A short history of small machines

Liesbeth de Mol, Maarten Bullynck, Edgar G. Daylight

A short history of small machines

Liesbeth De Mol[1], Maarten Bullynck[2], and Edgar G. Daylight[3]

[1] CNRS – UMR8163 STL, Université de Lille 3, France
liesbeth.demol@univ-lille3.fr
[2] Université Paris VIII, Vincennes Saint-Denis
maarten.bullynck@kuttaka.org
[3] Lonely Scholar, Geel, Belgium
egdaylight@dijkstrascry.com

This is a work in progress. Please do not quote or cite without permission from the authors.

An earlier version of this paper was written in 2012 during the Turing year under the condition that we had to stay below 12 pages. The current version is a slightly rewritten and extended version of that unpublished paper. Most of the sections will be extended significantly.

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

In recent years, there has been a renewed interest in the interactions between logic and engineering practices and how they helped to shape modern computing. The focus of these writings, however, is mostly on the work of Alan M. Turing and John von Neumann and the question if and how their logical and mathematical works affected the shaping of the modern computer and its coding practices. Even though this has resulted in some very interesting work, one runs the danger of not seeing the wider picture and so, rather than going against celebrations of said heroes one is actually reaffirming this (though with nuance).

The present paper, by contrast, shows how a broader dispositive which has a clear tradition within (mathematical) logic and which we will here call *logical minismalism* was recast into a diversity of computing practices in the 40s and 50s. In our approach, rather than looking at interactions between logic, mathematics and computing practices from the institutional or social perspective, we study how techniques from different practices work together at the level of the computing practices themselves. Hence, rathe then focusing on humans, we focus on code and machines. Such perspective allows us to render understandable the shaping of computing from the bottom-up and contributes to a pluralistic history of computing. For a more detailed motivation of our approach, see [13,14].

Logical minimalism is part of a more general research programme into the foundations of mathematics and logic that was carried out at 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 ties it to this older tradition of logical minimalism. Focus will be on how the transition of symbolic machines into real computers integrates minimalist philosophies as parts of more complex computer design strategies.

1 A tradition of logical minimalism in logic

In the early 20th century, mathematical or symbolic logic flourished as part of research into the foundations of mathematics. Research in algebraic logic and on the axioms for arithmetic and analysis had led to important work by people like Boole, Schröder, Dedekind, Cantor, Frege, Hilbert, Peano, Russell and Whitehead 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 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. Post's work from the early 20s can be characterized by a method of generalization through simplification with a focus on the "*outward forms of symbolic expressions, and possible operations thereon, rather than [on] logical concepts*" which ultimately resulted in an anticipation of parts of Gödel's, Church's and Turing's work in the 1930s [15]. This method which ultimately tries to eliminate all meaningful logical concepts such as variable, quantor etc, resulted in what Post himself called a "*more primitive form of mathematics*" known as tag systems and which is one of the simplest formal devices to be computationally equivalent to Turing machines.

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 [17, 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 M. Turing, two logicians/mathematicians who had the opportunity to access and think about the new electronic computers of the 40s.

2 Logical minimalism in switching theory

Independently of this mathematical research tradition, the problem of economy in developing electromechanical devices led engineers to consider algebraic and logical methods as aids in the design of their circuits. Claude E. Shannon's master's thesis "A symbolical analysis of relay switching circuits" (1938) [**?**] is the most famous example. In his thesis, Shannon showed how the equations for designing relay circuits could be symbolically rewritten using Boolean algebra, making the manipulation of the equations amenable to a simple calculus. In fact, the scope of the thesis was broader and wanted to address the general problem of network synthesis:

> Given certain characteristics, it is required to find a circuit incorporating these characteristics. The solution of this type of problem is not unique and methods of finding those particular circuits requiring the least number of relay contacts and switch blades will be studied.

Boolean algebra is actually but one of a number of mathematical (and graphical) techniques Shannon proposes in his thesis to attack the problem of designing specific circuits with a minimum number of elements.[4] Shannon's techniques were further explored and developed at Bell Labs in the 1940ies,in particular for some complex relay circuits needed in the No. 5 Crossbar for telephone switching or the Relay Calculators conceived by G.R. Stibitz. Karnaugh, McCluskey, Mealy, Moore and others, all came up with further techniques to find minimal circuits, E.F. Moore even exhaustively tabulated the most economical relay circuit for each Boolean function upto four variables in 1952 [**?**, 56-58]. Together with C.E. Shannon, Moore also patented a circuit analyzer developed in 1952-53:

> This machine (called the relay circuit analyzer) has as inputs both a relay contact circuit and the specifications the circuit is expected to satisfy. The analyzer (1) verifies whether the circuit satisfies the specifications, (2) makes systematic attempts to simplify the circuit by removing redundant contacts, and also (3) obtains mathematically rigorous lower

[4] It should be noted that many other researchers, mostly in Japan, Germany and Russia, came up with similar ideas and techniques around the same time.

> bounds for the numbers and types of contacts needed to satisfy the specifications.

This kind of research was bottom-up, from the elements to complex circuits. While it worked well for certain, not too complex, circuits with particular elements such as relays, there were many other aggregate components needed in complex machines that could not be synthesized using the known methods. This was because either the elements were not easily translatable into symbols (e.g. wave filters), or because the purpose and functions of the machines were too complex to be formulated as easy boundary conditions on the number of combinations. Therefore, some engaged in the study of the inverse problem, top-down, starting from a complex machine and trying to analyze it down to its elements.

In the early fifties, Moore and Shannon embarked on a study of simplifying universal Turing machines, with an eye on a possible application in the design of complex calculators. During this time, Shannon obtained his famous result that two symbols suffice for a universal machine (published in 1956 [18]). In 1952 Moore presented his 'simplified universal Turing machine' at the ACM conference, the paper appeared two years later. Moore described a 15-state two-symbol three-tape universal Turing machine. The significance of his result was 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." [**?**, 54] But at the same time, Moore remarks that it is "not economically feasible to use [such] a machine to perform complicated operations because of the extreme slowness and fairly large amount of memory required", though 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."

3 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 [5, 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* [19] 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' [19, 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 [7,10], 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*" [20]. 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 [10, 319].

As is argued in [7,10] 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 [10, 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*" [20]. Martin Davis identifies this attitude as being in line with the RISC (reduced instruction set computing) architecture [7, 189].

A similar minimalist philosophy was pursued by 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. At that time, von Neumann and Goldstine had already written two reports on the planning and coding of the EDVAC [9] which motivated an independent attack on the problem of coding and planning because he found the von Neumann and Goldstine approach too elaborate (see [12] for more details) and missing a proper theory of program composition which is general enough to be applicable for all kinds of programs. 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 [6]:

> [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 [6, 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 [12]).

4 Less is more in the fifties: '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 [29, 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.

The model by Wang laid the basis for the development of an even simpler model of computability which was created as a device more similar to real computers and is known as the register machine. Minsky is one of the researchers who independently developed this model and proved minimal results for it. Indeed, it was Minsky who proved that two registers suffice to have universality. It was also Minsky who wrote several of the other classic papers on small universal devices. He proved for instance, together with Cocke, that there is a universal Turing machine with four symbols and seven states, using the universality of Post's tag systems mentioned in Sec. 1.

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 [16]. 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 [16, 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 [16, 54]:[5]

> 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.* " [16, 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.*" [16, 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.*"

[5] Perhaps Wang's non-erasing model described in [29], was inspired by the need for a simple model for a punched-tape computer.

5 Less is more in the fifties: Simple digital computers

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 which are designed based on an idea of simplicity and economy of instructions. 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 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.

Our main focus will be on two so-called experimental machines, viz. machines that are not really intended to be commercialized but rather as a test battery of ideas. As we will see, it is exactly in such contexts that ideas of logical minimalism are put to the test. We will consider two such machines: van der Poel's ZERO machine, built as an experimental machine in the Netherlands and the TX-0, also an experimental machine developed at MIT's Lincoln lab, that would eventually influence the design of the PDP-1 in the beginning of the 1960s.

5.1 From ZERO to ZEBRA

In Europe the Dutch engineer Willem L. van der Poel pioneered investigations into the structure of simple digital computers. Around 1947 Nicolaas G. de Bruijn (1918–2012), just appointed professor, submitted a project proposal to the Hogeschoolfonds of Delft. His objective was to obtain funding for the construction of an "all-round" calculating machine, a "self-thinking" device.

The goal of de Bruijn's project was to build a calculating machine that would specifically carry out computations in optics. An important part of the research was the selection of adequate materials for the machine which could be mechanical, pneumatic, electrical, or electronic.

Leen Kosten, head of the Mathematics Department of the Central Laboratory of the PTT in The Hague, ensured that de Bruijn's project in Delft would receive multiple relays "for loan." This event naturally led to the construction of an electrical machine: the ARCO. Based on a selector switch, the ARCO could be configured to use one of several "recipes" (i.e., one of several hard-wired programs).

The actual project work, details of which lie outside the scope of the present article, was assigned to the technical physics student van der Poel whom had a clear and realistic picture of the limitations of calculating machinery. In 1947, he wrote that:

> Tasks such as indefinite integration and reduction of an equation to its most simple form can, of course, not be carried out by the [ARCO] machine for they rely on subjective criteria. We have yet to reach the stage in which machines will think for us. [22, p.1, my paraphrased translation]

Immediately after graduation in 1950, van der Poel joined Kosten's team, leaving other physics students to finalize the ARCO's construction, a machine that was later coined "the TESTUDO" (turtle) due to its extremely slow execution speed. Van der Poel's cautious attitude in 1947 with regard to computing machinery was more than an example of Dutch austerity, it was also a result of his thorough knowledge of the computing literature. Besides referring to published writings in connection with electronic circuits and the construction of calculating machines, van der Poel pointed to the recent reports of Burks, Goldstine, and von Neumann. Even the mathematical logic of Hilbert and Ackermann, and Turing's 1936 article 'On computable numbers,' belonged to his knowledge base. Van der Poel had characterized the most essential problems for the Delft project as "programming problems" [22, my translation].

> Solving a problem by means of programming amounts to choosing an appropriate set of instructions. Of course, the challenge is to accomplish as much as possible with as few instructions as is feasible. [22, p.60, my translation]

The electrotechnical engineer Kosten, in turn, had obtained his Ph.D. during World War II on the design of analog machines with the purpose of simulating congestion problems in connection with telephone traffic. With van der Poel now on his side, the choice was quickly made to build a radically new, digital, calculating machine, the PTERA,[6] which differed greatly from the TESTUDO.[7] Many years later, van der Poel described the transition from the TESTUDO to the PTERA as a transformation from "pre-von-Neumann" machines to "post-von-Neumann" computers [27, p.8, my translation]. According to van der Poel's recollections, it was the writings of von Neumann et al. which had made him realize that a program could be stored *inside* the machine. During his 1988 retirement speech, van der Poel recalled how he had obtained von Neumann's report from the Mathematical Centre of Amsterdam [27, p.8].

In 1952 then van der Poel's knowledge of this literature became applied in what he later called his "most beautiful machine ever": the ZERO machine which had only 7 instructions. The ZERO was really an experimental machine which "*not meant as a practical computer, but only serves the purpose of gaining experience*" [23, 368]. The idea was to build the simplest possible computer taking into account at least some practical limitations [23, 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.

The ZERO's beauty indeed exemplified frugality and logical minimalism as the following examples show. Van der Poel used the same register to serve both

[6] PTERA: "PTT Elektronische Reken Automaat" (PTT Electronic Calculating Automaton)

[7] For a more detailed account of Kosten, see Kranakis [11, p.70–72] who says that Kosten had already, before van der Poel's arrival, shifted the attention of his research group to developing a general-purpose computer.

as accumulator and control register. He avoided expensive multiplication and division components in hardware by programming them in terms of addition. He implemented the addition of two numbers in one and the same electronic component by means of bit-wise addition sequentialized in time. (These last two design choices led to slow computers.) Finally, van der Poel resorted to four "functionally independent bits" [24]. One bit b_1 expressed whether the machine's instruction had to read something from ($b_1 = 0$) or write something to ($b_1 = 1$) the drum. Another bit b_2 independently expressed whether the accumulator had to be cleared ($b_2 = 0$) or not ($b_2 = 1$). The two bits together ($b_1 b_2$) then defined four possible combinations: 00, 01, 10, and 11. Because the value of the first bit did not depend on that of the second and vice versa, no control box was required and, hence, less equipment was needed, which resulted in small and cheap computing machinery. Van der Poel, led by esthetics, was more than willing to pay the price that "not all combinations were practically useful"; indeed, only four of the seven instructions in the `ZERO` made practical sense [24][28, p.31]. The `ZERO` only existed for a couple of months and was quickly dismantled in favor of the `PTERA`.

However, 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.*" Later, in his PhD [25] he would push the idea to its limits by describing a one-instruction machine, the so-called "purely one-operation machine".[8] His inspiration for this theoretical machine came from the Sheffer stroke which is indeed the one-instruction version of propositional logic.

Given his combined experience with `PTERA` and `ZERO`, van der Poel started on another computer known as the `ZEBRA`, the "Zeer Eenvoudig Binair Rekenapparaat" (Very Simple Binary Calculating machine) which he described at length in his 1956 Ph.D. dissertation [25], It was inspired by `ZERO` and built to resolve some practical problems, mostly related to speed, of the `PTERA`. Just as `ZERO`, it made extensive use of the functional bits, though there were now 15 rather than 4 bits. Moreover, it strove for "complete duality," between the fetching of an instruction (which allowed jumps) and the execution of some operation[9] realizing full well that "this is seldom of practical importance" [25, p.18-19].

Neither the `PTT`, nor Philips, showed interest in building the `ZEBRA`. The firm `ZUSE` did but not for long. It was the English company `STANTEC` which eventually manufactured the `ZEBRA` from 1957 onwards, delivering dozens of `ZEBRA`s throughout Europe [11, p.74]. The lively correspondence between the users of those machines led to the formation of the `ZEBRA` club, with van der Poel at its center. Van der Poel's style of computer design and his complementary approach to computer programming (described below), along with the `ZEBRA` user club, made him an influential computer pioneer in the 1950s and 1960s [21].

[8] Needs to be studied still in much more detail....

[9] This was achived in the ZERO by setting the X-bit to 0 or 1; in the ZEBRA this technique basically remained unchanged.

Van der Poel's designs of "simple" and "very simple" machines, as reflected in the names of his machines, were surely motivated by some practical considerations including price, flexibility and reliability:

> [The ZERO has a flexible system of programming that can easily be learnt.[...] There are only a few parts [...] This makes fault location very easy. [...] As the memory constitutes the bulk of the machine, the price is mainly determined by its parts

However, despite these advantages it is clear that he is also led by a strong theoretical sense of beauty which one often would not expect from the " typical" engineer: simple instructions, independent bits, "Of course, the challenge is to accomplish as much as possible with as few instructions as is feasible" [22, p.60, my translation] and ideas of duality between usually differentiated parts of the machine and of the instruction.

The construction of a machine which is considered to be the practical approximation of the ideal of a one-instruction computer however also had one major drawback: speed. Combined with the very lengthy programs required because of the limited number of functional bits, it was not practical enough. It is for that reason that van der Poel developed intricate programming tricks, exploiting his thorough knowledge *while* programming. Specifically, he perfectionized two already existing techniques, optimum coding and underwater programming, and thereby demonstrated a strong similarity with Turing's own machine-specific programming habits (see e.g. [1]).

Optimum coding essentially meant accessing the drum economically;e.g., by interleaving instructions and data on the drum in conformance with the way the program would behave. The drum was, after all, the slowest part of the computer. The common practice of independently storing instructions and data on the drum, resulted in several drum rotations (during program execution). To reduce the number of drum rotations, van der Poel opted for a less orderly solution by interleaving the instructions and the data on the drum in conformance with the order in which they would be called by the processor [28, p.26]. Underwater programming amounted to minimizing the drum accesses; e.g., by copying an instruction I from the drum to the registers and subsequently modifying the contents of the registers in order to transform I into the next instruction I', and I' into I'', and so forth. Until the drum was accessed a second time, the program was executing "under water," using van der Poel's terminology [8]. The reduced number of accesses to the drum allowed the program to maintain a high execution speed. It was not easy to circumvent the drum by exclusively resorting to the registers. To be successful in this regard, the underwater programmer had to have a thorough understanding of the machine.

Support for optimum coding and underwater programming made the ZEBRA an economically competitive machine, as the firm STANTEC conveyed in 1958:

> Due mainly to the entirely new programming concept and the simple logical design upon which it is based, the cost of the STANTEC-ZEBRA is far less than might be expected for a machine of its capabilities.

Without this machine-specific programming technology, van der Poel's aesthetically pleasing machines would have remained extremely slow and, as a result, economically unattractive. The price van der Poel paid for his notion of beauty was programs that, presumably, were incomprehensive to almost everyone except the programmer himself.[10]

5.2 The TX-0 at Lincoln Lab

In the United States as well, at the Lincoln Laboratory, a small computer was developed, called the TX-0 (1956-1958). The TX-0 pioneered the use of transistors and the idea of personal computing, and would eventually influence the development of DEC's PDP-line of minicomputers. As Wesley A. Clark, the main engineer on the project, states [2]:

> "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, as a test-case for 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 [4, 144]. Clark's course contained six parts, each part building on the previous one:

- The Turing machine: a basic introduction to the Turing machine concept
- The universal Turing machine
- Boolean Algebra: Interestingly, this is in fact considered as the lower-level description of the Turing machine "The symbol-printing operations in a Turing machine can be described in terms of the tape cells themselves. For example, a machine which performs the sequence "If cell A holds "1" or if cells B holds "0", print "1" on cell C is described by the stamement: $A_1 or B_0 : C_1$"' The manipulative aspect of this notation can be exploited in demonstrating that therules for printing symbols define a Boolean algebra"

[10] In 1961, van der Poel gave his approach to programming a name: "trickology" [21, p.137]. In his words:

> The concept of micro-programming and the practice of devising tricks to do the more complicated composite action is so interwoven in ZEBRA that the volume of knowledge of these tricks has been given a special name: *trickology*. Without this knowledge of trickology and the standard programs based on it, ZEBRA would be a useless machine. [26, p.274, original emphasis]

– Boolean algebra (continued)
– Synthesis of Boolean machines
– Synthesis of Boolean machines continued

While the Sheffer stroke us mentioned in the discussion of Boolean algebra that it could be used as the sole building block for designing 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!*" [3, 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 still needs to be evaluated but we do believe there are some indications that it did play an important role. 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 and the Zebra, the logical simplicity of the computer itself entails a necessary, richly developed programming.

6 Discussion

Nowadays the result that a few basic operations suffice to compute anything computable (provided one accepts the Church-Turing thesis which basically states that all computable problems can be computed by a series of formalisms which are all logically equivalent to the λ-calculus) 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 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 general spirit of logical minimalism 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 trade-offs. 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 of programming proper.

Computer science as a practice was, from the beginning, characterized by the coming together of different practices, most notably, mathematics and logic on the one hand and engineering on the other. Today, this multi-faceted face of computer science is exactly what makes it so hard for practitioners to define their own field. By studying how, throughout the history of computing, these different practices came to be intertwined to build computers or to develop programming techniques, it becomes possible to make transparent how formal and engineering practices really constitute a new discipline that can perhaps not be classified using old schemes and fences. The challenge then is to pick up the relevant theoretical ideas and unravel how pure theory is transmuted into technology (and conversely) to constitute a practice that can be reduced to neither.

References

1. M. Campbell-Kelly. Alan Turing's other universal machine: Reflections on the Turing ACE computer and its influence. *Communications of the ACM*, 55(7):31–33, 2012.
2. Wesley A. Clark. Oral history interview by Judy E. O'Neill, 3 May 1990. Charles Babbage Institute, University of Minnesota, Minneapolis.
3. Wesley A. Clark. The logical structure of digital computers. Technical report, Course notes, Division 6 Lincoln Laboratory, MIT, 1955.
4. Wesley A. Clark. The Lincoln TX-2 computer development. *Proceedings WJCC*, pages 43–145, 1957.
5. Haskell B. Curry. The combinatory foundations of mathematical logic. *The Journal of Symbolic Logic*, 7(2):49–64, 1942.
6. Haskell B. Curry. A program composition technique as applied to inverse interpolation. Technical Report 10337, Naval Ordnance Laboratory, 1950.
7. Martin Davis. *Engines of Logic: Mathematicians and the Origin of the Computer*. W.W. Norton and Company, New York, 2001.
8. E.G. Daylight. Interview with Van der Poel in February 2010, conducted by Gerard Alberts, David Nofre, Karel Van Oudheusden, and Jelske Schaap. Technical report, 2010.
9. Herman H. Goldstine and John von Neumann. Planning and coding of problems for an electronic computing instrument. vol. 2, part I,II and III, 1947-48. Report prepared for U. S. Army Ord. Dept. under Contract W-36-034-ORD-7481.

10. Andrew Hodges. *Alan M. Turing. The enigma.* Burnett Books, London, 1983. Republication (1992), 2nd edition, Vintage, London.
11. E. Kranakis. Early Computers in The Netherlands. *CWI-Quarterly*, pages 61–274.
12. 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.
13. L. De Mol, M. Bullynck, and M. Carle. Haskell before Haskell. an alternative lesson in practical logics of the ENIAC. *Journal of Logic and Computation*, 25:1011–1046, 2015.
14. L. De Mol and G. Primiero. Facing computing as technique: Towards a history and philosophy of computing. *Philosophy and Technology*, 27:321–326, 2014.
15. Liesbeth De Mol. Generating, solving and the human mind. emil post's views on computation. In *A computable universe, Understanding Computation & Exploring Nature As Computation*, pages 45–62, Providence, Rhode Island, 2014. Worldscientific.
16. E.F. Moore. A simplified universal Turing machine. In *Proceedings of the meeting of the ACM, Toronto Sept. 8 1952*, pages 50–54, 1952.
17. 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.
18. C.E. Shannon. A universal Turing machine with two internal states. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 157–166. Princeton University Press, 1956.
19. 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.
20. 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.
21. A. van den Bogaard. Stijlen van programmeren 1952–1972. *Studium*, 2:128–144, 2008.
22. W. van der Poel. Inzending 1946/47 van Van der Poel op de prijsvraag genaamd "1+1=10". Technical report, Delft, 1948.
23. Willem L. van der Poel. A simple electronic digital computer'. *Applied Scientific Research Section B*, 2:367–400, 1952.
24. W.L. van der Poel. A simple electronic digital computer. *Appl. sci. Res.*, 2:367–399, 1952.
25. W.L. van der Poel. *The Logical Principles of Some Simple Computers.* PhD thesis, Universiteit van Amsterdam, February 1956.
26. W.L. van der Poel. *Digitale Informationswandler*, chapter Microprogramming and trickology, pages 269–311. Braunschweig: Vieweg, 1961.
27. W.L. van der Poel. Een leven met computers. TU Delft, October 1988.
28. C.J.D.M. Verhagen. Rekenmachines in Delft. Uitgave van de Commissie Rekenmachines van de Technische Hogeschool te Delft, 1960.
29. Hao Wang. A variant to Turing's theory of computing machines. *Journal of the ACM*, 4(1):63–92, 1957.