fashion? An historical answer lies in Minsky's own words:

> "A *Turing machine* is a finite-state machine associated with a special kind of environment—its *tape*—in which it can store (and later recover) sequences of symbols." [43, p.107, original emphasis]

According to Minsky (and computer scientists today), a Turing machine, is a "closed system" [43, p.119]. Moreover, in compliance with the Church-Turing Thesis, it is the most powerful of all sensible generalizations of a finite-state machine — which, in turn, is an open system [43, p.114]. Nowhere does Minsky peruse the ADM model of computation, which, in retrospect, is another potentially fruitful generalization of a finite-state machine. To the best of my knowledge, this critique holds not only for Minsky's 1967 book, but for all modern textbook treatments of the "Turing machine" concept, since the publication of Kleene's 1952 *Introduction to Metamathematics* [33].

My plea to embrace the concept of an ADM approaches, but does not trespass, the border between traditional computability theory (where mathematical functions prevail) and computation as interaction (where other mathematical objects can have their say too). See e.g. the seminal work of Robin Milner for an example of the latter [42]. Milner's views on the Church-Turing Thesis have been conveyed, correctly or incorrectly, by others, including Peter Wegner [60] and Samson Abramsky. The latter is quoted in S. Barry Cooper's 2012 review article as follows:

> "There is indeed a lack of a clear-cut Church-Turing thesis in this wider sphere of computation—computation as interaction, as Robin Milner put it." [12, p.76]

Contrast this viewpoint with that of yet other respectable scholars: Nachum Dershowitz & Yuri Gurevich [22] and Wilfried Sieg [52]. For example, Sieg has introduced the abstract concept of

[5] See Wilfried Sieg et al. [53, p.201]. A similar remark holds for Emil Post and his attempt "to capture all possible processes we humans can set-up to solve problems" [44].

a "computable dynamical system." He shares the view that the Church-Turing Thesis holds on a broad scale; that is, also for parallel computation, including distributed systems. Especially in this broader context, it is more than natural for a critic, such as myself, to scrutinize the interaction between a Turing machine and its outside world. For, the outside world can, again, consist of Turing machines and the like. Specifically, although I appreciate each of Sieg's four informal requirements for human computers [52, p.392], listed below, I take gentle issue with Sieg's implicit assumption that output produced by each of his (sub)systems always, and only, happens in one, full sweep — an assumption, that, to the best of my knowledge, has not been made explicit in the literature so far. In Sieg's words, human computers:

1. Operate on finite, but unbounded configurations.
2. Recognize, in each configuration, a unique sub-configuration or pattern from a fixed, finite list.
3. Operate on the recognized pattern.
4. Assemble the next configuration from the original one and the result of the local operation.

Presumably, researchers with Sieg's profile will initially, from the outset, have the inclination to scrutinize the present paper along the following lines:

> "Turing machines cannot be successfully analyzed by *a finite general procedure* to determine what they might do sometime in the future."

This statement, coming from Charles Petzold's insightful book [47, p.307], also holds — or, very likely, also holds — when the italicized words refer, not to Turing machines, but to ADM's. For, indeed, at least in the present paper I will not provide the reader with an ADM-based procedure to "successfully" analyze an arbitrary Turing machine M, unless it is permissible to thoroughly simulate M. Turing machines, in contrast, cannot, in the general case, successfully analyze M *even if they are allowed to thoroughly simulate* M — unless, of course, one tweaks the standard definition of a Turing machine [32, Ch.7], as I do in the present paper, which will then result in the ADM model of computation or something similar.

The crux of this exposition is that neither *a finite general procedure*, nor an *algorithm*, need to be modeled by a Turing machine: there are more powerful machines that can realistically take over the modeling role of Turing machines. And, as we shall see, ADM's reveal only the tip of the iceberg in this regard.

1.4 Theoretical Findings

Trained initially as an engineer, I hold the opinion that ADM's have a higher model fidelity than DTM's with regard to human (and electronic) computing. This modeling viewpoint may, or may not, have theoretical implications in the realm of computable functions; that is, in a world where space and time constraints do not apply. As already mentioned however, it turns out that ADM's make a small, yet noticeable, difference on this mathematical scene.[6] In the present article I will:

- Show that every function that is DTM computable is also ADM computable (Theorem 23).
- Prove the existence of a function partially computable by an ADM, but by no DTM (Theorem 25).

[6] I opine that multiple scholars would endorse the statements just made about ADM's high model fidelity *as long as* ADM's turn out to be no more than Turing complete. Alas, this is not the case.

- Provide an ADM-based method to generate a subset of $\mathbb{N}$ that is not computably enumerable (Theorem 29).

Whether these results suffice to unplug the Church-Turing Thesis is a matter that is preferably addressed by multiple — and, especially, future — generations of scholars, rather than by any single authority. Perhaps the following 1989 words from Israel Kleiner convey the potential, indirect merit of the present paper:

> "Counterexamples play an important role in mathematics. They illuminate relationships, clarify concepts, and often lead to the creation of new mathematics." [34, p.294]

Halting Problem

It seems to be a common misunderstanding (in mainstream computer science) that a prerequisite to invalidating the Church-Turing Thesis is solving the halting problem of ordinary Turing machines. It will turn out that:

- There are plenty of functions f on $\mathbb{N}$ which are, and are not, partially computable by ADM's, and DTM's, respectively.
- Knowledge of all $(x, f(x))$ pairs will still not suffice for any ADM to solve the halting problem of Turing machines.

What matters is that ADM's have no lower model fidelity than DTM's *and* they are provably stronger computationally (i.e., in terms of recursive function theory). For, suppose I were to present a model of computation that *is* provably stronger than the Turing machine model *but* which relies on, say, the capability of carrying out infinitely many computational steps in finite time. Then, the reader will rightfully complain that my model has a lower fidelity than the Turing machine model.

Nevertheless, to thoroughly understand the ADM model of computation, it does pay to examine how far ADM's can go further than DTM's in *attempting* to solve the latter's halting problem. Doing so, as summarized in the following paragraphs, will lead the reader through a maze of ever-more powerful machines and, finally, to the aforementioned *eventually correct machines* that have been described in the literature before.

Consider, to begin with, an arbitrary computably enumerable set S of naturals. The characteristic function of S, denoted χ_S, maps naturals to either 0 or 1 like this:

- $\chi_S(n) = 0$ if $n \notin S$
- $\chi_S(n) = 1$ if $n \in S$

If my charitable readership allows the number 0 to be encoded with the string 0 and the number 1 with the string 0σ, where σ is any nonempty string of the reader's choosing, then it is possible to construct an ADM N that computes χ_S in the limit, for the given set S. More generally, if the reader accepts any encoding function *enc* such that *enc*(0) is a strict prefix of *enc*(1), then one can construct an ADM that computes χ_S in the limit, for the given set S. For instance, set S can be the halting set for ordinary Turing machines (cf. Corollary 28 and Theorem 39).

An important remark is that many texts on computability theory, including Kleene's 1952 book [33], use unary notation. Specifically, if we define the encoding of n to be a string of ones of length $n+1$, then the characteristic function of the halting set of ordinary Turing machines is

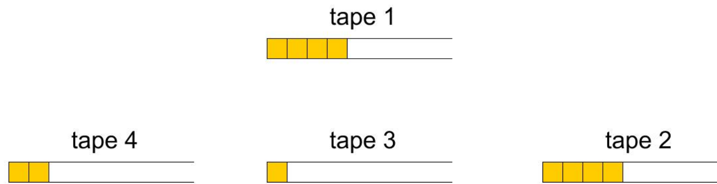

Figure 1.1: Illustrating the tapes of a CDM: tape 1 contains input, and tapes 2, 3, and 4 contain output.

computable in the limit by a potentially nonhalting ADM that outputs at most two symbols on every input. (That is, the ADM in hand need not have the capability to produce an unbounded, yet finite, number of output symbols.)

Now, coming to my critical readership. If they insist that, say, $enc(0)$ is string 0 and $enc(1)$ is string 10, then machines that can prepend bits to their output, rather than append bits, come in handy. I call machines that can both prepend and append bits BDM's. They naturally extend ADM's and amount to the following analogy: a mathematician typically writes the preface of her Ph.D. dissertation *after* having produced and discussed her lemmas and theorems. (The present author wrote the abstract of the present article after spending months on all the rest.) BDM's have a high model fidelity in this regard, while ADM's do not. One possible implementation choice for a BDM amounts to the following:

- Just like an ADM, a BDM has an input and work tape, which we call its *first* tape.
- Appending symbols σ_1 takes place on a one-way infinite, left-to-right output tape. This *second* tape is merely a clone of ADM's output tape.
- Prepending symbols σ_2 occurs on a *separate* one-way infinite, left-to-right output tape, called the *third* tape.
- The complete output string, $\sigma_2\sigma_1$, is read from left to right, starting with the leftmost symbol on the third tape, and ending with the rightmost symbol on the second tape.
- Just like an ADM, the output symbols of a BDM are observable to the outside world as soon as they have been printed (i.e., on either the second or third tape).

Figure 1.1 illustrates a BDM if tape 4 is discarded from the figure. A variant of the BDM concept is discussed formally in Section 10. Suffice to mention here that BDM's can be used to compute the characteristic function of the halting set of ordinary Turing machines for any of the aforementioned encoding schemes.

Coming to the devil's advocate. Suppose he insists that $enc(0)$ is, say, string 0 and $enc(1)$ is, say, string 1. BDM's cannot solve the halting problem of ordinary Turing machines in this particular setting (and, therefore, neither can ADM's). More powerful machines can, nevertheless, be concocted:

1. One approach is to add another output tape to a BDM, resulting in a CDM (illustrated in Figure 1.1). CDM's are computationally stronger than BDM's. In general, one can keep adding output tapes and, accordingly, keep obtaining computationally stronger machines (cf. Section 10). This can be done any finite number of times. None of these machines, though, will solve the halting problem of the devil's advocate.

2. Another, orthogonal, approach is to allow an ADM (or a BDM, or a CDM, ...) to *modify* r output symbols, where r is a fixed, natural number. Even more generally, one could concoct a machine that can modify output symbols a finite, yet unbounded, number of times.

This second idea can be found in the writings of e.g. Timothy McCarthy & Stewart Shapiro [39] and Mark Burgin [9]. It suffices to empower an ADM with the capability of modifying no more than one output symbol in order to solve the halting problem of the devil's advocate, as Burgin has demonstrated with his inductive Turing machines [9, Ch.4]. A discussion will follow in Section 11, where I provide a new incentive to also embrace these more powerful machines as natural models of algorithms.

1.5 Outline

In the interest of obtaining a bird's-eye view, the reader is encouraged to read the last section (i.e., Section 11) before delving into the rest of this article. Moreover, a distinction can be made between odd-numbered and even-numbered sections. Odd-numbered sections convey ideas mostly in plain English. They can be consulted before delving into the even-numbered sections, which, in turn, provide the mathematical rigor.

Section 2 contains no surprises, for it merely reintroduces the common "Turing Machine" setup. A few comments about this setup follow in Section 3, along with general remarks pertaining to Section 4. The latter section, in turn, formally introduces the ADM model of computation. Section 5, being an odd-numbered section, summarizes parts of both the previous and the next section. In Section 6 I demonstrate ADM's computational superiority over DTM's. Another intermezzo appears in Section 7, before I detail the limits of ADM's computational power in Section 8. Finally, Section 9 transitions from computing functions on the naturals (ADM's) to computing functions on the reals (AdM's). A treatment of AdM's, BdM's, and CdM's follows suit in Section 10.

2 Preliminaries and Deterministic Turing Machines

2.1 Strings

The terms "string" and "word" are used as synonyms. We assume the reader is familiar with string-based notation:

- the empty string ϵ of length zero,
- the concatenation $\alpha_1\alpha_2$ of two strings α_1 and α_2,
- Γ^*, which denotes the set of all strings of symbols in alphabet Γ,
- $\Gamma^+ = \Gamma^* \setminus \{\epsilon\}$,
- the length $|w|$ of string w.

Finally, both α_1 and $\alpha_1\alpha_2$ are *prefixes* ($\preceq$) of $\alpha_1\alpha_2$, but only α_1 is a *strict prefix* ($\prec$) of $\alpha_1\alpha_2$ and with the proviso that $\alpha_2 \neq \epsilon$.

2.2 Definitions from Hopcroft, Ullman, and Davis

Copying from Hopcroft and Ullman [32, Ch.7], we obtain the following formal setup.

All deterministic Turing machines under consideration have one tape only and, specifically, a one-way infinite tape. The tape has a leftmost cell and progresses infinitely to the right.

Definition 1. A *deterministic Turing machine* (abbreviated DTM) is denoted

$M = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$
where
Q is the finite set of *states,*
Γ is the finite set of allowable *tape symbols*,
b, a symbol of Γ, is the *blank*,
Σ, a subset of Γ not including b, is the set of *input symbols*,
δ is the *next move function*, a partial mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$,
$q_0 \in Q$ is the *start state,*
$F \subseteq Q$ is the set of *final states.*

Remark. For the sake of halting computations, it is desirable that δ is undefined for some arguments. We assume that Q and Γ are disjoint. Moreover, note that L and R are intended to mean "move left" and "move right," respectively.

We denote an *instantaneous description* (ID) of the Turing machine M by $\alpha_1 q \alpha_2$ with three provisos:

1. The current state q of M has to be in Q.
2. $\alpha_1\alpha_2$ is the string in Γ^* that is the contents of the tape up to the rightmost nonblank symbol or the symbol to the left of the head, whichever is rightmost.
3. The tape head is assumed to be scanning the leftmost symbol of α_2, or if $\alpha_2 = \epsilon$ (the empty string), then the head is scanning a blank.

We define a *move* of M as follows (also called a *computational step* in the present article). Let $X_1X_2 \cdots X_{i-1}qX_i \cdots X_n$ be an ID. We distinguish between two cases.

1. Suppose $\delta(q, X_i) = (p, Y, L)$, where if $i - 1 = n$, then X_i is taken to be the blank b. We distinguish between two subcases:
   (a) $i = 1$
   There is no next ID, as the tape head is not allowed to fall off the left end of the tape.
   (b) $i > 1$
   Then we write:
   $X_1X_2 \cdots X_{i-1}qX_i \cdots X_n \vdash_M X_1X_2 \cdots X_{i-2}pX_{i-1}YX_{i+1} \cdots X_n$
   However, if any suffix of $X_{i-1}YX_{i+1} \cdots X_n$ is completely blank, that suffix is deleted.
2. Suppose $\delta(q, X_i) = (p, Y, R)$. Then we write:
   $X_1X_2 \cdots X_{i-1}qX_iX_{i+1} \cdots X_n \vdash_M X_1X_2 \cdots X_{i-1}YpX_{i+1} \cdots X_n$
   Note that in the case $i - 1 = n$, the string $X_iX_{i+1} \cdots X_n$ is empty, and the right side of $\vdash_M$ is longer than the left side of $\vdash_M$.

If two ID's are related by $\vdash_M$, we say that the second results from the first by one move. If one ID results from another by some finite number of moves, including zero moves, they are related by the symbol $\vdash_M^*$.

Following Davis to some extent [16, Ch.1], we define various notions of computation.

Definition 2. An ID τ_1 is called *terminal* with regard to DTM M if for no ID τ_2 do we have $\tau_1 \vdash_M \tau_2$. By a *finite computation* of DTM M is meant a finite sequence $\tau_1, \tau_2, \ldots, \tau_p$ of ID's such that $\tau_i \vdash_M \tau_{i+1}$ for $1 \leq i < p$. By a *terminating* or *halting computation* of DTM M is meant a finite computation $\tau_1, \tau_2, \ldots, \tau_p$ of M in which the last ID τ_p is terminal with regard to M. We write $\tau_p = Res_M(\tau_1)$ and we call τ_p the *result* of τ_1 with regard to M. By an *infinite computation* of M is meant an infinite sequence $\tau_1, \tau_2, \ldots$ of ID's such that $\tau_i \vdash_M \tau_{i+1}$ for all $i \geq 1$. In this case we also say that M does not halt. By a *computation* of M is meant either a finite computation or an infinite computation of M.

2.3 Functions implemented by Turing Machines.

We denote a function f from domain X to co-domain Y by $f :: X \to Y$. For every $x \in X$, there exists $y \in Y$ such that $f(x) = y$. When we write $f :: X \to Y \cup \{\bot\}$, with $\bot \notin Y$, we have: for every $x \in X$, there exists either $y \in Y$ or $y = \bot$ such that $f(x) = y$. We say that two functions $f, g :: X \to Y \cup \{\bot\}$ are *equal*, denoted $f = g$, when either $f(x) = g(x) \in Y$ or $f(x) = g(x) = \bot$, for all $x \in X$.

Remark. We often write $Y_\bot$ as shorthand for $Y \cup \{\bot\}$.

Definition 3. A function $f :: X \to Y$ is an *injection* (also called *one-to-one*) when, for any $x_1, x_2 \in X$, $x_1 \neq x_2$ implies $f(x_1) \neq f(x_2)$. A function $f :: X \to Y$ is a *bijection* when f is an injection and $Y = \{f(x) \mid x \in X\}$.

Definition 4. We say that DTM M *(mathematically) implements* function $f :: \mathbb{N} \to \mathbb{N}_\bot$ when the following two equivalences hold, for $\forall x, y \in \mathbb{N}$:
(1) $f(x) = y$ if and only if M on $enc(x)$ halts with output $enc(y)$.
(2) $f(x) = \bot$ if and only if M on $enc(x)$ does not halt.
We say that *DTM M computes function $f :: \mathbb{N} \to \mathbb{N}_\bot$ in the limit*, when M implements f.

Remark 5. When DTM M implements $f :: \mathbb{N} \to \mathbb{N}_\bot$, then we can also say that DTM M *partially computes* $\tilde{f}$, with total function f and partial function $\tilde{f}$ on $\mathbb{N}$ closely related in the following way:
$\tilde{f}(x) = f(x)$, when $f(x) \in \mathbb{N}$
x does not belong to the domain of $\tilde{f}$, when $f(x) = \bot$.

Lemma 6. *Given two functions $f, g :: \mathbb{N} \to \mathbb{N}_\bot$ and some DTM M that implements both f and g. Then $f = g$.*

Proof. Take any DTM M that implements both functions f and g. Suppose $f \neq g$; that is, there exists some natural number x for which $f(x) \neq g(x)$. Without loss of generality, $f(x) = y \in \mathbb{N}$. Then M on $enc(x)$ halts and outputs $enc(y)$.
Case 1: Suppose $g(x) = y' \in \mathbb{N}$. Then M on $enc(x)$ halts and outputs $enc(y')$, with $y \neq y'$. But then $enc(y) \neq enc(y')$. Contradiction.
Case 2: Suppose $g(x) = \bot$. Then M on $enc(x)$ does not halt. Contradiction.
Therefore, $f = g$. □

Remark 7. Encoding function *enc* is introduced in the next section. Note that *enc* has to be injective for the previous proof to be correct.

Definition 8. We say that DTM's M_1 and M_2 are *functionally equivalent* when there exists a function $f :: \mathbb{N} \to \mathbb{N}_\bot$ such that both M_1 and M_2 implement f.

2.4 Encodings

The notation $\langle\cdot\rangle$ denotes some standard, bijective encoding of $\mathbb{N}^n$ into $\mathbb{N}$. Frequently we use $\langle\cdot\rangle$ as a pairing function to associate a unique number $\langle x, y\rangle$ with each pair of numbers (x, y). With abuse of notation, we also write $\langle x, y\rangle$ where x and y are strings instead of natural numbers. Likewise, we write $\langle M, w\rangle$ to denote either the Gödel number n or the corresponding string (which encodes that number n), where M is a description of a Turing machine and w typically serves as input string for M.

We are also interested in encodings *enc* from $\mathbb{N}_\perp$ to strings over the finite alphabet Σ; that is, $enc :: \mathbb{N}_\perp \to \Sigma^*$. Any standard, injective encoding is acceptable with the proviso that the empty string ϵ is the encoding of $\perp$; that is, $\epsilon = enc(\perp)$.

Remark 9. If the reader prefers $enc(\perp)$ not to be ϵ, but, rather, some string of length $l > 0$, then the proofs in the sequel will have to be modified accordingly. In particular, when seeking some string different from $enc(\perp)$, the reader can take 0^{l+1}, but not 0, as a candidate for the proofs to carry through.

Following Christos Papadimitriou [46, p.57] to some extent, without giving all the obvious details, we obtain the following definition:

Definition 10. A *universal (deterministic) Turing machine U* (abbreviated UTM U), when provided an input, interprets this input as the description of another DTM M, concatenated with the description of an input to M, say w. U simulates M's computation on input w. We write: $U(\langle M, w\rangle) = M(w)$.

3 From Preliminaries to Alternative Deterministic Machines

In the previous section I have mainly appropriated part of Hopcroft and Ullman's formal exposition (1979), and to a lesser extent that of Davis (1958). Besides defining the notions of "deterministic Turing machine" (DTM) and "computation," I have explicitly distinguished between:

- Partial functions on the naturals.
- Deterministic Turing machines that implement such functions.

Given some fixed, standard encoding function $enc :: \mathbb{N}_\perp \to \Sigma^*$, Lemma 6 tells us that each DTM M (which works with alphabet Σ) implements at most one function $f :: \mathbb{N} \to \mathbb{N}_\perp$. Now, given an arbitrary DTM M, may we also assert that M implements at least one function $f :: \mathbb{N} \to \mathbb{N}_\perp$? The answer is affirmative and the corresponding (trivial) proof is omitted. To recapitulate, then, we have the following result:

> Each DTM M implements precisely one function $f :: \mathbb{N} \to \mathbb{N}_\perp$.

However, and as is common in classical computability theory, the proviso at all times is that encoding function *enc* is fixed from the outset.

In the following section I tweak the previous definitions of "deterministic Turing machine" and "computation" in order to formally present the ADM model of computation. Definition 11 of an ADM N is key, while the details of the follow-up "move" relation, $\vdash_N$, will be of less concern to casual readers. ADM's are discussed in terms of language recognition (Section 4.1) and computers of functions (Section 4.2). Concerning computers, Definitions 17 and 18 stand out:

- Definition 17 introduces the notion of a converging sequence
  $S :: g_1, g_2, g_3, \ldots, f$ of functions, relative to encoding function *enc*.

- Each such sequence S may, or may not, be implementable by an ADM, as elaborated in Definition 18.

An understanding of these two definitions will allow the reader to easily grasp Definition 19; that is, the notion of some ADM N *computing* the aforementioned function f *in the limit.*

Another crucial, albeit trivial, result is Theorem 23, which states that every DTM computable function (on the naturals) is also ADM computable. That is, DTM's are not more powerful than ADM's.

4 Alternative Deterministic Machines (ADM's)

Concerning Alternative Deterministic Machines, two observations come in handy in preparation for the formal treatment.

1. When ADM N halts (on a given input $w \in \Sigma^*$), it signifies this event by printing out the end marker \$. When N has not (yet) halted, the output tape contains no end marker \$ and the already printed output $\alpha_1, \alpha_2, \ldots, \alpha_p$ (for some $p \in \mathbb{N}$) is a prefix of N's final output $\alpha_1, \alpha_2, \ldots, \alpha_p, \alpha_{p+1}, \ldots$, which, in the limit, could be either finite or infinite in length.
2. ADM N has two tapes: a read-write tape and an output tape, also called the *first* tape and the *second* tape, respectively.
   (a) The head of the first tape can move left (L) or right (R).
   (b) The head of the second tape can only move right (R) or remain idle (I); that is, not move.

Now, to keep the formal exposition concise, we refine the previous statement about printing symbols in the following way:

1. ADM N either prints the blank b, with $b \in \Gamma$, on the output tape and the tape's head remains idle (I).
2. Or, ADM N prints any nonblank symbol $\alpha \in \{\$\} \cup \Gamma \setminus \{b\}$ on the output tape and the tape's head moves one cell to the right (R).

Technically, then, N produces a symbol on its output tape during every computational step. However, N progresses one cell to the right on its output tape if and only if it has just printed a nonblank symbol (on the output tape). For completeness' sake we also explicitly remark that when ADM N prints the end marker \$, the head of the output tape moves one cell to the right.

Following Hopcroft & Ullman [32, p.148] to some extent, we can now formally define an ADM.

Definition 11. An *Alternative Deterministic Machine* N (abbreviated ADM) is denoted
$N = (Q, \Sigma, \Gamma, \delta, q_0, b, \$)$
where
Q is the finite set of *states.*
Γ is the finite set of allowable *work tape symbols.*
b, with $b \in \Gamma$, is the *blank.*
Σ, with $b \notin \Sigma \subset \Gamma$, is the set of *input symbols.*
\$, with $\$ \notin \Gamma$, is the *end marker.*
δ is the *next move function*, a partial mapping
from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\} \times \Gamma \cup \{\$\} \times \{R, I\}$;
that is, δ may be undefined for some arguments.
$q_0 \in Q$ is the *start state.*

Remark. It is important to note that ϵ denotes the string of length zero. Any ADM N initially contains a blank output tape; that is, a tape filled with infinitely many b symbols, containing not a single symbol from $(\Gamma \cup \{\$\}) \setminus \{b\}$. The output tape is then said to "contain ϵ as output."

We denote an *instantaneous description* (ID) of ADM N by $\alpha_1 q \alpha_2 \parallel \alpha_3$ with three provisos:

1. The current state q of N has to be in Q.
2. Concerning the first tape:
   (a) $\alpha_1\alpha_2$ is the string in Γ^* that is the contents of the first tape up to the rightmost nonblank symbol or the symbol to the left of the head, whichever is rightmost.
   (b) The head of the first tape is assumed to be scanning the leftmost symbol of α_2, or if $\alpha_2 = \epsilon$, then the head is scanning a blank.
3. Concerning the second tape:
   (a) α_3 is the string in $(\Gamma \cup \{\$\})^*$ that is the contents of the second tape.
   (b) The head of the second tape is assumed to be scanning a blank b, with:
      i. Either b appearing as the last symbol in α_3.
      ii. Or, if the previous case does not apply, b appearing as the leftmost blank on the tape following α_3.

Remark. We use an underscore to denote a "don't care" value. For instance, we write $\alpha_1 q \alpha_2 \parallel _$ to denote an ID in which we intentionally do not specify the second tape. Likewise, we write $_ \parallel \alpha_3$ to denote an ID in which we don't bother to describe the machine's control (q), nor the contents $(\alpha_1\alpha_2)$ on the first tape, nor the position of the head $(\alpha_1 q)$ of the first tape.

Inspired by Hopcroft & Ullman [32, p.149], we define a *move* of N as follows (also called a *computational step*). Let $X_1X_2 \cdots X_{i-1}qX_i \cdots X_n \parallel Y_1Y_2 \cdots Y_m$, with $Y_m \neq \$$, be an ID. We distinguish between two orthogonal concerns: 1 and 2.

1. For the control and the first tape, taken together, we have to consider two cases:
   (a) Suppose $\delta(q, X_i) = (p, Z, L, _, _)$, where if $i - 1 = n$, then X_i is taken to be the blank b. We distinguish between two subcases:
      i. $i = 1$
      There is no next ID, as the tape head is not allowed to fall off the left end of the tape.
      ii. $i > 1$
      Then we write:
      $X_1X_2 \cdots X_{i-1}qX_i \cdots X_n \parallel Y_1Y_2 \cdots Y_m$
      $\vdash_N$
      $X_1X_2 \cdots X_{i-2}pX_{i-1}ZX_{i+1} \cdots X_n \parallel _$
      However, if any suffix of $X_{i-1}ZX_{i+1} \cdots X_n$ is completely blank, that suffix is deleted.
   (b) Suppose $\delta(q, X_i) = (p, Z, R, _, _)$. Then we write:
   $X_1X_2 \cdots X_{i-1}qX_iX_{i+1} \cdots X_n \parallel Y_1Y_2 \cdots Y_m$
   $\vdash_N$
   $X_1X_2 \cdots X_{i-1}ZpX_{i+1} \cdots X_n \parallel _$

2. For the second tape, we have three cases to consider:

   (a) Suppose $\delta(q, X_i) = (_, _, _, b, I)$.
   Then we write:
   $X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \parallel Y_1 Y_2 \cdots Y_m$
   $\vdash_N$
   $_ \parallel Y_1 Y_2 \cdots Y_m$

   (b) Suppose $\delta(q, X_i) = (_, _, _, \$, R)$.
   Then we write:
   $X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \parallel Y_1 Y_2 \cdots Y_m$
   $\vdash_N$
   $_ \parallel Y_1 Y_2 \cdots Y_m \$$

   (c) Suppose $\delta(q, X_i) = (_, _, _, Z, R)$ with $Z \in \Gamma \setminus \{b, \$\}$.
   Then we write:
   $X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \parallel Y_1 Y_2 \cdots Y_m$
   $\vdash_N$
   $_ \parallel Y_1 Y_2 \cdots Y_m Z$

Remark. "$\tau_1 \vdash_N _ \parallel \alpha_3$" is shorthand for "there exists α_1, q, α_2 such that $\tau_1 \vdash_N \alpha_1 q \alpha_2 \parallel \alpha_3$". Likewise, "$\tau_1 \vdash_N \alpha_1 q \alpha_2 \parallel _$" is shorthand for "there exists α_3 such that "$\tau_1 \vdash_N \alpha_1 q \alpha_2 \parallel \alpha_3$".

If two ID's are related by $\vdash_N$, we say that the second results from the first by one move. If one ID results from another by some finite number of moves, including zero moves, they are related by the symbol $\vdash_N^*$. We write $\vdash_N^m$, for some specific $m \in \mathbb{N}$, to denote precisely m moves, with $\vdash_N$ as shorthand for $\vdash_N^1$.

Originally inspired by Davis [16, Ch.1], we provide the following definitions.

Definition 12. An ID τ_1 is called *terminal* with regard to ADM N if τ_1 is of the following form: $_ \parallel Y_1 Y_2 \cdots Y_m \$$, for some $m \in \mathbb{N}$. By a *finite computation* of ADM N is meant a finite sequence $\tau_1, \tau_2, \ldots, \tau_p$ of ID's such that $\tau_i \vdash_N \tau_{i+1}$ for $1 \leq i < p$. By a *terminating* or *halting computation* of ADM N is meant a finite computation $\tau_1, \tau_2, \ldots, \tau_p$ of N in which the last ID τ_p is terminal (with regard to N). We then write $\tau_p = Res_N(\tau_1)$ and we call τ_p the *result* of τ_1 with regard to N. By an *infinite computation* of N is meant an infinite sequence $\tau_1, \tau_2, \ldots$ of ID's such that $\tau_i \vdash_N \tau_{i+1}$ for all $i \geq 1$. In this case we also say that N does not halt. By a *computation* of N is meant either a finite computation or an infinite computation of N.

Remark 13. Given an ADM N and some input w. It is possible that the complete computation of N on input w is a finite computation but not a terminating computation. Concretely: N's run on w eventually halts, but N has not printed the end marker \$ on its output tape. With that said, one can construct a DTM M^* that takes a description of any ADM N and outputs a description of a similar ADM N', with N' having complete computations that are finite if and only if they are terminating. Without loss of generality, from now on we assume that any ADM N halts if and only if \$ appears on N's output tape.

4.1 The ADM as a Language Recognizer

If an ADM were merely a DTM, then the language accepted by ADM N, denoted $L(N)$, would be the set of those words w in Σ^* that cause N to eventually print \$ when w is placed, justified at the left, on the first tape of N, with N in start state $q_0 \in Q$, and the heads of both tapes of N at

their leftmost positions. Formally, the language accepted by ADM $N = (Q, \Sigma, \Gamma, \delta, q_0, b, \$)$ would then be $L(N) = \{w \mid w \in \Sigma^* \text{ and } q_0 w \parallel \epsilon \vdash_N^* \alpha_1 p \alpha_2 \parallel \alpha_3\$ \text{ with } p \in Q \text{ and } \alpha_1, \alpha_2, \alpha_3 \in \Gamma^*\}$. This definition poorly reflects the intuition provided in Section 1. Infinite computations of a specific brand need to be incorporated; i.e., infinite computations for which the output does not change any more after a finite number of moves. That is, we are interested in all words $w \in \Sigma^*$ that lead to computations (be it finite or infinite computations) for which the output becomes fixed after a finite number of computational steps. To recapitulate, even if ADM N on w does not halt, if the corresponding output is fixed from some computational step onward, then w should be in $L(N)$. All this, to prepare the following definition.

Definition 14. The *language accepted by ADM N*, denoted $L(N)$, is the set of those words w in Σ^* that cause N in the limit to print precisely n output symbols in Γ for any $n > 0$, when w is placed, justified at the left, on the first tape of N, with N in start state $q_0 \in Q$, and the heads of both tapes of N at their leftmost positions. Formally, the language accepted by ADM $N = (Q, \Sigma, \Gamma, \delta, q_0, b, \$)$ is
$L(N) = \{w \mid w \in \Sigma^* \text{ and } \exists \alpha \in \Gamma^+.\ \exists \text{m} \in \mathbb{N}.\ \forall \text{m'} \in \mathbb{N}.\ \ E \text{ or } F\}$, with:
$E =$ “ $q_0 w \parallel \epsilon \vdash_N^m _ \parallel \alpha\$$ ”
$F =$ “ $q_0 w \parallel \epsilon \vdash_N^m _ \parallel \alpha \vdash_N^{m'} _ \parallel \alpha$ ”

Remark 15. Consider a universal, deterministic Turing machine $\tilde{U}$. Consider ADM U that on input $\langle M^*, w^* \rangle$ simulates DTM M^* on input w^* just like $\tilde{U}$ does on input $\langle M^*, w^* \rangle$. Suppose the computation of M^* on w^* is infinite. Then M^* on w^* will never output anything, nor will $\tilde{U}$ when simulating M^* on w^*. In turn, ADM U will forever have the empty string ϵ as output, since U's simulation never terminates. But, the reader might now ask, what if some DTM M on some input w halts with ϵ as output? This is not possible in the present setup: let $enc(x) = w$, then $f(x) = y$ for some $y \in \mathbb{N}$, and $\forall y \in \mathbb{N}.\ enc(y) \neq \epsilon$. In other words, Turing machines (and ADM's) that halt with empty output (ϵ) are not considered to be relevant in the present setup.

4.2 The ADM as a Computer of Functions

A sequence S of functions from $\mathbb{N}$ to $\mathbb{N}$ is denoted by $S :: g_1, g_2, g_3, \ldots$ A sequence S of functions from $\mathbb{N}$ to $\mathbb{N}$ with a limit f is denoted by $S :: g_1, g_2, g_3, \ldots f$.

Definition 16. Two sequences $S :: g_1, g_2, g_3, \ldots f$ and $S' :: g'_1, g'_2, g'_3, \ldots f'$ are said to be *equivalent*, denoted $S \equiv S'$, when $g_i = g'_i$ for all $i \geq 1$ and $f = f'$.

Definition 17. A *converging sequence* $S :: g_1, g_2, g_3, \ldots$ *of functions* $g_1, g_2, g_3, \ldots$ *from* $\mathbb{N}$ *to* $\mathbb{N}$, *relative to enc, and with limit function* $f :: \mathbb{N} \to \mathbb{N}_\perp$ — denoted $S :: g_1, g_2, g_3, \ldots, f$ — is a sequence with the following properties, for $\forall x \in \mathbb{N}$:
(1) $\forall i > 0.\ [\ enc(g_i(x)) \preceq enc(g_{i+1}(x))\ ]$
(2) $\forall y \in \mathbb{N}.\ [\ f(x) = y$ implies (2a) and (2b)]
(2a) $\forall i > 0.\ [\ enc(g_i(x)) \preceq enc(y)\ ]$
(2b) If $enc(g_i(x)) \prec enc(y)$, for some $i \geq 1$,
then there exists a larger $j > i$, such that: $enc(g_i(x)) \prec enc(g_j(x))$

Remark. When unspecified, quantification is taken to be over the set $\mathbb{N}$ of natural numbers. We frequently write “iff” to abbreviate “if and only if.”

Definition 18. We say that ADM N *(mathematically) implements* a converging sequence $S :: g_1, g_2, g_3, \ldots, f$, with functions g_i from $\mathbb{N}$ to $\mathbb{N}$ and function f from $\mathbb{N}$ to $\mathbb{N}_\perp$, when the following three conditions hold, for all $x \in \mathbb{N}$:
(1) $\forall y.\ [\ f(x) = y$ iff $\exists t_0.\ \forall t \geq t_0.\ g_t(x) = y\ ]$

(2) $f(x) = \bot$ iff $\forall y.\, f(x) \neq y$
(3) $\forall t, y.$ [$g_t(x) = y$ iff N on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after t moves]

Remark. Condition (1) is called *convergence* and (2) is called *nonconvergence*.

Definition 19. We say that *ADM N computes function $f :: \mathbb{N} \to \mathbb{N}_\bot$ in the limit*, when N implements a converging sequence $S :: g_1, g_2, g_3, \ldots, f$ of functions $g_1, g_2, g_3, \ldots$ with limit function f.

Remark. What would it mean if some ADM N computes a function $f :: \mathbb{N} \to \mathbb{N}$ in the limit instead of $f :: \mathbb{N} \to \mathbb{N}_\bot$? Then N prints in the limit a finite, nonempty output on input $enc(x)$, for any $x \in \mathbb{N}$. Indeed, $f :: \mathbb{N} \to \mathbb{N}$ is merely a special case of $f :: \mathbb{N} \to \mathbb{N}_\bot$.

Definition 20. We say that two ADM's N and N' are *functionally equivalent (with regard to function $f :: \mathbb{N} \to \mathbb{N}_\bot$)* when N implements some converging sequence $S :: g_1, g_2, g_3, \ldots, f$ and N' implements some converging sequence $S' :: g'_1, g'_2, g'_3, \ldots, f$', with $f = f'$.

Lemma 21. *Given two converging sequences of functions from $\mathbb{N}$ to $\mathbb{N}$, called S and S', and some ADM N that implements both S and S'. Then $S \equiv S'$.*

Proof. Consider an arbitrary ADM N that implements both sequences:
$S :: g_1, g_2, g_3, \ldots, f$ and $S' :: g'_1, g'_2, g'_3, \ldots, f'$. Suppose it is not the case that $S \equiv S'$.

A first possibility is that $f \neq f'$; i.e., there exists a natural number x for which $f(x) \neq f'(x)$. Without loss of generality, $f(x) = y \in \mathbb{N}$. Then, by Definition 18, $\exists t_0.\, \forall t \geq t_0.\, g_t(x) = y$. We distinguish between two cases, each case results in a contradiction.
Case 1: $f'(x) = y' \in \mathbb{N}$. Then, by Definition 18, $\exists t'_0.\, \forall t' \geq t'_0.\, g_{t'}(x) = y' \neq y$.
Case 2: $f'(x) = \bot$. Then, by Definition 18, $\forall y.\, \forall t_0.\, \exists t \geq t_0.\, g_t(x) \neq y$.

A second possibility is that $g_i \neq g'_i$, for some $i \geq 1$. This means, that, for a specific $x \in \mathbb{N}$, we have: $y = g_i(x) \neq g'_i(x) = y'$, with $y, y' \in \mathbb{N}$. On the one hand, from Definition 18 we have: N on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after i moves. On the other hand, from Definition 18 we have: N on $enc(x)$ has $enc(y')$ or $enc(y')\,\$$ as output after i moves. But, $enc(y)$ differs both from $enc(y')$ and $enc(y')\,\$$. Likewise, $enc(y)\,\$$ differs both from $enc(y')$ and $enc(y')\,\$$. So we have our desired contradiction.

The conclusion is that $S \equiv S'$. □

Definition 22. We say that DTM M and ADM N are *functionally equivalent* when M computes f in the limit and N computes f in the limit, for some function $f :: \mathbb{N} \to \mathbb{N}_\bot$.

Theorem 23. *Each function $f :: \mathbb{N} \to \mathbb{N}_\bot$ that is DTM computable in the limit, is also ADM computable in the limit.*

Proof. Consider some DTM M that computes some function $f :: \mathbb{N} \to \mathbb{N}_\bot$ in the limit; i.e., M implements f. Then, we have:
(a) $\forall y.$ [$f(x) = y \in \mathbb{N}$ iff M on $enc(x)$ halts with output $enc(y)$]
(b) $f(x) = \bot$ iff M on $enc(x)$ does not halt

We have to construct a functionally equivalent ADM N, with, for all $x \in \mathbb{N}$:
(1) $\forall y.$ [$f(x) = y$ iff $\exists t_0.\, \forall t \geq t_0.\, g_t(x) = y$]
(2) $f(x) = \bot$ iff $\forall y.\, f(x) \neq y$
(3) $\forall t, y.$ [$g_t(x) = y$ iff N on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after t moves]

Combining (a-b) and (1-3), we have the following requirements for N:
+ $\forall y.$ [M on $enc(x)$ halts with output $enc(y)$ iff $\exists t_0.\, \forall t \geq t_0.\, g_t(x) = y$]
+ M on $enc(x)$ does not halt iff $\forall y.\, f(x) \neq y$
+ $\forall t, y.$ [$g_t(x) = y$ iff N on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after t moves]

So, we have the following requirements for N:
+ $\forall y.$ [M on $enc(x)$ halts with output $enc(y)$ iff $\exists t_0. \forall t \geq t_0. N$ on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after t moves]
+ M on $enc(x)$ does not halt iff $\forall y. \forall t_0. \exists t \geq t_0. N$ on $enc(x)$ does not have $enc(y)$ nor $enc(y)\$$ as output after t moves

Now we construct ADM N as follows. ADM N has DTM M hardcoded. N on input $enc(x)$ uses its first tape to simulate M on $enc(x)$. We distinguish between two cases.
Case 1: If the simulation stops,
then N copies M's output $enc(y)$ to its second tape, prints \$ and halts.
Case 2: If the simulation goes on forever,
then N's second tape contains ϵ forever, and N does not halt. □

Remark. The previous proof relies on the choices made in Section 2.4 and, specifically, on the following property: $\epsilon \neq enc(y)$ and $\epsilon \neq enc(y)\$$, for all $y \in \mathbb{N}$.

Definition 24. A *universal ADM U*, when provided an input, interprets this input as the description of another ADM N, concatenated with the description of an input to N, say $enc(x)$, with $x \in \mathbb{N}$. ADM U simulates N's computation on input $enc(x)$ such that: $f_U(\langle N, x\rangle) = f_N(x)$, with f_U and f_N implemented by U and N, respectively.

Remark. As is common in computability theory, we have fused three categories:
(1) "ADM N" in plain English
(2) "the description of ADM N" in language Σ^*
(3) the Gödel-number encoding of "the description of ADM N" in Σ^*.

5 From ADM's to a Comparison Between DTM's and ADM's

The previous section contains a detailed setup of the ADM model of computation. The crux is Definitions 17, 18 and 19 in concert. Suppose ADM N computes function $f :: \mathbb{N} \to \mathbb{N}_\perp$ in the limit. Then, N implements a converging sequence $S :: g_1, g_2, g_3, \ldots, f$. Only limit function f is partial, while $g_1, g_2, g_3, \ldots$ are total functions, mapping naturals to naturals.

Now, consider an arbitrary natural x with $w = enc(x)$. If N on input word w does not converge, then $f(x) = \perp$. On the other hand, if N on input w does converge, then it prints a finite number n of output symbols in the limit, with $n > 0$. We then write: $f(x) = y$, with output word $w' = enc(y)$ and $|w'| = n$. Convergence on input x implies that from some moment t_0, we have: $g_{t_0}(x) = g_{t_0+1}(x) = g_{t_0+2}(x) = \ldots = y$.

Perhaps an intuitive way to understand, for a given input x, the sequence S of functions $g_1, g_2, g_3, \ldots, f$, is as follows: Associate an Observer i with each function g_i, and associate a God-like Observer G with limit function f. Observer i takes a camera snapshot of machine N's output tape after i computational steps. And God G does the same at infinity. Loosely speaking, convergence implies that G's snapshot, which is either $enc\,(y)$ or $enc\,(y)\,\$$, is also observable by all but finitely many Observers i.

The next section contains the main results of this article: Theorem 25, Corollary 26, and Theorem 29. The overarching idea is relatively simple:

> An ADM V can simulate an arbitrary DTM M on arbitrary input w. Before starting the simulation, V prints the symbol 0 on its output tape. On the one hand, if the simulation does not terminate, then V has printed nonempty output (i.e., symbol 0 and nothing more), which differs from M's output (i.e., the empty string). On the other hand, if the simulation stops, then M can print one more output symbol

(i.e., 0 or 1) such that its entire output differs from M's output. The net result is that V computes a function on the naturals that differs from every DTM-computable function.

Another way to appreciate the power of ADM's is to zoom in on the distinction between a "computably enumerable" set of naturals and the new form of enumeration demonstrated by ADM D in the proof of Theorem 29. The remark following the proof highlights the conceptual difference between both forms of set enumeration.

6 DTM's Are Computationally Inferior to ADM's

We shall now prove the central result of this article: there is an an ADM that computes a function on $\mathbb{N}$ in the limit that no DTM computes in the limit.

Remark. For the sake of presenting concrete arguments, the following proofs implicitly rely on $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, b\}$ but in no essential way. Furthermore, we implicitly rely on a canonical enumeration of DTM descriptions M_1, M_2, ... and, likewise, on an enumeration of input words w_1, w_2, ... Specifically, M_d and w_d denote the d-th item in each enumeration, respectively.

Theorem 25. *There is a function $f_V :: \mathbb{N} \to \mathbb{N}_\perp$ that (i) some ADM V computes in the limit, and (ii) no DTM computes in the limit.*

Proof. Starting with part (i) of the theorem, we construct ADM V as follows. ADM V, on input w, first checks whether w is $\langle M_d, w_d \rangle$ for some $d \geq 1$, with DTM M_d and input word w_d. If this check does not pass, then ADM V prints infinitely many times the symbol 0; i.e., ADM V demonstrates nonconverging behavior. Else, ADM V on input $\langle M_d, w_d \rangle$ prints 0 on its output tape. Then, ADM V simulates DTM M_d on w_d. Two cases can be distinguished:
Case 1: M_d on w_d halts and outputs w'_d. Then, ADM V prints 0 or 1 on its output tape, such that its total output (00 or 01) differs from w'_d. V then prints \$ and halts.
Case 2: M_d on w_d does not halt and thus outputs nothing. In this case, ADM V merely simulates M_d on w_d forever. Note, however, that V's output tape does contain a string, 0, which differs from ϵ, i.e. the string representing $\perp$.

Having constructed ADM V, we now use Definitions 18 and 19 to specify function f_V that ADM V computes in the limit. ADM V implements a converging sequence $S :: g_1, g_2, g_3, \ldots, f_V$ with the following properties, for all $x \in \mathbb{N}$:
(1) $\forall y.\ [\ f_V(x) = y$ iff $\exists t_0. \forall t \geq t_0. g_t(x) = y\ ]$
(2) $f_V(x) = \perp$ iff $\forall y. f_V(x) \neq y$
(3) $\forall t, y.\ [\ g_t(x) = y$ iff V on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after t moves]
Clearly, for any well-formed input $\langle M_d, w_d \rangle$, ADM V prints a finite number n of symbols on its output tape, with $n > 0$. Therefore, condition (2) does not apply in the intended case; i.e., when x is such that $enc(x) = \langle M_d, w_d \rangle$ for some $d \geq 1$. Expressing (1) in terms of (3) results in the following property: for all intended $x \in \mathbb{N}$, and all $y \in \mathbb{N}$, we have that: $f_V(x) = y$ iff $\exists t_0. \forall t \geq t_0. V$ on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after t moves. In more plain English: function f_V maps any natural number x representing a DTM M_d and input w_d onto a number y, where $enc(y) \in \{0, 00, 01\}$ is different from any output produced by M_d on w_d.

We now come to part (ii) of the theorem: we show that function f_V cannot be computed in the limit by any DTM. Suppose that some DTM $\tilde{V}$ computes $f_V :: \mathbb{N} \to \mathbb{N}_\perp$ in the limit. By Definition 4, we have, for $\forall x$:
(a) $\forall y.\ [\ f_V(x) = y$ iff DTM $\tilde{V}$ on $enc(x)$ halts with output $enc(y)$]
(b) $f_V(x) = \perp$ iff DTM $\tilde{V}$ on $enc(x)$ does not halt.

We focus on (a); that is, we need only consider natural numbers x for which $enc(x) = \langle M_d, w_d \rangle$ in order to obtain our contradiction. From (a) we obtain the following: $f_V(x) = y$ iff DTM $\tilde{V}$ on $\langle M_d, w_d \rangle$, with $enc(x) = \langle M_d, w_d \rangle$, halts with output $enc(y) \in \{0, 00, 01\}$, which is different from the output produced by M_d on w_d. But, no DTM can in general decide for input $\langle M_d, w_d \rangle$ whether M_d on w_d terminates or not. Yet, DTM $\tilde{V}$ needs this information to output a string that M_d on w_d does not produce itself. □

Corollary 26. *If the model fidelity of ADM's is at least as good as that of DTM's, then the notion of algorithm is not fully captured by DTM's.*

Remark. The "model fidelity" refers to computing in the real world, and to human computing in particular. Arguments supporting the corollary's precondition are provided in Sections 1 and 11.

Definition 27. Let S be a set of natural numbers. The *characteristic function of* S, denoted χ_S, maps natural numbers to 0 or 1 as follows:
$\chi_S(n) = 0$ if $n \notin S$
$\chi_S(n) = 1$ if $n \in S$
A *generalized characteristic function of* S, denoted $\tilde{\chi}_S$, maps natural numbers to natural numbers with the proviso that:
$\tilde{\chi}_S(n) = 0$ iff $n \notin S$

Remark. A generalized characteristic function $\tilde{\chi}_S$ is not necessarily a characteristic function χ_S, for any fixed set S under consideration.

Corollary 28. *Suppose number* 0 *is encoded with string* 0*, and some other number with string* 0σ *and* $\sigma \neq \epsilon$*. Then, for each computably enumerable set* S*, there is an ADM* N *that computes a generalized characteristic function* $\tilde{\chi}_S$ *in the limit.*

Proof. Follows from (a) the construction of ADM V in the proof of Theorem 25, and from (b) the ability of an ADM to first compute 0, with $0 = enc(0)$, and subsequently — if need be — to print out a particular extension of 0. The latter, in turn, serves as the encoding of some number $n \neq 0$. □

Theorem 29. *There exists an ADM* D *that generates a non c.e. set* A*.*

Remark. The abbreviation "c.e." stands for "computably enumerable," and its definition is assumed [32, Sec.7.7]. Metaphorically speaking we shall use the phrase "$cell(j, k)$ contains ..." in the proof below, instead of writing "$cell(j, k)$ denotes ...".

Proof. Consider an enumeration of input words w_1, w_2, ..., and an enumeration of descriptions of DTM's M_1, M_2, ... The computation of DTM M_j on input w_k is either finite or infinite:
(a) In the finite case, let $cell(j, k)$ contain M_j's output word w'_k with regard to its computation on input w_k.
(b) In the infinite case, let $cell(j, k)$ contain $enc(\perp)$; that is, the empty string ϵ.

We shall construct a diagonalizer, ADM D. The construction relies on dovetailing and proceeds in stages i, with $i = 1, 2, \ldots$ Diagonalizer D will operate across the diagonal (M_1, w_1), (M_2, w_2), ... in order to provide the contents of the cells on its own a-diagonal ("antagonist diagonal"): $Dcell(1, 1)$, $Dcell(2, 2)$, ... We will show, for any $d \geq 1$, that the contents of $Dcell(d, d)$ will, from some stage onward, be fixed and different from the contents of $cell(d, d)$.

For any $d \geq 1$, Diagonalizer D stores the word 0 in $Dcell(d, d)$ when consulting this cell for the first time.

Stage i. Diagonalizer D uses $\langle M_1, w_1 \rangle$, $\langle M_2, w_2 \rangle$, ..., $\langle M_{i-1}, w_{i-1} \rangle$, and $\langle M_i, w_i \rangle$ for, respectively, i, $i - 1$, ..., 2, and 1 simulation moves in order to possibly modify the contents of

$Dcell(1,1)$, $Dcell(2,2)$, ..., $Dcell(i-1,i-1)$, and $Dcell(i,i)$, respectively. Diagonalizer D works with $\langle M_d, w_d \rangle$ and a budget of t moves in the following manner. Like a universal Turing machine, D simulates M_d on input w_d for t moves. However, during this simulation D distinguishes between three cases:
(1) If M_d's computation on input w_d terminates with length $p < t$,
then D does not modify $Dcell(d,d)$.
(2) If M_d's computation on input w_d has length $p > t$,
then D does not modify $Dcell(d,d)$.
(3) If M_d's computation on input w_d terminates with length $p = t$ and output w'_d, then D appends 0 or 1 to the contents of $Dcell(d,d)$, so that $Dcell(d,d)$ contains a word (00 or 01) different from w'_d.

Analysis. Suppose M_d on input w_d does not halt. Then $cell(d,d)$ contains $enc(\perp)$ of length zero while $Dcell(d,d)$ contains word 0 of length one. Suppose M_d on input w_d does halt with output word w'_d. Then $cell(d,d) = w'_d \neq Dcell(d,d) \in \{00, 01\}$ from some stage onward. The conclusion is, that, for any $d \geq 1$, from some stage onward we have $cell(d,d) \neq Dcell(d,d)$. □

Remark 30. A new form of enumerating a set A of pairs $\langle d, v \rangle$ follows from D's construction in the previous proof. ADM D can be viewed as generating a set A of natural numbers as follows: D gradually puts each pair $\langle d, 0 \rangle$ for every natural number d in set A and can modify each pair, albeit at most once and only by appending one bit to the second element in the pair to obtain either $\langle d, 00 \rangle$ or $\langle d, 01 \rangle$. We have shown via diagonalization that no DTM generates set A; that is, A is not computably enumerable.

Remark 31. Infinitely many diagonalizers $D, D', \ldots$ can be concocted to prove Theorem 29 and the forthcoming lemmas, resulting in oracle sets $A, A', \ldots$, respectively. In the present article we stick to ADM D and its corresponding set A. Note, moreover, that (strictly speaking) ADM D works with *strings* and set A contains *natural numbers*.

7 From DTM's vs. ADM's to an Elaboration of ADM's

From the previous sections it follows that DTM's are computationally inferior to ADM's. For, indeed, every DTM-computable function is ADM computable (cf. Theorem 23), and there is an ADM that generates oracle set A, which no DTM can generate (cf. Theorem 29). The question remains, however, to what extent ADM's are superior to DTM's — a question that is addressed in the next section by examining oracle set A in greater detail. Set A is formally defined in Section 8.1 and subsequently analyzed in terms of halting problems (Section 8.2) and printing problems (Section 8.3).

Theorems 39 and 43 are the two take-away messages of the next section. The first theorem conveys ADM's limited superiority over DTM's: for, ADM's do not have enough power to solve the halting problem of Turing machines in its most general form (cf. the devil's advocate in Section 1.4). Leaving Turing machines aside, the ADM model is, just like any other model of computation in classical computability theory, limited in terms of the functions it can implement. Theorem 43 shows that ADM's cannot solve their own printing problem.

8 Elaborating on ADM's Computational Power

ADM D in the proof of Theorem 29 provides a way to generate an oracle set A which is not computably enumerable. Note, in particular, that ADM D can be viewed as generating words of the form $\langle d, 0 \rangle$ with one possible rectification — i.e.,$\langle d, 00 \rangle$ or $\langle d, 01 \rangle$ — provided a finite number

of computational steps later. The focus now lies on set A, followed by a treatment of halting and printing problems.

8.1 The Set A

Definition 32. $A = \{\langle d, v\rangle \mid (P \text{ holds or } Q \text{ holds}\}$, with:
P = "DTM M_d on w_d halts with output $v_d \neq v \in \{00, 01\}$."
Q = "DTM M_d on w_d does not halt, $v = 0$."
Moreover, we have: $\forall d \in \mathbb{N}.\ \exists x \in \{0, 00, 01\}\ .\ \langle d, x\rangle \in A$

Remark. Abuse of notation. We simply write $\langle d, v\rangle$ instead of $\langle M_d, v\rangle$, with $d \in \mathbb{N}$, $v \in \Gamma^*$, and M_d a description in Σ^* of the d-th DTM in a fixed, canonical enumeration. We often write "ADM N on input $\langle d, v\rangle$" instead of "ADM N on input $enc\,(\langle d, n_v\rangle)$," where n_v is the number equivalent of string v.

In terms of language recognition, we know that $A = L(N_A)$, for some ADM N_A. We now analyze ADM N_A in detail. N_A on input $\langle d, v\rangle$ prints 0 and subsequently does the following:

1. If $v = 0$, then N_A runs M_d on w_d.
   - If M_d on w_d does not halt, then neither will N_A with convergence.
   - If M_d on w_d halts, then N_A prints 0 forever; i.e., nonconvergence.
2. If $v \in \{00, 01\}$, then M_d on w_d has to halt in order for $\langle d, v\rangle$ to be in A. ADM N_A runs M_d on w_d and during this simulation N_A prints 0 on every computational step.
   - If M_d on w_d does not halt, then neither will N_A and in a nonconvergent manner, as desired.
   - If M_d on w_d halts with output $= v$, then N_A prints 0 forever.
   - If M_d on w_d halts with output $\neq v$, then N_A prints \$ and halts.
3. If neither 1. nor 2. apply, then N_A demonstrates nonconvergent behavior.

8.2 Halting Problems

Definition 33. Halting sets:
$H = \{d \mid \text{DTM } M_d \text{ on } w_d \text{ halts}\}$
$H' = \{d \mid \text{ADM } N_d \text{ on } w_d \text{ halts}\}$

Lemma 34. *There is a DTM M with oracle H that computes χ_A in the limit.*

Proof. We construct DTM M with oracle H, denoted M^H, on input $\langle d, v\rangle$, such that:
M^H on $\langle d, v\rangle$ halts with output $enc(1)$ iff $\langle d, v\rangle \in A$.
M^H on $\langle d, v\rangle$ halts with output $enc(0)$ iff $\langle d, v\rangle \notin A$.

Now, $\langle d, v\rangle \in A$ iff either (a) or (b) holds, with:
(a) M_d on w_d halts with output $v_d \neq v \in \{00, 01\}$.
(b) M_d on w_d does not halt with $v = 0$.

M^H uses oracle H to check whether $d \in H$. That is, M^H checks whether M_d on w_d halts, resulting in two cases.
<u>Case 1</u>: M_d on w_d halts.
Then M^H simulates M_d on w_d.

M^H then checks if M_d outputs something $\neq v \in \{00, 01\}$.
If so, then M^H halts with output $enc(1)$.
Else, M^H halts with output $enc(0)$.
Case 2: M_d on w_d does not halt.
Then M^H checks if $v = 0$.
If so, then M^H halts with output $enc(1)$.
Else, M^H halts with output $enc(0)$. □

Lemma 35. *There is a DTM M with oracle A that computes χ_H in the limit.*

Proof. Construct DTM M with oracle A, denoted M^A, as follows.
We want M^A on input d to:
(a) halt with output $enc(1)$ iff $\chi_H(d) = 1$.
(b) halt with output $enc(0)$ iff $\chi_H(d) = 0$.

That is:
M^A on d halts with output $enc(1)$ iff M_d on w_d halts.
M^A on d halts with output $enc(0)$ iff M_d on w_d does not halt.

So, M^A checks if $\langle d, 0\rangle \in A$.
If not so, then M_d on w_d halts. Then, M^A halts with output $enc(1)$.
If so, then M_d on w_d does not halt. Then, M^A halts with output $enc(0)$. □

Remark 36. In the previous proof we have implicitly relied on a trivial lemma: M_d on w_d does not halt iff $\langle d, 0\rangle \in A$.

Lemma 37. *If $enc(0) \not\prec enc(1)$ and $enc(1) \not\prec enc(0)$, then no ADM computes χ_H in the limit.*

Proof. Suppose some ADM N computes χ_H in the limit. Then N implements a converging sequence $S :: g_1, g_2, g_3, \ldots, \chi_H$. Then, for all $x \in \mathbb{N}$:
$\chi_H(x) = 1$
iff N on $enc(x)$ has $enc(1)$ or $enc(1)\$$ as output from some point in time onward.
$\chi_H(x) = 0$
iff N on $enc(x)$ has $enc(0)$ or $enc(0)\$$ as output from some point in time onward.

That is, for all $x \in \mathbb{N}$:
DTM M_x on w_x halts
iff N on $enc(x)$ has $enc(1)$ or $enc(1)\$$ as output from some point in time onward.
DTM M_x on w_x does not halt
iff N on $enc(x)$ has $enc(0)$ or $enc(0)\$$ as output from some point in time onward.

Thus, N on any $enc(x)$ will have as output, from some point t^* onward:
either $enc(1)$ optionally followed with \$,
or $enc(0)$ optionally followed with \$.

Now, take ADM N^* to be both a "DTM in disguise" and the "antagonist" of N. That is, ADM N^* only provides output in one sweep, immediately before halting: ADM N^* is DTM N^*. Antagonist N^* will, on input $enc(x)$, use its work tape to simulate N on $enc(x)$ such that:
(1) When N outputs either $enc(1)$ or $enc(1)\$$, then N^* does not halt.
(2) When N outputs either $enc(0)$ or $enc(0)\$$, then N^* prints 0\$ and halts.
Antagonist N^* has the capability to distinguish between (1) and (2) due to the precondition stated in the present Lemma.

But, N^* is in the canonical enumeration of DTM's M_1, M_2, ... So, we have $N^* = M_i$, for some $i \geq 1$. But, then, M_i on input w_i will:
(1) halt
iff N on w_i has $enc(1)$ or $enc(1)\$$ as output from some point onward

iff N^* on w_i does not halt
iff M_i on w_i does not halt
(2) not halt
iff N on w_i has $enc(0)$ or $enc(0)\$$ as output from some point onward
iff ...
iff M_i on w_i halts

These results lead to the desired contradiction. □

Corollary 38. *If $enc(0) \not\prec enc(1)$ and $enc(1) \not\prec enc(0)$, then no ADM computes $\chi_{H'}$ in the limit.*

Theorem 39. *DTM's are computationally inferior to ADM's which, in turn, are computationally inferior to DTM's with oracle set A or, equivalently, oracle set H.*

Proof. Follows immediately from the previous results. □

Remark 40. The phrase "computationally inferior to" incorporates the choice on how to encode natural numbers. The phrase does *not* solely refer to partially computable functions in the abstract; that is, to treatments in which the encoding function *enc* is swept under the rug.

8.3 Printing Problems

Definition 41. Printing set $P = \{d \mid \text{ADM } N_d \text{ on } w_d \text{ prints finite output} \neq \epsilon\}$

Lemma 42. *If $enc(0) \not\prec enc(1)$ and $enc(1) \not\prec enc(0)$, then no ADM computes χ_P in the limit.*

Proof. Suppose some ADM N computes χ_P in the limit. Then N implements a converging sequence $S :: g_1, g_2, g_3, \ldots, \chi_P$. Then, for all $x \in \mathbb{N}$:
(1a) $\chi_P(x) = 1$ iff $\exists t_0. \forall t \geq t_0. g_t(x) = 1$
(1b) $\chi_P(x) = 0$ iff $\exists t_0. \forall t \geq t_0. g_t(x) = 0$
(2) $\chi_P(x) \neq \bot$, because χ_P is a total function
(3) $\forall t, y.$ [$g_t(x) = y$ iff N on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after t moves]

Then, for all $x \in \mathbb{N}$:
(a) ADM N_x on w_x prints finite output $\neq \epsilon$
iff N on $enc(x)$ has $enc(1)$ or $enc(1)\$$ as output from some point onward
(b) ADM N_x on w_x prints ϵ or infinite output
iff N on $enc(x)$ has $enc(0)$ or $enc(0)\$$ as output from some point onward

Thus, N on any $enc(x)$ will have as output, from some point t^* onward:
either $enc(1)$ optionally followed with \$,
or $enc(0)$ optionally followed with \$.

Now, take ADM N^* to be the "antagonist" of N. That is, Antagonist N^* will, on input $enc(x)$, use its work tape to simulate N on $enc(x)$ such that:
(1) When N outputs either $enc(1)$ or $enc(1)\$$, N^* prints infinitely many symbols.
(2) When N outputs either $enc(0)$ or $enc(0)\$$, N^* prints 0\$ and halts.
Antagonist N^* has the capability to distinguish between (1) and (2) due to the precondition stated in the present Lemma.

But, N^* is in the canonical enumeration of ADM's N_1, N_2, ... So, we have $N^* = N_i$, for some $i \geq 1$. But, then, N_i on input w_i will:
(1) Print infinitely many symbols iff N_i on w_i prints finite output $\neq \epsilon$.
(2) Print 0\$ and halt iff N_i on w_i prints ϵ or infinite output.

These results lead to the desired contradiction. □

We can strengthen the previous lemma:

Theorem 43. *No ADM computes χ_P in the limit.*

Proof. We refer to ADM N and the Antagonist N^* in the previous proof, and distinguish between three cases.
Case 1: $enc(0) \not\prec enc(1)$ and $enc(1) \not\prec enc(0)$. Then the result follows from the previous lemma.
Case 2: $enc(0) \prec enc(1)$. Then: $enc(1) = enc(0)\,\sigma$ for some nonempty string σ. Let Antagonist N^*, on input $enc(x)$, use its work tape to simulate N on $enc(x)$, such that:
(1) When N outputs $enc(0)$, N^* prints 0 and continues the simulation, such that:
(2a) If N outputs \$, then N^* prints \$ and halts.
(2b) If N outputs σ, then N^* prints infinitely many symbols.
Case 3: $enc(1) \prec enc(0)$. Then: $enc(0) = enc(1)\,\sigma$ for some nonempty string σ. Let Antagonist N^*, on input $enc(x)$, use its work tape to simulate N on $enc(x)$, such that:
(1) When N outputs $enc(1)$, N^* continues the simulation and prints 0 on every computation step, such that:
(2a) If N outputs \$, then N^* prints infinitely many more symbols.
(2b) If N outputs σ, then N^* prints \$ and halts. □

9 From Naturals to Reals

All previous sections are about machines implementing functions on the naturals (e.g., DTM's, ADM's, BDM's, CDM's). In contrast, the next section — which can be skipped by casual readers — discusses machines implementing functions on the reals (e.g., Turing's 1936 machines, AdM's, BdM's, CdM's). To be more precise, the objective will now be to scrutinize Turing's 1936 paper itself; that is, to apply a similar kind of reasoning as in the previous sections but this time with regard to Turing's 1936 machines. Specifically, I shall charitably model Turing's 1936 automatic machines with AdM's and then prove that AdM's are computationally inferior to BdM's, which, in turn, are inferior to CdM's, etc. This take-away message is conveyed formally in Corollary 60.

10 A Retrospective On Turing's 1936 Machines

Coming to Turing's 1936 paper [56], I build on the account of Charles Petzold [47], whom, in turn, has relied on the analysis and corrections provided by Emil Post [48] and Donald Davies [15].

Each of Turing's 1936 automatic machines starts with a blank tape, and, if all goes well, computes an infinitely long sequence of binary digits, which, in turn, represent a real number $z \in \mathbb{R}$. Specifically, z lies in between 0 and 1 — that is, $0 \leq z \leq 1$ — and we shall denote this as follows: $z \in \mathbb{R}^{01}$.

Storage-wise, Turing's automatic machines only have a one-way infinite tape. In Petzold's words:

> "Although the tape is theoretically infinite in both directions, the machines that Turing describes in [his] paper require only that the tape extend infinitely towards the right because that's where the digits of the computable sequences are printed [...]" [47, p.81]

With regard to this one-way infinite tape, the automatic machine in hand is expected to:

- Print output digits (0 or 1) *from left to right* on alternative (say, even-numbered) tape cells, which we call numeric squares.

- Turing called these F-squares for *figures.*
- These cells are comparable to the cells constituting the second tape of an ADM; i.e., the output tape of an ADM.

- Print erasable scratch data on all other cells (i.e., the odd-numbered cells), which we call non-numeric squares.
  - Turing called these E-squares for *erasable.*
  - These cells can be associated with the first tape of an ADM; i.e., the input and work tape of an ADM.

The following comparison between Turing's 1936 machines and Emil Post's generalization is important to note. In Petzold's words:

> "As the machine progressively computes the 0s and 1s, it prints them sequentially from left to right. Every new figure that the machine computes is printed on the next available blank numeric square. No numeric squares are skipped. These restrictions are a collection of rules (some explicit and some implied) that Emil Post later called a "Turing convention-machine," [...] which is a little more restrictive than the generalized "Turing Machine." A Turing convention-machine never erases a numeric square, or writes over an existing figure on a numeric square with a different figure." [47, p.86]

Furthermore, Turing distinguished between two kinds of automatic machines:

- *Circular* machines never fill in more than a finite number of numeric squares.
- *Circle-free* machines do fill in, from left to right, infinitely many numeric squares.

It is the latter type of machine that is desirable because it prints infinitely many digits (of the real number in hand). In Turing's words:

> "A number which is a description number of a circle-free machine will be called a *satisfactory* machine. [...] it is shown that there can be no general process for determining whether a given number is satisfactory or not." [56, p.242]

Charitable reading of Turing's 1936 paper leads the present author to view Turing's 1936 automatic machine as equivalent to a 2-tape DTM, containing:

- One work tape, on which symbols 0, 1, and the blank b can be printed and modified any number of times.
- One output tape, on which digits 0 and 1 can only be printed from left to right.

These 2-tape DTM's, which serve to compute real numbers, are nevertheless limited. To reveal the limited power of these machines, variants of ADM's and BDM's, called AdM's and BdM's, have to be introduced first.

10.1 AdM's

Similar to an ADM, an AdM contains:

- One input and work tape (containing symbols 0, 1, or the blank b).

- One output tape, on which digits 0 and 1 can be printed from left to right. Each digit is observable to the outside world from the moment it is printed.

While an ADM computes a function $f :: \mathbb{N} \to \mathbb{N}_\perp$ in the limit, an AdM computes a function $f :: \mathbb{N} \to \mathbb{R}^{01}_\perp$ in the limit.

Remark 44. Strings are finite in length, sequences are infinitely long. We extend the notion of "prefix" from strings to strings and sequences. Again, we denote $\preceq$ for "prefix" and $\prec$ for "strict prefix." We also extend the concept of an injective encoding function *enc* from natural numbers to both natural numbers x and real numbers z. The encoding of x, denoted $enc(x)$, is a string. The encoding of z, denoted $enc(z)$, is a sequence.

Remark. Not any standard encoding function *enc* will do. For example, if some AdM N spits out the digits 001... (from left to right), then we want $enc(n_1) = 0$, $enc(n_2) = 00$, and $enc(n_3) = 001$, for some numbers $n_1, n_2, n_3 \in \mathbb{N}$. More to the point, the common binary representation — and other standard representations — of real numbers will not do, as Turing, himself, came to conclude in 1937. Guido Gherardi provides an excellent account in his 2011 article [24].

Definition 45. We say that AdM N *(mathematically) implements* a converging sequence $S ::$ $g_1, g_2, g_3, \dots, f$, with functions g_i from $\mathbb{N}$ to $\mathbb{N}$, and limit function f from $\mathbb{N}$ to $\mathbb{R}^{01}_\perp$, when the following three conditions hold, for all $x \in \mathbb{N}$:
(1) $\forall z \in \mathbb{R}^{01}$. [$f(x) = z$ iff $\forall t_1. \exists t_2 > t_1. \, enc\,(g_{t_1}(x)) \prec enc\,(g_{t_2}(x)) \prec enc\,(z)$]
(2) $f(x) = \perp$ iff $\forall z \in \mathbb{R}^{01}. f(x) \neq z$
(3) $\forall t, y$. [$g_t(x) = y$ iff N on $enc(x)$ has $enc(y)$ or $enc(y)\$$ as output after t moves]

Definition 46. We say that *AdM N computes function $f :: \mathbb{N} \to \mathbb{R}^{01}_\perp$ in the limit*, when N implements a converging sequence $S :: g_1, g_2, g_3, \dots, f$ of functions $g_1, g_2, g_3, \dots$ with limit function f.

Remark 47. What would it mean if some AdM N computes a function $f :: \mathbb{N} \to \mathbb{R}^{01}$ in the limit instead of $f :: \mathbb{N} \to \mathbb{R}^{01}_\perp$? Then N prints in the limit infinitely many output symbols on input $enc(x)$, for any $x \in \mathbb{N}$. Indeed, $f :: \mathbb{N} \to \mathbb{R}^{01}$ is merely a special case of $f :: \mathbb{N} \to \mathbb{R}^{01}_\perp$. Note, however, that, since each of Turing's 1936 automatic machines M is supposed to correspond to an AdM N, and since each M doesn't accept input, we will primarily be interested in AdM's N that do not use their input; that is, $f_N(x) = f_N(x')$, for $\forall x, x' \in \mathbb{N}$. See Definition 50 in particular.

Definition 48. We say that two AdM's N and N' are *functionally equivalent (with regard to function $f :: \mathbb{N} \to \mathbb{R}^{01}_\perp$)* when N implements some converging sequence $S :: g_1, g_2, g_3, \dots, f$ and N' implements some converging sequence $S' :: g'_1, g'_2, g'_3, \dots, f$', with $f = f'$.

Lemma 49. *Given two converging sequences (i.e., of functions from $\mathbb{N}$ to $\mathbb{N}$, with a limit function from $\mathbb{N}$ to $\mathbb{R}^{01}_\perp$), called S and S', and some AdM N that implements both S and S'. Then $S \equiv S'$.*

Proof. Consider an arbitrary AdM N that implements both sequences:
$S :: g_1, g_2, g_3, \dots, f$ and $S' :: g'_1, g'_2, g'_3, \dots, f'$.
Suppose it is not the case that $S \equiv S'$.

A first possibility is that $f \neq f'$; i.e., there exists a natural number x for which $f(x) \neq f'(x)$. Without loss of generality, $f(x) = z \in \mathbb{R}^{01}$. Then, by Definition 45, $\forall t_1. \exists t_2 > t_1. \, enc\,(g_{t_1}(x)) \prec enc\,(g_{t_2}(x)) \prec enc\,(z)$. We distinguish between two cases, each case results (non trivially) in a contradiction. (Details are omitted.)
<u>Case 1</u>: $f'(x) = z' \in \mathbb{R}$, with $z' \neq z$. Then, by Definition 45,
$\forall t'_1. \exists t'_2 > t'_1. \, enc\left(g_{t'_1}(x)\right) \prec enc\left(g_{t'_2}(x)\right) \prec enc\,(z')$.
<u>Case 2</u>: $f'(x) = \perp$. Then, by Definition 45,
$\exists t'_1. \forall t'_2 > t'_1. \, enc\left(g_{t'_1}(x)\right) \not\prec enc\left(g_{t'_2}(x)\right)$ or $enc\left(g_{t'_2}(x)\right) \not\prec enc\,(z)$.

A second possibility is that $g_i \neq g'_i$, for some $i \geq 1$. This means, that, for a specific $x \in \mathbb{N}$, we have: $y = g_i(x) \neq g'_i(x) = y'$, with $y, y' \in \mathbb{N}$. On the one hand, from Definition 45 we have: N on $enc(x)$ has $enc(y)$ or $enc(y)\,\$$ as output after i moves. On the other hand, from Definition 45 we have: N on $enc(x)$ has $enc(y')$ or $enc(y')\,\$$ as output after i moves. But, $enc(y)$ differs both from $enc(y')$ and $enc(y')\,\$$. Likewise, $enc(y)\,\$$ differs both from $enc(y')$ and $enc(y')\,\$$. So we have our desired contradiction.

The conclusion is that $S \equiv S'$. □

Definition 50. We say that any one of Turing's 1936 circle-free machines M and any AdM N are *output equivalent* when M computes $z \in \mathbb{R}^{01}$ in the limit and N computes f in the limit, with $f(x) = z$, for $\forall x \in \mathbb{N}$ and some function $f\, ::\, \mathbb{N} \rightarrow \mathbb{R}^{01}_{\perp}$.

Theorem 51. *Each real number z that is computable with one of Turing's 1936 circle-free machines is also AdM computable in the limit.*

Remark 52. Proof omitted. The only major conceptual difference between Turing's 1936 automatic machines and AdM's is that the latter machines provide *more* freedom with regard to output: any finite prefix of the final, infinite output of an AdM is, from some moment in time and onward, observable to the outside world.

To diagonalize out of the class of AdM-computable functions (cf. Theorem 55), we shall, in the forthcoming exposition, extend the AdM concept to that of a BdM.

10.2 BdM's

BdM's are slightly extended AdM's in that they can both prepend and append bits to their output, while AdM's can only append. Just like an AdM, a BdM has:

- A one-way, left-to-right infinite tape that serves both as input and work tape; this tape is also called the *first* tape.
- A one-way, left-to-right infinite tape that serves for appending digits 0 and 1 to the output; this tape is also called the *second* tape.

Unlike an AdM, a BdM also has:

- A one-way, left-to-right infinite tape that serves for prepending digits 0 and 1 to the output; this tape is also called the *third* tape.

The inclusion of the third tape requires an extension of the way in which encoding function *enc* is used:

- The encoding of input x, denoted $enc(x)$, is placed on BdM N's first tape.
- After some t moves, BdM N has computed a natural number y, with $enc(y) = \sigma_1 \ldots \sigma_l \sigma_{l+1} \ldots \sigma_n$ and $1 \leq l \leq n$, such that:
  - N's second tape contains, from left to right, the string $\sigma_{l+1} \ldots \sigma_n$
  - N's third tape contains, from left to right, the string $\sigma_1 \ldots \sigma_l$

To recapitulate, prepending the string on the third tape ($\sigma_1 \ldots \sigma_l$) to the string on the second tape ($\sigma_{l+1} \ldots \sigma_n$), results in the complete output $enc(y) = \sigma_1 \ldots \sigma_n$ that has been computed by N after the aforementioned t moves.

Definition 53. We say that BdM N *(mathematically) implements* a converging sequence S :: $g_1, g_2, g_3, \ldots, f$, with functions g_i from $\mathbb{N}$ to $\mathbb{N}$, and limit function f from $\mathbb{N}$ to $\mathbb{R}^{01}_{\perp}$, when the following three conditions hold, for all $x \in \mathbb{N}$:
(1) $\forall z \in \mathbb{R}^{01}$. [$f(x) = z$ iff $\forall t_1 . \exists t_2 > t_1 . \, enc\,(g_{t_1}(x)) \prec enc\,(g_{t_2}(x)) \prec enc\,(z)$]
(2) $f(x) = \perp$ iff $\forall z \in \mathbb{R}^{01} . f(x) \neq z$
(3) $\forall t, y$. [$g_t(x) = y$ iff N on $enc(x)$ has $enc(y)$ or $enc(y)\$$ as output after t moves]

We say that *BdM N computes function f :: $\mathbb{N} \rightarrow \mathbb{R}^{01}_{\perp}$ in the limit*, when N implements a converging sequence S :: $g_1, g_2, g_3, \ldots, f$ of functions $g_1, g_2, g_3, \ldots$ with limit function f.

Theorem 54. *Each function f :: $\mathbb{N} \rightarrow \mathbb{R}^{01}_{\perp}$ that is AdM computable in the limit, is also BdM computable in the limit.*

Proof. Trivial. □

Theorem 55. *There is a function f_V :: $\mathbb{N} \rightarrow \mathbb{R}^{01}_{\perp}$ that (i) some BdM V computes in the limit, and (ii) no AdM computes in the limit.*

Proof. Starting with part (i) of the theorem, we construct BdM V as follows. V, on input w, first checks whether w is $\langle M_d, w_d \rangle$ for some $d \geq 1$, with AdM M_d and input word w_d. If this check does not pass, then V prints \$ and halts. Else, V on input $\langle M_d, w_d \rangle$ goes through the following stages.
Stage 1: V prints 0 on its second tape. V simulates M_d on w_d and waits for M_d to print its first bit on its second tape. While waiting, V prints 0 on its second tape on every move.
+ If M_d prints 1 on its second tape,
then V prints 1 on its second tape, and goes to Stage 2.
+ If M_d prints 0 on its second tape,
then V prints 1 on its third tape and 0 on its second tape, and goes to Stage 2.
Stage $i > 1$: V continues simulating M_d on w_d and waits for M_d to print its i-th bit on its second tape. While waiting, V prints 0 on its second tape on every move. If M_d prints bit σ on its second tape, then V prints σ on its second tape, and goes to Stage $i + 1$.
Termination: If the simulation of M_d on w_d terminates, then V prints 0 infinitely many times on its second tape.

Analysis: Three cases can be distinguished:
Case 1: M_d on w_d prints, in the limit, no more than a finite number of digits.
Then V computes a sequence different from M_d's output on w_d.
Case 2: M_d on w_d prints, in the limit, infinitely many digits $1\sigma_2\sigma_3 \ldots$.
Since V computes a sequence starting with 0, V's output differs from M_d's output.
Case 3: M_d on w_d prints, in the limit, infinitely many digits $0\sigma_2\sigma_3 \ldots$
Since V computes a sequence starting with 10, V's output differs from M_d's output.

We now come to part (ii) of the theorem. Suppose that some AdM $\tilde{V}$ computes f_V :: $\mathbb{N} \rightarrow \mathbb{R}^{01}_{\perp}$ in the limit. Then AdM $\tilde{V}$ prints, on $\langle M_d, w_d \rangle$, precisely the same bits as BdM V on $\langle M_d, w_d \rangle$. But, in order to accomplish this in the general case, $\tilde{V}$ needs to be able to flip its output bits, which it cannot do by construction. □

Corollary 56. *Turing's 1936 circle-free machines and AdM's are computationally inferior to BdM's.*

10.3 CdM's

CdM's are slightly extended BdM's in that they have three output tapes instead of two. Just like a BdM, a CdM has:

- A one-way, left-to-right infinite tape that serves both as input and work tape; this tape is also called the *first* tape.
- A one-way, left-to-right infinite tape that serves for appending digits 0 and 1 to the output; this tape is also called the *second* tape.
- A one-way, left-to-right infinite tape that serves for prepending digits 0 and 1 to the output; this tape is also called the *third* tape.

Unlike a BdM, a CdM also has:

- Another one-way, left-to-right infinite tape that serves for prepending digits 0 and 1 to the output; this tape is also called the *fourth* tape.

The inclusion of the fourth tape requires an extension of the way in which encoding function *enc* is used:

- The encoding of input x, denoted $enc(x)$, is placed on CdM N's first tape.
- After some t moves, CdM N has computed a natural number y, with $enc(y) = \sigma_1 \dots \sigma_l \dots \sigma_m \dots \sigma_n$ and $1 \leq l, m \leq n$, such that:
  - N's second tape contains, from left to right, the string $\sigma_{m+1} \dots \sigma_n$
  - N's third tape contains, from left to right, the string $\sigma_{l+1} \dots \sigma_m$
  - N's fourth tape contains, from left to right, the string $\sigma_1 \dots \sigma_l$

To recapitulate, prepending the string on the fourth tape ($\sigma_1 \dots \sigma_l$) to the string on the third tape ($\sigma_{l+1} \dots \sigma_m$) to the string on the second tape ($\sigma_{m+1} \dots \sigma_n$), results in the complete output $enc(y) = \sigma_1 \dots \sigma_n$ that has been computed by N after the aforementioned t moves.

Definition 57. We say that CdM N *(mathematically) implements* a converging sequence S :: $g_1, g_2, g_3, \dots, f$, with functions g_i from $\mathbb{N}$ to $\mathbb{N}$, and limit function f from $\mathbb{N}$ to $\mathbb{R}^{01}_{\perp}$, when the following three conditions hold, for all $x \in \mathbb{N}$:
(1) $\forall z \in \mathbb{R}^{01}.\ [\ f(x) = z$ iff $\forall t_1. \exists t_2 > t_1.\, enc\,(g_{t_1}(x)) \prec enc\,(g_{t_2}(x)) \prec enc\,(z)\ ]$
(2) $f(x) = \perp$ iff $\forall z \in \mathbb{R}^{01}. f(x) \neq z$
(3) $\forall t, y.\ [\ g_t(x) = y$ iff N on $enc(x)$ has $enc(y)$ or $enc(y)\$$ as output after t moves]

Theorem 58. *Each function f :: $\mathbb{N} \rightarrow \mathbb{R}^{01}_{\perp}$ that is BdM computable in the limit, is also CdM computable in the limit.*

Proof. Trivial. □

Theorem 59. *There is a function f_V :: $\mathbb{N} \rightarrow \mathbb{R}^{01}_{\perp}$ that (i) some CdM V computes in the limit, and (ii) no BdM computes in the limit.*

Proof. Starting with part (i) of the theorem, we construct CdM V as follows. CdM V, on input w, first checks whether w is $\langle M_d, w_d \rangle$ for some $d \geq 1$, with BdM M_d and input word w_d. If this check does not pass, then V prints \$ and halts. Else, V on input $\langle M_d, w_d \rangle$ goes through the following stages.

Stage 1: V prints 0 on its second tape. V simulates M_d on w_d and waits for M_d to print its first bit on its third tape. On every move while waiting, V prints 0 on its second tape, unless:
+ M_d prints 1 on its second tape.
For then V prints 1 on its second tape, and goes to Stage 1.

+ M_d prints 0 on its second tape.
For then V prints 0 on its second tape and 1 on its fourth tape, and goes to Stage 1.
+ M_d prints 1 on its third tape.
For then V prints 1 on its third tape and 0 on its fourth tape, and goes to Stage 2.
+ M_d prints 0 on its third tape.
For then V prints 0 on its third tape and 1 on its fourth tape, and goes to Stage 2.
Stage $i > 1$: V continues simulating M_d on w_d and waits for M_d to print its next bit on either its second or third tape. While waiting, V prints 0 on its second tape on every move. If M_d prints bit σ on its second (third) tape, then V prints σ on its second (third) tape, and goes to Stage $i + 1$.
Termination: If the simulation of M_d on w_d terminates, then V prints 0 infinitely many times on its second tape.

Analysis: Three cases can be distinguished:
Case 1: M_d on w_d prints, in the limit, no more than a finite number of digits.
Then V computes a sequence different from M_d's output on w_d.
Case 2: M_d on w_d prints, in the limit, infinitely many digits $1\sigma_2\sigma_3\ldots$.
Since V computes a sequence starting with 0, V's output differs from M_d's output.
Case 3: M_d on w_d prints, in the limit, infinitely many digits $0\sigma_2\sigma_3\ldots$
Since V computes a sequence starting with 1, V's output differs from M_d's output.

We now come to part (ii) of the theorem. Suppose that some BdM $\tilde{V}$ computes $f_V :: \mathbb{N} \to \mathbb{R}^{01}_{\perp}$ in the limit. Then BdM $\tilde{V}$ prints, on $\langle M_d, w_d \rangle$, precisely the same bits as CdM V on $\langle M_d, w_d \rangle$. But, in order to accomplish this in the general case, $\tilde{V}$ needs to be able to flip its output bits, which it cannot do by construction. □

Corollary 60. *Turing's 1936 automatic machines and AdM's are computationally inferior to BdM's, which, in turn, are inferior to CdM's.*

11 Closing Remarks

> Are ordinary Turing machines diamonds hidden in the depths of the universe, or are they synthetic stones manufactured by man, yet so brilliant nevertheless that they bedazzle those mathematicians who are already partially blinded by pride in their own creations?

That's my re-phrasing of Morris Kline's original 1980 words, in which I have substituted "Turing machines" for "mathematics".[7]

A solution to the halting problem of Turing machines is not required in order to invalidate the Church-Turing Thesis. There are infinitely many functions f on the naturals ($\mathbb{N}$) that are not partially computable by Turing machines, yet for which knowledge of all $(x, f(x))$ pairs will still not suffice to solve the halting problem (of Turing machines). I have presented the ADM model of computation, and shown that there are ADM's that implement infinitely many of the aforementioned functions f, even though no ADM can solve the halting problem (in its full

[7] Kline's original phrasing:

> "Is then mathematics a collection of diamonds hidden in the depths of the universe and gradually unearthed, or is it a collection of synthetic stones manufactured by man, yet so brilliant nevertheless that they bedazzle those mathematicians who are already partially blinded by pride in their own creations?" [35, p.323]

I came across this passage in Charles Petzold's 2008 book *The Annotated Turing* [47, p.300].

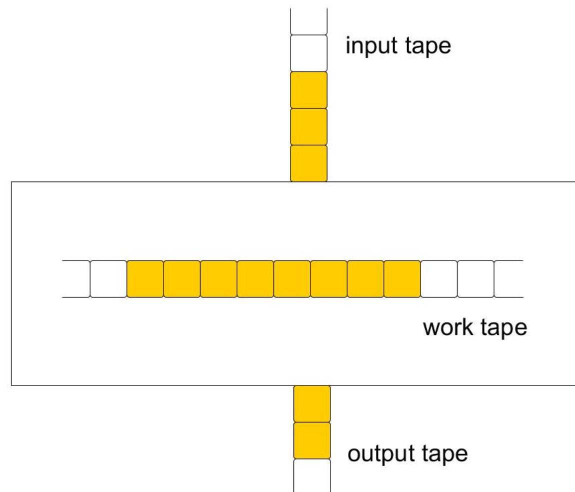

Figure 11.1: A 3-tape DTM or a 3-tape ADM.

generality). Add to these observations the following two, and we obtain reasons to, at the very least, doubt the validity of the Church-Turing Thesis:

1. Every partial function on $\mathbb{N}$ that is Turing computable is also ADM computable.
2. ADM's arguably have a higher model fidelity than Turing machines with regard to human (and electronic) computing.

Before elaborating on the second point, I emphasize, that, from a syntactical perspective, ADM's *are* DTM's (i.e., deterministic Turing machines). Suppose, for instance, that each DTM under consideration contains three tapes:

- an input tape
- a work tape
- an output tape

See Figure 11.1 for an illustration.

Without loss of generality, assume further that each DTM M uses an end marker \$ on its output tape to signify the termination of its computation. Let each ADM N contain the same three tapes and suppose N uses the end marker \$ in a similar fashion. (In all previous sections, DTM's are defined as 1-tape machines without \$ and ADM's as 2-tape machines with \$.) Even then, each ADM N differs semantically from any DTM M, namely as follows:

1. The user of M can only exploit M's output once M has halted, with \$ as final output symbol. The user of N, in contrast, can observe and exploit the contents of N's output tape, even if N has not printed \$.
2. Suppose M and N implement the same partial function f from $\mathbb{N}$ to $\mathbb{N}$, with $f(x) \in \mathbb{N}$ for some $x \in \mathbb{N}$. Let strings i and o encode numbers x and $f(x)$ in some standard fashion. Then we have that:

(a) M on input i has to terminate, with output o.

(b) N on input i does not have to terminate, it only has to output o.

There is no fundamental reason to prefer implementation scheme M instead of the more general scheme N. Both schemes treat mathematical functions $f :: \mathbb{N} \to \mathbb{N} \cup \{\perp\}$ as primary citizens. On the one hand, the critical reader is of course quite right to remark that only God (so to speak) may observe an infinite computation of N and declare that it is, indeed, providing a finite output o. On the other hand, however, God is also needed to tell whether the complete computation of an arbitrary DTM M is finite or infinite. In sum, critique pertaining to ADM's also holds for DTM's.

DTM's have at least two drawbacks that ADM's do not have. First, human (and even electronic) computers are not able to show to the outside world all digits of their output instantly, only incrementally. Second, in many computations, a prefix of the final output is known at some preliminary stage of computation, and can, thus, already attain meaning in the outside world. ADM's capture these two traits of computing, while DTM's do not.

11.1 Concurrency Theory

The idea to embrace nonterminating computations is far from novel. But, formalizing such computations *without* abandoning the aforementioned functions on $\mathbb{N}$ as primary citizens is less common. Consider, for example, the adjacent field of concurrency theory, which has an agenda that I briefly convey with the following three quotes.

- Robin Milner in 1973:
  - "Most of the computing agents with which computing science is concerned, for example digital computers themselves, their memories and peripheral hardware devices, and — more abstractly — computer programs, exhibit a behaviour which is not just the computation of a mathematical function of their inputs, but rather a possibly infinite sequence of communications with their environment." [41, p.157]
- Moshe Vardi and Pierre Wolper in 1994:

  "For many years, logics of programs have been tools for reasoning about the input/output behavior of programs. When dealing with concurrent or nonterminating processes (such as operating systems) there is, however, a need to reason about infinite computations." [59, p.1]
- Jos Baeten in 2005:

  "Here, two breakthroughs were needed: first of all, abandoning the idea that a program is a transformation from input to output, replacing this by an approach where all intermediate states are important, and, secondly, replacing the notion of global variables by the paradigm of message passing and local variables." [1, p.142]

In contrast to these authors, I have stuck to mathematical functions. With regard to Baeten's words, two remarks can be made to further position the present article in the literature. I have:

1. Continued viewing a program as a transformation from finite input to finite output.

2. Incorporated various (but not all) intermediate states of computation; i.e., those states in which output symbols are produced by ADM N.

Consider, for example, a payroll application in which the user can already use the first sheet of the machine's printout, *while* the ADM-modeled machine is still producing the other sheets. On the one hand, this example is intrinsically concurrent as the italicized word in the previous sentence highlights. On the other hand, since I have not explicitly modeled "the user," nor, thus, any explicit form of user interaction, the territory of concurrency theory remains foreign on the basis of the present paper alone.

11.2 Pandora's Box

In retrospect, some theorists might — and perhaps rightfully so — insist that the present article fails to debunk the Church-Turing Thesis. I beg to disagree for four main reasons. First, I have proved, by resorting solely to concepts from classical computability theory, that halting Turing machines are, in general, computationally inferior to potentially nonhalting Turing machines. To be more precise, DTM's are computationally weaker than ADM's. Second, ADM's do not have a lower fidelity than DTM's. (I argue that they have a higher fidelity.) Third, the ADM model of computation is only the tip of the iceberg. It lies strictly in between the DTM model of computation and the even more powerful models of Shapiro, Burgin, et al., which I shall discuss later. Fourth, accepting the ADM model *also* amounts to opening Pandora's box: We can add an extra output tape and subsequently analyze the BDM model of computation (Figure 11.2). Adding yet another output tape, results in the CDM model of computation. And so on. A study of these extensions would amount to the following results:

- ADM's are computationally weaker than BDM's.
- BDM's are computationally weaker than CDM's.
- ... ad infinitum ...

Indeed, it is easy to formalize a diagonal argument such that some, well-defined, BDM D diagonalizes out of the class of ADM-computable functions $f :: \mathbb{N} \to \mathbb{N} \cup \{\bot\}$. A similar remark holds for a CDM D' that diagonalizes out of the class of BDM-computable functions. And so on. The formal details follow naturally from a similar exposition that has been provided in Section 10, where the focus lay on the computations of real numbers by means of Turing's 1936 machines, AdM's, BdM's, CdM's, and so on. In that section I have charitably modeled Turing's 1936 automatic machines with AdM's, and subsequently proved that AdM's are computationally inferior to BdM's, which, in turn, are inferior to CdM's, etc.

11.3 Model Fidelity

Coming originally from engineering, I remain surprised that several scholars passionately defend the claim that the modern Turing machine model (of Kleene, Davis, Hopcroft & Ullman) captures *all* relevant aspects of the phenomenon being modeled: human computations. The reason for my surprise is best conveyed by Brian Cantwell Smith, a computer scientist and philosopher:

> "Every act of conceptualization, analysis, categorization, does a certain amount of violence to its subject matter, in order to get at the underlying regularities that group things together." [54, p.20]

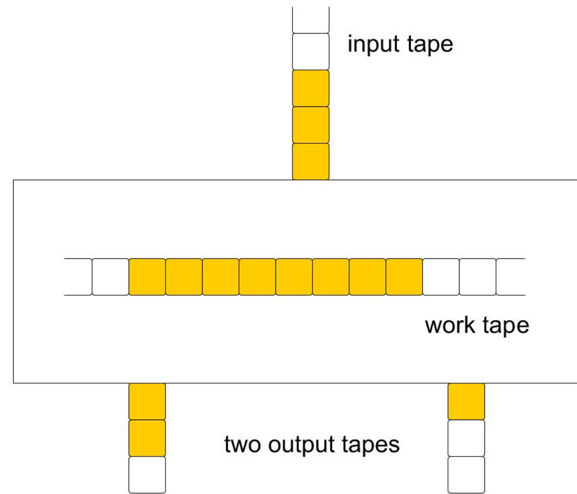

Figure 11.2: A BDM has two output tapes. Each output symbol is observable to the outside world from the moment it is printed. Each output symbol cannot be modified.

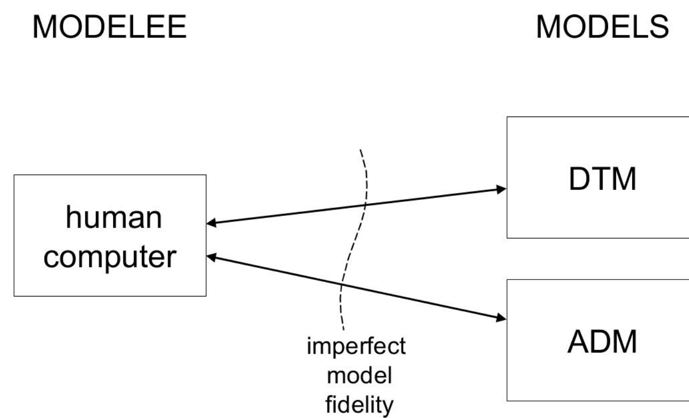

Figure 11.3: Human computation can be mathematically modeled in many ways, e.g., with Deterministic Turing Machines (DTM's) and with Alternative Deterministic Machines (ADM's). In various disciplines in which mathematical modeling is practiced, the model fidelity is always taken to be imperfect, because that is precisely what makes modeling so powerful [38, 54].

In engineering, the fidelity of a mathematical model is never perfect with regard to the real-world phenomenon being modeled [27, 37].

If I am right to call "mathematical modeling" the true name of computability theory, then it follows that neither DTM's nor ADM's capture all relevant traits of human (and electronic) computing — a viewpoint illustrated in Figure 11.3. In retrospect, this point of view seems to match every-day practice very well, as the following two examples indicate:

1. To adhere to the limitations of programming technology, various historical actors in computer science have preferred finite-state machines over Turing machines in their theoretical endeavors [21].

2. To capture the concurrent activities supported by a nonterminating operating system, Vardi and Wolper (quoted above) discarded the traditional input/output behavior, so dominant in automata theory. Instead, they viewed their finite-state programs as generators of infinite words [58].

Although both examples are about electronic computations (and about software in particular), it is straightforward to provide similar arguments for cooperative work practices of *humans*; e.g., two mathematicians proving a theorem together, optionally with the constraint that each mathematician has finite resources only.

Two take-away messages with regard to modeling can now be stated succinctly:

- Every model of computation — thus also the DTM model — is inherently partial and it has to be, for else it would not be tractable; cf. Smith [54].

- Therefore, mindful researchers will use their model only when the target is "operating within" the "regime of applicability of the model;" cf. Lee [38, p.42].

With these modeling insights, we can now briefly revisit Turing's work in the 1930s.

11.4 The Fidelity of Turing's 1936 Model

Before writing his 1936 paper, Alan Turing had grown up in an intellectual environment that is perhaps best described with the following passage from one of Turing's favorite books: Edwin Brewster's *Natural wonders every child should know.*[8]

> "We have taken up being, and doing, and thinking. [...] We shall learn about how the body of the plant or animal feeds itself and keeps alive, and how the different parts of it, the bones and skin and leaves and bark, manage to get on with one another, and work together like a well-made machine.
>
> For, of course, the body is a machine. It is a vastly complex machine, many, many times more complicated than any machine ever made by hands; but still after all a machine. It has been likened to a steam engine. But that was before we knew as much about the way it works as we know now. It really is a gas engine; like the engine of an automobile, a motorboat, or an airplane."

The repeated emphasis on machines in the previous passage is significant. Today, allegedly and largely due to Turing, it is common in various circles to state that a human *is* a machine.

[8] Thanks to Charles Petzold's lead [47] and Andrew Hodges's biography [31], it seems fair to state that Brewster's exposition, as illustrated here [3, Ch.XXXV], is historically relevant concerning Turing's maturing views on science. Note, moreover, that the "human-as-machine" metaphor was already "commonplace" by the end of the 18th century [40, p.45].

In particular, a human computer *is* a Turing machine (cf. some flavors of the Church-Turing Thesis) and a Turing machine *is* an electronic computer (cf. specific versions of the Physical Church-Turing Thesis). Moreover, it is not uncommon for computer scientists to endorse, not to mention conflate, both statements. Fortunately, critical voices have been raised before; see e.g. the writings of Carol Cleland [10, 11], Oron Shagrir [51], Selmer Bringsjord et al. [4, 5], and Mark Burgin [8, 9].

An emphasis on machines also appears in Robert Soare's 1996 retrospective account, in which he states that Turing might well have been "asking himself what was meant by calling a typewriter *mechanical*" [55]. The engineer in me, however, says that a typewriter can be modeled in various ways, and that Turing merely favored one of several viable approaches when he wrote his 1936 paper. For example, Turing chose *not* to model any of the following three aspects of a more encompassing *mathematician-as-typewriter* metaphor:

1. The sheets coming out of the typewriter — and, likewise, the symbols on each sheet — come out incrementally, not instantly (i.e., the central topic of the present article).

2. Typewriter A can output a sheet of paper that serves as input for typewriter B. Analogously, think of mathematician A who produces a proof sketch that is to be filled in further by mathematician B. I have commented on this slight form of interaction between agents A and B in Section 1.3, where I briefly scrutinized Wilfried Sieg's positive stance on the Church-Turing Thesis.

3. Mathematician A, who now uses a typewriter herself, typically makes a few mistakes. Subsequently, she rectifies her mistakes (e.g., with a pen and whiteout) on the printout itself; that is, she modifies one or more printed output symbols. More generally, mathematician A makes a few mistakes with her typewriter and lets another mathematician B correct A's output.

The third item captures a modeling approach that is orthogonal to the ADM-BDM-CDM-... story of the present paper. Following through on the third item would amount to allowing an ADM (or a BDM, or a CDM, ...) to *modify* r output symbols, where r is a fixed, natural number. Even more generally, one could concoct a machine that can modify output symbols a finite, yet unbounded, number of times. This type of extension of the *mathematician-as-typewriter* metaphor brings us, in the following paragraphs, to the notion of an ADM(n) machine, the writing of Shapiro et al., and Burgin's inductive Turing machines.

11.5 Temporarily Imperfect Computations

Machines much more general than ADM's — called ADM(n)'s, with $n \in \mathbb{N}$ — are based on the idea that a human computer can make n mistakes on "its output tape" and subsequently rectify those mistakes during "its" computation. So, not only can an ADM(n) append symbols to its output (in compliance with the ADM model of computation), it can also retroactively modify its output symbols, and it can do so at most n times.

We take ADM to be an abbreviation of ADM(0); that is, a machine that makes *no* mistakes (nor rectifications) during its computation. Taking $n = 1$, we can now refer to ADM(1) in order to convey a basic idea that (unfortunately) has hardly been addressed in the present paper:

> A mathematician who makes one mistake and subsequently rectifies her mistake (cf. ADM(1)) is more powerful than a similar mathematician who is deprived of temporarily being imperfect (cf. ADM).

The halting problem of Turing machines — i.e., the unconstrained halting problem of the devil's advocate, discussed in Section 1.4 — can be solved by an ADM(1) machine:

Theorem. *There is an ADM(1) machine V that solves the Halting problem for ordinary Turing machines.*

Proof. We construct ADM(1) machine V as follows. V, on input w, first checks whether w is $\langle M_d, w_d \rangle$ for some $d \geq 1$, with M_d a DTM description, and w_d an input word. If this check does not pass, then V prints the symbol 0 infinitely many times on its output tape (i.e., V demonstrates nonconverging behavior). Else, V on input $\langle M_d, w_d \rangle$ prints one occurrence of 0 on its output tape. Then, V simulates M_d on w_d. Two cases can be distinguished:
Case 1: M_d on w_d halts. Then, V flips 0 to 1 on its output tape, prints \$ and halts.
Case 2: M_d on w_d does not halt. Then V simulates M_d on w_d forever, with V's output tape containing string 0. □

Remark. The general idea underlying the above proof can be understood without depending on the previous sections. For simplicity, the proof relies on the assumption that all machines of interest work on binary digits (0 and 1) only. As we shall see shortly, the proof is far from novel. Moreover, and again, no claim is made in the present paper that the theorem itself has any immediate, novel applications.

In addition to the previous theorem and proof about ADM(1) machines, a more general observation ($n \geq 1$) can be made:

> A mathematician who can temporarily make n mistakes, for some fixed number $n > 0$, is computationally more powerful than a similar mathematician who can temporarily make at most $n - 1$ mistakes.

It is, after all, not uncommon in real life for a mathematician to derive results that are not to her satisfaction, to subsequently make n adjustments, and, finally, obtain a more desirable outcome. A similar remark holds for computer programmers, with the caveat that many of their bugs (software errors) are often only corrected after their software has been deployed.

In the limit, we can analyze mathematicians who can make *a finite, yet unbounded,* number of mistakes and corresponding rectifications in their computational work: we write ADM($-$) to denote the mathematical counterpart. These ADM($-$) machines are even more powerful than any of the ADM(n) machines. For example, the present author has written this article largely in compliance with a BDM($-$) model:

- I wrote the abstract of this article after spending months on all the rest. Hence, my reference to the *prepending capability* of a BDM, which an ADM does not have.
- I made some arbitrary, finite number of rectifications to draft versions of this article. Hence, my reference to a BDM($-$), and not a BDM(n), for any predefined number n.

Moreover, I have received feedback from colleagues on drafts of this paper. That is, I have *interacted* with other scholars. Arguably, then, a BDM($-$) model does not fit the bill completely. In retrospect, this is only to be expected, for no single model has a perfect fidelity with regard to the real-world phenomenon/activity that is being modeled. Attempts to more faithfully capture the interactive nature of my research would, perhaps, amount to using the *reactive Turing machines* of Jos Baeten et al. [2] and/or the *persistent Turing machines* of Dina Goldin et al. [25, 26].

11.6 Mark Burgin's Inductive Turing Machines

The extensions, just presented, of Turing's *mathematician-as-typewriter* metaphor provide an intrinsic motivation to embrace ADM(n) machines — and, likewise, BDM(n)'s, CDM(n)'s, and so on — as natural models of *algorithms.* My metaphor-based motivation is, I believe, a contribution to the present state of the art.

The main idea underlying the proof of the previous theorem, however, already appears in e.g. an article by Timothy McCarthy & Stewart Shapiro[9] and, more profoundly, in Mark Burgin's work on inductive Turing machines [9]. Credit for casting doubts on the Church-Turing Thesis, if not debunking the thesis altogether, goes first and foremost to Burgin.[10] In his words:

> "Working without halting, [an inductive Turing machine] ITM can occasionally change its output as it computes more. However, a machine that occasionally changes outputs does not deter humans. They can be satisfied when the printed result is good enough, even if another (possibly better) result may come in the future." [8, p.86]

This passage conveys a more general way of viewing computation than what modern Turing machines and even ADM's have to offer. For example, if a computer provides me with a weather forecast, then that forecast will influence my actions, even though I might receive an updated weather forecast a few hours later. Both forecasts are valuable, not only the last forecast. In retrospect, then, reconsider Kugel's complementary description of what is generally called an *eventually correct machine* in the literature:

> "When we use a computer to compute, we take its first output to be its result. But when we use it to compute in the limit, *we take its last output as its result* without requiring that it announce when an output is its last." [36, p.35, my emphasis]

Indeed, from a function-centric perspective, and in adherence to an ADM($-$) view on computation, we take the *last output* to be the official result. Historians of computability theory, in turn, will then not be surprised to come across the following kind of criticism, written by actors who have vested interests in classical concepts:

> "It is generally understood that in order for a computational result to be useful one must be able to at least recognize that it is indeed the result sought." [18, p.128]

The indented remark comes from Martin Davis, the father of the modern "Turing machine" concept; cf. [6, 20]. Davis dismisses the work of Mark Burgin yet does not distinguish between Burgin's conceptually simple, mathematical framework and the more extreme positions that dominate the rest of the field of *hypercomputation* [17, 18]. Specifically, the simplest of Burgin's inductive Turing machines are computationally superior to Turing machines yet do not rely on infinitely fast computations, nor on the ability to store infinitely many digits in a finite space. Critics of Burgin seemingly fail to make this observation. Naveen Sundar Govindarajulu et al. [28], in turn, also scrutinize Davis's reservations about hypercomputation in general — a topic that lies outside the scope of the present article.

To recapitulate, "Turing aimed at modeling the limits of human computation" [57, p.192] and the thesis carrying his name states that "Turing machines can do anything that could be described as rule of thumb or purely mechanical" [57, p.192]. In the present article, I have attempted to

[9] See McCarthy & Shapiro [39]. Slightly less related writings, also concerning *eventually correct systems*, are due to Hilary Putnam [49] and Jürgen Schmidhuber [50].

[10] I have not been able to access Burgin's original 1983 account on inductive Turing machines [7]. His 2005 book, although hard to read, reveals the thoughts of an original thinker [9]. I am not (yet) in a position to grasp Burgin's ideas concerning extensions of his simplest kind of inductive Turing machine [9].

improve Turing's modeling activity. In the process of doing so, I (and others before me) have arguably debunked Turing's Thesis and, by extension, the Church-Turing Thesis. Some, if not many, readers may still wish to ignore the previous statement without dismissing the presented mathematics altogether. To end with Minsky's 1967 words:

> "The reader who finds himself in strong disagreement either intellectually or (more likely) emotionally should not let that keep him from appreciation of the beautiful technical content of the theory developed ..." [43, p.105]

References

[1] J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335:131–146, 2005.

[2] J.C.M. Baeten, B. Luttik, and P. van Tilburg. Reactive Turing machines. *Information and Computation*, 231:143–166, 2013.

[3] E. Brewster. *Natural wonders every child should know.* Doubleday, Doran & Company, Inc., 1928.

[4] S. Bringsjord and K. Arkoudas. On the Provability, Veracity, and AI-Relevance of the Church-Turing Thesis. In A. Olszewski, J. Wolenski, and R. Janusz, editors, *Church's Thesis After 70 Years*, pages 66–118. ontos verlag, 2006.

[5] S. Bringsjord, O. Kellett, A. Shilliday, J. Taylor, B. van Heuveln, Y. Yang, J. Baumes, and K. Ross. A New Gödelian argument for Hypercomputing Minds Based on the Busy Beaver Problem. *Applied Mathematics and Computation*, 176(2):516–530, 2006.

[6] M. Bullynck, E.G. Daylight, and L. De Mol. Why did computer science make a hero out of Turing? *Communications of the ACM*, 58(3):37–39, March 2015.

[7] M. Burgin. Inductive Turing Machines. *Notices of the Academy of Sciences of the USSR*, 270(6):1289–1293, 1983. Translated from Russian, v.27, no.3.

[8] M. Burgin. How We Know What Technology Can Do. *Communications of the ACM*, 44(11):83–88, November 2001.

[9] M. Burgin. *Super-Recursive Algorithms.* Springer, 2005.

[10] C.E. Cleland. Is the Church-Turing Thesis True? *Minds and Machines*, 3:283–312, 1993.

[11] C.E. Cleland. Recipes, algorithms, and programs. *Minds and Machines*, 11:219–237, 2001.

[12] S.B. Cooper. Turing's Titanic Machine? *Communications of the ACM*, 55(3):74–83, March 2012.

[13] B.J. Copeland. Accelerating Turing machines. *Minds and Machines*, 12:281–301, 2002.

[14] J. Copeland, O. Shagrir, and M. Sprevak. Zuse's Thesis, Gandy's Thesis, and Penrose's Thesis. In M. Cuffano and S. Fletcher, editors, *Computational Perspectives on Physics, Physical Perspectives on Computation*, pages 39–59. Cambridge University Press, 2018.

[15] D.W. Davies. *The Essential Turing*, chapter Corrections to Turing's Universal Computing Machine, pages 103–124. Oxford University Press, 2004.

[16] M. Davis. *Computability and Unsolvability.* McGraw-Hill, New York, USA, 1958.

[17] M. Davis. The Myth of Hypercomputation. In *Alan Turing: Life and Legacy of a Great Thinker*, pages 195–212. Springer, 2004.

[18] M. Davis. The Church-Turing Thesis: Consensus and Opposition. In A. Beckmann et al., editor, *CiE 2006*, Berlin Heidelberg, 2006. Springer-Verlag.

[19] M. Davis, R. Sigal, and E.J. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science.* Morgan Kaufmann, second edition, 1994.

[20] E.G. Daylight. A Turing Tale. *Communications of the ACM*, 57(10):36–38, 2014.

[21] E.G. Daylight. The Halting Problem and Security's Language-Theoretic Approach: Praise and Criticism from a Technical Historian. *Computability*, Under Peer Review.

[22] N. Dershowitz and Y. Gurevich. A natural axiomatization of computability and proof of Church's thesis. *Bulletin of Symbolic Logic*, 14:299–350, 2008.

[23] A. Galton. The Church-Turing thesis: Still valid after all these years? *Applied Mathematics and Computation*, 178:93–102, 2006.

[24] G Gherardi. Alan Turing and the Foundations of Computable Analysis. *The Bulletin of Symbolic Logic*, 17(3):394–430, September 2011.

[25] D.Q. Goldin, S.A. Smolka, P.C. Attie, and E.L. Sonderegger. Turing machines, transition systems, and interaction. *Information and Control*, 194:101–128, 2004.

[26] D.Q. Goldin and P. Wegner. The interactive nature of computing: Refuting the strong Church-Turing thesis. *Minds and Machines*, 18(1):17–38, 2008.

[27] S.W. Golomb. Mathematical models: Uses and limitations. *IEEE Transactions on Reliability*, 20(3):130–131, August 1971.

[28] N.S. Govindarajulu and S. Bringsjord. The Myth of 'the Myth of Hypercomputation'. *Parallel Processing Letters*, 22(3), 2012.

[29] D. Hilbert and W. Ackermann. *Grundzuge der Theoretischen Logik.* Springer, 1928.

[30] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[31] A. Hodges. *Alan Turing: The Enigma.* Burnett Books, London, 1983.

[32] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[33] S.C. Kleene. *Introduction to Metamathematics.* Van Nostrand, Princeton, New Jersey, USA, 1952.

[34] I. Kleiner. Evolution of the Function Concept: A Brief Survey. *The College Mathematics Journal*, 20(4):282–300, 1989.

[35] M. Kline. *Mathematics: The Loss of Certainty.* Oxford University Press, 1980.

[36] P. Kugel. It's time to think outside the computational box. *Communications of the ACM*, 48(11):33–37, November 2005.

[37] E.A. Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15:4837–4869, 2015.

[38] E.A. Lee. *Plato and the Nerd: The Creative Partnership of Humans and Technology.* MIT Press, 2017.

[39] T.G. McCarthy and S. Shapiro. Turing projectability. *Notre Dame Journal of Formal Logic*, 28(4):520–535, 1987.

[40] P. McCorduck. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence.* A.K. Peters, Ltd, 2004.

[41] A.J.R.G. Milner. Processes: a Mathematical Model of Computing Agents. In Rose and Shepherdson, editors, *Logic Colloquium'73*, pages 157–174. North Holland, 1973.

[42] R. Milner. *Communicating and mobile systems - the Pi-calculus.* Cambridge University Press, 1999.

[43] M. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc, 1967.

[44] L. De Mol. Generating, Solving and the Mathematics of Homo Sapiens. Emil Post's Views on Computation. In H. Zenil, editor, *A Computable Universe — Understanding and Exploring Nature as Computation.* World Scientific, New Jersey, 2013.

[45] L. De Mol. Turing Machines. *The Stanford Encyclopedia of Philosophy*, 2018 Edition, 2018. plato.stanford.edu/entries/turing-machine/.

[46] C.H. Papadimitriou. *Computational Complexity.* Addison Wesley Longman, 1994.

[47] C. Petzold. *The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine.* Wiley Publishing, Inc., 2008.

[48] E. Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

[49] H. Putnam. Trial and error predicates and the solution to a problem of Mostowski. *Journal of Symbolic Logic*, 30(1):49–57, 1965.

[50] J. Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *International Journal of Foundations of Computer Science*, 3(4):587–612, 2002.

[51] O. Shagrir. Effective computation by humans and machines. *Minds and Machines*, 12:221–240, 2002.

[52] W. Sieg. What Is the Concept of Computation? In F. Manea et al., editor, *CiE 2018*, volume LNCS 10936, pages 386–396, 2018.

[53] W. Sieg, M. Szabó, and D. McLaughlin. Why Post Did [Not] Have Turing's Thesis. In E.G. Omodeo and A. Policriti, editors, *Martin Davis on Computability, Computational Logic, and Mathematical Foundations*, pages 175–208. Springer International Publishing Switzerland, 2016.

[54] B.C. Smith. The Limits of Correctness. *ACM SIGCAS Computers and Society*, 14,15:18–26, January 1985.

[55] R.I. Soare. Computability and recursion. *Bulletin of Symbolic Logic*, 2:284–321, 1996.

[56] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series*, 42:230–265, 1936.

[57] R. Turner. *Computational Artifacts: Towards a Philosophy of Computer Science.* Springer, 2018.

[58] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *1st IEEE Symposium on Logic in Computer Science*, pages 332–334, 1986.

[59] M. Vardi and P. Wolper. Reasoning about Infinite Computations. *Information and Computation*, 115:1–37, 1994.

[60] P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5), 1997.

About the Author

Edgar G. Daylight, also known as Karel Van Oudheusden, has a PhD in computer science from KU Leuven (2006) and an additional MSc in logic from the University of Amsterdam (2009). He is currently a post-doc researcher in the history of computer science at Siegen University.
This research was funded by SFB 1187 "Medien der Kooperation" (Siegen University) and ANR-17-CE38-0003-01 "PROGRAMme" (Lille University).

Acknowledgments

Thanks to Lorenz Demey, Simone Martini, Liesbeth De Mol, and Máté Szabó for commenting on technical details in this article. I am also grateful for the feedback I received from Kurt De Grave, Erhard Schüttpelz, Selmer Bringsjord, and Jean Paul Van Bendegem in the last quarter of 2018. The quote in my abstract — "Any process which could naturally be called an effective procedure can be realized by a Turing machine." — comes from Marvin Minsky [43, p.108].

"The decision problem is solved when we know a procedure with a finite number of operations that determines the validity or satisfiability of any given expression [...]. The decision problem must be considered the main problem of mathematical logic."
— Translation [47, p.260] from Hilbert & Ackermann [29, p.73,77]

"The decision problem is only really solved when we know a procedure with a finite number $n(e)$ of operations that determines the validity or satisfiability of any given expression e, and some Turing machine M on input e computes n. Preferably, we, humans, also posses a specification of machine M."
— The present author.