Tangled Tapes: Infinity, Interaction and Turing Machines

Paul Cockshott(1), Greg Michaelson(2)

(1) School of Computer Science, University of Glasgow, Sir Alwyn Williams Building, Lilybank Gds, Glasgow, Scotland, G12 8QQ
william.cockshott@glasgow.ac.uk
(2) School of Mathematical and Computer Sciences, Heriot-Watt University, Riccarton, Scotland, EH14 4AS
G.Michaelson@hw.ac.uk

Key words: Turing machine, tape, infinite, interaction

Abstract. In the sixty-five years since Turing introduced his eponymous machines, many popular misconceptions about their properties have become current. In this paper, we explore the notions that Turing machines have infinite tapes and that their expressive power is limited by by an inability to interact with the wider environment during computations.

1 Introduction

As time passes since the death of a great scientist, they become better known. More people have heard of them, and more people have heard of their work. But it does not follow that more people actually know their work or still read it. Newton like Darwin and Freud is still famous, but how many still read Newton in the original?
As time passes their work becomes known at second or third hand, and in the process is changed. Things are omitted, and things which, whilst they were the result of the great author's work, were other's ideas, are attributed to them.
For instance one of us who was brought up on Dawkins and Gould, with their accounts of the opposition between Darwinian and Lamarkian evolution, was surprised to discover in Darwin's *Descent of Man*[4] that he was a firm believer in the inheritance of acquired characteristics. A later, post Mendel, orthodoxy has been projected back onto Darwin. One encountered not the original Darwin, but a stylised version, modified by more recent concerns.
Turing, who is so much more recent than Darwin and whose publications were much less extensive, is already showing signs of being transformed in a similar way. Far more computer scientists have read accounts of the Turing Machine in textbooks than have read his original description, and as a result a number of misconceptions have taken hold. In this paper, we explore two misconceptions about properties of Turing machine tapes, that is that they are infinite and that they may not be subject to external change during computations.

2 Finite and infinite tapes

There is a widespread notion that Turing machines have infinite tapes. However, in Turing's original paper[18] there is no mention of the tape having to be infinite; instead Turing says: "The machine is supplied with a "tape", (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol S(r) which is "in the machine"". The length of the tape is not specified.
The tape is not specified as infinite, and, since he is concerned with finite computation, it is clear that any terminating programme will have only modified a finite portion of the tape, so what he was assuming was effectively a finite but unbounded tape. An infinite tape would also have required an infinite amount of energy to accelerate and decelerate it on each step and would thus have been incompatible with requirement that computable numbers be "calculable by finite means".
It is possible that this notion of an infinite Turing machine tape arose from the independent elaboration of Emil Post's eponymous machine[16] at around the same time. Post also uses the analogy of a "problem solver or worker" manipulating "spaces or boxes" but "the symbol space is to consist of a two way infinite sequence.
Perhaps more significantly, in 1947 Turing said in a lecture that, some years earlier, he had been investigating the problem of machines with "an infinite memory contained on an infinite tape"[20]. It is not clear whether he was misrecollecting what he published in [18] or referring to unpublished investigations. to be infinite but had at least to be 'very extensive'. It is probably from this lecture by Turing himself, that the idea that the original Turing Machine required an infinite memory arose.
In practice this difference between infinite and finite but unbounded memories is not important. We know that in reality all our digital computers are finite machines, but such is the vastness of the finite, that even a tiny embedded microprocessor like the PIC with a paltry 68 bytes of RAM has so many potential states that it would take about 2^{450} times the age of the universe to run through them all. What is important for a general purpose digital computer is that its memory be 'extensive' and that this memory is modifiable during programme execution. The size of the memory is important, bur only insofar as size constrains the datasets that can be practically processed. The organisation of the memory, whether it has random or sequential access, is also of practical importance, since random access stores are faster than sequential access ones. But neither the size nor the organisation affect the type of problem that the computer can address.

> Machines such as the ACE may be regarded as practical versions of this same type of machine. There is at least a very close analogy. Digital computing machines all have a central mechanism or control and some very extensive form of memory. The memory does not have to be infinite, but it certainly needs to be very large.[20]

Subsequently, there is a curious divergence in accounts of Turing machine tapes in books on computability theory. In a collection of twenty two such

books held by one author, eleven refer directly or obliquely to the tape as infinite - see Figure 3. These include the classic texts by Davis[5], "two-way infinite sequence"(p3), Hopcroft and Ullman[10], "infinity of cells"(1969), Manna[13],"The tape...is infinite to the right"(p228), and Harel[9], "an infinite tape"(p21). Finally, Minsky[14] offers a compromise:

> "The tape is regarded as infinite in both directions. But...one can think of the tape as really finite at any particular time."(p118)

3 Interaction and Computation

The infinite tape seems to have been the first stylisation of the Turing's universal computer. A more recent stylisation is the idea that according to Turing, universal computers are not allowed to perform input output. Wegner proposes that Turing Machines correspond to the batch mainframes running procedural languages, and that workstations running object oriented languages correspond to an entirely new class of machines which he calls interaction machines.

> Turing machines transform strings of input symbols on a tape into output strings by sequences of state transitions. Each step reads a symbol from the tape, performs a state transition, writes a symbol on the tape, and moves the reading head. Turing machines cannot, however, accept external input while they compute; they shut out the external world and are therefore unable to model the passage of external time.[22]

We will consider the two issues of input output and the passage of time separately. If one looks at Turing's 1937 paper the tape is both the input output device and the read write store.

The lack of any dedicated input output hardware in the 1937 proposal could clearly have been circumvented by appropriate programming. One could arrange that say every 3rd square on the tape was to be reserved for input data, and allow the programmer to write input data onto these 3rd squares as the computation proceeds. These 3rd squares would then constitute an input channel multiplexed onto the tape.

Allow the input channel to contain blanks, Xs, 0s or 1s,. Once a 0 or 1 has been read from the input channel the machine overwrites it with an X. The machine can poll for input by inspecting the next character in the input channel, if it is blank, move back a certain number of positions to eject the tape and allow the programmer to optionally write a 0 or 1 on the square and then repeat. Obviously we would have to arrange the machine to run slow enough that there was time to write on the tape, but this does not affect the essence of the argument.

We have to ask if the lack of dedicated input output was a significant feature of Turing's ideas. We would argue that the key computational innovation he was concerned with in [18] was the Universal Computer, the computer that could emulate any other computer, and thus perform any computation. Special purpose computing machines were well known prior to Turing[17,8,6,11,7]. What was new, or at least a recovery of Babbage[12], in [18] was the idea of the Universal Computer. It is the implementation of this universality that has led to the subsequent success

of the computer industry, since it allows a single design of machine to be applied to an arbitrary number of tasks.

> It can be shown that a single special machine of that type can be made to do the work of all. lt could in fact be made to work as a model of any other machine. The special machine may be called the universal machine, it works in the following quite simple manner. When we have decided what machine we wish to imitate we punch a description of it on the tape of the universal machine. This description explains what the machine would do in every configuration in which it might find itself. The universal machine has only to keep looking at this description in order to find out what it should do at each stage. Thus the complexity of the machine to be imitated is concentrated in the tape and does not appear in the universal machine proper in any way.
>
> If we take the properties of the universal machine in combination with the fact that machine processes and rule of thumb processes are synonymous we may say that the universal machine is one which, when supplied with the appropriate instructions, can be made to do any rule of thumb process. This feature is paralleled in digital computing machines such as the ACE. They are in fact practical versions of the universal machine. [20]

To establish the minimum hardware requirement for a Universal Computer, a dedicated input output mechanism was irrelevant, but as soon as Turing starts to actually build the Universal Computer he includes dedicated input output in the design: "It may be divided for the sake of argument into the following parts Memory, Control, Arithmetic part, Input and output" [20]. From the earliest practical design of the Universal Computer, Turing saw the need for input and output. Back in 1947 he chose Hollerith cards for this task, but he saw this as a temporary restriction: "It would be quite possible to arrange to control a distant computer by means of a telephone line. Special input and output machinery would be developed for use at these out stations, and would cost a few hundred pounds at most."[20] He also proposed the use of digitisers to input analogue data : "As time goes on the calculator itself will take over the functions both of masters and of servants. The servants will be replaced by mechanical and electrical limbs and sense organs. One might for instance provide curve followers to enable data to be taken direct from curves instead of having girls read off values and punch them on cards."[20] By masters he means what are now called programmers, and by servants he means what used to be called operators. He correctly foresaw that as the machines developed 'electrical sense organs' the need for operators would diminish.

From this very early stage, prior to the first computer being built, Turing's idea of the computer was as something that was interacting with its environment. He discusses the possibility of it playing chess or modifying its own 'tables', what we would now call its programme. Wegner's idea of an interaction machine is not new, the Turing Machine design of 1947 was already an interaction machine.

Wegner further claims that Turing machines, and algorithmic computers in general, are not aware of the passage of time. In a contemporary

machine if you want to find out the time you have to make a special system call to examine a hardware clock. The algorithm in a high-level language knows nothing of the passage of time. Although most computers are provided with an auxiliary clock circuit to count the passage of time, this is just a matter of economy, not necessity. Suppose one has a simple RISC that takes a fixed number of cycles for every instruction then, at the machine code level, it becomes easy to use software emulate the existence of a hardware clock. One simply arranges the programme so that every alternate instruction increments a register used as the real time clock. Inspecting the register then gives one a measure of the passage of time. The fact that programmes in a high level language do not directly see time, does not mean that it can not be seen at the machine code level.

While identifying object oriented programming languages with a new super-Turing class of computing, might seem tempting, the fact that universal computers can be programmed to interpret object oriented languages shows that it must be a mistake. Object oriented programming is actually closer to the model of a more primitive class of computer - the finite state automaton.

The finite state automaton is the interaction machine par-excellence. On each cycle it reads an input and generates an output in response, possibly changing its state in the process (Minsky [14], page 13). Hardware im-

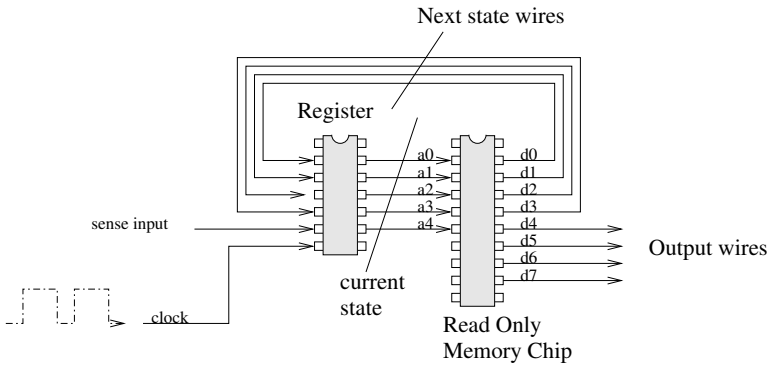

Fig. 1. Finite state automaton suitable for simple interaction and sequencing can be built out of a register and a read only memory(ROM) chip. In the picture the register latches a new address for the ROM each clock cycle. The ROM outputs a next state and a set of wires which send output actions to the environment. Thus each cycle it jumps to a new state. The behaviour of the machine is then defined by the table loaded into the ROM. One or more sense inputs can also be provided which provide additional address inputs. The presence of the sense inputs allows the next state transitions to be on the sense input.

plementations of this abstraction, Fig. 1 gain their interactiveness from

a very simple construction and this makes them suitable for high speed operations like microcode sequencing in a general purpose processor.
It is arguable that any general purpose computer will need such an interaction machine or finite state automaton as part of the control unit. When a conventional CPU interprets conditional branch instructions it relies, at a lower level, on the ability of its micro sequencer to invoke different actions on the data-path dependent on outputs from that path. This was true even of the earliest know design for a general purpose computer, the Analytical Engine of Charles Babbage. As Bromley shows [1] this had what amounts to mechanical microcode, with a mechanical microcode sequencer, Fig 2.
Rather than being the invention of fundamentally new model of computation object oriented programming, is a software embodiment of finite automata, the ur-interaction machines. The parametrised methods of a class correspond to the set of basic input symbols that are accepted by the automaton. The fields of the class to the state of the automaton.
Finite state automata are well known to be less general than von Neumann and Turing computers([14] page 26). Object oriented languages are typically more powerful than finite state machines since they allow recursion and the allocation of new heap storage locations during method allocation. These additional powers are easy to provide since we typically compile object oriented languages to general purpose CPUs. The benefit of object oriented languages is that, by modelling their main control structure on the paradigm of the finite state machine, they give software engineers easy access to the sort of interaction that hardware engineers have long been familiar with in finite state automata components.

4 Misrepresenting Turing

Wegner's paper makes a number of assertions which at best are inaccurate, both historically and technically.
For example, he states that:

> "...Turing showed in the 1930's that algorithms in any programming language have the same transformation power as Turing machines[6]"(p82)

but there were no programming languages in the 1930's and the citation "[6]" is to Turing's 1950 paper on artificial intelligence.
Wegner next states:

> "We call the class of functions computable by algorithms and Turing machines the "computable functions." This precise characterisation of what can be computed established the respectability of computer science as a discipline. However, the inability to compute more than the computable functions by adding new primitives proved so frustrating that this limitation of Turing machines was also called the "Turing tarpit." Interactive computing lets us escape from the gooey confines of the Turing tarpit."(p82/83)

Once again, computer science did not exist as a discipline until the last years of Turing's life.

More important, it is a contradiction in terms to want to compute more than is computable. Wegner, fails to distinguish "computable", corresponding to decidable decision problems where the characterising machine is guaranteed to terminate, from "semi-decidable" decision problems where, although the characterising machine is not guaranteed to terminate, nonetheless equivalent *heuristics* can still provide useful results, for example in unbounded search.

The "Turing tarpit" is from Alan Perlis[15]:

> "54. Beware of the Turing tarpit in which everything is possible but nothing of interest is easy."(p10)

This is a jocular characterisation of the limits of *very simple* computable systems, of which Turing machines are an excellent example, rather than of Turing computable systems in general.

Wegner, later says that:

> "Turing was born in 1912 and matured at about the time Gödel delivered his coup de grace to formalist mathematics. But the effects of Gödel's incompleteness result were slow to manifest themselves among such logicians as Church, Curry, and Turing who shaped the foundations of computer science. ... Before Gödel, the conventional wisdom of computer scientists assumed that proving correctness was possible (in principle) and simply needed greater effort and better theorem provers. However, incompleteness implies that proving correctness of interactive models is not merely difficult but impossible. This impossibility result parallels the impossibility of realising Russell and Hilbert's programs of reducing mathematics to logic. The goals of research on formal methods must be modified to acknowledge this impossibility."(p88)

It is a misrepresentation to say that Gödel's results were "slow to to manifest themselves among such logicians as Church, Curry, and Turing". Church and Turing, at the heart of international exploration of Hilbert's programme, were acutely aware of Gödel's work. Thus, Church cites Gödel in[2] and Turing[18] notes:

> ..."what I shall prove is quite different from the well known results of Gödel."

Furthermore, there were no "computer scientist" before, or indeed for many years after, Gödel's 1930 result, and the only "theorem provers" were human, much like the then common understanding of "computer".. Wegner's claims for the liberating power of interaction machines are by assertion rather than argument:

> "Abstraction is a key tool in simplifying systems by focusing on subsets of relevant attributes and ignoring irrelevant ones. Incompleteness is the key distinguishing mechanism between rationalist, algorithmic abstraction and empiricist, interactive abstraction. The comfortable completeness and predictability of algorithms is inherently inadequate in modelling interactive computing tasks and physical systems. The sacrifice of completeness is frightening to theorists who work with formal models like Turing machines, just as it was for Plato and Descartes." (p88)

Here, Wegner seems gratuitously patronising of theorists, but maybe that is his privilege. Turing clearly embraced Gödel's results. In[19] he further developed Gödel's "well known theorem"by associating increasingly complete systems of logic with constructive ordinals to explore whether classes of unsolvable problems for lower ordinal systems might be solvable in higher ordinal systems. Turing concludes that only an *oracle*, that is a device that somehow knows the right answer, would enable the full mechanisation of "some intellectually satisfying system of logical inference with some ordinal logic". There is no suggestion that some alleged inability to interact with a machine during a computation is the source of this fundamental limitation.

Wegner seems at odds with his own characterisation of computations and computers. In his 1971 book[21], he introduces the notion of an instantaneous description of a computation step, which he attributes to Turing. First of all, he discusses a simple machine with just a processing unit and a memory. He next presents a more sophisticated model of a digital computer, which includes input and output as uni-directional linear streams. Here, he explicitly adds the input and output records to the instantaneous description. This is equivalent to denoting some portion of a Turing machine as input and output areas, where the input plainly cannot be known in advance of the computation. It is wholly legitimate for people to change their minds: it is also customary to explain why the change has been made.

5 Conclusion

As foundational ideas mature and become widespread, so opportunities for memeish misconceptions increase. Some are probably harmless, like the alleged infinite length of Turing machine tapes: the equivalence of Turing machines with finite but unbounded tapes and Post machines with infinite tapes is long established.

Other misconceptions have wider implications. Thus, Wegner's claims that interaction transcends alleged limitations on Turing machines[23] form the basis of what we argue[3] are fruitless attempts to construct systems and machines which are not subject to classic decidability results.

If we return to Turing's original papers we are not struck by their limitations. Instead we see remarkable foresight as to the potential of the Universal Computer.There are, of course, limits in Turing: the fundamental limits to computability which he established. We are as bound by them as we are by the laws of thermodynamics.

References

1. A.G. Bromley, *Charles babbage's analytical engine, 1838*, Annals of the History of Computing **4** (1982), no. 3, 196–217.
2. A. Church, *An unsolvable problem of elementary number theory*, American Journal of Mathematics **58** (1936), 345–363.

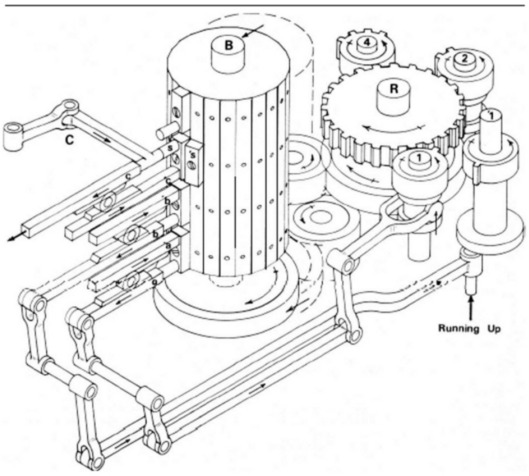

Fig. 2. The finite state automaton that controlled Babbage's Analytical Engine. A microprogram word is represented by a vertical row of studs screwed to the barrel. These act on the ends of the control levers when the barrel moves sideways. Figure and description from [1].

Author	Year Tape Size Comment	Page
Kleene	1952 (potentially) infinite	p357
Davis	1958 infinite in both directions	p3
Arbib	1964 potentially infinite	p148
Hermes	1965 implies infinite but with finite non-empty cells	p32
Hopcroft and Ullman	1969 infinity of cells	p81
Salomaa	1969 extending the tape	p216
Wegner	1971 unlimited memory	p6
Minsky	1972 infinite in both directions v think of as really finite	p118
Manna	1974 infinite to right	p21
Brainerd and Landweber	1974 two-way unbounded	p104
Hamlet	1974 finite...no limit	p62
Bird	1976 infinite	p65
Hopkin and Moss	1976 finite... not bounded	p68
Brady	1977 extra memory or tape stretches to infinity	p31
Hopcroft and Ullman	1979 infinite to right	p148
Harel	1992 infinite tape	p228
Gruska	1997 bi-infinite tape	p217
Jones	1997 two-way infinite	p115
Kozen	1997 infinitely many to right	p210
Cohen	1997 rest of tape initially filled with blanks	p435
Moret	1998 unbounded	p99
Savage	1998 potentially unlimited	p118

Fig. 3. Computability books and TM tape size.

3. P. Cockshott and G. Michaelson, *Are There New Models of Computation? Reply to Wegner and Eberbach*, The Computer Journal **50** (2007), no. 2, 232.
4. C. Darwin, *The descent of man and selection in relation to sex*, John Murray London, 1906.
5. M. Davis, *Computability and Unsolvability*, McGraw-Hill, 1958.
6. T. Dawson, *Range Keeper For Guns*, November 30 1909, US Patent 941,626.
7. M. d'Ocagne, *Le calcul simpifié par les procédés mécaniques et graphiques: Esquisse générale compernant: calcul mécanique, calcul graphique, cacul graphomécanique, calcul nomographique, calcul nomomécanique*, Gauthier-Villars, 1928.
8. J.S. Dumaresq, *A Rate of Change of Range and Deflection Finder to Facilitate the Accurate Shooting of Guns.*, (1905), GB patent GBD190417719 19040815.
9. D. Harel, *Algorithmics: The Spirit of Computing(2nd ed)*, Addison-Wesley, 1992.
10. J. E. Hopcroft and J. E. Ullman, *Formal Langauges and their Relation to Automata*, Addison-Wesley, 1969.
11. L. Jacob, *Le calcul mecanique*, Bibliothèque de mathématiques appliquées, Octave Doin, Paris, 1911.
12. D. Lardner, *Babbage's calculating engines*, Edinburgh Review **59** (1834), 263–327.
13. Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, 1974.
14. M.L. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, 1967.
15. A. Perlis, *Epigrams on Programming*, SIGPLAN Notices **17** (1982), no. 9, 7–13.
16. E. L. Post, *Finite Combinatory Processes. Formulation 1*, Journal of Symbolic Logic **1** (1936), 103–105.
17. W. Thomson, *Mechanical Integration of the Linear Differential Equations of the Second Order with Variable Coefficients*, Proceedings of the Royal Society of London (1875), 269–271.
18. A. Turing, *On Computable Numbers, With an Application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society **42** (1937), 230–65.
19. ———, *Systems of Logic Based on Ordinals*, Proc. London Mathematical Soc. (1939), 161–228.
20. ———, *Lecture on the Automatic Computing Engine, 1947* , The Essential Turing (B. J. Copeland, ed.), OUP, 2004.
21. P. Wegner, *Programming Languages, Machine Structures and Machine Organisation*, McGraw-Hill, 1971.
22. P. Wegner, *Why interaction is more powerful than algorithms*, Communications of the ACM **40** (1997), no. 5, 80–91.
23. P. Wegner and E. Eberbach, *New Models of Computation*, Computer Journal **47** (2004), 4–9.