Thinking Like a Machine: Alan Turing, Computation and the Praxeological Foundations of AI

Dipanjan Saha

University of Liverpool, United Kingdom/ dipanjan.saha@liverpool.ac.uk

Phillip Brooker

University of Liverpool, United Kingdom

Michael Mair

University of Liverpool, United Kingdom

Stuart Reeves

University of Nottingham, United Kingdom

Abstract

As part of ongoing research bridging ethnomethodology and computer science, in this article we offer an alternate reading of Alan Turing's 1936 paper, "On Computable Numbers". Following through Turing's machinic respecification of computation, we hope to contribute to a deflationary position on AI by showing that the activities attributed to AIs are achieved in the course of methodic hands-on work with computational systems and not in isolation by them. Turing's major innovation was a demonstration that mathematical and logical operations could be broken down into elementary, mechanically executable operations, devoid of intellectual content. Drawing out lessons from a re-enactment of Turing's methods as a means of reflecting on contemporary artificial intelligence (AI), including the way those methods disappear into the technology, we will suggest the interesting question raised in "On Computable Numbers" is less about the possibilities of designing machines that "can think" (cf. Turing, 1950), but the practical work we do, and which is made possible, when we ourselves set out to think like machines.

Keywords: Turing Machine, Computation, Artificial Intelligence, Ethnomethodology, Re-enactment

Introduction

The foundations of Turing's 'thinking machines' (Turing, 1950)—and by extension the aspirational research programme(s) of 'artificial intelligence' (AI) and core assumptions about the *computational* character of 'intelligence' that AI mobilises—are built upon Turing's earlier work on computability and Turing Machines (Turing, 1936). In this paper we want to examine the overlooked praxeology of Turing Machines (TMs) as an imagined—and widely claimed—precursor for AI.

By attempting to create a simple TM as part of a course of "technical self-instruction" (Sormani, 2016), we reveal *how instructions come to constitute machines that do things 'on their own'*—and in doing so advance an ethnomethodologically-informed corrective to what we think are lingering reifications of 'machine autonomy' in AI. Our core argument is that the circumstances in which machines are brought off as 'autonomous', as demonstrating 'artificial intelligence', can be traced back to the same kinds of underexamined practical work revealed when we attempt to piece together TMs for ourselves: familiar activities to any Computer Science undergraduate, but largely unremarked in any explicit fashion by Turing himself and others since.

Partnered with what we could call Turing's 'disappearing act'—the dematerialisation of the practical construction of TMs into the TMs themselves—is a conflation of *human* computation with *machine* computation as a way of conceiving of the machine in the first place[1]. This starts with Turing's original focus, namely, human computers doing computations, being seamlessly transformed into machines doing computations in what are presented as equivalent ways. That originating conflation is stubborn and has underpinned misunderstandings about the capabilities of machines and humans in discussions of AI ever since (cf. Collins, 1990; Brooker et al., 2019a). What we seek to provide here is a praxeologically-oriented corrective to that conflation, a corrective which at the same time will make Turing's work visible again. We will do so via an account of 'getting the TM to compute', which displays just how machine computation rests on human activity (e.g., on the production and delivery of machine instructions), in every case, at all points. Though these machines can be used to do profoundly impressive things, they do not set up or operate themselves, and focussing on the practicalities of what must be done (by people) to get the machines to work affords some clarity to a set of fields where distortingly inflationary discourses can tend to prevail (cf. Elish and boyd, 2018; Brooker et al., 2019a; Campolo and Crawford, 2020; Mair et al., 2021).

Modern computational technologies are, of course, far more sophisticated than Turing's original examples, but, as with Turing's machines, we will continue to misunderstand their capabilities if we insist on recasting them as somehow either directly mirroring human practices or as working 'on their own' (cf. Suchman, 2006; Holton and Boyd, 2021). While many regard attempts to define AI as a fools' errand—part of the phenomenon not a means of bringing it into view (e.g., Seaver, 2019) —we believe the only serious means of addressing AI is to get a more precise handle on what these technologies do and how. This is why we use this paper to develop an "alternate" (Garfinkel, 2002: 72-73) reading of Turing's 1936 paper, "On Computable Numbers, with an Application to the *Entscheidungsproblem*", a return to Turing that helps us recover what is involved in building a machine that, allegedly, and at its most foundational level, 'does things for itself'.

More specifically, by drawing out lessons from a re-enactment of Turing's methods as a site which opens up the praxeological "foundations" of AI in under-appreciated ways (Lynch et al., 1983: 208; Garfinkel 2022: 182), we will suggest the interesting question raised in "On Computable Numbers" is less about the possibilities of designing machines that "can think" (Turing, 1950), but the practical work we do, and which is made possible, when *we ourselves* set out to think like machines, i.e., when we are devising instructions and frameworks for instructing machines to performs tasks such as calculating.

Turing's contribution in "On Computable Numbers"—an imaginative as well as formal one that cut across logic, mathematics, engineering, philosophy and psychology—was a demonstration that mathematical and logical operations could be broken down into elementary, mechanically executable operations that are devoid of intellectual content and can be implemented without understanding. If we follow Turing's computational methods, these are operations which can be carried out by machines; Turing Machines, as they have since come to be known. Indeed, by respecifying the doing of mathematics and logic in the particular ways that he did, i.e., as strings of elementary non-intellectual processes, Turing showed machines could be constructed which could in principle carry out any operation capable of being computed whatsoever[2], depending only on the ingenuity, accuracy and precision of the instructions they were supplied with, becoming in the process what he termed "universal machines" (Turing, 1936: 242; Turing, 2005[1945]: 371-372), conceptual counterparts to contemporary digital computers (see, e.g., Piccinini, 2003: 28). The question remains, however, as to how instructions for such machines are to be formulated in any actual case. In dialogue with a growing literature on data, algorithms, automation, machine learning, artificial intelligence and programming (e.g., Agar, 2003, 2017; Benbouzid, 2019; Brock, 2016; Brooker et al., 2019b; Burrell, 2016; Burrell and Fourcade, 2021; Elish and boyd, 2018; Fazi, 2016, 2018; Jaton, 2020; Lee, 2020; Mackenzie, 2017; Rieder, 2020; Seaver, 2019; Smith, 2019; Ziewitz, 2016), we seek to open up features of the work involved; work which cannot be recovered from the machine through cognitive analogies or models of thought or mind, but only by attending to the practical activities through which it is accomplished (Lynch et al., 1983; Garfinkel, 2022). Based on our re-enactment of the work of instruction in Turing's paper as a "tutorial problem" (Garfinkel, 2002: 145), we argue that acquainting ourselves with ways of thinking with and through the kinds of methods found in Turing's work helps us recover the computational foundations of AI via an understanding of the practices involved in its achievement. Seen thus, as we shall argue in conclusion, AI, as "engineered design" (Garfinkel, 2002: 268), emerges as a reproducibly instructable phenomenon (Lynch and Lindwall, forthcoming). Following Turing's machinic respecification of computation through and clarifying its grounds, we hope to contribute to a more consistently deflationary position on AI, dispelling AI's "magic" (Elish and boyd, 2017; Campolo and Crawford, 2020) and defusing its "drama" (Ziewitz, 2016) by showing the activities attributed to AIs are achieved in the course of methodic hands-on work with computational systems and not exclusively *by* them. If we are to recover the work practices through which AI systems are crafted, however, we need to be alive to the ways in which those practices are made to disappear into those systems once built. Understanding how Turing first formalised that 'disappearing act', we argue, provides important lessons for anyone seeking to unpick its contemporary equivalents in the field of AI, something we tease out in the discussion with reference to AlphaGo and its successor algorithms but which is an issue with broader relevance still.

Conceptualising computation: two ways of thinking about Turing's work

Whether or not Turing's work had any significant bearing on the construction of modern computers is a contested issue (see, e.g., Sloman, 2002 on Turing's "irrelevance" there). Indeed, Agar (2003, 2017) has argued, "On Computable Numbers" looked less to the future and the digital computer than back to the general purpose 'machinery' of government and the bureaucratic reorganisation of clerical work within it into simplified tasks arranged in both serial and parallel orders as part of a procedural, we might even say algorithmic, division of epistemic labour. Nonetheless, Shanker's (1995) notes in his reflection on "Computing Machinery and Intelligence" (Turing, 1950), Agar's (2003) 'governmental' reading and 'Turing realists' like Sloman share common ground. All agree Turing's early work (a) did influence those involved in building the first generation of computers, such as von Neumann, by offering them an initial logical model (see Gandy, 1988), and (b) retrospectively played a pivotal role in the formation of AI as a field in the 1950s by figures such as McCarthy, McCulloch, Minsky and Simon, as it provided them with a clear sense of what the phrase "Artificial Intelligence" could be taken to mean when rendered computationally. In this sense, "On Computable Numbers" represents a pivotal move because in it Turing effects a logical refutation of the proposal that computation could be treated as part of analytic philosophy, i.e., as aprioristic, deductive and strictly logically derived. In refuting that proposal, he instead relocated computation to a domain of practical, empirical, trial and error work – comput*ing*, in the active sense – involving the construction of devices for stabilising and testing computability as a contingent matter (cf. Fazi, 2018). The paper crucially, therefore, worked as a ground clearing exercise that helped establish space for subsequent developments in computing *and* what would come to be called AI. Central to Turing's contribution in this regard as the earliest published breakthrough in formulating potential bases for specifying 'machine intelligence' were what Shanker terms his "two questions": "the philosophical question …: Can machines think? … and the psychological question: Do thinkers compute?" (Shanker, 1995: 52). For Shanker (1995),

> These two questions belong to very different traditions. The former was a central concern of English mathematicians in the nineteenth century (e.g., Babbage, Jevons and Marquand); the latter a mainstay of empiricist psychology in Germany, England, and America. But Turing not only regarded these two questions as intimately connected: in fact, he thought they were internally related—that in answering one you would *ipso facto* be answering the other. The result was a remarkable synthesis. (p. 53)

This intended synthesis had many strands and we do not have space to fully set out Shanker's detailed examination of them here, though we would encourage readers to consult Shanker's work for themselves (e.g., 1987, 1995, 2002). However, we do want to offer an outline of one aspect of Turing's work in "On Computable Numbers" as part of rethinking how we might approach the second question in particular.

That said, offering an easily-digestible exegesis of even a small part "On Computable Numbers" is a far from straightforward task. For one thing, ten of its eleven sections plus the introduction and the 1937 appendix are given over almost entirely to working through what for lay readers are formidably complex problems of mathematical logic—without the requisite background in mathematics and mathematical logic, a background which Turing could reasonably assume his contemporary readers had, these sections are opaque to say the least (though see Petzold, 2008 for a helpful line-by-line discussion as well as, e.g., Agar, 2003, 2017, Fazi, 2016, 2018 and Gandy, 1988 for more wide-ranging and differently oriented accounts). For another, the work Turing does within the paper is transgressive; grounded in mathematical logic but crossing into philosophy, speculative engineering and psychology in an idiosyncratic fashion. Nonetheless, despite—indeed, because of—these difficulties, the paper contains valuable lessons. It occupies an important place in Turing's intellectual programme because it is there where Turing formalises his respecification of computation with reference to the activities of *human* computers, that is, *individuals* undertaking the work of calculation. This is part of the framing of the paper in §1— "We may compare a man in the process of computing a real number to a machine" (1936: 231)—and the focus of §9, more specifically Turing's "appeal to intuition" (1936: 249), which elaborates on the basis of that comparison. It is from there, following Shanker, that we take the argument up.

The first point to note is that during the time Turing was writing his 1936 paper, (i.e., prior to the introduction of machines to do the work), human computers were employed in government and industry to perform long and time-consuming mathematical operations based on instructions issued to them (see Agar, 2006). In a context where calculative operations had already been rendered increasingly 'mindless' through an atomising clerical division of labour driven by large corporations and governments over decades if not a century or more (Agar, 2003, 2017)[3], Turing sought to reduce their work even further to its behavioural minima. In Turing's analysis, while the outcomes of that work could be highly sophisticated, the individual tasks these human computers performed seemed simple and did not appear to have to be treated as involving any deep, complex mathematical reasoning beyond writing down symbols one after another according to a prescribed series of steps accessed by looking up input tables and logbooks. As Shanker puts it, Turing was reimagining a typical human computer "performing the most routine of calculating tasks in order to … break calculation down into its elementary … units" (1995: 74). As those units, those behavioural minima, could be shown to be "devoid of intelligence" and mechanistic in their operations, Turing notes we "may now construct a machine to do the work of this computer" (Turing, 1936: 251). Such machines came to be dubbed "Turing Machines" and proved influential as Turing showed that they could be used for computation on a formal and thus provably rigorous, logical and mathematical basis. With this machinic respecification of the problem of computability in hand, Turing could argue that "the machine's 'behaviour' … satisfies our criteria for saying that it is 'calculating' because its internal operations are isomorphic with those guiding the human computer" (Shanker, 1995: 80). The equivalence is established on this basis: bracketing the material and situational differences between them, machines can be said to be calculating, on Turing's analysis, because they are doing what human computers do when they 'compute', i.e., working through 'recursive functions', algorithmic chains of elementary operations; and when human computers calculate they are doing what computing machines do, i.e., working through the same recursive algorithmic functions composed of simple steps albeit at the time in lengthier and more complex combinations. In Davis's (1978) summary,

> Turing based his precise definition of computation on an analysis of what a human being actually does when he computes. Such a person is following a set of rules which must be carried out in a completely mechanical manner. Ingenuity may well be involved in setting up these rules so that a computation may be carried out efficiently, but once the rules are laid down, they must be carried out in a mercilessly exact way. (as cited in Shanker, 1995: 73)

In Turing's work, exactly the same parameters were to be applied to computing machines as to human computers because their operations were designed to "include all those which are used in the computation of a number [by the former]" (1936: 118; see also Gandy, 1988; Piccinini, 2003; Sieg, 2009).

Shanker goes on to critically deconstruct Turing's account with respect to the drawing of that equivalence, "question[ing] the whole basis of Turing's interpretation of the logical relation in which algorithms stand to inferring, reasoning, calculating, and thinking" (1995: 81-82). He does so, with reference to Wittgenstein, by showing that our practices of calculation are not the same as the operations of the computing machine (see also Collins, 1990 for related arguments). Shanker's response to Turing's second question is consequently a negative one: humans do not compute in the same terms machines do. However, while we endorse Shanker's analysis, we want to take the discussion in a somewhat different direction, picking up on matters Shanker and others have left unremarked. Those matters are foreshadowed in Davis's gloss above, "Ingenuity may well be involved in setting up … rules so that a computation may be carried out …" and take us to Turing's stated objective of "construct[ing] a machine to do the work of … [a] computer" (again, as cited in Shanker, 1995: 73) and not just one capable of handling single computations but whole classes of them – Turing's "universal computing machine" (1936: 241). Just what is this ingenuity and just how is to be embodied in the construction of such a machine? Insofar as Turing is presenting a conceptual blueprint for that machine and *was thus himself engaged in computing work*, what was *he* doing and *how*? Finally, how might that work be recovered from Turing's published accounts of it?

Just as with the work of the human computer it is said to derive from, the work of the machine as it computes is entirely unlike the (human) work that goes into setting it up to do so. Yet while distinct, in this case the two *are* intertwined. Indeed, and in an important sense, 'the machine', such as it is, can be seen *as* being constituted by its tables of instructions and thus the work that has gone into formulating them (Turing, 1936: 243). Here then, contra Shanker, we do have an internal relation. However, when we start to look within Turing's paper for the work of devising those instructions, of thinking in mechanical terms about computation for the purposes of building an instructed and instructable machine, we find we can locate the machine easily enough but the instructive work that constitutes it proves much more elusive.

In his later 1945 report on the construction of the Automatic Computing Engine or ACE, *Proposed Electronic Calculator* (reproduced in Copeland, 2005), Turing (2005) notes

> It is evident that if the machine is to do all that is done by the normal human operator it must be provided with the analogues of three things, viz. firstly, the computing paper on which the computer writes down his results and his rough workings; secondly, the *instructions as to what processes are to be applied*; … thirdly, the function tables used by the computer must be *available in appropriate form to the machine*. (p. 371, emphasis added)

He (Turing, 2005) goes on:

> It is intended that the setting up of the machine for new problems shall be virtually *only a matter of paper work*. Besides the paper work nothing will have to be done except to prepare a pack of … [punch] cards in accordance with this paperwork, and to pass them through a card reader connected with the machine. There will positively be no internal alterations to be made even if we wish suddenly to switch from calculating the energy levels of the neon atom to the enumeration of groups of order 720. It may appear somewhat puzzling that this can be done. How can one expect a machine to do all this multitudinous variety of things? The answer is that we should consider the machine as doing something quite simple, namely *carrying out orders given to it in a standard form* which it is able to understand. (p. 372, emphasis again added)

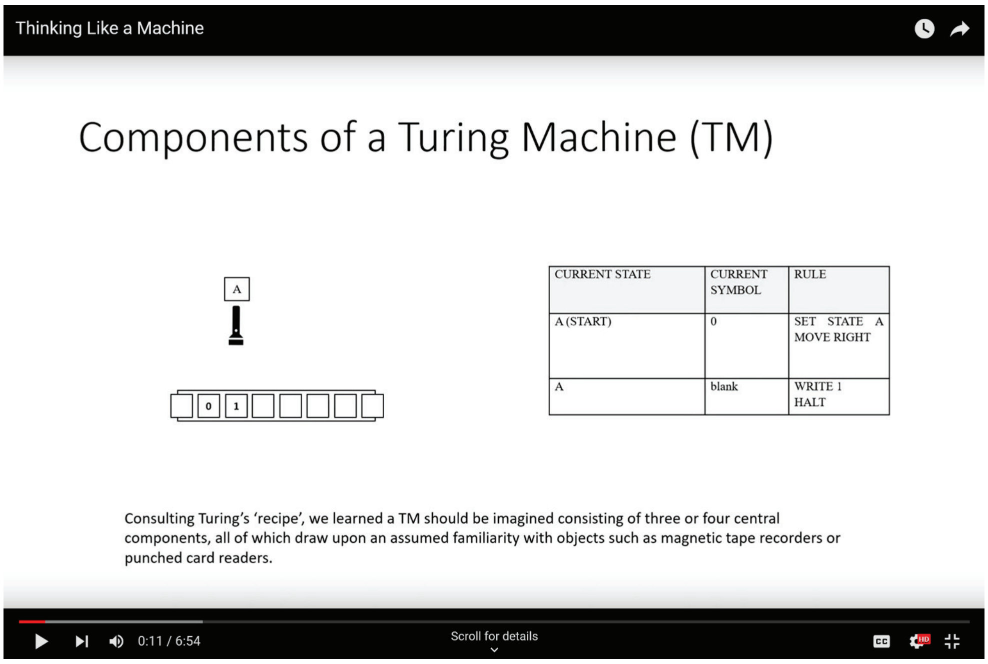
The phrases "instructions as to what processes are to be applied … available in appropriate form"/ "orders given … in a standard form" make it clear that Turing was seeking to devise a framework—a set of reproducible methods—for working through the instructions these machines were to be given as the basis of that "standard form". Turing's machines could do all manner of things if instructed in the right way but, as we can see, that hinged on working out the instructions, exercising the "ingenuity" Davis points to along the way. Put differently, there is lots of relevant action—i.e., the careful design of instructions that can be supplied to and carried out by a machine, yet which will have some meaningful purpose resulting from their execution (i.e., will be able to be made sense of and thus made relevant in a specific social context)—bracketed off here as "only a matter of paper work". Yet nowhere is it made clear just how that "paper work" is to be done. While this unspecified set of activities is almost entirely glossed over by Turing, we argue it sits at the foundations of AI then and now, foundations we attempt to excavate praxeologically in what follows.

Human-machine asymmetries as a tutorial problem

As we have begun to explore in a preliminary way above, it is not exactly the case that Turing's model treats humans and machines as engaged in the same activities—the very articulation of the mechanisms of a Turing Machine by Turing himself shows us otherwise. There is, instead, a 'pairing' here, i.e., the framing of the instructions and the instructed operations of the machine, but they are not reducible to one another, and their internal relations are alternately *a*symmetric as Garfinkel (2002: 114) puts it. That is, it is possible to get from the first to the second (indeed the instructions, which are often materially encoded in the case of digital computers, define 'the machine') — *but not the other way round*. Reading Turing's 1936 paper, however, as we have noted above, provides little sense of how *he* constructed the machine at least explicitly[4]. Nor can the work which went into figuring out those instructions and putting them into a "standard form" be recovered by looking at the inputs and outputs of the machine alone. The practical work of "making a universal machine" (Jaton, 2020: 103) is thus missing from the picture, and while that is not problematic for all purposes, it does highlight a praxeological gap for those seeking to understand 'autonomous' machines more generally, not least because a consideration of Turing's own methods shows us the two cannot be meaningfully separated out.

One way to recover Turing's prototype methods could be via a detailed examination of "On Computable Numbers", working through what Turing was doing in its successive stages as part of a critical textual hermeneutics of computation (cf. Fazi, 2018). However, following Garfinkel's later work (e.g., 2002, 2022) and the example of studies by the likes of Livingston (1986), Bjelić (1996), Sormani (2016), Sharrock and Ikeya (2000), and others (e.g., Brooker and Mair, 2022), we want to instead proceed somewhat differently and "misread" Turing's 1936 paper, treating it not as an established logico-mathematical proof but an instructional guide for "construct[ing] a [Turing] machine" that we ourselves can attempt to follow as part of a course of "technical self-instruction" (Sormani, 2016: 102-136) in the foundations of AI. In this, the text of the paper furnishes "clues" (Bjelić, 2003: 133-136). While we would certainly not suggest Turing's methods are the *only* way of constructing them, our misreading provides an occasion for a pedagogic tutorial in the work of making 'computing machines' via a consideration of their instructed character. Through that tutorial, we want to consider the lessons that might be learned from gaining a first-hand appreciation of their constitution. In this case we will use different diagrams as part of a re-enactment designed to demonstrate the workings of a paper TM assembled from scratch for a particular computational purpose. These diagrams are not part of Turing's paper but intended as pedagogical devices to guide readers through the machinery of TMs and make them instructably observable, in Garfinkel's phrase (2002: 136), in relation to the specific courses of specific action and reasoning that machinery is coupled to. It is with this in mind that we invite readers to draw out their own TM, as per the instructions below, and follow along with the operation of those instructions to see the praxeological sense they have first-hand. For those who might benefit from a further talk-through of our TM, see Clip 1 below. Voiced by a machinic narrator, the choppy, robotic delivery is fitting given our subject matter.

Our approach here is novel in the sense that most analyses of Turing's work take up its implications for two domains. On the one hand, its implications for philosophy, logic, mathematics and formal theories of computation and AI (as in, e.g., Fazi's (2018) work); and on the other hand, for the practical construction of digital computers and 'artificially intelligent' systems and the economic, social, political and cultural developments, positive and negative, both have shaped but have also been shaped by (as in, e.g., Agar's 2003, 2017 work). As a result, Turing's machinic respecification of 'intelligence' as a practically reproducible matter of computational engineering, has not been traced through in relation to the situated courses of methodic work in and through which Turing developed it. This 'missing' element in treatments of Turing's work, how it might be opened up and what it might reveal is something we came to notice on the basis of reading Turing alongside prior ethnomethodological studies of scientific and technical practice, including Ziewitz's (2017) "experiments with the ethnomethods of the algorithm". In circumstances where the original courses of practical activity being explored are inaccessible, as is the case in relation to Galileo's demonstrations (Garfinkel, 2002; Bjelić, 1996, 2003; Livingston, 1995b) or Goethe's experiments with colour (Bjelic and Lynch, 1992), through re-enactments of demonstrations and experiments, ethnomethodologists have sought to make at least some of the contingent specificities of the practices involved available again, "for



Components of a Turing Machine (TM)

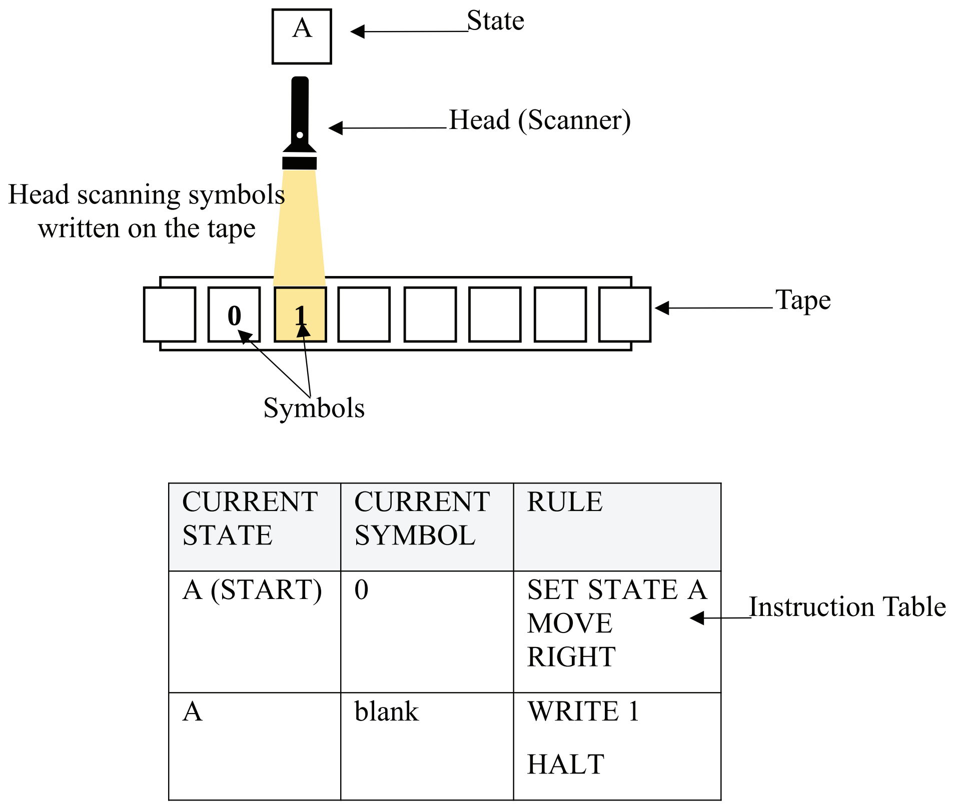
CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A	blank	WRITE 1 HALT

Consulting Turing's 'recipe', we learned a TM should be imagined consisting of three or four central components, all of which draw upon an assumed familiarity with objects such as magnetic tape recorders or punched card readers.

Clip 1. Video: "A machinic talk-through of a Turing Machine". This video is available at: https://www.youtube.com/watch?v=Ln_WC9pARoE another next first time" (Garfinkel, 2002: 98, 216). Since re-enactments are by their nature subject matter specific, this study is, thus, a contribution to, rather than an application of, a growing body of work in ethnomethodology that mobilises re-enactments in engagements with science and technology. At the same time, it is also a contribution to debates about diversifying methodological repertoires within STS (Lippert and Mewes, 2021; Silvast and Virtanen, 2023). On the latter, ethnomethodological re-enactments can profitably be read alongside related work being developed in other areas of STS (see, e.g., Kirksey et al., 2021). While the ethnomethodological character of studies such as ours is distinctive, we also view such studies as sites for productive dialogue in STS, as our own engagement with the work of Agar, Jaton, Fazi and others goes some way to demonstrating.

Re-enacting Turing: "On Computable Numbers" as a site of technical self-instruction

An important initial question for any such endeavour, as Bjelić notes (2003: 133), is; where to start? Before we could begin to construct our own TM, our initial engagement with Turing's text made it clear we needed to familiarise ourselves with its constituents. Consulting Turing's 'recipe', we learned a TM should be imagined consisting of three or four central components, all of which draw upon an assumed familiarity with objects such as magnetic tape recorders or punched card readers, key technologies of Turing's day. First, a 'tape' which is divided into equal-sized blocks where each block can contain a single symbol at most (see Figure 1). Second, a 'head' or a scanner which can move either left or right to scan the symbols written on the tape, with the machine also having the capacity to erase an existing symbol or write a new symbol on the tape. At any given time, Turing tells us, the machine is in a particular 'state'—therefore, a means of recording and identifying the current 'state' of the machine is required too. That identifiable machine 'state', i.e., what at any moment it is set up to do, is the third component of a TM. Based on the state and the symbol currently being scanned, a TM determines which instruction is to be carried out next. These instructions are to be represented in a table-like format (see Figure 1) akin to the tables



CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A	blank	WRITE 1 HALT

Figure 1. Components of a Turing Machine

human computers would follow. This 'instruction table'—Turing calls them 'configuration tables' because they constitute structural arrangements of the machine—is the fourth and final component of a TM. While the third and fourth components can be combined, it made sense to us to keep them separate as it allowed us to more easily track and make explicit the logic of the machine's parts in terms of their respective functions, particularly important given our machine was to be used as a demonstration device. Moving from the formal recipe to a working version of the diagrammed schematic depicted in Figure 1 helped us in that regard but it remained a preliminary step. We still needed to work through a set of operations which would enable us to both explore and elaborate how we could animate the machinery, putting it to computational work, and for that we needed a concrete application, something the TM could process and in as clearly followable a form as possible.

For our present purposes, we decided to take a simple arithmetic operation as our "tutorial problem" (Garfinkel, 2002: 145) so we set out to build a TM based on our schematic that could check if a number is divisible by three. The first issue we faced was this: how would we use the "engineered design" (Garfinkel, 2002: 268) Turing bequeathed us and which we had just familiarised ourselves with to determine divisibility by three? We needed something that could be sequentially processed through elementary non-intellectual steps and which could operate in line with the components listed above. We thus had to formalise the problem. We settled on finding the remainder left when we divide a number by three as it allowed us to introduce a binary logic to the machine's operations. That is, if the machine indicated that a remainder was zero, we could then conclude the number was divisible by three. If the machine gave us back any number other than zero, we could conclude the reverse. This way of finding remainders is called a 'modulo operation' in computing. While modulo operations can be performed with any two numbers, to keep our TM as simple as possible we restricted the divisor to 3. However, the dividend in this case had to be extended to any possible natural number if our TM was to do its projected job. In setting the TM up, we were, then, directing it to work through how many times the divisor would go into the dividend—whatever number that might happen to be—and we were to call that number the quotient, and whatever was left over we were to call the remainder. For example, if we divide 7 by 3, we get 2 as the quotient (as 3 goes twice into 7) and 1 as the remainder $(7-(3*2)=7-6=1)$. If we were able to set our TM up effectively, it should indicate the remainder is not zero in this case, enabling us to conclude that 7 is not divisible by 3.

From many examples like this one, we can derive the following formalisation/formula:

$$remainder = dividend - (divisor * quotient)$$

It was this formula we generalised into a method for finding remainders that we wanted to implement using a TM. To simplify our local specification of Turing's design further, reducing the parameters of the problem operationally, we realised we should restrict the symbols on our TM's input tape to 0, 1, 2 and 'blank'—for 'do nothing else' or 'halt'—as the only symbols which could be scanned, thus limiting the number of instructions we would have to write for it. On top of this, informed by the way digital electronics offers simpler implementation for binary systems, we would opt to use binary numbers to represent the dividend as part of reducing the number of symbols required to represent it. If we were to opt instead for the decimal system, we would need a set of 10 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9) and a much more complex set of accompanying instructions by extension. Using the binary system to codify the dividend, by contrast, meant we would only need to use two symbols (0 and 1) to represent any natural number. This would also reduce the number of configurations required to perform the computation. These choices had a neat symmetry: our TM would only need to be able to read and write 0, 1 and 2 as we could use 0 and 1 to represent every possible dividend from a given input sequence; and we could also use all three of them for our output sequence with 0, 1 and 2 as the only possible remainders when we divide a number by three. The 'blank' symbol would be there to instruct the TM to stop. Finally, to further simplify our TM in comparison to those in Turing's paper, we disallowed backtracking and instead restricted the TM to moves in one direction. That is, our TM would not go backwards and forwards along the tape selectively, but instead 'dumbly' proceed through the symbols it was presented with one by one in linear sequential order.

Our strategy from there in implementing this 'solution', as such things are called, was to start at the simplest possible point, at first working on and testing instructions we'd need to set out for checking the three-divisibility of an 'easy' number that would have 0 as the dividend. Then we wanted to gradually increase that number with every subsequent step so as to ensure our TM would not skip potentially relevant cases and help us to see what we would need to do to get the TM to handle *any* number. For each of those steps we would write down the tabular instructions—the machine configuration—required to perform the computation in that step, a way of "reverse engineering" the computational mechanisms we needed the TM to be built around step-by-step in parallel with the unfolding logic of the solution we were seeking to develop via the TM (see Brooker and Mair, 2022). In other words, rather than work out the instructions in advance, we would specify them *as we went along* to give us the results we expected in relation to the computational problem at hand (an approach we might characterise as central to programming's work more generally).

Understanding the components and having a plan is one thing, however, putting both into action another. How to get the TM going? As we learnt from Turing, a TM can be started with an input string—a sequence of symbols written somewhere on the tape—as long as we specify all the states at which the machine can start scanning that input string. However, the machine can only have a finite number of states so some of these states need to be denoted as 'START' states. Similarly, we also needed 'END' states to instruct the machine when to halt its operations i.e., at the end of computation. In any given state, a TM can find any possible symbol accepted by the machine, and thus, we needed to write *all* the possible combinations of states and symbols in the instruction table so as to avoid our TM encountering trouble in the form of missing instructions. This way a TM instructably moves from a START

CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT

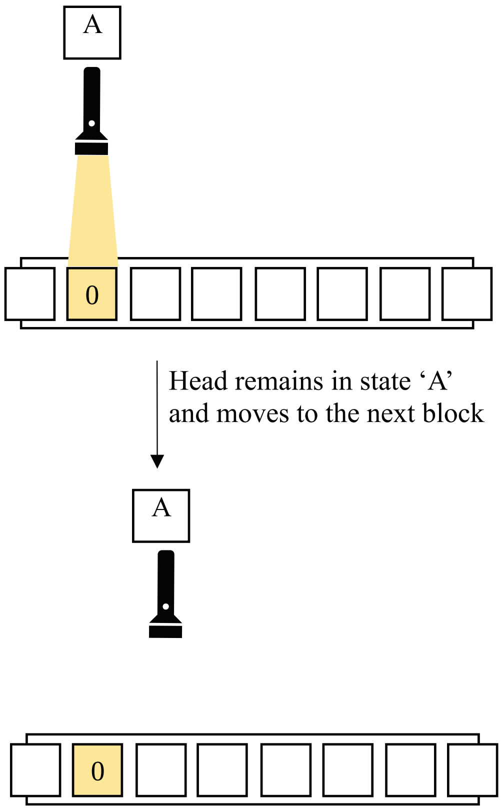


Figure 2. Three-divisibility of '0'

CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A (END)	blank	WRITE 0 HALT

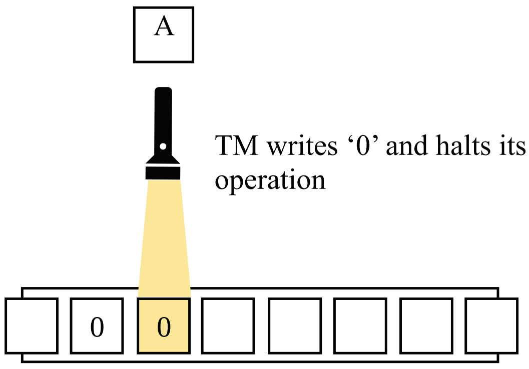


Figure 3. Three-divisibility of ‘0’ (contd.)

state to an END state to perform a given computation.

With all that covered and retracing the ground of our re-enactment, let us start by checking the three-divisibility of 0: as 3 is not contained within 0, both the quotient and the remainder $(0-(3*0)=0)$ are 0 in this case. The binary representation of 0 is also 0 and we write it down on the tape (see Figure 2). So, in this case, at the beginning of the computation, the TM will find 0 on the tape. Following Turing’s instructions more or less, this is the ‘START’ state of the machine and we represent it as ‘A’. So, the current symbol at the current state ‘A’ is 0. When the machine is in this situation, we instruct it to remain in state A and move the scanner to the right. We write down this instruction under the ‘RULE’ column of our instruction table. At this stage, our instruction table looks like Figure 2.

As instructed, the TM’s scanner moves to the next block, and it finds a ‘blank’, an empty block that does not contain any symbol (see Figure 2). So, currently the state is A, and the symbol is blank. In this case we already know that the remainder should be 0. So, we instruct the machine to write down the output 0 at the current empty block and halt the computation (Figure 3). This particular state where the TM halts its operations is one ‘END’ state of the machine. This way for input 0, we can find the correct output 0 as the remainder, with zero being divisible by any integer.

Next, we move from considering the three-divisibility of 0 to considering that of 1, which is 01 in binary. So, instead of a blank, imagine that our scanner encounters a 1 in its place in the last step (i.e., the sequence becomes 01 (see Figure 4)). However, in this case, the machine needs to be moved into a new state because our instruction table does not yet contain any instructions to address the case when the remainder is 1 ($1-(3*0)=1$). We will call this state ‘B’.

As instructed, the TM will move its scanner to the next block, and it will find another blank there. Where the current symbol is a ‘blank’ and

CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A	blank	WRITE 0 HALT
A	1	SET STATE B MOVE RIGHT

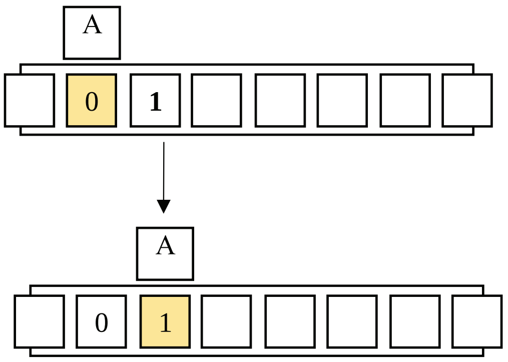


Figure 4. Three-divisibility of ‘01’

CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A (END)	blank	WRITE 0 HALT
A (START)	1	SET STATE B MOVE RIGHT
B (END)	blank	WRITE 1 HALT

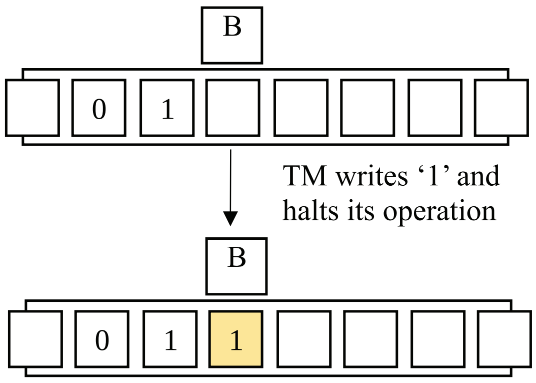


Figure 5. Three-divisibility of '01'

the current state is 'B' (see Figure 5) we have a new situation for our machine, and it needs to be represented in the configuration table. As we know the remainder in this case is 1, we will instruct the machine to write 1 before halting its operations as per the updated instruction table n Figure 5. In terms of how we progressively built our learning into the instructions we were developing as we moved along, because we were representing our dividend in binary, at the beginning of a computation, i.e., in the TM's START state, our TM could encounter either 0 or 1.

CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A (END)	blank	WRITE 0 HALT
A (START)	1	SET STATE B MOVE RIGHT
B (END)	blank	WRITE 1 HALT
B	0	SET STATE C MOVE RIGHT

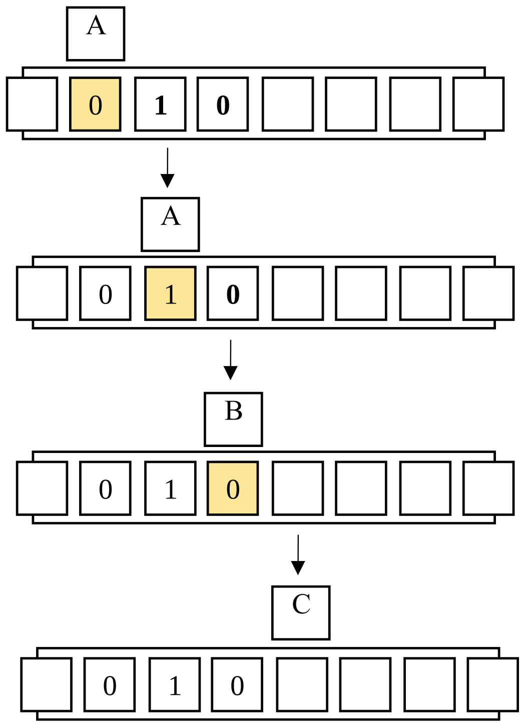


Figure 6. Three-divisibility of '010'

We had just worked through the situation where our TM encountered 0 in its START state (Figure 3) and thus, to incorporate that, we also needed to add the '(A, 1)' configuration to the table in Figure 5 as another possible START state. In this way, our list of instructions started to grow and feed into one another.

Now, if the sequence did not end there and the scanner finds 0 instead of blank in the last step, the sequence now becomes 010 (see Figure 6) which equals 2 in the decimal system. In this case, the remainder should be 2 ($2-(3*0)=2$). This is *again* a new situation, so we again need to instruct the machine as to what should be done in this case.

First, following Turing again, we will call this state 'C'. After scanning 0, 1 and 0 respectively, as the machine is in state C, if it finds a blank in the next block, we need to instruct the machine to write 2 as the remainder in this place before halting the operation (see Figure 7). This is another possible END state where the machine could terminate its operations. As even a small number of initial cases makes clear, we could continue the same kind of procedure for all the subsequent numbers, devising instructions for different scanned sequences as we go.

Knowing a priori that mathematically there cannot be a remainder larger than two, we now anticipate that when applying these instructions to numbers larger than two we will see a pattern in the output where 0, 1 and 2 keep appearing as outputs in an orderly manner as we add digits at the end of our sequence. It is also notable that we have developed three categories of states to deal with three-divisibility through the step-by-step work we outlined above. We capture this pattern in our final instruction table (see Figure 8) where all the potential outcomes are accounted for and it is in that way that we determine whether it is possible to find the remainder when we divide a number by 3 using our TM. We can therefore now test our TM with a binary sequence like 1001 to find out if it can correctly find the remainder and check if our intuition about the pattern in our sequences is correct. 1001 equals 9 and the remainder in this

CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A (END)	blank	WRITE 0 HALT
A (START)	1	SET STATE B MOVE RIGHT
B (END)	blank	WRITE 1 HALT
B	0	SET STATE C MOVE RIGHT
C (END)	blank	WRITE 2 HALT

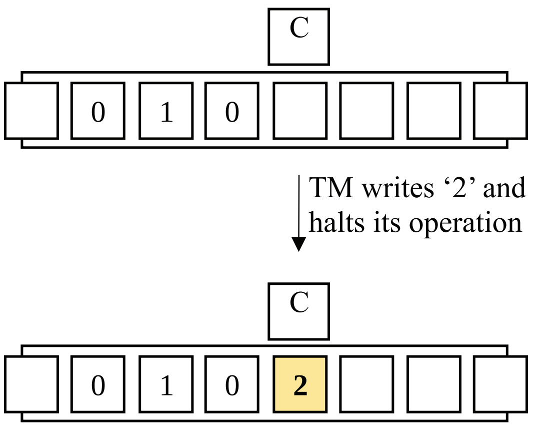


Figure 7. Three-divisibility of '010' (contd.)

case is 0. If we follow the instruction table in the following order, the machine will eventually write 0 as output. We again invite readers to verify our TM's instruction table by working through their own test inputs at this point.

The diagrammed demonstrations above prove that the instruction table we have devised works for any possible natural number: this TM can solve not just a problem but a "class of problems" (Livingston, 1995a: 113). Our solution thus constitutes an 'effective procedure', i.e., a mathematically sound algorithm, because it is a generalised solution to the computational problem we set ourselves where the solution is reached by following a finite set of instructions. The process by which we have determined divisibility by 3 is effective in these terms because it adequately captures elementary, mechanisable and thus 'computable' steps in Turing's sense adequate to undertaking the task as specified (i.e., ascertaining the three-divisibility of any natural number).

The divisibility problem in our demonstration is described in terms of the observable and observed constituents of the problem's arithmetic properties as they became computationally relevant in the context of building our TM; "normal troubles"

CURRENT STATE	CURRENT SYMBOL	RULE
A (START)	0	SET STATE A MOVE RIGHT
A (END)	blank	WRITE 0 HALT
A (START)	1	SET STATE B MOVE RIGHT
B (END)	blank	WRITE 1 HALT
B	0	SET STATE C MOVE RIGHT
B	1	SET STATE A MOVE RIGHT
C (END)	blank	WRITE 2 HALT
C	0	SET STATE B MOVE RIGHT
C	1	SET STATE C MOVE RIGHT

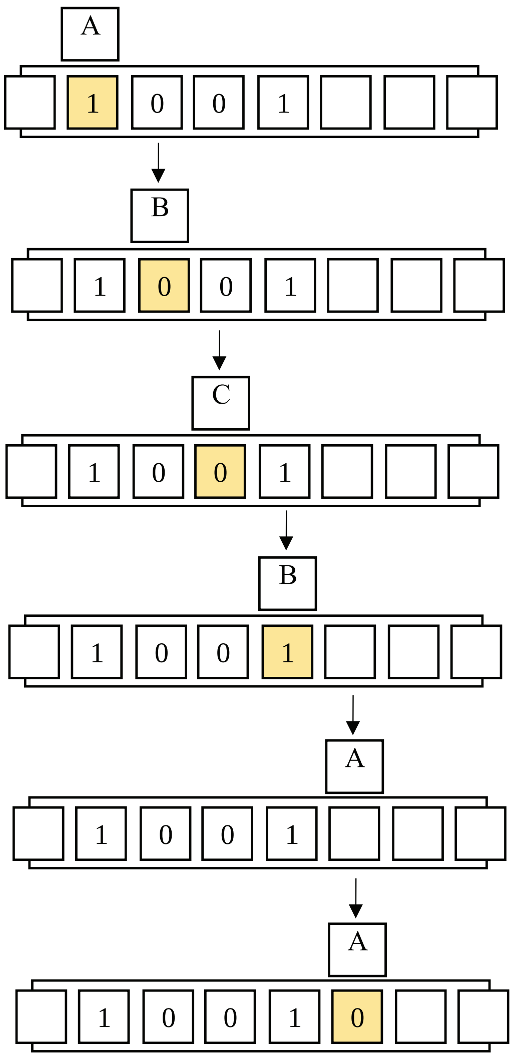


Figure 8. Three-divisibility of '1001'

(Garfinkel, 1967) of getting the machine to work, as and when we encountered them. If we were to set out to solve this problem using a modern programming language like Python or Java, we would not have to build a TM at all—indeed, such languages are often dubbed 'high level' precisely because their operations are rarefied far beyond the mechanical aspects of transistors switching between binary states. However, each time programmers write a program to solve a mathematical or logical problem like this, regardless of their language of choice, they have to engage in a process of mapping that problem into processes that can be handled within the computational systems they are working with, just as we have here. The formal possibility of so doing is exactly what Turing demonstrated in his paper.

Discussion

Turing wanted to make his machines "automatic", dependent only on a set of pre-defined configurations for their operation. These machines run "automatically" in the sense that once initialised, "external operators" are only needed when the computation cannot move forward without further inputs from them (Turing, 1936: 232). As a consequence, the role of human workers (allegedly) ends in designing and implementing the instruction table, and once it has been implemented, as long as all outcomes have been anticipated and are handled accordingly, the machine should be able to carry out the instructions in the prescribed order. Hence, as we have seen first-hand by virtue of undertaking this exercise, mechanising a computational procedure also includes *eliminating* the work that went into devising that procedure in the first place. Once it was complete, we no longer appeared in the TM's running, in spite of the TM's operations sense and meaning *only* being furnished by reservoirs of practical, mundane reasoning about problem decomposition that we had to engage in, through and as part of the TM's very construction. Our situated courses of practical reasoning in assembling our TM were progressively 'enchained', to adapt a phrase from some of Garfinkel's (2022: 189) recently published work, *in* the TM's operations.

This leads us back to the computing work Turing was doing in the 1936 paper. Seen in a praxeological light, Turing's paper furnishes a logico-mathematical or conceptual programme—a set of methods—for assembling a computing machine, with the sections offering instructions as to what goes into their assembly and how they are to be engineered to execute calculations. We showed that this involved putting the operations of the Turing Machine centre stage while backgrounding the methodic work Turing did in setting out the instructions it could be capable of following. What makes the latter difficult to recover—and what necessitated the re-enactment—is the intentional elision of the operations of the machine and the methods for instructing it, with the latter seemingly written 'into' the arrangements of the machine (the sense of which, albeit, can *only* ever be recovered via further practices of local reasoning). This is, therefore, a phenomenon that consists of two irreducible parts and so is 'paired' in ways that Garfinkel (2007) as well as Livingston (1986) and Bjelić (2003) sought to elaborate in their work from the 1970s on. That is, we have the formalisation of the computation in the form of the TM itself, on the one hand, and the practical work of composing the instructions that constitute it, on the other, and the two are inextricably linked.

In our attempt to solve an arithmetic problem using a TM built 'from scratch', the computational work involved became recognisable in and through the steps of ordering the instructions to it. That is, the solution's generality became evident in the followable character of those constitutive instructions from within the process of implementing that solution via the specifics of the TM's engineered design. In the course of that computational work, when those instructions were followed in a 'mechanical' fashion, we arrived at something that could be worked through as a solution to our problem, which in turn proved that an effective procedure or algorithm exists that solves an entire class of arithmetic problems, however limited those problems might have been. In other words, the formal construction of our abstract machine *through* the composition of instructions was what yielded an effective procedure or algorithm, albeit an unwieldy one.

The formal representation of our efforts—the instruction table—does not, however, make the situated and contingent character of the work that has informed it evident. This is precisely why we sought to specify the practices that informed the TM's computational workings. The practical 'details' of our computing work do not have to be and are *not* made explicit in the process of achieving such things as formalisation, generalisation and reduction, just as they are not made explicit in Turing's (1936) original demonstration. In our case, it was the instruction table which made our abstract machine 'automatic' in Turing's terms, while we found the work of formalisation, generalisation and reduction as its "shopfloor problem" constituents, i.e., practical problems we had to solve to get going with the building of a working machine (Garfinkel, 2002). These constituents can only be accessed in and through the 'lived work' of computation, be it on paper while building TMs or on screen while writing computer programs. In the case of programming, it is the computer programs that make those constituents recognisable in the work of writing them. As such, thinking like a machine emerges as *the* praxeological supplement to 'the thinking machine'; this 'thinking machine', then, is silently supported by the wealth of underlying reasoning practices and hands-on work by and through which it is produced.

Turing's practice shows us, therefore, that methods of writing instructions in machine executable terms are constitutive of the machines so instructed. While we have applied rather than rediscovered Turing's design, our tutorial problem has supplied us with important practical lessons in that regard. To adapt Bjelić's (2003) work on Galileo to Turing,

> When … [Turing] proposed the specifications for the … [machine], he unintentionally left a set of practical contingencies for … practitioners to find and resolve according to the specific local conditions of their work … [The] structures and their descriptions of the discovery of … [effective algorithmic procedures using those machines] are available only where the discovery is reproduced. (Bjelić, 2003: 135)

For instance, our capacity to produce a TM-based solution to an arithmetical problem depended on such things as: our choice of problem, an elementary mathematical and hence potentially culturally more accessible one (including, for instance, unstated assumptions around the significance and utility of operations such as determining divisibility); the formatting of inputs to the device as part of the 'language of instruction'; and the way in which we built the TM around (and in line with) equally elementary computational steps undertaken in a sequence which we established as we worked through it. Major issues Turing's paper did not help us settle but which we had to resolve by 'best guess' included: just how many components can be said to be minimally involved in the construction of a TM, three or four, and what might formalising that either way make visible? And how does 'the tape' being scanned come to us? Are numbers already printed or are we to conceive of ourselves as writing it as we go for demonstration and testing purposes? The way we developed our procedure, the latter was more accurate even though that meant the TM could ultimately handle the former too. Our TM calculates remainders as part of mechanically determining divisibility by 3; it does so 'on its own', but we now have a much better sense of how this 'on its own' is foundationally reliant upon a scaffold of elided, reasoned activities. What we have come to see, by virtue of our course of instruction in the TM's specific mode of operation, is that the relation and categorical shift between humans and machines is something we are *diverted* from seeing—no different to the case for many new AIs—not because we lack an understanding of intelligence, the brain or mind but because of the very practices through which computing machines are produced.

Now, our TM does things for sure, but not in the ways we ordinarily do nor even in the ways we specifically did in working its design through; it runs its operations on binary, for example, and we worked them out that way, but we *chose* binary over a decimal system, where the point (at least on this particular aspect) is that *we* saw the sense in which working with binary would be a useful thing to do in this domain in just the same way that the designers of contemporary AI systems do, even those described as 'autonomously intelligent'. How the machines work is not a surprise, in other words, but the outcome of a process of practically stipulating parameters in pursuit of a working model. Most importantly, as in Turing's work but as is also the case in programming work more generally, all the choices and decisions we made assumed and traded upon an open-textured background of shared practices and understandings against which an activity of this sort acquired whatever cultural intelligibility it may be taken to have. This is a lesson learned that may lead us to take a more cautious approach to claims made on behalf of new AI technologies which (some have claimed, as outlined above) comprise AI's much heralded 'autonomous systems' that 'do things for themselves'. Take AlphaGo; one of the headlines grabbing AI systems of the past five years. Our re-enactment of Turing's methods furnishes insights into how we might approach such technologies. How so? We return to Jaton :

> I shall … temporally define computer programming as the situated activity of inscribing numbered lists of instructions that can be executed by computer processors to organize the movement of bits and to modify given data in desired ways … If I place emphasis on the practical and situated aspect of computer programming in my operational definition, it is because important historical events have progressively set it aside … [Once] computer systems started to be presented as input-output instruments controlled by a central unit – following the successful dissemination of the so-called von Neumann architecture – the entangled sociotechnical relationships required to make these objects operate in meaningful ways had begun to be placed in the background. If electronic computing systems were, in practice, intricate and highly problematic sociotechnical processes, von Neumann's modelization made them appear as functional devices transforming inputs into outputs. The noninclusion of practices – hence their *invisibilization* – in the accounts of electronic computers … led to serious issues. (Jaton, 2020: 93)

While von Neumann's formalisation of the computer was a significant achievement, in other words, it involved a specific kind of disappearing act; that is, it problematically disappeared the practical work of "making a universal machine" (Jaton, 2020: 103) as well as the people who made critical contributions to that work, engineers and *computers*, many of whom were not, contra to the received histories, white and male as Jaton points out. But if von Neumann effected a disappearing act of this kind, we believe it depended on a prior one initiated by Turing who in his 1936 paper succeeded in disappearing *himself*. As we have shown above, a non-praxeological reading of Turing is liable to direct us away from the point that even before hardware is built and ways to operate that hardware to perform meaningful tasks are designed, the work of computation (e.g., mathematics) has to be *done*; it will not do itself. Hence, we must be alive to the contemporary versions of Turing's self-disappearing act if we are to properly get the measure of computation, especially for "the new AI" (Fuchs and Reichert, 2018) where the accompanying sales pitches and commentary often obfuscate rather than illuminate just how these systems work and have come into being (cf. Holton and Boyd, 2021).

Even those with an otherwise deep understanding of the issues can still fall foul of these problems when it comes to assessing these technologies. In a reflection on AlphaGo Zero, a much more powerful successor to the AlphaGo algorithm (created by Google DeepMind) which beat the human world Go champion, Lee Sedol, in 2016, Fazi (2021) makes allusions to a machine operating purely autonomously from human involvement:

> While much of computer programming has historically consisted in making human abstraction significant and operative within the instrumental remit of algorithmic machines, with deep learning we face the opposite case: the abstractions and consequent instructions the machine gives itself now require interpretation for them to be significant and operative for humans. The modes of organisation, categorisation and classification that belong to the abstractive operations of these computational cognitive agents are indeed incommensurable. Maintaining a theoretical focus on the nature and possibilities of abstraction as the balance moves between autonomy and automation within AI thus involves acknowledging and working with the prospect of modes of abstracting that might arise within calculation but also surpass the boundaries of human

> cognitive representation … [The] 'autonomy of automation' … regarding abstractive operations is demonstrated by a deep learning system producing internal representations independently from the phenomenological or experiential ground of the human programmer … [In the] example of AlphaGo Zero, such an autonomy is doubled: not only the outputs but also the training inputs are somewhat independent from human knowledge. (Fazi, 2021: 15)

We take very seriously Fazi's point that we need to avoid conflating the operations of new AIs with our practices, an incommensurability argument which parallels that of Shanker's, and share her scepticism with respect to totalising systems. However, Fazi has also here succumbed to Google DeepMind's successful disappearing act in hinting at 'independence'. For what is entirely missing here is any account of how the researchers involved got from AlphaGo to the successor algorithm and the work that went into *it* as an "engineered design"—where to illuminate this and recover the ways in which AIs are woven both out of and into practices, an approach of the kind we have outlined above is required. While Turing's machines are certainly unwieldy when judged by contemporary standards—for instance, our 'three-divisibility' algorithm could be optimised further rather than sequentially proceed through numbers one by one *ad infinitum*—it is worth noting that with enough time, patience and "ingenuity", to return to Davis, we could simulate AlphaGo Zero using Turing's components. The resulting TM programme would be extremely complicated, however, extending far beyond the instruction table sketched above. That alone should alert us to the dangers of any claim that automation has been 'automated' or that an AI has achieved 'independence' in this domain: AIs cannot produce themselves, any more than any computational system can, and we lose sight of that point—and by corollary, the practices and material set ups that do such important enabling work in the realm of these machines—at our conceptual and methodological peril.

Conclusion: grappling with Turing's 'disappearing act'

As the burgeoning literature attests, the social sciences and humanities, like much of the rest of the world, are in the process of getting to grips with the disparate technologies which comprise the contemporary field of artificial intelligence (AI) and which underpin its rapid and often highly problematic advances over the last decade and more. Real strides have undoubtedly been made along that path—as interested publics, we all understand a great deal more than we did even a few years ago—but, we would contend, erasures and misunderstandings persist. Here in particular, and precisely because they have been designed that way, it is all too easy to accept claims regarding the agentic status of the new AI's signature systems without looking any further. In that context and building on important work already conducted on that front, we have tried to open up the praxeological foundations of machine computation as a corrective to lingering reifications of the 'thinking machine' (Garfinkel, 2002). Reading Turing alternately, to draw on Garfinkel a final time, we have argued that the construction of such machines as a formal accomplishment constitutes a paired phenomenon connecting the execution of a function with the writing of instructions which enable that function to be so executed while working within a particular computational architecture. On this basis, we have argued that the work of instruction represents an irreducible praxeological supplement to the construction of 'the autonomous machine' and while they are asymmetrically related, they are mutually dependent and mutually informative. Jones-Imhotep (2020) has recently argued that machine autonomy is a carefully crafted performance on a stage set for an audience with specifically cultivated sensibilities who are primed to see the machine in quite particular ways, i.e., as operating without external intervention. If Jones-Imhotep is right, we need to understand what goes into stabilising such performances in the field of contemporary AI, including the various disappearing acts performed along the way, if we are to arrive at a more consistently deflationary rather than inflationary view of contemporary AI's actual achievements. It is only by proceeding in that way that we will be in a viable position to show in any particular case, as we hope to have done via our re-enactment, what computers can do and how we help them to do it

Acknowledgements

We would like to thank the two anonymous reviewers and the editor for their valuable comments and suggestions. This research has been funded by UK Research and Innovation (project reference: 2273902) through the North West Social Science Doctoral Training Partnership, and the Engineering and Physical Sciences Research Council (grant number EP/T022493/1).

References

Agar J (2003) *The government machine: a revolutionary history of the computer*. Cambridge: The MIT press.

Agar J (2006) What difference did computers make? *Social Studies of Science* 36(6): 869-907.

Agar J (2017) *Turing and the Universal Machine (Icon Science): The Making of the Modern Computer*. London: Icon Books Ltd.

Benbouzid B (2019) Values and Consequences in Predictive Machine Evaluation. A Sociology of Predictive Policing. *Science & Technology Studies* 32(4): 119-136.

Bjelic D and Lynch M (1992) The work of a (scientific) demonstration: Respecifying Newton's and Goethe's theories of prismatic color. In: Watson G and Seiler RM (eds) *Text in context: Contributions to ethnomethodology*. London: Sage Publication, pp. 52-78.

Bjelić DI (1996) Lebenswelt structures of Galilean physics: The case of Galileo's pendulum. *Human Studies* 19(4): 409-432.

Bjelić DI (2003) *Galileo's Pendulum: Science, Sexuality, and the Body-Instrument Link*. Albany: State University of New York Press.

Brock K (2016) The 'FizzBuzz' Programming Test: A Case-Based Exploration of Rhetorical Style in Code. *Computational Culture* (5).

Brooker P, Dutton W and Mair M (2019a) The new ghosts in the machine: 'Pragmatist' AI and the conceptual perils of anthropomorphic description. *Ethnographic Studies* 16: 272-298.

Brooker P, Sharrock W and Greiffenhagen C (2019b) Programming Visuals, Visualising Programs. *Science & Technology Studies* 32(1): 21-42.

Brooker P and Mair M (2022) Researching Algorithms and Artificial Intelligence. In: Housley W, Edwards A, Beneito-Montagut R and Fitzgerald R (eds) *The SAGE Handbook of Digital Society*. London: SAGE Publications Ltd, pp. 228-246.

Burrell J (2016) How the machine 'thinks': Understanding opacity in machine learning algorithms. *Big Data & Society* 3(1): 205395171562251.

Burrell J and Fourcade M (2021) The Society of Algorithms. *Annual Review of Sociology* 47(1): 213-237.

Campolo A and Crawford K (2020) Enchanted determinism: Power without responsibility in artificial intelligence. *Engaging Science, Technology, and Society* 6: 1-19.

Collins HM (1990) *Artificial experts: Social knowledge and intelligent machines.* Cambridge: The MIT press.

Elish MC and boyd d (2018) Situating methods in the magic of Big Data and AI. *Communication Monographs* 85(1): 57-80.

Fazi MB (2016) Incomputable aesthetics: open axioms of contingency. *Computational Culture* (5).

Fazi MB (2018) *Contingent computation: abstraction, experience, and indeterminacy in computational aesthetics.* Lanham: Rowman & Littlefield.

Fazi MB (2021) Beyond Human: Deep Learning, Explainability and Representation. *Theory, Culture & Society* 38(7-8): 55-77.

Fuchs M and Reichert R (2018) Introduction: Rethinking AI. Neural Networks, Biometrics and the New Artificial Intelligence. *Digital Culture & Society* 4(1): 5–13.

Gandy R (1988) The confluence of ideas in 1936. In: Herken R (ed) *A Half–Century Survey on The Universal Turing Machine*. Oxford: Oxford University Press, pp. 55-111.

Garfinkel H (1967) *Studies in Ethnomethodology*. Englewood Cliffs: Prentice-Hall, Inc.

Garfinkel H (2002) *Ethnomethodology's Program: Working Out Durkheim's Aphorism*. Lanham: Rowman & Littlefield Publishers, Inc.

Garfinkel H (2007) Lebenswelt origins of the sciences: Working out Durkheim's aphorism. *Human studies* 30(1): 9-56.

Garfinkel H (2022) *Harold Garfinkel: Studies of Work in the Sciences*. London: Routledge.

Holton R and Boyd R (2021) 'Where are the people? What are they doing? Why are they doing it?'(Mindell) Situating artificial intelligence within a socio-technical framework. *Journal of Sociology* 57(2): 179-195.

Jaton F (2020) *The Constitution of Algorithms: Ground-Truthing, Programming, Formulating*. Cambridge: The MIT Press.

Jones-Imhotep E (2020) The ghost factories: histories of automata and artificial life. *History and Technology* 36(1): 3-29.

Kirksey E, Hannah D, Lotterman C and Moore LJ (2021) The Xenopus pregnancy test. In: Rogers HS, Halpern MK, Hannah D and de Ridder-Vignone K (eds) *Routledge Handbook of Art, Science, and Technology Studies*, pp. 163-178.

Lee F (2020) Enacting the Pandemic: Analyzing Agency, Opacity, and Power in Algorithmic Assemblages. *Science & Technology Studies* 34(1): 65-90.

Lippert I and Mewes JS (2021) Data, Methods and Writing: Methodographies of STS Ethnographic Collaboration in Practice. *Science & Technology Studies* 34(3): 2-16.

Livingston E (1986) *The Ethnomethodological Foundations of Mathematics*. London: Routledge & Kegan Paul plc.

Livingston E (1995a) *An anthropology of reading*. Bloomington: Indiana University Press.

Livingston E (1995b) The idiosyncratic specificity of the methods of physical experimentation. *The Australian and New Zealand Journal of Sociology* 31(3): 1-22.

Lynch M, Livingston E and Garfinkel H (1983) Temporal order in laboratory work. In: Knorr-Cetina KD and Mulkay M (eds) *Science observed: Perspectives on the social study of science*, pp. 205–238.

Lynch M and Lindwall O (eds) (forthcoming) *Instructed and Instructive Actions*. Routledge.

Mackenzie A (2017) *Machine learners: Archaeology of a data practice.* Cambridge: The MIT Press.

Mair M, Brooker P, Dutton W, and Sormani P (2021) Just what are we doing when we're describing AI? Harvey Sacks, the commentator machine, and the descriptive politics of the new artificial intelligence. *Qualitative Research* 21(3): 341-359.

Petzold C (2008) *The annotated Turing: a guided tour through Alan Turing's historic paper on computability and the Turing machine*. Indianapolis: Wiley Publishing.

Piccinini G (2003) Alan Turing and the Mathematical Objection. *Minds and Machines* 13(1): 23-48.

Rieder B (2020) *Engines of order: A mechanology of algorithmic techniques.* Amsterdam: Amsterdam University Press.

Seaver N (2019) Knowing Algorithms. In: Vertesi J and Ribes D (eds) *digitalSTS: A Field Guide for Science & Technology Studies*. Princeton: Princeton University Press, pp. 412-422.

Shanker SG (1987) Wittgenstein versus Turing on the nature of Church's thesis. *Notre Dame Journal of Formal Logic* 28(4): 615-649.

Shanker S (1995) Turing and the Origins of AI. *Philosophia Mathematica* 3(1): 52-85.

Shanker SG (2002) *Wittgenstein's remarks on the foundations of AI*. New York: Routledge.