Less is more in the Fifties. Encounters between Logical Minimalism and Computer Design during the 1950s

Liesbeth de Mol, Maarten Bullynck, Edgar G. Daylight

Less is more in the Fifties. Encounters between Logical Minimalism and Computer Design during the 1950s

Liesbeth De Mol (CNRS – UMR8163 STL, Université de Lille 3, France)
Maarten Bullynck (Université Paris VIII, Vincennes Saint-Denis)
Edgar G. Daylight (Universität Siegen & KU Leuven)

In recent years, there has been a renewed historiographic 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 Turing and von Neumann and the question if and how their logical and mathematical works affected the shaping of the modern computer and its coding practices. For many still, the early computers are "variations on the protean design of a limited Turing machine" [31, p. 119], a perspective where the EDVAC-design, the universal Turing machine and the stored-program computer are often conflated into one single concept, hiding the complexity of earlier computers, their many differences and different histories and settings. This conflation is both historically and conceptually wrong. Rather, the Turing machine was appropriated a posteriori by computer scientists to serve as the conceptual model of a general-purpose digital computer [14], and the stored-program computer is a construction after the facts started off by IBM [22]. This demystification, however, should not distract us from the fact that logic and some of its concepts were important in the development of the digital computer, on the contrary, it should stimulate research into how these concepts were actually integrated in the practices of the rapidly developing computer field.

The present paper wants to embed some important developments of the 1950s in two older traditions, one within (mathematical) logic and one within engineering. Both traditions could be termed *logical minimalism*, meaning the *systematic* use of (mathematical) logic in designing minimal systems and devices. These forms of logical minimalism were recast into a diversity of computing practices in the 40s and 50s. The logical tradition 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. The engineering tradition then emerged during the 1930s to design relay circuits and is part of a more general trend of using mathematical techniques in engineering. In the 1940s and 1950s, however, these traditions were redefined and appropriated when computer engineers, logicians and mathematicians started searching for the small(est) and/or simple(st) machines with an eye on engineering a small and relatively cheap digital computer. Of course, minimalism on one level does not imply overall simplicity, and nearly always, these logically small machines came with tradeoffs, mostly more involved and complex programming and a need for more memory for efficient operation. This paper studies the search for small machines, both physically and logically, and ties it to these older traditions 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. This was the result of the confluence of many different lines and agendas of research, among them research in algebraic logic (Boole, Schröder, Peirce etc.), the question on how to define real numbers (Weierstrass, Cantor, Dedekind etc.), set theory and the ensuing discussions (Cantor, Hilbert, Borel, Poincaré etc.) or the logicist programmes of Frege, Peano and Russell and Whitehead. A quite exhaustive panorama of the many strands coming together in the birth of this foundationalist movement in mathematics can be found in [21]. Landmarks were the publication of Whitehead and Russell's *Principia Mathematica* (3 volumes 1910-1913) and David Hilbert's metamathematical research agenda of 1921, both attracting many logicians, mathematicians and philosophers to work on foundational issues.

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, quite some of the advances made in mathematical logic during this period 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. Of particular importance for logical minimalism was Whitehead and Russell's *Principia Mathematica* that tried to formalise the entirety of mathematics in logical symbols and propositions. Of their own confession, they could not garantuee that their sets of basic propositions (or premisses) were minimal, nor that their set of primitive ideas was minimal, but they stated the ambition:

> It is to some extent optional what ideas we take as undefined in mathematics; the motives guiding our choice will be (1) to make the number of undefined ideas as small as possible, (2) as between two systems in which the number is equal, to choose the one which seems the simpler and easier. We know no way of proving that such and such a system of undefined ideas contains as few as will give such and such results. [64, p. 95]

These are indeed the two obvious lines of research informed by this minimalist philosophy. On the one hand, finding the smallest set of logical primitives, on the other hand reducing and/or simplifying existing axiom systems. A famous example of the first strand is Sheffer's 1913 paper that showed that one operation, the Sheffer stroke, suffices as primitive operation for a Boolean algebra [53]. An example of the second strand would be Jean Nicod's small set of basic propositions for the propositional calculus, or similar work on minimal sets of premisses in the Polish school of logic.

In the 1920s then, when Hilbert with his own brand of formalism had added the metamathematical questions to the agenda, 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 [47, 46]. This method which ultimately tried 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 situated 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 was no less than to eliminate more fundamental and undefined notions of logic, including the variable. His reason for doing so was not purely methodological but also philosophical [49, p. 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.

After having made the classic trip to Europe, spending quite some time at Göttingen, Curry started working on his PhD, taking up Schönfinkel's ideas. This led to what is now known as combinatory logic. In combinatory logic, only three basic operations are used: the combinators K, W and C, corresponding respectively to Kxy = x, Wxy = Wxyy and Cxyz = xzy. In an address to the Association of Symbolic Logic (1942), Curry identified simplification as one of two major tendencies (the other is formalization) in investigations on the foundations of mathematics [10, 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.

This analysis down to the most elementary operations would help Curry later, in 1946-1950, to come up with an elaborate theory of combining simple programs so as to develop complex programs. (see section 3)

Turing's work is well-known, his famous *On computable numbers* [58] was written as a (negative) answer to one of Hilbert's problems, the Entscheidungsproblem of propositional logic. To do this he developed the formalism now known as Turing machines, which he obtained by observing a man in the process of computing a number and then try to make abstraction of this and reduce it to its most elementary and simple 'operations' [58, p. 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.

Those basic operations according to Turing are four: go left, go right, erase, and print a symbol. This primitiveness of the operations of the Turing machine allowed to define the universal Turing machine which is capable to compute what any other Turing machine can do. Indeed, the elementarity of these acts made it possible to translate them into a code understandable by a (Turing) machine. Much later, in the late 1950s, Turing's theory would be reclaimed by those founding computer science (see [14]), but in 1936-37, it was still a result in mathematical logic. It would also have, through von Neumann's and Turing's own work, some, limited impact on the design of the first digital computers (see section 3).

2 A tradition of 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. Of course, various forms of minimalism have always guided engineers in designing systems, but they often remained informal or ad-hoc, marrying the intuition of Ockham's razor with pragmatism. In tune with the contemporary import of more advanced mathematical techniques in engineering, engineers of relay switching circuits started looking for systematic techniques they could borrow from discrete mathematics to advance their art in the early 20th century. Claude E. Shannon's master's thesis "A symbolical analysis of relay switching circuits" (1938) [50] 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 [50, p. 713]:

> 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 was actually but one of a number of mathematical (and graphical) techniques Shannon proposed in his thesis to attack the problem of designing specific circuits with a minimum number of elements. It should also be noted that many other researchers, mostly in Japan, Germany and Russia, came up with similar ideas and techniques around the same time (see [61, Part 2]).

Minimalism in the engineering sense forbade easy definition, as Shannon later explained:

> the economy of elements [...] may be of several types, for example:
>
> 1. We may wish to realize our function with the least total number of switching elements, regardless of which variables they represent.
> 2. We may wish to find the circuit using the least total number of relay springs [...]
> 3. We may wish to distribute the spring loading on all the relays or on some subset of the relays as evenly as possible.
>
> [...] Although all equivalent circuits representing a given function *f* [...] can be found with the aid of Boolean Algebra, the most economical circuit in the any of the above senses will often not be of this type. [...] The difficulty springs from the large number of essentially different networks available and more particularly from the lack of a simple mathematical idiom for representing these circuits. [51]

Both the variety of elements used in engineering as the multiple facets of their design complexified an easy application of Boolean algebra, but also the integration of larger circuits into such a calculus was far from obvious.

In short, Shannon's techniques were not so simply amenable to engineering practice and it took a while before it became useful. They were further explored and developed at Bell Labs in the 1940s, in particular for some complex relay circuits needed in the No. 5 Crossbar for telephone switching or the Relay Calculators conceived by George R. Stibitz. Other researchers such as Karnaugh, McCluskey, Mealy, Moore at Bell Labs and M.I.T., or Aiken and his collaborators at Harvard all came up with further techniques to find minimal circuits.

Edward F. Moore even exhaustively tabulated the most economical relay circuit for each Boolean function of four variables in 1952 [33, pp. 56-58]. Together with Shannon, Moore also patented a circuit analyzer developed in 1952-53 [38]:

> 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 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. This made some engineers look at the Turing machine as a possible way to achieve this top-down strategy in the 1950s (see section 4).

3 Early minimalist computing: Von Neumann, Turing and Curry

With the development of the general-purpose digital electronic computer after World War II, a number of people were involved that were well-versed in mathematical logic. Most notable were John von Neumann and Alan Turing, and through their work some implicit logical minimalism entered into the design of early digital computers. It must be remarked, however, that neither von Neumann nor Turing formulated a systematic framework or method to design logically minimal machines, their minimalism remained ad-hoc and implicit. In the pioneering years of the 1940s, only the logician Haskell B. Curry developed a systematic framework guided by logical minimalism, though his work would find no echo in the field, probably because Curry's reports did not circulate widely.

The famous *First Draft of the EDVAC* (1945) written up by von Neumann is characterised by a reduction of the complexity in the early computing machines. Walter Pitts and Warren McCullough's paper on neural networks helped von Neumann to schematise the relationships between the various units, reducing the many units of the ENIAC to the five classic units of what is now known as the von-Neumann-architecture (though von Neumann consistently underestimated the complexity of input-output). A similar minimalism was at work in von Neumann's desire to keep the number of (machine) instructions to a minimum:

> To keep the number of components at an absolute minimum, the machine [EDVAC] had only a few built-in instructions. That was a wise decision. Each "instruction" demanded dozens of tubes and hundreds of handwired connections. And each increased the computer's cost and multiplied the probability that it would experience a failure well before any significant computational task could be completed. [4, p. 237]

Not only did each machine instruction add quantitatively to the complexity of the machine (more elements), but also qualitatively, because each new machine instruction required dedicated design and much manual wiring instead of reusing standard elements. Evidently, the number of programming instructions may be much greater, an instruction in a code being a combination of one or more machine instructions.

The EDVAC built at the Moore School would eventually have 12 machine instructions. The IAS machine would originally have 26 instructions (according to the preliminary reports), but would ultimately come down to 20:

> Von Neumann also believed that his machine should have very few commands. The smaller the number of commands, the less internal circuitry that would be needed. Following his mandates, his engineers were able to reduce the number of components in the IAS machine. It had only two-thirds the number of tubes of the EDVAC. [4, p. 240]

Other American machines built after the EDVAC-design also had a moderately small machine instruction set: the Whirlwind had 31, the ERA 1101 had 38. The number of instructions was not determined by any logical principle, rather, instructions were added if the main application of the machine needed it frequently, or if a new kind of peripheral was added to the system.

Haskell B. Curry, professor of logic at the university of Pennsylviana, had read Goldstine and von Neumann's papers on the design and programming of the IAS machine, but found their flowchart approach too elaborate. Therefore, he developed an independent attack on the problem of coding and planning using his experience with the ENIAC on the one hand, and his combinatory logic on the other hand. Indeed, Curry had helped in 1946 to devise computation schemes for the ENIAC and had tried to find a general way to combine two or more programs. His concrete experience with the ENIAC coupled with the combinatorial logic and its minimalist philosophy led Curry to 'design' a theory of programming.

One key aspect of this theory was the analysis of programs into basic programs and the development of a theory which allowed to compose more complicated programs from these basic programs in an automatable fashion. This analysis into basic programs and their composition explicitly displayed a minimalist philosophy [11]:

> [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 one's insight into the most profound and abstract theories by considering trivially simple examples

Curry went on to give a method which reduced a specific class of 26 basic programs from the original list of IAS-machine instructions to only 4 basic programs. The proof of this reduction to 4 basic programs was through the detailing of a (programmable) method which resynthesises the original 26 basic programs from these 4 initial programs. Curry commented that one might save machine memory when compiling programs. He therefore made the following hardware recommendation [11, pp. 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.

Through his logical analysis, Curry could indeed recommend to hardwire only 4 instructions and resynthesise all other order code instructions through appropriate and automatic programming, if enough machine memory was available. Hence, a theoretical result so in line with logical minimalism, serves as a blueprint for automatically compiling complex order instructions as combinations of simple machine instructions (for more details on Curry's theory of programming see [36]). It also points down quite exactly an important and recurring trade-off: With fewer machine instructions, you will need both memory and speed to compile instructions that are not wired-down. Or put differently: A small machine instruction set comes at the price of more complex programming, felt either in hardware (loss of memory and speed) or software (more elaborate planning and coding).

Finally, Turing was evidently also guided by a minimalist logic philosophy when he helped to develop one of England's first computers, the ACE (Automatic Computing Engine). Just after World War II, Turing was recruited by John Womersley of the NPL to help design the ACE. As has been argued elsewhere [12, 24], 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 stated that computers such as the ACE "are in fact practical versions of the universal machine" [59]. Even though a good theoretical model, it needed to be adapted. Thus, for instance, Turing made 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 [24, p. 319].

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. Or, to put it in Turing's words, "[W]e have often simplified the circuit at the expense of the code" [59]. The same philosophy is echoed in Harry D. Huskey's 1947 report on computer developments at NPL:

> Certain principles have been established of which one is:
> USE THE MINIMUM AMOUNT OF EQUIPMENT,
> that is, do everything possible by programming unless it has to be done extremely frequently [25, p. 536]

This trade-off between a simple computer architecture and a more complex programming system will also be a recurrent theme in the small machines realised in the 1950s. One of the main strategies developed at NPL to achieve reasonable execution speed despite longer instruction code was "optimum coding", exploiting the timing of instruction cycles to cram a maximum of (machine) instructions in a time slot. As Huskey remarked, "This machine has been planned on the premise that switching can be accomplished between pulses" [25, p. 537]

Between Turing's plan and the ACE's eventual realisation(s) many things happened (see [9]), and the simplicity of the conception was not always recognisable in the implementation:

> There are logical facilities in the ACE not needed in most computing problems. The numerical operations of addition, subtraction (with complements), and multiplication, along with a discrimination process seem sufficient.With these other logical operations can be programmed. [...] The computing portions of the ACE can be considerably simplified and certainly should be in any small machine that is built. [25, p. 539]

However, there were other computers derived from the ACE-design, e.g. the DEUCE (see [9]], and especially the Bendix G-15. This machine was developed by Huskey, who was the main engineer on the (Test Assembly) ACE in 1947. He took up the basic conception of the ACE and pursued the development of a small machine as such. Huskey had worked in the ENIAC team in the mid-1940s and had thereafter spent some years travelling back and forth between the U.S. and the U.K. Upon returning definitively to the States, Huskey developed the design of a small general-purpose computer, inspired by the ACE-design, and tried to sell his idea to manufacturers [27].

> high speed electronic computers [...] prior to the present invention tend to be extremely complex and extremely costly. [...] The present invention is [...] a small high speed general purpose computer which will perform a great variety of computations thus giving it a wide range of utility as a machine aid to computation [bringing it] within the economic reach of potential users [...] who cannot afford the more costly equipment heretofore needed for general computation purposes. [26]

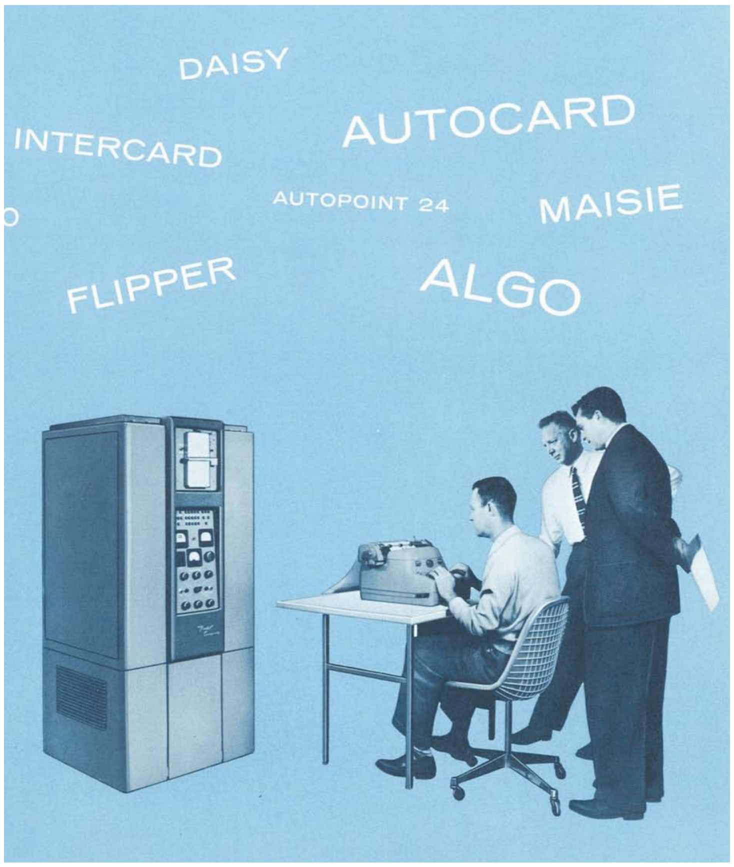

Figure 1: The Bendix-G15 machine. Hovering over the machine are the names of the various programming aids that were provided for the machine.

Huskey's redesigned computer design was eventually sold to the highest bidder, Bendix.

Upon engineering the machine itself, which would be marketed as the Bendix G-15, Bendix demanded Huskey to add multiplication and division to the small instruction set, but Huskey did not manage to include it, and finally, one of Bendix's engineers, Robert Beck finally got it in [39, p. 4]. Later still, in 1961, Beck together with Max Palevsky would design the Packard-Bell 250 computer (1961). This computer was another ACE- or G-15-inspired design, but transistorised and using magnetostrictive delay lines instead of a magnetic drum. Stanley Frankel, incidentally, was a consultant for that same computer.

The command structure of the G-15 had the following form:

(T or L_k;N;C;S;D)

T in combination with S (for Source) or D (for Destination) was the operand,

L_k was used for I/O that did not need an operand. N indicated where the next instruction was to be taken. And finally, C had a value between 0 and 7, referring to the time cycle of the magnetic drum memory, 4 words being on one short line (cyclic on the drum), but, when using double-precision, 8 words being available on two short lines. Because programming with this code was not straightforward, especially if one wanted a minimum acces code reducing the time lost in waiting on the drum to rotate (details on minimum access coding and magnetic drums in Section 5), quite some programming devices were developed by Huskey and the users of the Bendix G-15. Bendix offered POGO as an automatic coding aid that helped avoid the intricacies of machine coding, but it was not very popular with the users. Huskey himself developed another approach, an interpretive routine, which means that one line at a time could be translated in machine code and executed. This approach contrasts with loading and compiling the entire program first before it can be executed. INTERCOM 500, or in double precision, INTERCOM 1000 in 1959 that proved more suited to the users' needs. Later still, Huskey would also port his European ALGOL experience to the machine, introducing the ALGO language (1960).

4 Less is more in the Fifties I: 'Automata studies'

Quite independently of Curry's, von Neumann's and Turing's more practical work of the 1940s, a renewed interest into the benefits of systematic applications of logical minimalism emerged in the 1950s. Several researchers coming from different backgrounds, but with the same keen interest in the theory *and* practice of the new computing machines, became 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. But also the 'rapprochement' between Boolean logic, circuit design and mathematical logic contributed considerably to more interaction between mathematical logicians and computer scientists. In this context, a tradition of logical minimalism was 'transmuted' to the context of machines. 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 (cf. also [32, Part III]).

With the foundation of the *Journal of Symbolic Logic* in 1936 and through its driving force, Alonzo Church, a 'rapprochement' between circuit switching and mathematical logic slowly grew from the late 1940s onwards. The tenth meeting of the Association for Symbolic Logic in December 1947 featured talks by E.C. Berkeley and T.A. Kalin both addressing how symbolic logic and the new digital computing machines could interact. From then onwards, Church and others scrupulously reviewed all publications on the use of logic in switching theory, and, reciprocally, more advanced logical techniques became known to the engineers. This began to bear some fruits in the early 1950s onwards. An important example of this transmission of knowledge between engineers and logicians was the logician Willard V. Quine's method from 1952 to find the simplest Boolean function [48] that was promptly picked up by some circuit designers such as David A. Huffman. Another important stepping stone was Stephen C. Kleene's theoretical treatment of McCullough and Pitts's nerve nets. This rather informal model had been the model used by von Neumann for his description of the EDVAC. Now Kleene's report from 1951 [28] written on order for the RAND corporation brought a formal treatment, thus connecting the logical design of computers with metamathematical theory, and starting off quite some theoretical research on finite automata and Turing machines. A substantial part of this RAND report was later published in the volume *Automata Studies* (1956).

During the same early fifties, Edward F. Moore and Claude E. Shannon at Bell Labs 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 [52]). In 1952 Moore presented his 'simplified universal Turing machine' at the ACM meeting in Toronto, the paper appeared two years later. Moore described a 15-state two-symbol three-tape universal Turing machine.

Moore starts out from Martin 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.

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." [37, p. 54] But at the same time, Moore remarks that 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." Moore also explains how magnetic tape memory is more suited in this context than punched tape [37, p. 54]:" 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." Perhaps Wang's non-erasing model described in [63], was inspired by the need for a simple model for a punched-tape computer.

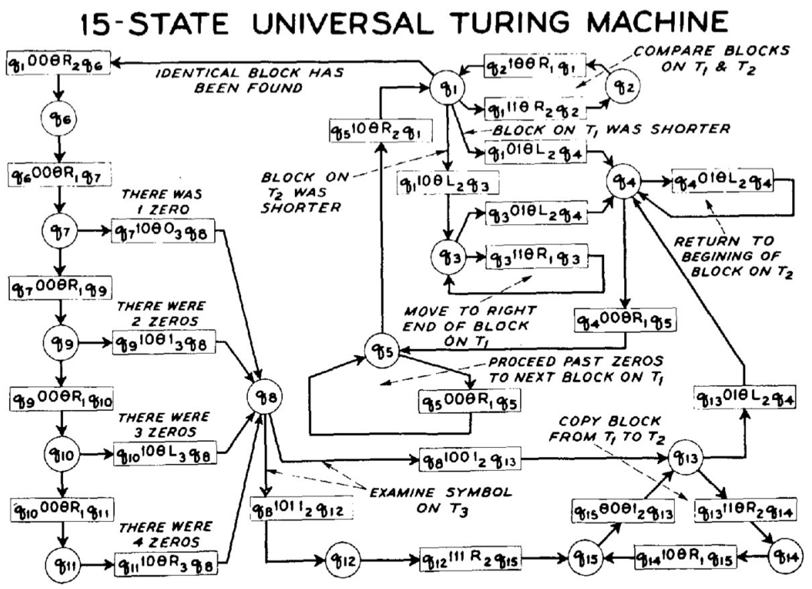

Figure 2: The diagram of E.F. Moore's 15-state three-tape universal Turing machine

Starting at the other end, trying to adapt Turing machines to resemble actual computers, 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 [63]. He explicitly placed 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 [63, p. 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.

Wang's model only has four basic instructions: shift the head one square to the right; shift the head one square to the left; mark the square of the tape under scan; and a conditional transfer. He remarks that his universal machine would be less economical to realise than Moore's, but that its interest lies in reducing the instructions to "a bare minimum" [63, p. 88]. Note that Wang's B-machines are basically non-erasing Post machines [46].

Wang was actually not the first to develop a variant to Turing's machine, already the German logician Hans Hermes had provided a proof that a digital computer, provided with infinite memory, is equivalent to a Turing machine [23]. In Hermes's formalisation, there are five 'Elementarmaschinen': go right, go left, mark, zero (erase) and decision (conditional transfer). Using these five machines, Hermes showed how a digital computer can emulate a sequence of such elementary machines that make up a universal Turing machine. Hermes's contribution, due to its being written in German, would not be widely read, but Wang's congenial model would later be developed by others and be the basis for the register machine model [54]. Indepently, in the late 1950s, Marvin Minsky at MIT developed a similar theoretical model and using Post's tag systems (mentioned in section 1), Minsky proved that two registers suffice to have universality. [34]. Minsky would later write several of the other classic papers on small universal (paper) devices.

5 Less is more in the fifties II: Simple digital computers

At around the same time that Edward F. Moore was thinking on small universal Turing machines and their practical feasibility, several engineers started to effectively implement similar ideas by building "small" computers, viz. computers

Machine	Year	Technology	Number produced
ACE	1949	Vacuum tubes and delay lines	1
Bendix G-15	1955	Vacuum tubes and magnetic drum	300
ZERO	1952	Vacuum tubes and magnetic drum	(1)
Stantec Zebra	1956	Vacuum tubes and magnetic drum	44
MINAC	1954	Vacuum Tubes and magnetic drum	1
LGP-30	1956	Vacuum tubes and magnetic drum	460
TX-0	1956	Transistors and magnetic core	1
PDP-1	1960	Transistors and magnetic core	53

which are designed around the idea of simplicity of architecture and economy of (machine) instructions. Their goal was to find a design for a digital computer that was small and relatively cheap, while at the same time maintaining the general purpose character of the computer. Some engineers involved in the development of small computers were well versed in modern mathematical logic and its possible applications to computer design. However, it turned out that minimalist philosophies had to be redefined as parts integrating more complex computer design strategies.

Most of these simple, small computers started off as either thought experiments or as experimental machines before a commercial version would be marketed. These commercial versions, quite pragmatically, modified the initial experimental concept considerably, they appeared in the latter half of the 1950s. The idea was to build a (relatively) cheap machine, hence the economy of (machine) instructions and of hardware elements and the combination with a magnetic drum memory, which was the cheapest kind of memory on the market in the early 1950s. The ACE-computer (1949) was redeveloped into the commercial Bendix G-15 by Harry Huskey (1955) (see section 3). Van der Poel's ZERO machine, built as an experimental machine in the Netherlands (1952) was the basis for his ZEBRA (1956) that was commercialized by STANTEC (1957). Stanley Frankel's MINAC (1954) was the model for Librascope's LGP-30 (1957), and, finally, at MIT, the transistorized TX-0 (1955) was an experimental machine to test transistors and would, eventually, serve as the blueprint for DEC's PDP-1 computer (1960).

Except for the TX-0 at Lincoln Lab, which had an advanced ferro-magnetic core memory, all these computers used a magnetic drum memory, being the cheapest and quite reliable memory on the market back then. Because the drum revolved rather slowly, in comparison to the addition time, techniques were developed to exploit the time in between drum cycles. There is "a minimum interval between the location of the command and its operands or operands [...] attention to these minimum (coding intervals in construction of a program is called minimum access programming" [40, p. 173] Using minimum access coding (or optimum coding or minimal latency coding as others would call it), viz. letting multiple instructions execute within a drum cycle, ERA's engineers, who were the experts for magnetic drums, achieved speedups, making, e.g., matrix inversion 50 times faster. (Turing's 'optimum coding' mentioned in Section 3 is close in spirit to this minimum access programming, though it exploits the cycles of the delay lines rather than the magnetic drum.)

Another hardware innovation of the 1950s was also used by a number of these small machines: microprogramming. Maurice Wilkes and John Stringer had published a description of this hardware technique in 1953 [65], though the idea had been discovered independently by a number of other engineers. Microprogramming is an example of engineering minimalism. The idea was to economize on the hardware parts by analyzing programming instructions as sequences of machine transfers and signals.

> While arithmetic operations are usually considered as basic by programmers, to the engineer it is simpler operations, namely shifts and transfers, that are basic. Microprograming is the same process as programming, but with engineering operations as the basis. The distinction between microprogramming and programming is a question of drawing a line between the machine's control and the programer's control. [56, p. 74]

A sequencing control unit, often a matrix circuit, "brings the machine to life by supplying the pulses which cause the suboperations required in the execution of an order to take place." [66, p. 19] These so-called microprograms could be hardwired in the machine. Most vendors did not provide the user access to this matrix, but the user could pay the vendor extra to provide a new microprogram to be wired in the matrix. The design of the sequences was generally not guided by a systematic logico-mathematical framework (with the possible exception of a Boolean West Coast 'rational design' as Wilkes seems to suggest [66, p. 20]), though experience and context provided the pragmatic guidelines to decide on how and which microprograms to provide. The microprogramming idea proved to be a useful hardware medium to implement the small computer designs of the 1950s: the Zebra used a (rather slow) sequencing unit, and the Bendix G-15 and the TX-0 extensively used a magnetic core matrix to stimulate various microprograms.

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. Van der Poel had been recruited as a graduate student in 1947 to work on the ARCO, a relay-based "all-round" calculating machine, a "self-thinking" device. ARCO was a project started by Nicolaas G. de Bruijn (1918–2012), who had just been appointed professor in Delft. 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." Immediately after graduation in 1950, van der Poel joined Kosten's team 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 had characterized the most essential problems for the Delft project as “programming problems”.

> Solving a problem by programming amounts to choosing an appropriate set of instructions. Of course, the challenge is to accomplish as much as is feasible with as few instructions as possible. [41, p. 60]

At PTT, Kosten and van der Poel decided to build a radically new, digital, calculating machine, the PTERA, [PTT Elektronische Reken Automaat” (PTT Electronic Calculating Automaton)] which differed greatly from the TESTUDO (more details in Kranakis [29, p. 70-72]). 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.

> [The TESTUDO] was still a pre-von-Neumann machine. At that time I did not yet have the idea to store instructions and numbers in the same memory. I became aware of that idea when I read the famous report ‘Preliminary discussion of the logical design of an electronic computing instrument’ of Burks, Goldstine and von Neumann [...] [45, p. 8]

In 1952 then van der Poel’s knowledge of the literature (next to IAS-reports also Shannon and Turing) 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” [42, p. 368]. The idea was to build the simplest possible computer taking into account at least some practical limitations [42, p. 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 economy of design and logical minimalism as the following examples show. Van der Poel used the same register to serve both 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” [42]. 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. 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 did not push the minimalist philosophy to its limits on the ZERO: "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." [42] Later, in his PhD [43] he would go further. First, van der Poel made an analysis of the 7 instructions, remarking that, e.g., shift instructions can be omitted because (on a binary machine) shift to the left is actually doubling, a number, or, a number once added to itself. He concluded, "there are left the operations store with clear (T), add (A), subtract (S), jump (X) and some form of test order." [43, p. 95] But one could do with less, van der Poel showed how subtraction could be done using additions (and vice versa), and also how a test order is not strictly necessary. "It is a widespread opinion that automatic computers are universal because they have a facility to discriminate, a test order" [43, p. 96], but van der Poel showed it is superfluous, or rather, that one out of the three orders (test, shift to the right and conjunction) suffice to build the other two. This leaves only the order X, S and T, but "S and T can be combined into a single order" which is Bn, it substracts n from the value in the accumulator and stores the result [43, p. 97]. Van der Poel concludes: "Now the machine only knows two types of instructions X and B. As a last step we shall discuss the cancelling of the X-operation." [43, p. 98] This is done by automatic alternating between B and X orders. Depending on the n of Bn, the machine simply jumps to the next line, or (if $n = 0$) jumps to the next line and interprets the next Bn as jump to location n. This led ultimately to the one-instruction machine, the so-called "purely one-operation machine", that is "a purely jumpless machine" without "disguised jump". Van der Poel showed the possibility of this one instruction, going over all three registers (accumulator, memory and control register), first extracting 0 or 1 (in case of 1 a B instruction will be executed, in case of 0 an X instruction), followed by a quite intricate round of transfers between the registers [43, pp. 100-102]. At the very end of his dissertation van der Poel concluded:

> It is more a question of economy to determine the optimum capacity of the store, and the complication of the operational part, in relation to the speed and the price. It is very remarkable that the ZERO is already a practicable machine, though it is hardly more complicated than the one-operation machine, which is completely impractical. [43, p. 105]

In other words, there is a trade-off between the size of the instruction list and the working memory available, a one-instruction machine would need much memory all the time to be workable, whereas a machine with a reasonable number of instructions can do with less memory.

Given his combined experience with the ZERO and the PTERA, 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 [43], supervised by professor van Wijngaarden from the University of Amsterdam. It was inspired by ZERO

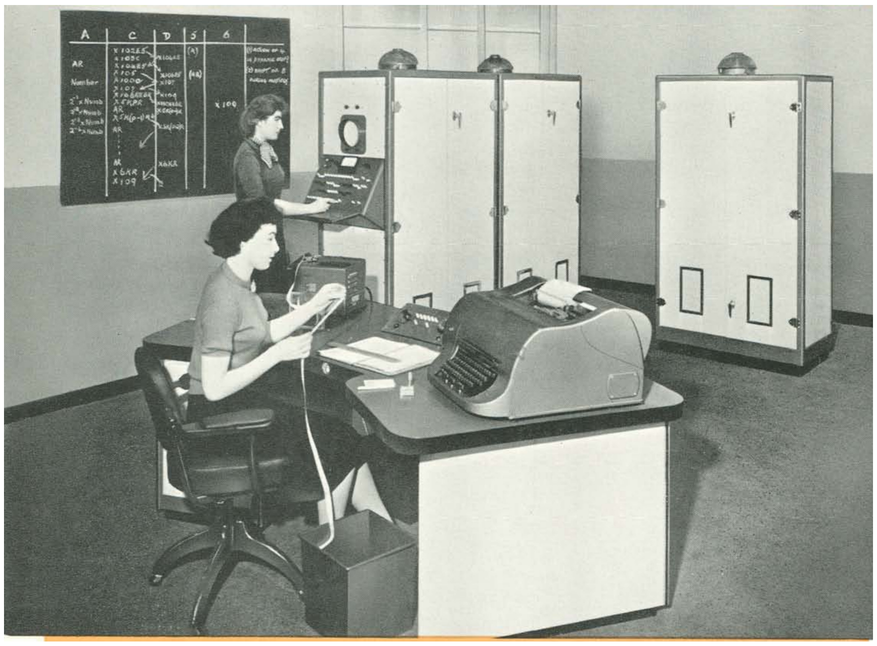

Figure 3: The STANTEC ZEBRA computer. The blackboard displays a typical example of Van der Poel's programming style with arrows indicating the timing relationships between the instructions.

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. This was achieved in the ZERO by setting the X-bit to 0 or 1; in the ZEBRA this technique basically remained unchanged. Van der Poel realized fully well that "this is seldom of practical importance" [43, 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 ZEBRAs throughout Europe [29, 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 [2].

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. 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 [62, 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 [13]. 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. However, as the STANTEC manual noted: "a good deal of skill and experience is required in the programmer". Therefore a Simple Code was developed that could be used instead of the Normal Code: "a special instruction code known as the 'simple code' has been devised, which works in conjunction with interpretive routines stored in the machine" [60, p. 13]. Using this interpretative programming language "day to day problems" could be programmed "after brief training", making the machine only "one-sixth to one-fifth", sometimes even "half as fast" as when operated under Normal Code [60, p. 14]. Without the support for optimum coding and underwater programming and the development of the Simple Code, van der Poel's aesthetically pleasing machines would have remained both slow and difficult to program, and, as a result, economically unattractive.

5.2 Frankel's MicrocephalAC and the LGP-30

Stanley Frankel was a physicist who had been part of the Los Alamos team that had used the ENIAC for their calculations back in 1945-48. After that period, Frankel became a professor at CalTech, but remained interested in computing, in particular in developing a small and cheap computer. Through his contacts at Hughes Aircraft, Frankel got free diodes and thanks to his consultancy at Northrop he became familiar with the usage of Boolean algebraic equations in computer design, typical of the Western Coast style of designing computers [55]. Together with a CalTech student, James Cass, Frankel managed to (partially) build a small computer, the MINAC (1954). Frankel subsequently sold his idea to Librascope who had been interested earlier in Huskey's design, but lost the bid to Bendix. With James Cass as main engineer, they developed the LGP-30,

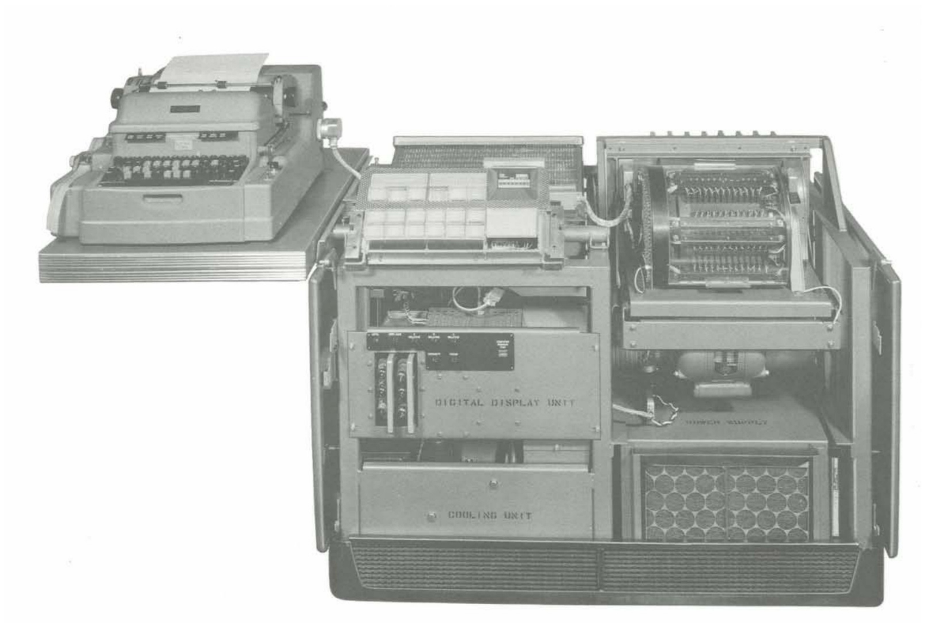

Figure 4: Librascope's General Purpose computer LGP-30.

LGP standing for Librascope General Purpose.

Frankel's machines had a theoretical background that was laid out in three journal articles. The first explained how a small computer with a magnetic drum could be a general-purpose machine, using Turing's 1936-paper:

> One remarkable result of Turing's investigation is that he was able to describe a single computer which is able to compute any computable number. He called this machine a universal computer. It is thus the 'best possible' computer mentioned. [...] This surprising result shows that in examining the question of what problems are, in principle, solvable by computing machines, we do not need to consider an infinite series of computers of greater and greater complexity but may think only of a single machine. Even more surprising than the theoretical possibility of such a 'best possible' computer is the fact that it need not be very complex. The description given by Turing of a universal computer is not unique. Many computers, some of quite modest complexity, satisfy the requirements for a universal computer. In particular, it will be seen in the following that any of the modern general purpose computers, such as the relatively simple LGP-30, is a universal computer, as is the Analytical Engine mentioned previously. We now have a partial answer to our question as to the range of problems which can, in principle, be solved by a general purpose computer (GPC); namely: What one GPC can do

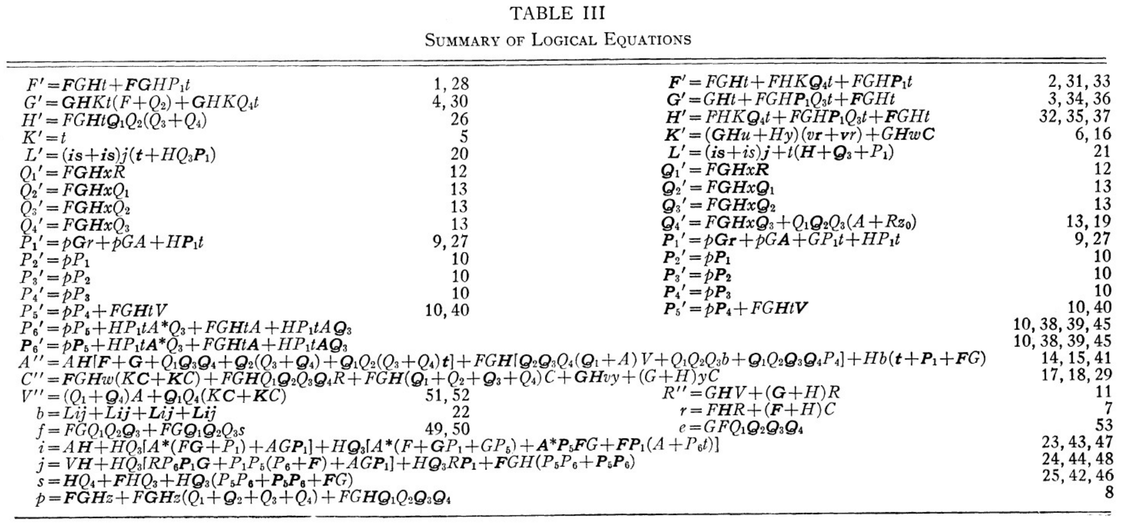
TABLE III

Summary of Logical Equations

$F'=\boldsymbol{F}G\boldsymbol{H}t+\boldsymbol{FG}HP_1t$	1, 28	$\boldsymbol{F}'=FG\boldsymbol{H}t+FHK\boldsymbol{Q}_4t+FGH\boldsymbol{P}_1t$	2, 31, 33
$G'=\boldsymbol{GH}Kt(F+Q_2)+\boldsymbol{G}HKQ_4t$	4, 30	$\boldsymbol{G}'=G\boldsymbol{H}t+FGH\boldsymbol{P}_1Q_3t+\boldsymbol{F}GHt$	3, 34, 36
$H'=FG\boldsymbol{H}t\boldsymbol{Q}_1Q_2(Q_3+Q_4)$	26	$\boldsymbol{H}'=FHK\boldsymbol{Q}_4t+FGH\boldsymbol{P}_1Q_3t+\boldsymbol{F}GHt$	32, 35, 37
$K'=t$	5	$\boldsymbol{K}'=(\boldsymbol{GH}u+Hy)(v\boldsymbol{r}+\boldsymbol{v}r)+G\boldsymbol{H}w\boldsymbol{C}$	6, 16
$L'=(\boldsymbol{is}+\boldsymbol{i}s)j(\boldsymbol{t}+HQ_3\boldsymbol{P}_1)$	20	$\boldsymbol{L}'=(\boldsymbol{is}+is)\boldsymbol{j}+t(\boldsymbol{H}+\boldsymbol{Q}_3+P_1)$	21
$Q_1'=F\boldsymbol{GH}xR$	12	$\boldsymbol{Q}_1'=F\boldsymbol{GH}x\boldsymbol{R}$	12
$Q_2'=F\boldsymbol{GH}xQ_1$	13	$\boldsymbol{Q}_2'=F\boldsymbol{GH}x\boldsymbol{Q}_1$	13
$Q_3'=F\boldsymbol{GH}xQ_2$	13	$\boldsymbol{Q}_3'=F\boldsymbol{GH}x\boldsymbol{Q}_2$	13
$Q_4'=F\boldsymbol{GH}xQ_3$	13	$\boldsymbol{Q}_4'=F\boldsymbol{GH}x\boldsymbol{Q}_3+Q_1\boldsymbol{Q}_2Q_3(A+Rz_0)$	13, 19
$P_1'=p\boldsymbol{G}r+pGA+H\boldsymbol{P}_1t$	9, 27	$\boldsymbol{P}_1'=p\boldsymbol{Gr}+pG\boldsymbol{A}+GP_1t+HP_1t$	9, 27
$P_2'=pP_1$	10	$\boldsymbol{P}_2'=p\boldsymbol{P}_1$	10
$P_3'=pP_2$	10	$\boldsymbol{P}_3'=p\boldsymbol{P}_2$	10
$P_4'=pP_3$	10	$\boldsymbol{P}_4'=p\boldsymbol{P}_3$	10
$P_5'=pP_4+FG\boldsymbol{H}tV$	10, 40	$\boldsymbol{P}_5'=p\boldsymbol{P}_4+FG\boldsymbol{H}t\boldsymbol{V}$	10, 40
$P_6'=pP_5+HP_1tA^*Q_3+FG\boldsymbol{H}tA+HP_1tA\boldsymbol{Q}_3$			10, 38, 39, 45
$\boldsymbol{P}_6'=p\boldsymbol{P}_5+HP_1t\boldsymbol{A}^*Q_3+FG\boldsymbol{H}t\boldsymbol{A}+HP_1t\boldsymbol{AQ}_3$			10, 38, 39, 45
$A''=A\boldsymbol{H}[\boldsymbol{F}+\boldsymbol{G}+Q_1\boldsymbol{Q}_3\boldsymbol{Q}_4+\boldsymbol{Q}_2(Q_3+\boldsymbol{Q}_4)+\boldsymbol{Q}_1Q_2(Q_3+Q_4)\boldsymbol{t}]+FG\boldsymbol{H}[\boldsymbol{Q}_2\boldsymbol{Q}_3Q_4(\boldsymbol{Q}_1+A)V+Q_1Q_2Q_3b+\boldsymbol{Q}_1Q_2\boldsymbol{Q}_3\boldsymbol{Q}_4P_4]+Hb(\boldsymbol{t}+\boldsymbol{P}_1+\boldsymbol{F}G)$			14, 15, 41
$C''=\boldsymbol{F}G\boldsymbol{H}w(K\boldsymbol{C}+\boldsymbol{K}C)+FG\boldsymbol{H}Q_1\boldsymbol{Q}_2Q_3\boldsymbol{Q}_4R+FG\boldsymbol{H}(\boldsymbol{Q}_1+Q_2+\boldsymbol{Q}_3+Q_4)C+\boldsymbol{GH}vy+(G+H)yC$			17, 18, 29
$V''=(Q_1+\boldsymbol{Q}_4)A+\boldsymbol{Q}_1Q_4(K\boldsymbol{C}+\boldsymbol{K}C)$	51, 52	$R''=G\boldsymbol{H}V+(\boldsymbol{G}+H)R$	11
$b=Lij+L\boldsymbol{ij}+\boldsymbol{L}i\boldsymbol{j}+\boldsymbol{L}ij$	22	$r=F\boldsymbol{H}R+(\boldsymbol{F}+H)C$	7
$f=FGQ_1Q_2\boldsymbol{Q}_3+FG\boldsymbol{Q}_1\boldsymbol{Q}_2Q_3s$	49, 50	$e=GFQ_1\boldsymbol{Q}_2\boldsymbol{Q}_3\boldsymbol{Q}_4$	53
$i=A\boldsymbol{H}+HQ_3[A^*(F\boldsymbol{G}+P_1)+AG\boldsymbol{P}_1]+H\boldsymbol{Q}_3[A^*(F+\boldsymbol{G}P_1+GP_5)+\boldsymbol{A}^*\boldsymbol{P}_5\boldsymbol{F}G+\boldsymbol{FP}_1(A+P_6t)]$			23, 43, 47
$j=V\boldsymbol{H}+HQ_3[RP_6\boldsymbol{P}_1\boldsymbol{G}+P_1P_5(P_6+\boldsymbol{F})+AG\boldsymbol{P}_1]+H\boldsymbol{Q}_3R\boldsymbol{P}_1+\boldsymbol{F}GH(P_5P_6+\boldsymbol{P}_5\boldsymbol{P}_6)$			24, 44, 48
$s=\boldsymbol{H}Q_4+\boldsymbol{F}HQ_3+H\boldsymbol{Q}_3(P_5P_6+\boldsymbol{P}_5\boldsymbol{P}_6+\boldsymbol{F}G)$			25, 42, 46
$p=\boldsymbol{FGH}z+F\boldsymbol{GH}z(Q_1+\boldsymbol{Q}_2+Q_3+Q_4)+FG\boldsymbol{H}\boldsymbol{Q}_1Q_2\boldsymbol{Q}_3\boldsymbol{Q}_4$			8

Figure 5: Frankel's Boolean equations for the LGP-30.

so can another. [17, p. 635]

For the LGP-30, using a 4-bit-order-code, 16 instructions were chosen. While one instruction normally would take 17 microseconds, due to the speed of the magnetic drum, "by exercising moderate care in coding", this could be sped up by a factor of four.

> Memory locations are so spaced around a track of the drum that eight word periods elapse between the presentations of two consecutively numbered words (in particular, two consecutively obeyed instructions). Problems may be so planned that the operand word usually appears in one of the middle six of these word periods. Planning a problem in this way is called "minimum latency coding." [17, p. 638]

This minimum latency coding is yet another name for the typical magnetic drum technique appearing on the ZEBRA as "optimum coding" or on the Bendix G-15 as "minimum acces coding".

In a follow-up article, Frankel dug deeper into the question of small, general-purpose machines. In "The Logical Design of a Simple General Purpose Computer" [18], Frankel explicitly deduces the more than 40 logical Boolean equations that describe the logical structure of the LGP-30 (see figure 5). In 1958, Frankel went for the "minimum logical complexity" in his description of the M'AC (MicrocepalAC) [19], whose informal description resembles a Turing machine:

> The computer described here is designated M'AC (from MicrocephalAC). Its memory organ is a magnetic tape which presents and receives information in several channels. Each of two channels is served by two heads spaced by integral multiples of the distance corresponding to a digit period. One of these channels acts as a circulating register: the first of its heads records a bit in each digit period, the second (in the direction of tape motion) presents to the computer proper the bit which was recorded in a correspondingly earlier digit period. The other of these channels holds the main memory. [19, p. 283]

In spite of the stored-program idea, in actual computers, storage and working memory are neatly separated, as are numerical and programming data. For his theoretical 'paper' machine M'AC, however, Frankel used one and the same tape both as register and as memory, though they are separable because of their numerical spacing.

Pursuing his analysis, Frankel noted that three distinguishable operations suffice for general-purposeness:

> The elementary operations of M'A C are intended to provide a minimum set of operations into which the activities of a computation can be broken. [...] The basic set of operations required for a gpc thus appears to be 1) subtract, 2) record in memory, and 3) branch. It is not necessary, however, that these operations have separate orders. In M'A C each instruction execution accomplishes all three of these operations – a number read from memory (by head M) is subtracted from that held in the circulating register, the result is simultaneously recorded in memory (by head M*), and the next instruction is then read either from M or from M*, depending on the outcome of the subtraction. [19, p. 284]

Pushing three instruction into one, Frankel established one kind of minimal machine, the one-instruction general-purpose machine that can be described by a mere seven logical equations (see figure 6). But this machine is hardly practical, because, as Frankel remarked, "the simplification of reducing the order list to one item is obtained at the cost of less efficient use of memory capacity and greater complexity of the programmer's task."

5.3 The TX-0 at Lincoln Lab

At MIT's Lincoln Laboratory, another kind of small computer was developed, called the TX-0 (1956-1958). The Semi-Automatic Ground Environment (SAGE) project had been the start for Lincoln Laboratory and they had developed the Whirlwind computer as its central data processor. They had also built a copy of the Whirlwind, the Memory Test Computer (MTC), for testing the newly developed ferrite-core magnetic memory technology (1953). Now, for testing yet another new and promising technology, transistors, a team at Lincoln Lab

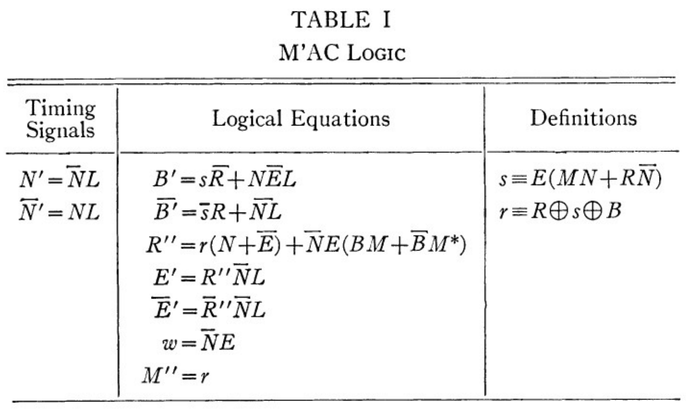

TABLE I

M'AC LOGIC

Timing Signals	Logical Equations	Definitions
$N' = \overline{N}L$	$B' = s\overline{R} + N\overline{E}L$	$s \equiv E(MN + R\overline{N})$
$\overline{N}' = NL$	$\overline{B'} = \overline{s}R + \overline{N}L$	$r \equiv R \oplus s \oplus B$
	$R'' = r(N + \overline{E}) + \overline{N}E(BM + \overline{B}M^*)$	
	$E' = R''\overline{N}L$	
	$\overline{E}' = \overline{R}''\overline{N}L$	
	$w = \overline{N}E$	
	$M'' = r$	

Figure 6: Frankel's Boolean equations for the M'AC.

started out to build a test computer before embarking on the development of a transistor-based remplacement for the Whirlwind (that would ultimately become the TX-2). This test computer was the TX-0, it pioneered the use of transistors but also some ideas that would only become current in the era of personal computing.

As Wesley A. Clark, the main engineer on the project, stated [8]:

> "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 [7, p. 144].

Clark's course contained six courses, each part building on the previous one:

- The Turing machine: a basic introduction to the Turing machine concept
- The universal Turing machine (2 courses)
- Boolean Algebra
- Synthesis of Boolean machines (2 courses), including a discussion on the Sheffer stroke as candidat for sole building block in designing circuits.

Clark had used Moore's 1952 article as the main source for the part on Turing machines and his small universal machine was explained in detail. As Clark remarked, this universal machine constitutes a "critical complexity beyond which no further increase in generality can be guaranteed!" [6, p. 13] During the course, Clark followed Moore's encoding and construction, but mashed the three tapes back into one tape using a specific encoding scheme.

The part on Boolean logic, apart from the obvious reference to Shannon, relied mostly on work done by Richard C. Jeffrey and Irving S. Reed at Lincoln Lab. The connection between Turing machines and Boolean logic was quite essential, the Boolean logic was 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 the rules for printing symbols define a Boolean algebra" [6, p. 26]

The course ends with the development of minimal circuits for encoding and for cyclic counting.

It is remarkable that this group of engineers familiarised itself with rather advanced techniques from mathematical logic to develop a transistor-based computer. How much of the logical minimalism taught in Clark's course found its way into the TX-0's design is difficult to evaluate, but it surely was important in their development of minimal modules (such as flip-flops) that could be standardised for use in building the TX-0 and TX-2. As Kenneth Olsen, who did the circuit design, remarked, "circuits which are repeated often were designed with as few components as possible." [35, p. 100] And because transistors were physically better behaved than vacuum tubes, they could be formalised more easily "transistors also can give improvements in speed and tolerance to parameter variations, and that they lend themselves to standardized building blocks." [35, p. 98]

As was the case with the other small machines, the minimal architecture of the TX-0 entailed a rich unfolding of programming practices. The TX-0 itself only had a small instruction set of four instructions:

1. sto x, place the value of the accumulator in the register
2. add x, add the value in the register to accumulator
3. trn x, if the value in the accumulator is negative, take next instruction from register x, if positive, go to the next instruction
4. opr x, execute an operate class command

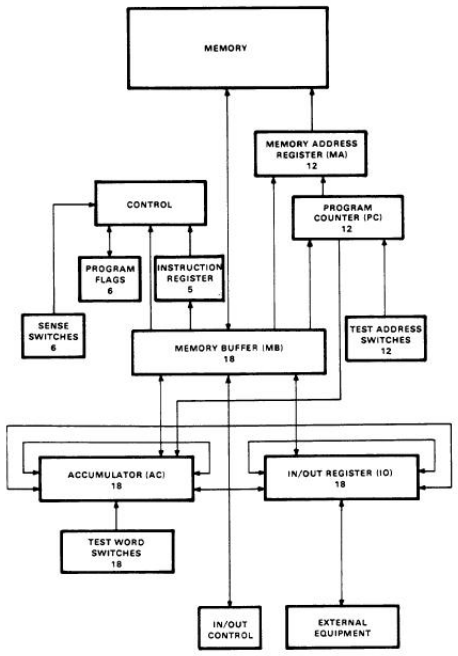

Figure 7: A block diagram of the general structure of the PDP-1. The triangle of Accumulator, Memory Buffer and In/Out register echoes the TX-0 design.

The fourth `opr` command triggered an elaborate vocabulary of 'class commands', viz. the operands of the `opr` command are actually special bit-encoded instructions. Through the class commands, a wide rang of specially wired micro-programs could be addressed. Because of its processing speed (thanks to the transistors) and its large and fast memory (thanks to the ferrite core memory), the TX-0 was much faster than the other small machines discussed here (Bendix G-15, LGP-30 and ZEBRA), all using tubes and a much slower magnetic drum. This made it worthwhile to not only have a potent interpretive routine (as had the Bendix G-15 and the ZEBRA), but also interactive command-line like features. A versatile symbolic assembler language and a powerful interpretative routine, the Direct Input routine, were programmed for the TX-0 along with many utility routines. Moreover, the TX-0 possessed a range of interactive possibilities: a flexowriter, a cathode ray tube, and later, a light pen. In combination with the interpretive routines, this made for an interactive system.

When DEC would later develop its PDP-1, the first minicomputer, it would take its cue from the TX-0 design [1, pp. 124-128]. Ben Gurley, the head engineer, had worked on the Whirlwind and its successors, and now transported his know-how to the PDP-1. The idea of standardized building blocks was pursued, but also parts of the general design. While the number of registers was reduced, they kept the core of three registers that pulse the flow of information: an accumulator, a memory buffer and and an I/O register (called live register on the TX-0). The instruction set was, as with the other commercial versions of small computers, expanded. Instead of just four, 28 instructions would be included, plus a great number of microprograms known as the “Operate Group”.

6 Discussion

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. 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. Within computing itself, many people have pointed out the multidisciplinary nature of computing and have invested a lot of work into bringing together different approaches (see, e.g., the final ACM/IEEE committee report on the core of computer science [16], cf. with [57].) 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.

Seen from this angle, the Turing machine has often been hailed as a most practical outcome of the foundational debates of the early 20th century, the positive face of the negative Entscheidungs-result. But as we have shown here, it also fits in a tradition of logical minimalism: the search for a minimum of operations, of axioms, of length of propositions etc. This facet of logical research proved to be quite useful in the early days of computing when the results from mathematical logic were ported to computer design and programming. Such a transformation was pioneered by people like Curry, von Neumann and Turing, but was further exploited in the 1950s leading to commercial computer designs such as the STANTEC ZEBRA, Bendix G-15 or LGP-30. There also was a reverse influence: the (informal) fact that four elementary instructions suffice to compute anything computable, that nowadays belongs to the lore of theoretical computer science, is constituted during this encounter between logic and engineering in the 1950s.

After 1960, the search for an optimum in combining a minimum of elements to have a general-purpose computer became less pressing, due to the arrival of new, very scalable technologies (especially transistors) and cheaper and/or faster memories. However, the topic would remain quite important in theoretical computer science, mostly because it provided a simple model (such as the register machines) for theoretical research on computers. More even, triggered by Minsky’s results on constructing small universal Turing machines, a kind of competition on the smallest machine developed during the 1960s (cf. the survey in [67]). Of course, the question of simplicity would remain relevant for computer engineers. For instance, in the 1980s there was a debate between the RISC (Reduced Instruction Set) and CISC (Complex Instruction Set) philosophies for developing microchips, where the RISC proponents sought to minimize the instruction set to achieve higher effiency.

Though hardly any detail of Turing's original construction of a universal machine made its way into an actual computer, the very idea that 1) such a thing as a universal machine exists, and 2) it does not need more than 4 or 5 instructions, was very important to this select group of engineers. It helped them to articulate the essence of the 'general-purpose' character of the digital computer. They all remarked that there was no need to complexify computer architecture beyond a certain point to make it more general-purpose, actually, quite a simple machine already sufficed. Although both van der Poel and Frankel deduced a one-instruction machine (be it in quite different ways!), this was merely a theoretical game, proving a lower bound on the set of instructions. More important was the question of van der Poel of finding an optimum, since every simplification of hardware brought on a complexification of software. There was more or less an implicit consensus around Turing's number of four instructions, certainly including store, add (or subtract) and a conditional transfer. In practice, most computers actually built rather had 16 instructions (Bendix G-15, LGP-30, ZEBRA) or a special order addressing tens of microprograms (TX-0).

In the process of porting a rather theoretical paper machine to actual computers, the tradition of using Boolean logic in developing switching circuits proved to be important. Although Shannon's methods did not immediately and easily extend to design complex machines such as computers, progress was made. The Boolean equations used at Northrop (used by Frankel) or Jeffrey and Reed's algebra (used in the TX-0) helped to translate a theoretical model into actual, minimal circuits. Further, the microprogramming hardware technique was important to implement in a flexible way the minimal machine instruction set, both on the Bendix G-15 and on the TX-0. On a still more concrete level, all machines needed a capable engineer to make them work. Huskey, though himself a capable engineer, had the help of Robert Beck; Frankel was helped by James Cass; and at Lincoln Lab a number of engineers, such as Wes Clark, Ken Olsen or Ben Gurley were working on the TX-0.

The minimalist philosophy in computer architecture necessarily had 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 had to find a compromise between different trade-offs. A simple logical structure of the computer asks for extensive programming possibilities and, if possible, more (and faster) memory. In the practice of the 1950s, this meant magnetic drum memories, a bit slow, but reliable and cheap. Because, together with the small instruction set, this might cripple the processing speed of the machines, special programming techniques such as optimum or minimum latency coding were developed. But for finding a commercial market for these machines, this was not enough. Van der Poel developed his Simple Code, user groups developed libraries of subroutines for the Bendix G-15 and the LGP-30. Especially Harry Huskey was very active in developing interpretative schemes such as the Intercom system to make the G-15 computer more accessible for a variety of users that did not necessarily wanted to wade through the intricacies of machine coding. The TX-0, as the expensive experimental machine it was, had the luxury of lots of fast memory. Therefore, a new kind of programming style altogether could be developed, an interpreted command line with many utility routines and some interactive possibilities at the fingers of the programmer.

Nowadays one can find claims on the internet that the Bendix G-15, the LGP-30 or the TX-0 were the first personal computers, but this is, of course, anachronistic. These computers did provide blueprints for the architecture of minicomputers in the 1960s. The TX-0 inspired the PDP-1 design, and the Bendix G-15, through its family member, the Packard Bell 250, had an influence on the first SDS minicomputer. But none of these small computers from the 1950s are based on a microprocessor and none was developed for a mass market of personal users. On the contrary, the market these small computers addressed were the smaller businesses and universities that could not afford the bigger computers. They were also often used, even by bigger institutes and companies, as a cheap and flexible data-processing solution for handling communication with special-purpose machines. And these computers were rather successful at that too if one looks at the sales numbers they achieved.

Acknowledgments

The authors would like to thank the audience at Computability in Europe 2012, Cambridge University, where an early version of this paper was presented, and the people present during a later presentation at a workshop ‘Beyond ENIAC. Early platforms and practices’ at Siegen University (2016). Thomas Haigh, Mark Priestley, Peter Denning, and Pierre Mounier-Kuhn provided particular helpful comments as did an anonymous reviewer of the Annals. Part of this research was supported by the ANR project PROGRAMme, ANR-17-CE38-0003-01.

The section on Willem van der Poel’s work builds on Chapter 3 in Edgar Daylight’s “Turing Tales,” (Lonely Scholar, 2016), which is based on a translated and modified version of part of Daylight’s contribution in “De geest van de computer: Een geschiedenis van software in Nederland,” (Uitgeverij Matrijs, 2016). All translations of Van der Poel’s Dutch quotes in [41, 45] have been taken from [15, p. 49-58]. Special thanks in this regard go to fellow Dutch historians: Eric Berkers, Harry Lintsen, and Gerard Alberts. The latter provided multiple source materials pertaining to Willem van der Poel.

References

[1] C. Gordon Bell, C.J. Mudge and J.E. McNamara. Computer Engineering, A DEC view of hardware systems design. Maynard, Digital Press (1978).

[2] A. van den Bogaard. Stijlen van programmeren 1952-1972. Studium, 2:128-144, 2008.

[3] Nicolaas G. de Bruijn. Verslag inzake onderzoek betreffende electronische en electrische rekenapparatuur over het cursusjaar 1947/48. Technical report, Delft, May 1948.

[4] Colin B. Burke. It Wasn't All Magic: The Early Struggle to Automate Cryptanalysis, 1930s – 1960s. United States Cryptologic History, Special Series, Volume 6. Fort Meade (MD), NSA, 2002.

[5] M. Campbell-Kelly. Alan Turing's other universal machine: Reactions on the Turing ACE computer and its infuence. Communications of the ACM, 55(7):31-33, 2012.

[6] Wesley A. Clark. The logical structure of digital computers. Technical report, Course notes, Division 6 Lincoln Laboratory, MIT, 1955.

[7] Wesley A. Clark. The Lincoln TX-2 computer development. Proceedings WJCC, pages 43-145, 1957.

[8] Wesley A. Clark. Oral history interview by Judy E. O'Neill, 3 May 1990. Charles Babbage Institute, University of Minnesota, Minneapolis.

[9] Jack Copeland (ed.). Alan Turing's Automatic Computing Engine. Oxford, OUP, 2005.

[10] Haskell B. Curry. The combinatory foundations of mathematical logic. The Journal of Symbolic Logic, 7(2):49-64, 1942.

[11] Haskell B. Curry. A program composition technique as applied to inverse interpolation. Technical Report 10337, Naval Ordnance Laboratory, 1950.

[12] Martin Davis. Engines of Logic: Mathematicians and the Origin of the Computer. W.W. Norton and Company, New York, 2001.

[13] Edgar 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.

[14] Edgar G. Daylight. Towards a historical notion of 'Turing – the father of computer science'. History and Philosophy of Logic, 36 (3), pp. 205-228, 2015.

[15] Edgar G. Daylight. Turing Tales. Geel: Lonely Scholar.

[16] Peter Denning, Douglas E. Comer, David Gries et al. Computing as a Discipline. Communication of the ACM, 32 (1), pp. 9-23, 1989.

[17] Stanley P. Frankel. Useful applications of a magnetic-drum computer, Electrical Engineering, 75 (7), 634-639 (1956).

[18] Stanley P. Frankel. The Logical Design of a Simple General Purpose Computer, IRE Transactions on Electronic Computers, 6 (1), pp. 5-14, 1957.

[19] Stanley P. Frankel. On the Minimum Logical Complexity Required for a General Purpose Computer, IRE Transactions on Electronic Computers, 7 (4), pp. 282-285, 1958.

[20] 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.

[21] Ovor Grattan-Guinness. The search for mathematical roots, 1870–1940 : logics, set theories and the foundations of mathematics from Cantor through Russell to Gödel. Cambridge, CUP (2000).

[22] Thomas Haigh, Mark Priestley and Crispin Rope. Reconsidering the Stored Program Concept. IEEE Annals of the History of Computing, 36:1 pp. 4-17, 2014.

[23] Hans Hermes. Die Universalität programmgesteuerter Rechenmaschinen. Math.-Phys. Semsterberichte (Göttingen) 4, pp. 42-53 (1954).

[24] Andrew Hodges. Alan M. Turing. The enigma. Burnett Books, London, 1983. Republication (1992), 2nd edition, Vintage, London.

[25] Harry D. Huskey. The state of the art in electronic digital computing in Britain and the United States (1947). In [9], chapter 23, pp. 529-540.

[26] Harry D. Huskey. Binary Digital computer with Magnetic Drum Storage. US Patent 2,289,472, filed May 2, 1955, granted May 2, 1961.

[27] Harry D. Huskey. From ACE to the G-15. Annals of the History of Computing, 6(4), pp. 350-371, 1984.

[28] Stephen C. Kleene. Representation of events in nerve nets and finite automata. RAND Memorandum 704 (1951). Partially reprinted as: Representation of events in nerve nets and finite automata, in Automata Studies, ed. by C.E. Shannon and J. McCarthy, Princeton, pp.3–42 (1956).

[29] Edna Kranakis. Early Computers in The Netherlands. CWI-Quarterly, 1 (4), pp. 61-84, 1988.

[30] W. McCulloch, W. and W. Pitts, W. A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, 5, pp. 115133 (1943).

[31] Michael Sean Mahoney. The histories of computing(s). Interdisciplinary Science Reviews, 30 (2), pp. 119-132.

[32] Michael Sean Mahoney. Histories of Computing, ed. by Thomas Haigh. Cambridge, Mass.: MIT Press, 2011.

[33] S. Millman. A History of Engineering and Science in the Bell System: Communications Sciences (1925-1980). AT & T, 1984.

[34] Marvin Minsky. Recursive Unsolvability of Post's Problem of 'Tag' and Other Topics in Theory of Turing Machines, Annals of Mathematics, 74 (3), pp. 437-455 (1961).

[35] J.L. Mitchell and K.H. Olsen: TX-0: A Transistor Computer. AFIPS Conferece Proceedings EJCC 10, pp. 93-101, 1956.

[36] Liesbeth De Mol, Martin Carlé et Maarten Bullynck, Haskell before Haskell. An alternative lesson in practical logics of the ENIAC, Journal for Logic and Computation, 25 (4), pp. 1011-1046, 2014.

[37] Edward F. Moore. A simplified universal Turing machine. In Proceedings of the meeting of the ACM, Toronto Sept. 8 1952, pages 50-54, 1952.

[38] Edward F. Moore and Claude E. Shannon. Electrical circuit analyzer. US Patent 2,776,405., 1953. Filed May 18, 1953, granted January 1, 1957.

[39] Robina Mapstone. Interview with Max Palevsky, February 15, 1973. Computer Oral History Collection. Repository: Archives Center, National Museum of American History.

[40] David. P. Perry. Minimum Acces Coding. Mathematical Tables and other Aids to Computation, 6 (39), pp. 172-182, 1952.

[41] Willem L. van der Poel. Inzending 1946/47 van Van der Poel op de prijsvraag genaamd "1+1=10". Technical report, Delft, 1948.

[42] Willem L. van der Poel. A simple electronic digital computer'. Applied Scientific Research Section B, 2:367-400, 1952.

[43] Willem L. van der Poel. The Logical Principles of Some Simple Computers. PhD thesis, Universiteit van Amsterdam, February 1956.

[44] Willem L. van der Poel. Digitale Informationswandler, chapter Microprogramming and trickology, pages 269-311. Braunschweig: Vieweg, 1961.

[45] Willem L. van der Poel. Een leven met computers. TU Delft, October 1988.

[46] Emil L. Post. Finite Combinatory Processes – Formulation I. Journal of Symbolic Logic, 1 (3) pp. 103-105, 1936.

[47] Emil L. Post. Absolutely unsolvable problems and relatively undecidable propositions- Account of an anticipation. In: Martin Davis (ed.), The undecidable. Basic papers on undecidable propositions, unsolvable problems and computable functions, Raven Press, New York, 1965, pp. 340433.

[48] Willard V. Quine.The Problem of Simplifying Truth Functions, American Mathematical Monthly, vol. 59, pp. 521-531, 1952.

[49] Moses Schönfinkel. Uber 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.

[50] Claude E. Shannon. A symbolic analysis of relay and switching circuits. Transactions of the American Institute of Electrical Engineers, Vol. 57, 1938.

[51] Claude E. Shannon. The Synthesis of Two Terminal Switching Circuits, Bell System Technical Journal, 28 (1), pp. 59-98 (1949, but written 1940).

[52] Claude 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.

[53] H.M. Sheffer. A set of five independent postulates for Boolean algebras, Transactions of the AMS, 14, pp. 481-488 (1913).

[54] John C. Shepherdson and H. E. Sturgis. Computability of Recursive Functions, Journal of the ACM, 10, pp 217-255 (1963).

[55] Richard Sprague, A Western View of Computer History. Communications of the ACM, 15 (7), pp. 686-692, 1972. 691-692.

[56] John Stringer. Microprogramming and the choice of order code. Automatic Digital Computation, Proceedings of a symposium held at the National Physical Laboratory March 1953, pp. 71-74.

[57] Matti Tedre. The Science of Computing. Shaping of a Discipline. Boca Rato: CRC, 2015.

[58] Alan M. Turing. On computable numbers with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, 42:230-265, 1936. A correction to the paper was published in the same journal, vol. 43, 1937, 544-546.

[59] 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.

[60] Standard Telephones and Cables Ltd. An Outline of the Functional Design of the Stantec Zebra Computer. Newport, 1958.

[61] Radomir S. Stankovic, Jaakko Astola (eds.). From Boolean Logic to Switching Circuits and Automata. Towards Modern Information Technology. Springer, 2011

[62] C.J.D.M. Verhagen. Rekenmachines in Delft. Uitgave van de Commissie Rekenmachines van de Technische Hogeschool te Delft, 1960.

[63] Hao Wang. A variant to Turing's theory of computing machines. Journal of the ACM, 4(1):63-92, 1957.

[64] Alfred North Whitehead and Bertrand Russell, Principia mathematica, volume 1 (1st ed.), Cambridge: Cambridge University Press (1910).

[65] Maurice V. Wilkes and John B. Stringer. Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer. Proceedings of the Cambridge Philosophical Society, v. 49, pp. 230-238 (1953).

[66] Maurice V. Wilkes. Microprogramming. Proceedings of the Eastern Joint Computer Conference, December 1958, pp. 18-20.

[67] Damien Woods and Thurlough Neary. The complexity of small universal Turing machines: A survey. Theoretical Computer Science, 410(4-5), pp. 443-450, 2009