A short history of small machines

Liesbeth De Mol[1⋆] and Maarten Bullynck[2]

[1] Center for Logic and Philosophy of Science, University of Ghent, Blandijnberg 2, 9000 Gent, Belgium
elizabeth.demol@ugent.be
[2] Université Paris VIII, Vincennes Saint-Denis
maarten.bullynck@kuttaka.org

"*This operation is so simple that it becomes laborious to apply*" (Lehmer, 1933)

One of the most famous results of Alan M. Turing is the so-called universal Turing machine (UTM). Its influence on (theoretical) computer science can hardly be overestimated. The operations of this machine are of a most elementary nature but nonetheless considered to capture all the (human) processes that can be carried out in computing a number. This kind of elementary machine fits into a tradition of '*logical minimalism*' that looks for simplest sets of operations or axioms. It is part of the more general research programme into the foundations of mathematics and logic that was carried out in the beginning of the 20th century. In the 1940s and 1950s, however, this tradition was redefined in the context of 'computer science' when computer engineers, logicians and mathematicians re-considered the problem of small(est) and/or simple(st) machines in the context of actual engineering practices. This paper looks into this early history of research on small symbolic and physical machines and tie it to this older tradition of logical minimalism. Focus will be on how the transition and translation of symbolic machines into real computers integrates minimalist philosophies as parts of more complex computer design strategies. This contextualizes Turing's machines at the turn from logic to machines.

1 Logical minimalism, ACE and Curry's compositions

1.1 A tradition of logical minimalism?

In the early 20th century, mathematical or symbolic logic flourished as part of research into the foundations of mathematics. This research followed the 'agenda' set out by Hilbert in his celebrated 1900 *Mathematical problems* lecture and was rooted in the work by people like Dedekind, Cantor, Peano, Boole etc. The search for simplicity, whether through the development of simple formal devices or the study of small and simple axiom sets, was part of this development. Indeed, a lot of the advances made in mathematical logic during that time can be characterized by (but surely not reduced to), what we will here call, logical minimalism. This kind of formal simplicity often served as a guiding methodology

⋆ Postdoctoral Fellow of the Fund for Scientific Research – Flanders (FWO)

to tackle foundational problems in mathematics. For some, it was a goal in itself to find the ultimate and simplest 'building blocks' of mathematics and, ultimately, human reasoning.

There are two obvious lines of research in mathematical logic that were informed by this minimalist philosophy. On the one hand, there were the several results aimed at finding the smallest set of logical primitives, with Sheffer's 1913 paper containing the description of the Sheffer stroke as one of the highlights. On the other hand, there are the attempts to reduce and/or simplify existing axiom systems, as was e.g. done by Nicod for the propositional calculus. In the 20s then, people like Post and Schönfinkel pushed this minimalism one step further.

Schönfinkel situates his work on combinators in the tradition of attempts to reduce and simplify axiom systems as well as to lower the number of undefined notions. His goal is no less than to eliminate more fundamental and undefined notions of logic, including the variable. His reason for doing so is not purely methodological but also philosophical [12, 358]:

> We are led to the idea [...] of attempting to eliminate by suitable reduction the remaining fundamental notions, those of proposition, propositional function, and variable. [T]o examine this possibility more closely [...]it would be valuable not only from the methodological point of view [...] but also from a certain philosophical, or, if you wish, aesthetic point of view. For a variable in a proposition of logic is, after all, nothing but a token that characterizes certain argument places and operators as belonging together; thus it has the status of a mere auxiliary notion that is really inappropriate to the constant, "eternal" essence of the propositions of logic. It seems to me remarkable [that this] can be done by a reduction to three fundamental signs.

It is exactly this more 'philosophical' idea of finding the simplest building blocks of logic and ultimately human reasoning that drove (part of) the work by Haskell B. Curry and Alan Turing, two logicians/mathematicians who had the opportunity to access and think about the new electronic computers of the 40s.

1.2 From combinators and Turing machines to computing techniques

Both Curry and Turing have done work that features logical minimalism. Indeed, Curry, who took up the work by Schönfinkel and developed it into what is now known as combinatory logic, explicitly states simplification as one of two major tendencies (the other is formalization) in investigations on the foundations of mathematics [4, p. 49]:

> On the other hand, [...] there is [the problem of] simplification; one can seek to find systems based upon processes of greater and greater primitiveness [...] In fact we are concerned with constructing systems of an extremely rudimentary character, which analyse processes ordinarily taken for granted.

Turing then had quite different motivations when he wrote his *On computable numbers* [13] in which he develops the Turing machine. His way of arriving at these devices was to start out from the process of a man in the process of computing a number and to try and reduce this process to its most elementary and simple 'operations' [13, 250]:

> Let us imagine the operations performed by the computer to be split up into "simple operations" which are so elementary that it is not easy to imagine them further divided.

It is not surprising that the minimalist philosophy also affects Curry's and Turing's work on *real* machines. Indeed, a formal apparatus which is simple and powerful enough to 'translate' ones problems to a *physical* machine is exactly what was required for the successful development of these machines.

Just after World War II Turing was recruited by Womersley of the NPL to help design the Automatic Computing Engine (ACE). As has been argued elsewhere [6,7], Turing definitely was inspired by and relied on the symbolic Turing machines developed in his *On computable numbers* for the design of the ACE. In fact, in a lecture to the London Mathematical Society, Turing explicitly states that computers such as the ACE '*are in fact practical versions of the universal machine*" [14]. Even though a good theoretical model, the UTM needed to be adapted. Thus, for instance, he makes clear that the one-dimensional tape as the memory of the Turing machine is not desirable in a real machine since it would take too long to look up information [7, 319].

As is argued in [6,7] the general philosophy behind the design of the ACE is minimalist in nature. Knowing that but a minimal set of symbols and operations is needed to have universal computation, Turing designed a machine with a hardware that is kept very simple and primitive, leaving the 'hard' work to the programmer, preferring to have less machine and more instructions. Indeed, as Hodges explains [7, 320]:

> His priorities were a large, fast memory, and then a hardware system that would be as *simple as possible*. His side was always that anything in the way of refinement or convenience for the user, could be performed by thought and not by machinery, by instructions and not by hardware. In his philosophy it was almost an extravagance to supply addition and multiplication facilities as hardware, since in principle they could be replaced by instructions applying only the most primitive logical operations of OR, AND and NOT.

Or, to put it in Turing's words, "*[W]e have often simplified the circuit at the expense of the code*" [14]. Martin Davis identifies this attitude as being in line with the RISC (reduced instruction set computing) architecture [6, 189].

A similar minimalist philosophy was pursued by H. B. Curry after his experience with the ENIAC, the first electronic and programmable US computer. In this context Curry was confronted with the need to develop a theory of program composition. Contrary to Turing, Curry did not design a computer like the ACE

but, instead, 'designed' a theory of programming inspired by, on the one hand, his work on combinatorial logic and the minimalist philosophy which guides it, and, on the other, his concrete experience with the ENIAC.

Curry's theory of programming was based on his knowledge of the IAS machine, the computer built on the basis of von Neumann's EDVAC report. One key aspect of this theory is the analysis of programs into basic programs and the development of a theory which allows to compose more complicated programs from these basic programs in an automatable fashion. This analysis into basic programs and their composition explicitly displays a minimalist philosophy [5]:

> [The] analysis can, in principle at least, be carried clear down until the ultimate constituents are the simplest possible programs. [...] Of course, it is a platitude that the practical man would not be interested in composition techniques for programs of such simplicity, but it is a common experience in mathematics that one can deepen ones insight into the most profound and abstract theories by considering trivially simple examples

Curry went on to give a method which reduces a specific class of 26 basic programs from his original list to only 4 basic programs. This is a good example of what kind of results Curry's minimalism led to. This reduction to 4 basic programs is proven by providing a (programmable) method which resynthesises the original 26 basic programs from these 4. This leads Curry to comment that one might save machine memory when compiling programs. He therefore makes the following hardware recommendation [5, 38–39]:

> Now the possibility of making such [arithmetic] programs without using auxiliary memory is a great advantage to the programmer. Therefore, it is recommended that, if it is not practical to design the machine so as to allow these additional orders [the 26 original basic orders], then a position in the memory should be permanently set aside for making the reductions contemplated.

Hence, a theoretical result so in line with logical minimalism, becomes an automatable method which allows to save computer memory (for more details on Curry's theory of programming see [9]).

2 Less is more in the Fifties

2.1 'Automata studies'

Through Curry's and Turing's more practical work, logical minimalism had a direct and immediate influence on the development of the early digital and programmable machines. However, this is not where this influence stops. In the 50s, several researchers coming from different backgrounds, but with the same keen interest in the theory *and* practice of the new computing machines, become familiarized with the results of the computability related work by Church, Curry, Kleene, Post, Turing etc. They regarded the Turing machine and related

concepts as useful theoretical tools and models to think about actual, physical machines. In this context, a tradition of logical minimalism is 'transmuted' to the context of machines when people like Minsky, Moore, Shannon, Wang, Watanabe etc start developing and studying small and/or simple theoretical devices with an eye on real computers. Much of this research was done under the heading 'automata theory', a domain that prefigured in some way the establishment of (theoretical) computer science proper.

Hao Wang, who was trained as a logician and worked for some time at Bell labs and the Burroughs company, developed a variant of the Turing machine model which is simpler and smaller and laid the basis for the register machine model. He explicitly places his approach in the tradition of logical work on reducing the number of logical operators, mentioning for instance the Sheffer stroke, but with a different motivation, viz. to bridge the gap between research in logic and digital computers [17, 63]:

> The principal purpose of this paper is to offer a theory which is closely related to Turing's but is more economical in the basic operations. [...] Turing's theory of computable functions antedated but has not much influenced the extensive actual construction of digital computers. These two aspects of theory and practice have been developed almost entirely independently of each other. [...] One is often inclined whether a rapprochement might not produce some good effect. This paper will [...] be of use to those who wish to compare and connect the two approaches.

Not only logicians like Wang saw the possible practical relevance of research on small and/or simple models of computability. In the 50s a 'rapprochement' from the side of the engineers also took place. The influential volume *Automata Studies* (1956), edited by McCarthy and Shannon, is a perfect example of this 'rapprochement', featuring contributions from logicians, engineers and mathematicians. The volume contains the paper by Shannon which proves that two-state Turing machines and two-symbol Turing machines are capable of universal computation and that there is no one-state UTM. This has become one of the classic references for research on smallest (universal) devices.

Another contributor to *Automata Studies* was E. F. Moore. He had embarked on the systematic study of minimal circuits for given functions at Bell Labs in the 1950s. Most probably during this research, Moore came up with a new, simplified construction of a UTM described in [10]. In this paper he gives the details of a rather small 3-tape UTM with two symbols and 15 states as a simplification of the original one-tape UTM [10, 51]: "*the method of storing all of this information on one tape is rather complicated, the internal structure of the universal Turing machine [...] is also rather complicated*".

Moore starts out from Davis's quadruple notation for Turing machines, where the quadruple $q_i S_j I q_l$ means: When in state q_i the symbol S_j is scanned then do operation I (left, right or print S_k then go to state q_l. Since Moore is using three tapes instead of one, he transforms this notation to a sextuple notation $q_i S_1 S_2 S_3 I_n q_l$ where S_1, S_2, S_3 are the symbols scanned on tapes 1 to 3 respectively and I_n is operation I to be performed on tape n. On tape 1 the description

of the Turing machine to be simulated is stored (as a circular loop), tape 2 is an infinite blank tape that will contain the active determinant of the machine to be imitated, and finally tape 3 will be a copy of the infinite tape that would be on the machine being imitated. To put it in more ordinary computer speak: Tape 1 is the program, tape 2 is the active register and tape 3 the output.

Moore devotes a complete section to the physical realizability of his model and explains how magnetic tape memory is more suited in this context than punched tape [10, 54]:[3]

> since holes in punched tape cannot be erased once they are punched, in order to make a machine using punched tape capable of imitating the behavior of an ordinary Turing machine which has this erasing property the coding of the description of the machine would have to be in a more complicated fashion [...] It should be mentioned [...] that the properties of the tapes assumed in Turing machines are very much like the properties attained by magnetic tapes, which have erasability, reversibility, and the ability to use the same reading head for either reading or writing. If a tape mechanism were available which had the properties assumed and which could be connected directly to relay circuits, it would be possible to build a working model of this machine using perhaps twenty or twenty-five relays. These figures are mentioned since the number of relays is frequently used as a measure of the complexity of a logical machine

For Moore the significance of his result lies in the fact that it suggests "*that very complicated logical processes can be done using a fairly small number of mechanical or electrical components, provided large amounts of memory are available.*" [10, p. 54] One of the biggest obstacles for the physical realization of the model is an economical and technological one, viz. the cost of memory and speed. Moore concludes that at that time it was not "*economically feasible to use a machine to perform complicated operations because of the extreme slowness and fairly large amount of memory required.*" [10, p. 54] but nonetheless sees the value of his result in the fact that it "*suggests that it may be possible to reduce the number of components required for logical control purposes, particularly if any cheap memory devices are developed.*"

2.2 Simple digital computers: Zebra, Minima and TX-0

At around the same time Moore was thinking on small UTM's and their practical feasibility, several engineers started to effectively implement similar ideas by building "small" computers, viz. computers with a small instruction set. Although Turing's universal machine is often mentioned in passing, its details seem to have had little direct influence on actual research. Wilkes's idea of microprogramming and the ACE design seem to have played a more important role. Some groups of engineers involved in the development of small computers were well

[3] Perhaps Wang's non-erasing model described in [17], was inspired by the need for a simple model for a punched-tape computer.

versed in modern mathematical logic and its possible applications to computer design. However, it turns out that minimalist philosophies have to be redefined as parts integrating more complex computer design strategies.

In Europe the Dutch engineer W.L. van der Poel pioneered investigations into the structure of simple digital computers. In 1952 he built the ZERO with only 7 instructions. The machine was "*not meant as a practical computer, but only serves the purpose of gaining experience*" [15, 368] The zero was the precursor for the Zebra computer (1957), the 'Zeer eenvoudige binaire rekenautomaat' ('very simple binary computer'). Both the Zero and the Zebra can be called practical versions of minimalist philosophies. As van der Poel remarked [16, 361]:

> The main idea of the [Zebra] machine is to economise as far as possible on the number of components by simplifying the logical structure. For example, multiplication and division are not built in but must be programmed.

Van der Poel clearly knows that he needs the (possibility of a) conditional jump to have universal computation. The reference to Turing's [13] in the description of the ZERO shows that the concept of a UTM belongs to van der Poel's general knowledge.

Van der Poel's motivation for designing a simple machine seems mainly theoretical: he considers the logical simplicity of the machine as an advantage in itself [15, 376]. Of course, there are several trade-offs of which van der Poel is very aware. First, there is a certain loss in speed, and this applies in particular for the Zero [15, 367]:

> In this article will be described the logical principles of an electronic digital computer which has been simplified to the utmost practical limit at the sacrifice of speed.

Secondly, more (and, if possible, faster) memory is required [15, 376]:

> Complexity of the circuits has been exchanged for capacity of memory. [...] Although there would have been room for two instruction in one register, no simple arrangement can be made for that purpose without upsetting the logical structure of the machine.

The memory is needed for flexible programming that becomes capital in using the machine [15, 375-376]:

> [the ZERO] has a flexible system of programming that can easily be learned. Straight-forward programming is simple, because there are only seven instructions. [...] the user can make the programs just as he wants them, use of the machine is very flexible.

Because usability would suffer, van der Poel does not push the minimalist philosophy to its limits: "*Of the seven instructions that are possible only three are strictly necessary [...] Of course, many more instructions, even for quite simple programs, are then required.*"

Van der Poel's work found resonance in Germany and the U.K. where actual simple computers were constructed. The Standard Telephones and Cables Limited contracted van der Poel to construct the Zebra (1957). In Germany, H. Poesch and Th. Fromme, inspired by van der Poel's work, proposed a simple machine, the MINIMA, as a pedagogic tool [11, 307, our translation]:

> For the study of programming and for education in questions of programming we have thought of a model of a machine that has a very simple instruction code so that one can learn it fast, but also can do anything that occurs as complex operations in a modern computer

As a practical machine, however, Poesch and Fromme recognize that "*This machine could execute all instructions (also conditionals), but if one would realize it, the operation time would be too long.*" Konrad Zuse's commercialized Z22 (1955) was then an "*outgrowth*" of the Minima, which was designed after "*discussions on the problem of the smallest meaningful program-controlled computer*" during which also the UTM was discussed but considered "*useless*" "*practically speaking*" [18, 135].

In the United States as well, at the Lincoln Laboratory, a small computer was developed, called the TX-0 (1956-1958). The TX-0 would be the first of an impressive line of computers that pioneered the use of transistors and the idea of personal computing, leading up to the PDP minicomputers. As Wesley A. Clark, the main engineer on the project, states [1]:

> "Well, all right, let's build the smallest thing we can think of," and designed the TX-0, which was very primitively structured and quite small, quite simple - small for the day. Not physically small - it took lots of space; it still took a room.

The TX-0 was designed as an experimental machine, testing both transistors and elaborate input/output facilities, that would eventually lead up to the monumental TX-2. Interestingly, the design of the TX-0 was done by a group of engineers who had been immersed in mathematical logic and computers. From October 1955 to January 1956 the engineers at Lincoln Laboratory had followed an intensive course on "the logical structure of digital computers", organized by Wes Clark. The course was part of discussions *"about the various possible minimal machines"* that could be designed [3, 144]. Clark's course contained two parts: a part on Turing machines and a part on Boolean algebra and its use in circuit design. While the part on Boolean algebra features a session on how to use the Sheffer stroke as the sole building block for circuits, the part on Turing machines details the construction of Moore's 1952 small universal machine. As Clark remarks, this universal machine constitutes a "*critical complexity beyond which no further increase in generality can be guaranteed!*" [2, p. 13] Clark follows Moore's encoding and construction, but mashes the three tapes back into one tape using a specific encoding scheme. This scheme later returns as the read-in procedure for the lines on the program tape on the TX-0.

How much of the logical minimalism taught in Clark's course found its way into the TX-0's design is hard to evaluate, but it certainly had an impact. In

the description of the TX-0 circuitry, Shannon's work is explicitly referenced, but also the logical organisation of the machine seems to have inherited from the course. A small instruction set (only four) was used in combination with an elaborate vocabulary of 'class commands' (special bit-encoded instructions triggered by the fourth `opr` command) and with a versatile symbolic assembler language. Again, as was the case with the Zero, the Zebra and the Minima the logical simplicity of the computer itself entails a necessary, richly developed programming.

3 Discussion

Nowadays the result that, for instance, four elementary instructions suffice to compute anything computable (provided one accepts the CTT) has become near trivial in computer science. One would almost forget that a complex history underlies such results.

The Turing machine has often been interpreted as a most practical outcome of the foundational debates of the early 20th century, the positive face of the negative Entscheidungs-result. It fits in a tradition of logical minimalism: the search for a minimum of operations, of axioms, of length of propositions etc. This research proved to be most useful in the early days of computing where the results from mathematical logic were transmuted into ideas on computer design and programming, a transformation pioneered by people like Curry and Turing.

The 1950s pursued the development of ideas from logical minimalism to a more practical, engineering context. Moore's and Wang's work develop theoretical models that fit more closely a real computer. Actual small machines, such as the Zero, the Minima or the TX-0, were all conceived as experimental machines to test ideas and design philosophies. Direct translations of logical minimalism such as a small UTM cannot be traced down in these experimental small computers, but the UTM certainly functions for their engineers as a warrant that small computers are possible and that they can execute whatever program.

The minimalist philosophy in computer architecture necessarily has to adapt itself to the realities of the time, money and hardware available. Instead of infinite tape, lots of time etc. that abound in theoretical research, the computer engineers have to find a compromise between different tradeoffs. A simple logical structure of the computer asks for extensive programming possibilities and hence more (and faster) memory. Since in general more instructions were needed for a program, a loss in speed could be expected, though, as witnessed by van der Poel's coding for the Zebra or the TX-0's class commands, devices were developed to remedy this situation. In general, the experimental small machines led to the development of new programming techniques. Just as Turing's work on the ACE preempts aspects of RISC architecture, the small experimental machines gave rise to elaborate schemes of microprogramming and, in the case of the TX-0, to a symbolic on-line assembly language.

In the last 20 years, research on small universal computational devices has known a renaissance (see [8] for an overview). Again, this research has an eye

an practical realizations, for instance in the form of DNA-computing. Also here, theoretical research on small universal devices is expected to help in the design and study of these new kinds of 'machines' and to help explore the boundaries of what can and cannot be done with such machines.

References

1. Wesley A. Clark. Oral history interview by Judy E. O'Neill, 3 May 1990. Charles Babbage Institute, University of Minnesota, Minneapolis.
2. Wesley A. Clark. The logical structure of digital computers. Technical report, Course notes, Division 6 Lincoln Laboratory, MIT, 1955.
3. Wesley A. Clark. The Lincoln TX-2 computer development. *Proceedings WJCC*, pages 43–145, 1957.
4. Haskell B. Curry. The combinatory foundations of mathematical logic. *The Journal of Symbolic Logic*, 7(2):49–64, 1942.
5. Haskell B. Curry. A program composition technique as applied to inverse interpolation. Technical Report 10337, Naval Ordnance Laboratory, 1950.
6. Martin Davis. *Engines of Logic: Mathematicians and the Origin of the Computer*. W.W. Norton and Company, New York, 2001.
7. Andrew Hodges. *Alan M. Turing. The enigma*. Burnett Books, London, 1983. Republication (1992), 2nd edition, Vintage, London.
8. Maurice Margenstern. Frontier between decidability and undecidability: A survey. *Theoretical Computer Science*, 231(2):217–251, 2000.
9. L. De Mol, M. Bullynck, and M. Carlé. Haskell before Haskell. Curry's contribution to a theory of programming. In *Programs, Proofs, Processes, Computability in Europe 2010*, volume 6158 of *LNCS*, pages 108–117. Springer, 2010.
10. E.F. Moore. A simplified universal Turing machine. In *Proceedings of the meeting of the ACM, Toronto Sept. 8 1952*, pages 50–54, 1952.
11. H. Pösch and Th. Fromme. Programmorganisation bei kleinen Rechenautomaten mit innerem Programm. *Zeitschrift für angewandte Mathematik und Mechanik*, 34:307–308, 1954.
12. Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. Republished and translated in J. van Heijenoort, From Frege to Gödel: A source book in Mathematical Logic 1879–1931, 1967, 357–366.
13. Alan M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936–37. A correction to the paper was published in the same journal, vol. 43, 1937, 544–546.
14. Alan M. Turing. Lecture to the London Mathematical Society on 20 february 1947. 1947. in: Brian E. Carpenter and Robert W. Doran (eds.), A.M. Turing's ACE Report of 1946 and Other papers, MIT Press, 1986, 106–124.
15. Willem L. van der Poel. A simple electronic digital computer'. *Applied Scientific Research Section B*, 2:367–400, 1952.
16. Willem L. van der Poel. Zebra, a simple binary computer. In *Proc. ICIP, UNESCO*, pages 361–365, 1959.
17. Hao Wang. A variant to Turing's theory of computing machines. *Journal of the ACM*, 4(1):63–92, 1957.
18. Konrad Zuse. *The Computer – My Life*. Springer Verlag, Berlin, 1993.