Towards a Historical Notion of "Turing — the Father of Computer Science"

Third and last draft, submitted in August 2013 to the Journal *History and Philosophy of Logic*

Edgar G. Daylight[⋆]

Eindhoven University of Technology
Department of Technology Management
egdaylight@dijkstrascry.com

Abstract. In the popular imagination, the relevance of Turing's theoretical ideas to people producing actual machines was significant and appreciated by everybody involved in computing from the moment he published his 1936 paper 'On Computable Numbers'. Careful historians are aware that this popular conception is deeply misleading. We know from previous work by Campbell-Kelly, Aspray, Akera, Olley, Priestley, Daylight, Mounier-Kuhn, and others that several computing pioneers, including Aiken, Eckert, Mauchly, and Zuse, did not depend on (let alone were they aware of) Turing's 1936 universal-machine concept. Furthermore, it is not clear whether any substance in von Neumann's celebrated 1945 'First Draft Report on the EDVAC' is influenced in any identifiable way by Turing's work. This raises the questions: (i) When does Turing enter the field? (ii) Why did the Association for Computing Machinery (ACM) honor Turing by associating his name to ACM's most prestigious award, the Turing Award? Previous authors have been rather vague about these questions, suggesting some date between 1950 and the early 1960s as the point at which Turing is retroactively integrated into the foundations of computing and associating him in some way with the movement to develop something that people call computer science. In this paper, based on detailed examination of hitherto overlooked primary sources, attempts are made to reconstruct networks of scholars and ideas prevalent to the 1950s, and to identify a specific group of `ACM` actors interested in theorizing about computations in computers and attracted to the idea of language as a frame in which to understand computation. By going back to Turing's 1936 paper and, more importantly, to recast versions of Turing's work published during the 1950s (Rosenbloom, Kleene, Markov), I identify the factors that make this group of scholars particularly interested in Turing's work and provided the original vector by which Turing became to be appreciated in retrospect as the father of computer science.

[⋆] The author, also known as Karel Van Oudheusden, thanks Nancy R. Miller and J.M. Duffin of the University of Pennsylvania Archives, and also Jos Baeten of the 'Centrum voor Wiskunde & Informatica' for funding his visit to the Archives.

1 Introduction

In August 1965, Anthony Oettinger and the rest of the Program Committee of the ACM met and proposed that an annual "National Lecture be called the Allen [sic] M. Turing Lecture" [4, p.5].[1] The decision was also made that the ACM should have an awards program. The ACM Awards Committee was formed in November 1965 [5]. After having collected information on the award procedures "in other professional societies", Lewis Clapp — chairman of the ACM Awards Committee — wrote in August 1966 that

> [a]n awards program [...] would be a fitting activity for the Association as it enhances its own image as a professional society. [...] [I]t would serve to accentuate new software techniques and theoretical contributions. [...] The award itself might be named after one of the early great luminaries in the field (for example, "The Von Neuman [sic] Award" or "The Turing Award", etc.) [6].

The mathematician Alan J. Perlis officially became ACM's first A.M. Turing Lecturer and Turing Awardee in 1966. Besides having been the first editor-in-chief of the *Communications of the ACM,* Perlis had earned his stripes in the field of programming languages during the 1950s and had been President of the ACM in the early 1960s. Perlis was thus a well-established and influential computer scientist by the mid-1960s. In retrospect, decorating Perlis was only to be expected. But why did the ACM honor Turing? Turing was not well known in computing at large in the 1960s and early 1970s [41]. Apparently, his name was preferred over John von Neumann's and Emil Post's, yet all three researchers had deceased by 1957 and all three were highly respected by some very influential actors in the ACM — including John W. Carr III, Saul Gorn, Perlis, and Oettinger.[2] And why are we celebrating Turing today? The latter question was posed repeatedly during Turing's centennial in 2012.

The ACM, founded in 1947, was an institutional response to the advent of automatic digital computers in American society. Only some of those computers, most of which were still under construction, were based on the principle of a large store containing both numbers and instructions — a principle that is also known today as the "stored program" principle.[3] That principle, in turn, paved the way for computer-programming advancements which were sought by men like Carr, Gorn, and Perlis. The Cold War was also responsible for massively funded research in machine translation (— later also called automatic language translation). The young student Oettinger at Harvard University was one of several embarking on Russian-to-English translation projects. Another example is Andrew Booth, a British computer builder who had met in 1946 with the prominent American scientist Warren Weaver to discuss the possibility of mechanically translating one language into another. In between 1946 and 1947, Booth visited several computer laboratories in the United States, including John von Neumann's lab at the Institute for Advanced Study in Princeton. By 1952, he was both an accomplished computer builder and a programmer of a mechanical dictionary. As the postwar years progressed, several computer specialists turned into applied linguists and many professional linguists became programmers [14, p.7][70, p.2,3,12,24][90, p.207]. Documenting this technological convergence leads to a better understanding of when, how, and why Turing assumed the mantle of "the father of computer science".

In this article, I describe the convergence by zooming in on the work of Booth, Carr, Gorn, and Oettinger.[4] I will show that, not until the 1950s, did computer programmers like Booth and Gorn begin to reinterpret the electronic computer in terms of the universal Turing machine. They did this with the purpose of developing higher level programming languages.[5] Turing thus assumed the mantle of "the father of computer science" for reasons that are orthogonal to the commonly held belief that he played an influential role in the design or construction of "universal computers". My historical account is primarily about the 1950s and ends with a brief discussion of Turing's 100th birthday in 2012 and with published work related to this article.

The take-away message in general terms is that the 1950s constitute a decade of cross fertilization between linguistics, computer programming, and logic. That decade is preferably *not* viewed as a smooth road from modern logic to computing; if there was any road at all in the history of computing, then it was most definitely from practice to theory.

2 Machine Translation — a Bird's Eye View

"See what you can do with your Russian" — the student Oettinger was told around 1949 by the American computer pioneer Howard Aiken at Harvard after the latter had corresponded with Weaver on the vexing topic of machine translation.[6] Weaver had directed American war work of hundreds of mathematicians in operations research. Fully aware of the developments in electronic computing machines, he had come to believe around 1946 that code-breaking technology from the war could help detect certain invariant properties that were common to all languages. Weaver had expressed his ambitious ideas in writing in his 1949 memorandum *Translation* [70, Ch.1] which, in turn, sparked intense American interest, including Aiken's interest, in the possibility of automatic language translation. In comparison with Booth's mechanical dictionary from 1952, Weaver's original idea on machine translation was more ambitious — namely to go "deeply into the structure of languages as to come down to the level where they exhibit common traits". Instead of trying to directly translate Chinese to Arabic or Russian to Portuguese, Weaver supported the idea of an indirect route: translate from the source language into an "as yet undiscovered universal language" and then translate from that language into the target language [70, p.2,3,15,23].

In the academic year 1949–1950 Oettinger started thinking about mechanizing a Russian-to-English dictionary. He also stayed at Maurice Wilkes's computing laboratory in Cambridge for a year where he met Alan Turing on a regular basis. Wilkes, in turn, also visited leading figures in computing. He regularly traveled from England to the United States where he met with Aiken, von Neumann, and many others. Apart from being an accomplished computer designer, Wilkes was also practicing and advancing the art of computer programming. In this regard, around 1952, he met Perlis and Carr who were working with Project Whirlwind at `MIT` [113, p.31]. Perlis had obtained his Ph.D. in mathematics from `MIT` in 1950. Carr had done the same in 1951 and had also spent some weeks in Wilkes's computer laboratory in England together with Oettinger. The world of computer practitioners was thus a very small one: many practitioners knew each other on a personal level and several became involved in the `ACM` [7, 113].[7]

Initially, most linguists were rather pessimistic about Weaver's memorandum, relegating his aspirations about machine translation to the realm of the impossible. Gradually, they started to see opportunities [70, p.4,137]. By 1951 the computer had made a clear mark on linguists. The Israeli linguist Bar-Hillel conveyed that message by making an analogy with chemistry. Chemists, he said, need "special books *instructing* students *how* to proceed in a fixed *sequential* order [... in their] attempted analysis of a given mixture" [14, p.158–159, my emphasis]. Likewise, special books will have to be written for the linguist, books that contain "sequential instructions for linguistic analysis, i.e., an *operational syntax*" [14, p.158–159, original emphasis]. According to the historian Janet Martin-Nielsen, American linguistics at large transformed from elicitation, recording and description before the war to theory and abstract reasoning after the war. It "rose to prominence as a strategic and independent professional discipline" [73].

Researchers knew that literal translations would yield low quality machine translation. Therefore, some of them sought methods to construct "learning organs"; that is, machines that "learn" which translation to prefer in a given context [14, p.154]. As Weaver had already put it in 1949: the "alogical elements" in natural language, such as "intuitive sense of style" and "emotional content", rendered literal translation infeasible [70, p.22].

Weaver's remarks can, *in retrospect*, be viewed as part of a grander intellectual debate in which fundamental questions were posed such as whether machines can think (cf. E.C. Berkeley's *Giant Brains, or Machines That Think* [18]). Weaver had addressed this issue optimistically in 1949. A year later, Turing's 1950 article 'Computing Machinery and Intelligence' was published [105]. It was followed up by Wilkes's 'Can Machines Think?' [112].

Weaver's memorandum was based on an appreciation for the theoretical 1943 work of McCulloch & Pitts, entitled 'A Logical Calculus of the Ideas Immanent in Nervous Activity' [74]. McCulloch and Pitts had essentially tried to find a mathematical model for the brain. Weaver described their main theorem as a "more general basis" for believing that language translation was indeed mechanizable by means of a "robot (or computer)".[8] In other words, according to Weaver there was no theoretical obstacle to machine translation: learning organs could, at least in principle, resolve the translation problem.

After leaving Aiken's lab at Harvard to temporarily join Wilkes's research team in Cambridge, Wilkes made clear to Oettinger that he had "no use for language translation". Instead, he urged Oettinger to address the question whether computers might be able to learn [90, p.208]. It is in this setting that, for Oettinger, “Alan Turing and others became familiars at meetings of the Ratio Club” [90, p.208]. Oettinger’s work on the `EDSAC` led up to his 1952 paper ‘Programming a Digital Computer to *Learn*’ [86, my emphasis], in which he targeted an audience of psychologists, neuro–physiologists, and everyone concerned with non-numerical applications of digital computers. He connected the McCulloch & Pitts’s paper to automatic digital computers and wrote that “digital computers can be made to serve as models in the study of the functions and of the structures of animal nervous systems” [86, p.1243].[9] Due to Turing’s direct influence, Oettinger used the words “universal digital computer” in his paper [86, p.1244].[10] But, although Oettinger included both Turing’s 1936 and 1950 papers in his bibliography, it was Turing’s 1950 “imitation game” that mattered most to him. Oettinger was, after all, concerned with programming a real computer, the `EDSAC`, so that it could “learn”.

After returning to Harvard, Aiken disapproved of his student’s turn towards learning organs and forced him to return to his automatic language translation project [90], which would lead up to his 1954 Ph.D. dissertation, *A Study for the Design of an Automatic Dictionary.* In the immediately following years, he dived into the needs of the milk and banking industries [7, p.6]. Oettinger’s papers on data processing [51, 87] and his comprehensive 1960 book *Automatic Language Translation* [88] — which was partially about programming the Univac I for the purpose of machine translation — did not contain any references to Turing and the like.

By 1961, Oettinger *was* referring — through Paul Rosenbloom’s 1950 book *The Elements of Mathematical Logic* — to the works of Alonzo Church, Turing, and especially Emil Post [100, Ch.IV]. As professor of Mathematical Linguistics, Oettinger was giving the bigger picture of the *converging* developments that had taken place during the 1950s. “Syntactic analysis”, he said, had received considerable attention, not only from the “mathematical linguists” (such as Noam Chomsky and himself), but also from “applied mathematicians” (such as Carr and Perlis) and “mathematical logicians”. The mathematical linguists were seeking algorithms for automatic translation among *natural* languages. The applied mathematicians (i.e. the computer programmers) were concerned with the design and translation of languages suitable for *programming* machines. The mathematical logicians, in turn, had been exploring the structure of formal *artificial* languages [89, p.104].

By 1963, the theoretical work of Post, Turing and several others had become common currency among some influential academics in both mathematical linguistics and in computing, particularly in automata theory and automatic programming. The professional linguist Bar-Hillel had already expressed his appreciation for “recursive function theory, Post canonical systems, and the like” in 1960 [14, p.84].[11] And in the Classification System of the December 1963 issue of the Computing Reviews, “Turing Machines” was explicitly mentioned next to “Automata”.

3 Automatic Programming — a Worm's Eye View

Besides the machines housed at MIT, Harvard University, and Cambridge University, also the ENIAC, EDVAC, and IAS machines should not go unmentioned. The ENIAC and EDVAC were constructed at the Moore School of Engineering, Philadelphia, and were later housed in Maryland at Aberdeen Proving Ground — where John von Neumann was a consultant for several years [10][24, p.886]. During the second half of the 1940s, von Neumann and his team built the IAS computer at the Institute for Advanced Study, Princeton [48].

During the early 1950s, Andrew Booth was, together with his wife Kathleen, writing a book entitled *Automatic Digital Calculators* (cf [20]). In the preface of their first edition, published in 1953, the authors explicitly acknowledged "their indebtedness to John von Neumann and his staff" at Princeton for "the stimulating period spent as the guests of their Project" [20, p.vii]. Reflecting on the history of computer building, the authors referred to several people, including Leibniz, Pascal, Babbage, Jacquard, and Hollerith. No mention was made of Turing throughout the whole book, except for a brief reference to the ACE computer which had been "under the direction of Womersley, Turing and Colebrook" [20, p.16].[12]

Moreover, in the section "The universal machine", the authors referred to "the Analytical Engine" of Babbage and essentially described him as the father of the universal computer.[13] In this connection, they mentioned the machines built by Howard Aiken (1937–1944) and at Bell labs (1938–1940, 1944), describing them as "universal" and "general purpose". More specifically, the authors distinguished between a special purpose and a general purpose machine in the following manner. A "special purpose machine" was constructed to perform one set of operations, to solve one particular problem. For example, a computer that can only compute Income Tax (for different sets of input) was special purpose. General purpose computers, by contrast, were "capable of being set up to solve any problem amenable to treatment by the rules of arithmetic" [20, p.1]. The authors also clarified their notion of "universality" by noting that multiplication and addition can be defined in terms of subtraction; therefore:

> Whatever type of computing machine is projected, so long as it is to have the attribute of *universality*, it must necessarily have in its structure some component capable of performing at least the most elementary operation of arithmetic — subtraction. [20, p.22, my emphasis]

It was due to the "expense" and "difficulty" of incorporating *several* electrical or electronic components that "most modern general purpose computers" did not include square root units, dividers, and — in some cases — even multipliers [20, p.3].[14]

Some of the first computers, like the ENIAC, were — at least initially — provided with a plugging system "so that the various units [could] be connected together and sequenced to suit the particular problem to be solved." [20, p.14]. Some of the later machines of the 1940s and early 1950s, like the EDSAC and the EDVAC, were based on the principle of a large store containing both numbers and instructions. The authors established an epistemological relationship between Babbage's work and the principle of a large store.[15] Furthermore, from the authors' perspective, the incentive to use a large store was primarily an engineering one.

> When ENIAC was under construction, von Neumann and his co-workers devoted considerable attention to the problem of the *optimum* form of a calculating machine. As a result of this it was shown that a calculating machine should have a storage capacity of the order of 1,000 [8,000][16] words and that, *given this*, both operating instructions and pure numbers could be stored in the same unit. [20, p.15, my emphasis]

Choosing to store both "operating instructions" and "pure numbers" in the same store — as opposed to storing each in a separate store — was based on practical concerns, not theoretical reasoning.[17] Likewise, the incentive among many computer builders to list all instructions consecutively in one part of the store and the pure numbers in another part (of the same store) was, according to the Booths, based on the practical concerns of "simple process control" and economic use of "memory capacity" [20, p.15, 23–24].[18]

Common storage of numbers and instructions inside the computer opened the door for new programming techniques. And, the realization that many computations can be reduced to iterative processes made it "most desirable" that data could be *erased* in *any* given memory location and be replaced by new material [20, p.23, my emphasis] — a topic which I shall return to shortly.[19]

Common storage was, however, not a prerequisite to embark on programming. Some people, like Konrad Zuse, stored commands apart from numbers (cf. [53, p.76]). Similarly, Haskell Curry made a sharp conceptual distinction between commands and pure numbers [77]. Another important name in this regard is Aiken, as the following words from Grace Hopper in 1978 indicate:

> Aiken was totally correct in saying that the programs and the data should be in separate memories, and we're just discovering that all over again in order to protect information. [111, p.21]

Indeed, in practice today, most software never treats data as code [49].[20]

Space Cadets

Specifying the orders to accomplish a computation was a tedious task. Relieving the programmer from that task was the job of the "space cadets" — a name that Carr gave in 1953 to those programmers who wanted to invest time and money in building interpreters and compilers [113, p.210–211]. Carr and his space cadets (including Booth, Gorn, Perlis, and Wilkes) advocated automatic programming. They wanted to design "pseudo codes" — "each order of which will, in general, represent a number of orders in the true machine code" [20, p.212].

In my eyes, Carr stood out in the 1954 'Symposium on Automatic Programming for Digital Computers' [91] in that he consistently and repeatedly described *both* the true machine code and the pseudo code as a "language".[21] Carr viewed automatic programming as the problem of closing the "gap" between the "external decimal language" and the "intermediate binary language", of recognizing a "dual language system" and the need to program the "translation between the two languages" [22, p.85].[22] Finally, the "lack of compatibility" between existing digital computers was the incentive to search for a "universal language" — an idea that was mostly, although not solely, Gorn's [22, p.89][85].

Indeed, it was Carr's later good friend, Saul Gorn, who — in the eyes of Grace Hopper — stole the show at the 1954 symposium on automatic programming. "Looking forward to what Dr. Gorn will tell us", Hopper expressed her curiosity in her opening address towards the development of a "universal code" that can be taught to mathematicians and that *each* computer installation can provide by means of an interpreter or compiler [91, p.4–5]. To counter the proliferation of "specialized codes", Gorn stressed in his talk the need for "a code more or less independent of the machine", so that the "artists, scientists, and professions can then return to a common language and creative thinking" [53, p.74–75].[23]

During the first half of the 1950s, Gorn worked at Aberdeen Proving Ground where three "general purpose machines" were in use simultaneously: the ENIAC, EDVAC, and ORDVAC. This diversity in machinery prompted a "universal coding" experiment. And so Gorn developed a "general purpose common pseudo-code" that could serve as input to both the EDVAC and the ORDVAC [56, p.6].

Loop Control

At the symposium, Gorn enumerated four machine requirements that had to be met in order to provide for a universal code, and he did this by referring to Carr's prior research on loop controlled machines — research that I have yet to come across in primary sources. Gorn's account, reproduced below, serves to back up my conjecture that up and till around 1954 Carr and Gorn did *not* view loop controlled computers as practical realizations of universal Turing machines.

The first machine requirement to obtain loop control, Gorn said, was due to the universal code being "translatable by the machine into the machine language". Therefore, the machine had to be able to "construct and adjust its own commands".[24] To do the latter, "[w]e will therefore assume that commands are stored internally just like other words, so that the only way in which a word is recognized as a command is that it reaches the control circuits through the sequencing of previous commands. (In the terminology of J.W. Carr III, we are requiring a Type II Variable Instruction Machine.)".[25] In addition to common storage, the machine should also have "the other two properties described by Carr as necessary for 'loop control', namely that they be able to transfer control out of sequences, and that they be able to make decisions". Finally, as a fourth requirement, the machine should be capable of carrying out elementary commands, such as "adding, multiplying, subtracting, extracting a digit, taking absolute values, taking negatives, shifting digits, and, most elementary of all, stopping, reading, writing, and generally transferring information from one place to another."[26] To do this, the machine should have "one or more forms of *erasable* internal storage" [53, p.76–77, my emphasis].

In later years, machines were simply said to have "loop control" if they possessed three properties: "common storage, erasable storage, and discrimination" [55, p.254].[27] These properties allowed recursive problems to be programmed iteratively instead of being programmed as straight lines of code. For example, instead of coding the summation $S = \sum_{i=1}^{n} x_i$ as a straight line of code — as in
$s_1 = x_1$
$s_2 = s_1 + x_2$
$s_3 = s_2 + x_3$
$\ldots$
$s_{n-1} = s_{n-2} + x_{n-1}$
$s_n = s_{n-1} + x_n$
$S = s_n$
— loop control allowed the summation S to be programmed iteratively on the basis of the following three equations
$s_1 = x_1,$
$s_{i+1} = s_i + x_{i+1}$ $(i = 1, 2, \ldots, n-1),$
$S = s_n.$
The sequence of values $i = 1, 2, \ldots, n-1$ indicates that $s_{i+1} = s_i + x_{i+1}$ had to be performed for each value of i from 1 through $n-1$. The crux was that only one computer instruction was stored to correspond to $s_{i+1} = s_i + x_{i+1}$. The machine *modified* the addition instruction to perform the addition over and over again the required number of times. The machine furthermore decided, all by itself, when it had performed the addition the proper number of times [58, p.2-46]. In other words, the programmer did not have to specify up front the number of times the addition had to be carried through; instead, the "thinking" machine — using Berkeley's terminology [19, p.5] — could do this autonomously.[28]

Kleene's *Introduction to Metamathematics*

Practically oriented as they were, Carr and Gorn were also theoretically inclined, trying hard to make connections between their loop controlled machines and mathematical logic. One book that clearly had an impact on their thinking was Kleene's *Introduction to Metamathematics* [64]. According to Carr, automatic programmers had to deal with "the generation of systems rather than the systems themselves", and with "the 'generation' of algorithms by other algorithms", and hence with concepts akin to metamathematics [22, p.89]. Gorn, in turn, described the machine requirement of common storage by saying that "the language of the machine includes its own syntax, so that not only may the machine be directed to tell us something, but it may also be directed to tell us something about the way in which it tells us something". For example, the machine could, besides generating the output of a calculation, *also* print out "the storage locations chosen for the key commands and variables", and, by doing so, tell the programmer something about the way it computes (i.e., the way in which it tells something). Gorn linked this to the "quasi-paradoxical properties, as revealed in the researches of Gödel" [53, p.75–76].

Gorn continued appropriating ideas from metamathematics in a 1955 technical report, which he distributed to several colleagues, including Carr, Perlis, Juncosa, Young, Herman Goldstine, and the logician Barkley Rosser [54]. On the one hand, Gorn remarked, every computational procedure by an automatic machine "must follow the steps of a constructive proof".[29] On the other hand, Gorn argued, "every constructive existence proof, with a few slight modifications, provide[s] a possible method of solution. (What we have just said is an intuitive way of stating 'Turing's Thesis'. See reference [64])".[30]

Later on in his report, Gorn described an "ideal general purpose machine" as one that is

> effectively equivalent to a 'universal Turing machine.' (See reference [103]. The universal Turing machine can copy the description of any special purpose Turing machine and imitate its operation by means of these copied specifications.)

By referring to a universal Turing machine, Gorn made clear that, to him in 1955, an "ideal" machine had unlimited storage capacity. Existing machines were, however, not ideal in that they could not store exact real numbers but only finite approximations of them. This deficiency, Gorn noted, could, without additional precaution on behalf of the programmer, easily result in computing errors.[31]

To summarize, by 1955, Carr and Gorn *were*, at times, viewing a universal Turing machine as a conceptual abstraction of a loop controlled computer.

"General-Purpose Machines"

In a 1956 booklet *The electronic brain and what it can do* [57], Gorn distinguished between "older machines" in which "switches were set externally" and manually, and the newer "four-address machines" like the `EDVAC` in which "each order contains a part that automatically sets the switches for the carrying of the next order". The bottom line was never having to plug or unplug a single wire ever again. Because orders were coded as numbers, these newer machines gained "a flexibility and power that were undreamed of only a few short years ago" [57, p.42, 58].

A year later, Gorn wrote a letter to his good friend Perlis who, as editor-in-chief of the *Communications of the ACM*, published the letter on the first page of the very first issue of the magazine [93]. In that letter, Gorn continued reflecting on `EDVAC`'s design which, he said, had "introduced the major step forward" in computer history "of having a common variable storage of instructions and data" [93, p.2].[32] The "manipulation of hardware" had been replaced by the "programming of coded symbols".

In his letter to Perlis, Gorn also recalled how "it has been recognized that all general purpose machines, from Edvac on, are essentially equivalent, any one being capable of the same end results as any other, with varying degrees of efficiency".[33] This recognition did not happen overnight, nor did it become common knowledge to all researchers in automatic programming.[34] It was few men like Carr and Gorn who repeatedly, during the 1950s, tried to see the metamathematical — and in their eyes, grander — picture of their practical accomplishments.

In the same spirit, Gorn noted that two fundamental principles had been recognized in recent years.[35] First, that hardware and programming were within-limits interchangeable. Second, that "our general purpose machines" are equivalent with "a certain, as yet ill-defined, universal command language". During the late 1950s, Gorn, Carr, and Perlis played leading roles in the USA in the development of the universal programming language `ALGOL 60` (cf. [83]).[36]

Gorn and Carr were engaged with Burks's research agenda on cellular automata, and with the overarching theme of *thinking* and *learning* machines. The "mathematical machine of the future", Carr wrote in 1959, "may well be an *even more general* purpose machine" than what we have today [28, p.12, my emphasis]. Carr's dream was to achieve "Completely Automatic Programming". This meant that, once a proper algorithm was developed, it should be possible for the machine to decide on its own machine code. The machine of the future "should be allowed to engineer its own construction", in contrast to present "fixed code machines" that are "not self-encoding" [28, p.11].

Emil Post

Researchers in the Soviet Union were also interested in the generation of algorithms by other algorithms; that is, in the design of "compiler-producing compilers". Versed in Russian, and having visited several computer laboratories in the Soviet Union, Carr was aware of the Russian state of the art. Particularly impressed by Markov's 1954 book *Theory of Algorithms* [72], Carr described Markov's "language" as something that "could be used directly as an input to compilers which are to perform symbol manipulation (such as compiler-producing compilers)". Moreover, Carr mentioned that several Russian papers [61, 63, 98] discussed the "extension of such language application to [. . .] other indirect reasoning processes [such] as theorem proving, *natural language translation*, and complex decision making" [28, p.190, my emphasis].

Responsive to the books of Rosenbloom and Markov (and to the Russian literature in general), Carr became attracted to the strong underlying kinship between Perlis's work on automatic programming and Chomsky's research in mathematical linguistics [29–31]. The kinship was due in no small part to Post whose work, as Carr observed, had influenced Perlis via Markov and Chomsky via Rosenbloom [28, p.230, 259].[37]

The fields of automatic programming and machine translation were *converging*. Perlis's "string language manipulator", for example, was proposed to be used not only with "symbolic operational programs", but also with "algebraic language compiler-translators" and "natural language translators" [28, p.230]. Likewise, Carr noted, the construction of automatic dictionaries had "already begun" with "algebraic languages as Fortran, Unicode, and GP at one end, and with the language translation experiment at the other [end]" [28, p.259–260].

Alan Turing

In 1959, Carr used the notion of a "Universal Turing Machine" to explain a fundamental insight about automatic programming: one can "simulate a new not-yet produced digital computer" on an older computer, provided that there is sufficient storage for the simulation. The first such situation, Carr reflected, "was the preparation of an interpretative program simulating the `IBM 704` on the `IBM 701` before the former was completed" [28, p.236, 239].

Much in line with Carr and Gorn's 1954 reception of metamathematics and self-referential arguments in particular, Carr elaborated on the idea of having a computer simulate *itself*.

> If one universal machine can simulate any other machine of a somewhat smaller storage capacity (which is what Turing's statement on universal machines means), it should therefore be possible for a computer to simulate a version of itself with a smaller amount of storage.

The practical implication of self simulation, Carr continued, is that it allows programs to be run that "would perform in the standard fashion, and at the same time print out pertinent information such as location, instruction, previous contents of [the accumulator], and the previous contents of the address of the instruction" [28, p.240].

Insights into interpreters, compilers, and — what we today call — program portability was, in short, what a universal Turing machine had to offer to Carr and some of his space cadets.[38]

Also British space cadets, like Booth and Stanley Gill, used Turing's 1936 paper to paint the bigger picture of what they had been accomplishing independently of Turing, and what they were going to accomplish with Turing's theory as a road map. Booth accredited Turing as the founder of automatic programming in his opening address at the Working Conference on Automatic Programming of Digital Computers, held at Brighton in 1959.[39] It was Turing, Booth proclaimed, who "first enunciated the fundamental theorem upon which all studies of automatic programming are based" [52, p.1]. Booth continued as follows:

> In its original form the theorem was so buried in a mass of mathematical logic that most readers would find it impossible to see the wood for the trees. Simply enunciated, however, it states that any computing machine which has the minimum proper number of instructions can simulate any other computing machine, however large the instruction repertoire of the latter. All forms of automatic programming are merely embodiments of this rather simple theorem and, although from time to time we may be in some doubt as to how FORTRAN, for example, differs from MATHMATIC or the Ferranti AUTOCODE from FLOW-MATIC, it will perhaps make things rather easier to bear in mind that they are simple consequences of Turing's theorem. [52, p.1]

Booth was appropriating Turing's work in the field of automatic programming, and he most likely did this on the basis of Carr and Gorn's earlier metamathematical insights.[40]

Equally noteworthy — for the purpose of documenting Turing's scholarly legacy — is Booth's historiographical remark about Turing's 1936 paper.

> Why was it, then, that Turing's original work, finished in 1937 before any computing machine of modern type was available, assumed importance only some years after machines were in common use? The reasons, I think, stem entirely from the historical development of the subject. [52, p.2]

According to Booth, the first computing machines were used almost exclusively by their constructors and, hence, by people who were intimately aware of their internal construction. It took some years before the machines were used for scientific applications, devised by people who were and wanted to remain ignorant of the machine itself and, hence, had to rely on automatic programming techniques [41, p.24–25].

Two Metaphors

Apart from Booth's opening address, also Gill discussed some practical implications of Turing's work at the 1959 workshop in Brighton in his talk "The *Philosophy* of Programming" [my emphasis]. On the one hand, Gill noted, automatic programming can be viewed as a "language translation problem"; that is, "from the language used by the programmer into the actual instructions required by the computer". This language metaphor — still perceived as novel by many in 1954 — was a well established metaphor in 1959.[41] On the other hand, Gill continued, "*another* way of looking at automatic coding is based on the idea expressed by *Turing*"; namely that "one computer can be made to imitate or to simulate another computer" [52, p.182, my emphasis]. Turing's work, Gill continued, implies that

> an actual computer, together with a suitably designed program, [is] equivalent to another computer which may not actually exist as a separate entity. When looked at in this way, an automatic coding scheme behaves rather like the skin of an onion, which if removed, reveals another onion underneath. [52, p.182]

Gill articulated this *onion-skin metaphor* in terms of the words "hierarchies of coding schemes" and "hierarchy of programming languages". To explain this hierarchy, Gill made an analogy with a hierarchy of natural languages. An English man may, at a later stage of his life, supplement his native language with a glossary of terms related to his profession, say, nuclear physics. Then, when studying a particular topic in nuclear physics, he might further supplement his language with a list of particular symbols and definitions, and so on. Likewise, for a computer which is supplied with coding scheme No. 1, one can supplement it in turn with scheme 1a or scheme 1b, "each adapting the system in a little more detail to some particular class of problems" [52, p.186].[42]

By 1960, Gorn was preaching about *languages* and Turing *machines* all over the place. He called his research agenda "Mechanical Languages and their Translators" and he described his agenda in terms of the words: "decidable languages", "universal mechanical languages", and "universal formal mixed languages" [2, p.29].[43] Most striking (to the historian) is his *summarized* curriculum of Electrical Engineering (1959–1960) in which he used the words "Turing machine" three times(!) and, specifically, as follows:

- Digital Computers - Engineering Logic
  - Elements of number theory; Turing machines and the foundations of computation theory; [...]
- Theory of Automata
  - Advanced problems in varieties of automata, including combinatorial nets, sequential nets, and Turing machines. Languages for describing the behavior of automata. A brief account of recursive function theory, formal axiomatic systems and Goedel's theorem; the relation of these to automata and their significance for an understanding of the general nature of machine computation.
- Introduction to Digital Computers: Systems and Devices
  - Principles of mechanization of computations derived from special and universal Turing machines. General purpose computers: system organization, input-output, logic, storage, and memory devices and timing. [...] [2, p.33–38]

Clearly, to Gorn and Carr, Turing machines did not solely belong to the domain of automata theory. Turing machines were also relevant in the practical context of automatic programming. And, as we have seen, other automatic programmers, like Booth and Gill, were also delivering speeches about some *practical* implications of Turing's *theoretical* work, during the late 1950s.[44]

4 Half a Century Later

Like Booth and Gorn in the 1950s, computer scientists in the 21st century have the tendency to construct a set of "foundations" for computer science in which ideas and events that had not been understood as connected when they happened are retrospectively integrated. (This, I believe, is characteristic of scientific progress!) Subsequent generations of computer scientists then tend to assume that these things have always been related. The connection between universal Turing machines and computers is one notable example. On the other hand, critical, self-reflecting, questions *are* put into the open every now and then. For example, the question "Why are we celebrating Turing?" was asked at several Turing events in the year 2012, the centennial of Alan M. Turing. Nobody gave an historically-accurate answer however, hence my incentive to write the present paper.

From a sociological perspective, I postulate that a majority of researchers gathered together in 2012 to celebrate their common belief that theory really does come before practice: mathematics and logic are indispensable in computing; that is, after all, what von Neumann and Turing have shown us, is it not? Or, as Robinson puts it:

> We can today see that von Neumann and Turing were right in following the logical principle that precise engineering details are relatively unimportant in the essential problems of computer design and programming methodology. [99, p.12]

Many researchers, including myself, concur with the idea that one should first grasp the logical principles (of, say, a software project) before working out the "engineering" details. But I object when people — and even the community at large — rephrase history in support of their cause. Robinson, for example, *also* writes that "Turing's 1936 idea had started others thinking" and that "[b]y 1945 there were several people planning to build a universal [Turing] machine." [99, p.10]. But, in fact, most people during the 1940s did not know what a universal Turing machine was, nor had they come across Turing's 1936 paper or a recast version of his work.

Alas, also trained historians have fallen into the trap of merely asserting, rather than proving, Turing's influence on computer building. Lavington, for example, writes in his recent book *Alan Turing and his Contemporaries: Building the world's first computers*, that

> *Turning theory into practice* proved tricky, but by 1948 five UK research groups had begun to build *practical stored-program* computers. [66, p.xiii, my emphasis]
> All of the designers of early computers were entering unknown territory. They were struggling to build practical devices based on a novel abstract principle — a *universal computing machine*. It is no wonder that different groups came up with machines of different shapes and sizes, [...] [66, p.8, original emphasis]

These words contradict what ironically seems to be Lavington's main thesis: that Turing had almost no direct influence in computer building [66, Ch.8]. A similar, yet more severe, critique holds for Mahoney's entire oeuvre — *Histories of Computing* [71] — which I have presented elsewhere [43].

As the previous examples illustrate, getting Turing's legacy right isn't easy. By hardly mentioning Turing in their joint book, *Computer: A History of the Information Machine* [25], Campbell-Kelly and Aspray put Turing into the right context. Similar praise holds for Akera's prize-winning book *Calculating a Natural World: Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research* [8]. (A critical side remark here is that Akera's book is also about computing in general and, as the present article shows, Turing's work did play an important role in this more general setting.) The research of De Mol [75], Olley [92], Priestley [94, 95], Daylight [41, 45], and the promising work in progress by Haigh, Priestley, and Rope all attempt to

1. clarify what we today call the "stored program" concept, and/or
2. put Turing's and von Neumann's roles into context by explaining what these men *did* do or how their work *did* influence other computing pioneers.

Finally, although Mahoney uncritically documented Turing's legacy [43], he also presented an impressive coverage of the rise of "theoretical computer science" — a topic that, to the best of my knowledge, has so far only been scrutinized by Mounier-Kuhn [79–82]. Two findings of Mounier-Kuhn that complement the present article are, in brief, that

1. Turing's work, and modern logic in general, only gained importance in French computing during the 1960s, and

2. it is a "founding myth of theoretical computer science" to believe that the "Turing machine" was "a decisive source of inspiration for electronic computer designers" [81].

5 Closing Remarks

The existing literature is rather vague about how, why, and when Turing assumed the mantle of "the father of computer science". This article has partially filled that gap by showing that Turing's 1936 paper became increasingly relevant to the influential ACM actors Carr and Gorn around 1955. Alan Perlis's reception of Turing has yet to be documented in future work.

Carr and Gorn were inspired by re-cast versions of Church, Post, and Turing's original writings. These versions included:

- Rosenbloom's 1950 *Elements of Mathematical Logic* [100],
- Kleene's 1952 *Introduction to Metamathematics* [64], and
- Markov's 1954 *Theory of Algorithms* [72].

In the late 1950s, also British automatic programmers, including Booth and Gill, appropriated ideas from Turing's 1936 paper.

All aforementioned men were attracted to the "universal Turing machine" concept because it allowed them to express the fundamental interchangeability of hardware and language implementations (of almost all computer capabilities). Unsurprisingly, insights such as these came during a decade of cross-fertilization between logic, linguistics, and programming technology — a decade in which computing was still a long way from establishing itself as an academically respectable field.

Just like some space cadets, also the reader may have noticed a strong similarity between Weaver's 1949 "interlanguage" or "universal language" on the one hand, and the notion of an intermediate machine-independent programming language on the other hand.[45] Another example of technological convergence is the pushdown store which, as Oettinger explained in his 1961 paper, served a unifying technological role across the domains of automatic programming and machine translation [89].

Understanding the birth of *computer science* — a term which I use here for the first time — amounts to grasping and documenting the technological convergence of the 1950s. On the theoretical side of this convergence, Church's, Turing's, and especially Post's work became increasingly relevant — a topic that concerns the notion of undecidability (see [41, Ch.2]) and which lies outside the scope of the present article. On the practical side, the universal Turing machine played a clarifying role in that it helped some experienced programmers, the space cadets, to grasp the bigger picture of what they had been accomplishing in conformance with the language metaphor (— a metaphor that became well established during the 1950s). To be more precise, the universal Turing machine paved the way for the complementary onion-skin metaphor, thereby allowing the space cadets to view their translation problem (from one *language* into another) in the interrelated, operational, setting of *machines*.[46] In modern terminology: the programming *languages* `ALGOL 60` and `FORTRAN` can be viewed as equivalent computational tools to that of a universal Turing *machine*. From this specific theoretical perspective, it does not matter which programming language or which computer one prefers, they are all equivalent and (to a very large extent) interchangeable.

By the end of the 1950s, two metaphorical seeds had thus been sown for future advances in computer programming. The concepts of language and machine became increasingly interchangeable. Dijkstra in the 1960s, for example, frequently described a layer in his hierarchical designs as either a language or, equivalently, as a machine.[47] The origins of Structured Programming are, in my opinion, firmly rooted in the language metaphor and the onion-skin metaphor [34, Ch.1].

Viewing the universal Turing machine as a mathematical model of a general-purpose computer — as Booth, Carr, Gill, and Gorn did in the second half of the 1950s — also meant that the *ideal* general-purpose computer was one of *infinite* storage capacity. Nofre et al. appropriately use the words "conceptualization of the computer as an *infinitely* protean machine" [85, my emphasis] (to describe this transformational point of view) but they do this without mentioning the pivotal role the universal Turing machine played in this regard.[48] Moreover, many historical actors never joined Gorn et al. in viewing computers as infinitely protean machines. Dijkstra, Parnas, and others frequently insisted throughout their careers not to ignore the finiteness of real machinery, even when reasoning abstractly about software.[49]

It was *some* mathematicians — most notably Carr, Gorn, and Perlis — who in the field of automatic programming tried to seek unifying principles and who advocated making a science. In 1958, for example, Carr explained his desire for "the creation of translators, techniques for using them, and, finally, a *theory* of such formal translators. [...] The development of 'automatic problem solutions' requires formalism, interchangeability of procedures, and computability of languages if it is to become a true discipline in the *scientific* sense." [27, p.2, my emphasis].[50]

Carr, Gorn, and Perlis played a large part in helping form and shape the `ACM` during the 1950s and 1960s. Under the initiative of Oettinger, the `ACM` chose to honor Turing in 1965. This was well *before* Turing's secret war work in England came into the open. In other words, Turing's influence was already felt in automatic programming, automata theory, and other research fields before popular books were published about his allegedly significant role in the makings of the first universal computers.

It was Babbage who — rightly or wrongly — was put front and center during the 1950s as the father of the universal computer (cf. Alt [9, p.8]). During the 1950s and 1960s, Turing was never portrayed as the father of the universal computer. Since the 1970s, popular claims have been made, literally stating that Turing is the "inventor of the computer" or "the inventor of the universal computer" — see the romantic accounts of Copeland [32], Davis [36, 37], Dyson [48], Leavitt [67, 68], and Robinson [99] for some typical examples; see Burks's fitting rebuttals [23, 24] and van Rijsbergen's and Vardi's sober reflections on Turing's legacy [108, 110].

"What would Turing be doing today if he were still alive?" — a question that was posed multiple times in celebration of Turing's Centennial. He would be countering the now-prevailing belief that he is "the inventor of the computer"!

Acknowledgments

I am particularly grateful to Thomas Haigh for his written and oral feedback on multiple drafts of this article, starting in the spring of 2013. I also thank an anonymous reviewer, the editors Liesbeth De Mol and Giuseppe Primiero, and my mentor Gerard Alberts.

Endnotes

[1]The minutes of that meeting state:

> Bright reported that the Program Committee recommends that the National ACM Lecture be named the Allen [sic] M. Turing Lecture.
> Oettinger moved, seconded by Young that it be so named. Several council members indicated they were not satisfied with this choice.
> Juncosa suggested we consider a lecture name that is not that of a person.
> van Wormer moved, seconded by Juncosa to table the motion.
> The vote was: for-15; opposed-5; abstention-2. [4, p.11, original emphasis]

[2]Carr was President of the National Council of the ACM in 1957–59, founding editor of Computing Reviews in 1960–62, and member of the Committee for the Turing Award during the second half of the 1960s. Gorn was committee member on programming languages, a Council Member in 1958–68, and Chairman of the Standards Committee in 1962–68. Oettinger was President of the ACM in 1966-68 [7, 26, 69, 84, 90].

[3]The phrase "a large store containing both numbers and instructions" is akin to the terminology used by Andrew & Kathleen Booth in 1956 [20]. The primary sources listed in the bibliography suggest that the Booths, Carr, Gorn, and several other actors did not use the words "stored program" during the first half of the 1950s.

As Mark Priestley notes in his book, *A Science of Operations* [94], it was the early machines (such as the ASCC and ENIAC) that were described as revolutionary: "It was several years until the first machines based on the stored-program design became operational, and even longer until they were widely available" [94, p.147].

[4]The reader should bear in mind that although I have extensively studied the "Saul Gorn Papers", which also contain a lot of correspondence between Gorn and Carr, the papers do not contain many primary sources of the 1940s and early 1950s. Moreover, I have yet to travel to the Great Lakes in order to study the "Alan J. Perlis Papers" at the Charles Babbage Institute, and the "John W. Carr Papers" and "Arthur W. Burks Papers" at the Bentley Historical Library. In future research I also aspire to cover the work of Maurice Wilkes and Christopher Strachey. Complementary to all this, I have written a chapter for a Dutch book *De geest van de computer* [46] in which I discuss the

accomplishments of the Dutch computer builders Gerrit Blaauw and Willem van der Poel and the programmers Aad van Wijngaarden and Edsger W. Dijkstra.

[5]A complementary history of programming languages is covered in Knuth's 'The Early Development of Programming Languages' [65, Ch.1]. While Carr and Gorn are not mentioned in the main body of Knuth's account, they are key players in the present article.

[6]The words "machine translation" and "automatic language translation" are used interchangeably in this article.

[7]In retrospect, it is safer to restrict the claim that "everybody knew each other" to British and American practitioners. Grace Hopper's 1978 recollection illustrates this point:

> I had absolutely no idea of what [the Swiss] Rutishauser, [the German] Zuse, or anyone else was doing. That word had not come over. The only other country that we knew of that was doing anything with the work was Wilkes in England. The other information had not come across. There was little communication, and I think no real communication with Germany until the time of ALGOL, until our first ALGOL group went over there to work with them. I think it's difficult for you in a seminar here to realize a time when there wasn't any ACM, there wasn't any IEEE Computer Society. There was no communication. There was no way to publish papers. [111, p.23]

[8]That "robot", which was of a "certain formal character", could deduce "any legitimate conclusion from a finite set of premises". Weaver adapted McCulloch and Pitts's theorem to mean that "insofar as written language is an expression of logical character", the problem of machine translation is at least solvable in a formal sense [70, p.22].

Neither Weaver nor McCulloch and Pitts explicitly referred to Turing's papers. Moreover, I have given no evidence here to suggest that Weaver was, by 1949, well versed in Turing's theoretical work. Likewise, McCulloch and Pitts's sole, brief, and inadequate reference to a "Turing machine" in one of their concluding remarks suggests that they were not well versed in Turing's theory of computation either. Their sole reference to a "Turing machine" was, strictly speaking, incorrect. In their words:

> [E]very net, if furnished with a tape, scanners connected to afferents, and suitable efferents to perform the necessary motor-operations, can compute only such numbers as can a [universal] Turing machine [74].

Nevertheless, an indirect link between Weaver's research agenda on machine translation and Turing's 1936 notion of a universal machine had been established by 1949, even though Turing's work had yet to really surface in both linguistics and computing. For example, no reference was made to Turing, Post, and the like in the comprehensive 1955 book *Machine Translation of Languages* [70].

[9]For a good understanding of the connection between McCulloch & Pitts's 1943 paper and von Neumann's work on the `EDVAC`, see Akera [8, p.118–119] *and* Burks [23, p.188] in conjunction.

[10]By 1946 Turing had become well aware of the fact that his 1936 notion of a universal machine could, essentially, serve as a mathematical model of a general purpose computer [60, p.4,6][95, p.76]. Turing, von Neumann, and their close associates may well have been the only people to have seen this connection during the 1940s. It was by presenting his 1950 paper, in which he devoted a section to 'The Universality of Digital Computers', that Turing gradually changed the common perception among some of his contemporaries, including Wilkes and Oettinger [95, p.79,84][94, p.152–153].

During the 1950s, a continually increasing number of computer practitioners began to view a universal Turing machine as a mathematical model for a general-purpose computer that was based on the principle of a large store containing both numbers and instructions. The practitioners included computer designers, switching theorists, and logicians — such as Willem van der Poel, Edward F. Moore, and Hao Wang [41, Ch.2]. Moore, for example, wrote in 1952 that a universal Turing machine "can, loosely speaking, be interpreted as a completely general-purpose digital computer" [78, p.51].

[11]Bar-Hillel was moreover coming to the conclusion that "even machines with learning capabilities [...] will not be able to become fully autonomous, high-quality translators" [14, p.9].

[12]Likewise, no mention of Turing was made in the comprehensive 1971 book *Computer Structures: Readings and Examples* [16] (— I thank David L. Parnas for bringing this book to my attention). Turing's involvement with computer *building* was *popularized* later, by Randell (1973), Hodges (1983), Robinson (1994), Davis (2000), Dyson (2012), and others [97, 59, 99, 37, 48]. David Kahn's 1967 book *The Codebreakers* [62] did not cover much about Turing either (— I thank Vinton G. Cerf for bringing this book to my attention.)

[13]Contrast this with George Dyson's claim that Booth saw von Neumann's computer project at Princeton as the practical implementation of both Babbage's *and* Turing's ideas [48, p.132]. For further scrutiny of Dyson's book *Turing's Cathedral*, I refer to my review [42].

[14]In modern terminology: Most computer designers who considered using a small instruction set did *not* do this in connection with practically realizing a universal Turing machine. An exception in this regard, other than Turing himself, was the successful Dutch computer builder Willem van der Poel. In his 1952 design of his `ZERO` computer, van der Poel did refer to Turing's theoretical 1936

notion of universality in connection to the ZERO's arithmetical unit. Van der Poel sought the most simple computer that was still universal in Turing's theoretical sense [106, p.376]. He elaborated this principle in his Ph.D. dissertation [107]. For brief accounts, see De Mol & Bullynck [76] and Daylight [45]. An extensive account is forthcoming [46].

[15]Several chapters in Bowden's 1953 book *Faster than Thought* [21] honored Babbage in a similar manner. Chapter 8 on the ACE *also* made a connection between Turing and Womersley's computer building aspirations on the one hand, and Turing's 1936 paper on the other hand. But, as the "Türing Machine" entry on page 414 indicates, the vast majority of computer builders — and practitioners in general — did not read Turing's 1936 paper, let alone grasp it's practical implications (cf. [41]).

[16]I thank Thomas Haigh for making this correction.

[17]So far, my exposition aligns well with one of Akera's main claims — namely that

> [a]side from the relative ease by which a program could be set up on such a system, the major aim of [EDVAC's] design was to reduce the amount of hardware by severely limiting the machine's parallelism. [8, p.115–116]

[18]A discussion of von Neumann's own writings lies outside the scope of this article. I speculate that he knew that a store was *not* a necessary condition (nor a sufficient condition) for realizing a practical version of a universal Turing machine (cf. endnote 14 in [41, p.202]). Or, as Allan Olley befittingly puts it, "the stored program [computer] is not the only way to achieve [what was later called] a Turing complete machine" [92]. The terms "Turing" and "completeness" were in use by 1965 (cf. J.G. Sanderson [102]).

Von Neumann was in a position to see that the universal Turing machine was a mathematical model of his IAS computer, the EDVAC, *and* the ENIAC. There is no primary sources supporting the popular belief that he thought he was building the first practical realization of a universal Turing machine. Priestley has drawn very similar conclusions in his recent book [94, p. 139, 147–148]. More important then is Akera's observation that von Neumann was embarking on a "formal theory of automata" — a theory which pertained to "machines capable of modifying their own programs" [8, p.119]. And, just to be clear, the ability to modify its own programs is *not* a prerequisite to practically realizing a universal Turing machine.

[19]An erasable store was *not* a necessary condition (nor a sufficient condition) for realizing a practical version of a universal Turing machine — a fact known to the mathematical logician Hao Wang in 1954 who *did* ponder about these theoretical issues [41, p.21].

[20]From a *sociological* perspective, it is interesting to compare and contrast Hopper's words with the following words made by Andrew Appel in 2012:

> [Appel, 27 minutes and 44 seconds into his talk:] The machines we use today are von-Neumann machines. And von Neumann was quite clearly influenced by Turing in building universal machines with a finite-state control and infinite tape. Really the only difference is that the infinite tape is random access rather than linear access. So von-Neumann machines are what we now call computers. The Harvard architecture in which the program is not stored in the memory of the computer; well, we saw a quotation from Howard Aiken [in the previous talk by Martin Davis [38] — a quotation which Davis repeatedly uses to ridicule Aiken and which I object to in my book [41, Ch.8]]. By the 90s, when you called something a Harvard architecture, you meant that the program was stored in a memory but in a read-only memory because nobody would even imagine that you could have a computer program that wasn't even representable as bits. [11]

Both Hopper and Appel were using the historical actor "Howard Aiken" and "history" in general to make a case for their own research aspirations. I view Hopper as a pluralist; in her 1978 keynote address, she repeatedly encouraged a multitude of programming styles as opposed to "trying to force" every practitioner into "the pattern of the mathematical logician":

> [Hopper:] I'm hoping that the development of the microcomputer will bring us back to reality and to recognizing that we have a large variety of people out there who want to solve problems, some of whom are symbol-oriented, some of whom are word-oriented, and that they are going to need different kinds of languages rather than trying to force them all into the pattern of the *mathematical logician.* A lot of them are not. [111, p.11, my emphasis]

I view Appel as a unifier; that is, as a theoretically-inclined researcher who seeks unifying principles to further his research. Appel's 2012 address was all about the importance of mathematical logic in computing and thus, unsurprisingly, about Turing and von Neumann's allegedly important roles in the history of the computer.

[21]An anonymous reviewer is quite right in noting that the words "In my eyes" run the risk of judging Carr's work in anticipation of later ways of thinking. To rectify this, I include this footnote as a disclaimer (which has another, intended, effect than simply removing the quoted words from the narrative).

[22]While it was common practice to refer to the "universal language of mathematics" and while it was customary to associate a language with machine code, as in the words "machine language", it was in 1954 still uncommon to describe the pseudo code of a computer as a "language". Brown and Car *did* consistently do this in their joint paper (— Backus and Herrick came in a close second at

that same symposium). And, as Nofre et al. point out, the words "programming language" in fact only entered the computing arena during the mid-1950s [85].

Nofre et al. state that by the early 1950s the translation from a mathematical language to machine language had already "become a central metaphor used to make sense of the activity of programming". They then refer to an illustration made by Hopper, an illustration in which a robot translates "Language X" into "Language A", *but* it should also be stressed that this illustration dates from October 1958. No evidence has been provided that Hopper described X as a language in 1954.

[23]Nofre et al. describe the origins of the idea of a universal code, going back to C.W. Adams of `MIT` in 1951.

[24]Note that Gorn made this implication at a time when computer storage was limited.

[25]As the 1950s progressed, Carr and Gorn refined their expositions. For example, "compilers" did "not absolutely require their machines to possess common storage of instructions and the data they process" but they were "considerably *simpler* when their machines" did have this property [55, p.254, my emphasis].

[26]This fourth requirement, by itself, already indicates quite strongly that Carr and Gorn were not pursuing logical minimalism à la Turing and van der Poel (cf. De Mol & Bullynck [76]).

[27]These properties were sufficient (but *not* necessary) for realizing a practical version of a universal Turing machine. Note, again, that I am making the theoretical connection here *instead of* the historical actors. Carr and Gorn had the knowledge to defer this conclusion themselves, but I conjecture that they were *not* reasoning along these lines *at all,* during the first half of the 1950s (and possibly throughout their whole careers).

[28]Machines that were not based on the principle of common storage of numbers and instructions were instructed differently than those that did possess loop control. The Swiss numerical analyst Rutishauser, for example, used (a modification of) Zuse's `Z4` machine through part of the 1950s, a machine that did not possess loop control. Although more research on Rutishauser is required, it is already interesting to note here that only by 1963 did he see the need to program recursively in his field of numerical analysis [101] (see also [39, 41]).

[29]That is, in the finitist terms set forth by "intuitionist mathematicians such as Kronecker, Brouwer, and Weyl" [54].

[30]Some of Gorn's 1955 insights were later published in his 1957 paper [55].

[31]For Gorn, infinite memory represented the simplest case of analysis. This was similar to Curry's reasoning (cf. [33] [1, p.65]) and unlike van der Poel's

insistence to view results from mathematical logic solely in the setting of finite machines (cf. [45, 107]).

[32]Besides Gorn in 1956, also Berkeley in 1958 discussed the importance of machines that have "loop control", and hence "think", compared to the "non-thinking machines" [19, p.5]. Contrast these observations with Akera's claim that the historiographical interest in the "stored-program concept" really only started around 1964 with the emergence of theoretical computer science [8, p.120].

[33]Notice that Gorn excluded the "older machines", such as the `ENIAC`. He was implicitly referring to all loop controlled computers and to the principle of a common store in particular. According to Gorn, those machines provided the flexibility for the development of a universal code.

[34]Moreover, note that Gorn's recognition was, strictly speaking, incorrect. We know, today, that also many of the "older machines" were "Turing complete" as well. For example, it suffices to "wire in" the program of a universal Turing machine and then represent any (to-be-simulated) Turing machine program, along with working store, on punched paper tape — tape which serves both as input and output for the calculation at hand. It is *not* a theoretical obstacle that the punched holes cannot be removed (cf. [41, p.22]).

An anonymous reviewer has commented that

> [E]ven if `ENIAC` was later identified as being Turing complete, the important historical thing is that the first machines to be thought of as Turing complete, and to inspire the integration of Turing's work into the growing field of computer science, were modeled on `EDVAC`.

This assessment by the reviewer seems fair with regard to Gorn's career. I have not given evidence to suggest that the claim holds in its general form; i.e., that it holds for other actors like von Neumann, van der Poel, Turing, and Wang. I would not be surprised if von Neumann did connect the `ENIAC` to Turing's 1936 universal machine around 1945, thereby (partially) contradicting the reviewer's comment in its general form.

In this regard I also present Priestley's words which I think nicely summarize a complementary part (of the total discussion):

> None of the writings originating from the Moore School group mention a logical or theoretical rationale for the introduction of the stored-program concept. It remains a possibility, of course, that the idea was suggested by von Neumann's knowledge of Turing's work, but even if that was so, its inclusion in the design was justified by practical, not theoretical, arguments. A logical pedigree for the idea would not, on its own, have been sufficient to ensure its incorporation in the design without a detailed examination of its engineering implications. [94, p. 138–139]

I conjecture that von Neumann knew that *both* the `ENIAC` and the `EDVAC` were "Turing complete" and he knew that this insight had no immediate practical relevance on his computer-*building* aspirations.

[35]Thanks in no small part to the writings of Burks (cf. [41, p.25]).

[36]Gorn continued by stating that the as yet ill-defined, universal command language is "probably, in terms of mathematical logic" equivalent with "the class of expressions obtainable by 'general recursive definitions"'. In the early 1960s, the recursive-function theorist Henry Gordon Rice substantiated Gorn's hunch by making an explicit connection between `ALGOL 60`'s computational power and the class of general recursive functions [41, Ch.8]. Important to note here is that, once again, the *theoretical* insights came *after* programming *practice.*

[37]Post's work was, together with Davis's 1958 book [35], also largely responsible for the advent of "decision procedures" [28, p.223, 225] in automatic programming, automata theory, and mathematical linguistics in general — a topic that lies outside the scope of this article but which is very relevant to understanding the emergence of (what many people seem to call) "theoretical" computer science.

It was researchers in computational linguistics (like Chomsky, Bar-Hillel, and especially Perles and Shamir) and soon-to-be-called theoretical computer scientists (like Rabin, Scott, Ginsburg, and Rice) who demonstrated a thorough reception of Turing's work and especially Post's "correspondence problem" during the late 1950s and 1960s (see e.g. [15, 50, 96]).

[38]Note that some of these insights were already several years old. I am discussing Carr's 1959 work here because I have yet to come across much of his earlier writings.

[39]Turing's 1936 paper and his follow-up correction [103, 104] were reprinted in Appendix One of the proceedings [52].

[40]I conjecture that Turing never viewed his 1936 paper from this angle — i.e., that he was not (and did not want to be) a space cadet — and that Booth and other space cadets did not completely grasp Turing's work. On the one hand, of course, *that* is exactly where the force of Turing's theory lies; it helped some researchers — and, more importantly, practitioners working in or with industry — to see the wood for the trees in unanticipated ways. On the other hand, many practitioners were not really helped by Turing's work: there is no evidence suggesting that if Turing had not existed, that these practitioners would then have been doing something else. Many advocates of various programming languages knew that these languages were equivalent. Several languages were implemented by translating into another competing language. The real issue was their suitability for human use. (— I thank Parnas for discussing this matter with

me.) Having said *that*, I now refer to Robert Bemer's experience with equivalent languages. In 1957 he wrote:

> Although the ultimate in language does not exist yet, we can console ourselves meanwhile with compatible (as against common) language. There is much current evidence that existing algebraic languages are all mappable into one another by pre-processors. [17, p.115][85]

Bemer's words make me conclude that some researchers were in need of a solid, theoretical, unifying principle, such as the universal Turing machine or the universal programming language `ALGOL`. Others were satisfied with experimental evidence. My conclusion here is merely another way of expressing my respect for the *multitude* of personal styles in tackling technical problems (— a topic that I have discussed at length with Peter Naur [40]). Unsurprisingly, it was the *theoretically-inclined* space cadets (like Carr, Gorn, and Perlis) who talked about some implications of Turing's work during the 1950s.

[41]In line with Nofre et al. [85], I am introducing the word "metaphor" here instead of the historical actors. Nofre et al. elegantly discuss the origins of the metaphor, going back to postwar cybernetics and Stibitz's role in 1947 in this regard. I take gentle issue with their supposition that the language metaphor had become a central metaphor in the computer programming arena by the early 1950s. Carr and his space cadets were few in number, exactly because they took the language metaphor seriously.

[42]Regardless of whether Gill was the first to articulate the onion-skin metaphor, it should at least not go unnoticed here that hierarchical design would become a hot research topic in later years. In what I take as support for my narrative, Priestley's 'The Invention of Programming Languages' [94, Ch.8] suggests that (i) men like Turing (1951), Wilkes (1952), and others understood the idea of "an interpretive routine enabling one machine to simulate another" and that (ii) in *later* years, Booth, Gill, and others associated the universal Turing machine concept with interpreters for high level programming languages [94, p.191–192].

[43]Gorn's technical reports from the 1960s and onwards were all about: Chomsky — Algol — Turing — Gödel — Wang — Curry — Carnap — Quine — Rosenbloom — . . . and so on.

[44]There are other examples. Philip R. Bagley, for instance, connected the Universal Computer-Oriented Language (`UNCOL`) to Turing's work. I take his understanding of Turing's work to be weak (and without much practical significance to him), as his words from 1960 partially indicate:

> If `UNCOL` can express the basic computing steps required by a [**universal**] Turing machine, then since all present-day computers are equivalent to [finite realizations of **universal**] Turing machines, it can lay claim to being "universal" from the standpoint of computation alone. [. . .] The limitation of `UNCOL`

> that we are really concerned with is whether or not we can achieve tolerable efficiency of execution [. . .] [13, p.21] [See also [12]]

[45]Weaver supported the idea of an indirect route: translate from the source language into an as yet undiscovered *universal* language and then translate from that language into the target language [70, p.2,23]. Likewise, Dijkstra (and others) supported a two-step translation of the universal programming language `ALGOL 60`. First, translate the `ALGOL 60` program into an intermediate *machine-independent* object language. Second, process the obtained object program by an interpreter (which is written in the machine code of the target computer) [41, p.68]. Also `UNCOL` was based on this principle, which was

> the design of a *universal*, intermediate language, independent of specific hardware but similar in character to machine languages, as a bridge over the growing gap between problem oriented languages (POLs) and machine languages (MLs). [3, p.2, my emphasis]

[46]In other words: not only did the space cadets associate the machine with a "machine language" and the pseudo code with a "programming language", they also began to view the programming language as a machine. Both metaphors together led to hierarchical decompositions of software and hardware in which each level was characterized as both a machine and a language (cf. [46]).

[47]Already in 1962, Dijkstra wrote: "machine and language are two faces of the same coin" [47, p.237]. Likewise, van Wijngaarden wrote: "we rather see the language as a machine" [109, p.18].

[48]Nofre et al. furthermore state that "two prominent senses of the notion of universality emerged" during the 1950s; namely, the idea of "machine-independence" and "a sense connecting directly with the universality of the notations of science, and in particular mathematical notation". Two more prominent senses of universality can now be mentioned, namely the idea of a universal Turing machine and a sense connected directly or indirectly with the universality of Weaver's interlanguage.

[49]I thank Parnas for discussing this matter with me. See also my subsequent blog post on this topic [44].

[50]I thank Gerard Alberts for bringing this quote to my attention.

References

1. *Symposium on Advanced Programming Methods for Digital Computers — Washington, D.C., June 28,29, 1956*, 1956. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 43.
2. University of Pennsylvania Computer Activity Report: July 1, 1959 – December 31, 1960. Technical report, Office of Computer Research and Education, 1960. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 39.
3. UNCOL Committee Report. From the Charles Babbage Institute collections, May 1961. Thanks to David Nofre for giving me a copy of this letter.
4. ACM Council Meeting, 27 August 1965. Available from the "Saul Gorn Papers", the University of Pennsylvania Archives (unprocessed collection).
5. ACM Council Meeting, 29 April 1966. Available from the "Saul Gorn Papers", the University of Pennsylvania Archives (unprocessed collection).
6. ACM Council Meeting, 29 August 1966. Available from the "Saul Gorn Papers", the University of Pennsylvania Archives (unprocessed collection).
7. A. Akera. Anthony Oettinger interview: January 10–11, 2006. In *ACM Oral History interviews*, 2006.
8. A. Akera. *Calculating a Natural World: Scientists, Engineers, and Computers During the Rise of U.S. Cold War Research.* MIT Press, 2007.
9. F.L. Alt. *Electronic Digital Computers: Their Use in Science and Engineering.* Academic Press, 1958.
10. F.L. Alt. Archaeology of computers — reminiscences, 1945–1947. *Communications of the ACM*, 15(7):693–694, 1972.
11. A. Appel. Turing, Gödel, and Church at Princeton in the 1930s. YouTube: www.youtube.com/watch?v=kO-8RteMwfw. Presentation at the Turing Centennial Celebration at Princeton, 10–12 May 2012.
12. P.R. Bagley. Letter to Mort Bernstein. From the Charles Babbage Institute collections, June 1960. Thanks to David Nofre for giving me a copy of this letter.
13. P.R. Bagley. Letter to the SHARE UNCOL Committee and other interested parties, 26 May 1960. From the Charles Babbage Institute collections, May 1960. Thanks to David Nofre for giving me a copy of this letter.
14. Y. Bar-Hillel. *Language and Information: Selected Essays on their Theory and Application.* Addison-Wesley Publishing Company, Inc. and the Jerusalem Academic Press Ltd, 1964.
15. Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschungr Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961.
16. C. Gordon Bell and A. Newell. *Computer Structures: Readings and Examples.* McGraw Hill, 1971.
17. R.W. Bemer. The status of automatic programming for scientific problems. In F.C. Bock, editor, *The Fourth Annual Computer Applications Symposium, October 24–25, 1957*, pages 107–117. Armour Research Foundation, 1958.
18. E.C. Berkeley. *Giant Brains, or Machines That Think.* Wiley, New York, 1949.
19. E.C. Berkeley. Memorandum for the Association for Computing Machinery — Committee on the Social Responsibilities of Computer Scientists. Technical report, October 1958. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 3.

20. A.D. Booth and K.H.V. Booth. *Automatic Digital Calculators*. Butterworths Scientific Publications, 2nd edition edition, 1956.
21. B.W. Bowden, editor. *Faster Than Thought: A Symposium on Digital Computing Machines*. Pitman, 1953.
22. J. Brown and J.W. Carr, III. Automatic programming and its development on the MIDAC. In *Symposium on Automatic Programming for Digital Computers*, pages 84–97, Washington D.C., May 1954. Office of Naval Research, Department of the Navy.
23. A.W. Burks. Turing's theory of infinite computing machines (1936–1937) and its relation to the invention of finite electronic computers (1939–1949). In *Theory and Practical Issues on Cellular Automata*. Springer, 2001.
24. A.W. Burks. The invention of the universal electronic computer—how the Electronic Computer Revolution began. *Future Generation Computer Systems*, 18:871–892, 2002.
25. M. Campbell-Kelly and W. Aspray. *Computer: A History of the Information Machine*. Basic Books, 1996.
26. J.W. Carr. Inaugural presidential address. Presented at the meeting of the Association, August 1956.
27. J.W. Carr. Computing Programming and Artificial Intelligence. Ann Arbor, University of Michigan, Summer 1958. An intensive course for practicing scientists and engineers: lectures given at the University of Michigan.
28. J.W. Carr. Programming and coding. In E.M. Grabbe, S. Ramo, and D.E. Wooldridge, editors, *Handbook of Automation, Computation, and Control: Computers and Data Processing*, volume 2, chapter 2. John Wiley and Sons, Inc., 1959.
29. N. Chomsky. *Transformational Analysis*. PhD thesis, University of Pennsylvania, 1955.
30. N. Chomsky. Three models for the description of language. *I.R.E. Trans. on Information Theory*, IT-2, 1956.
31. N. Chomsky. *Syntactic Structures*. The Hague/Paris: Mouton, 1957.
32. B. Jack Copeland. *Turing: Pioneer of the Information Age*. Oxford University Press, 2012.
33. H.B. Curry. On the composition of programs for automatic computing. Memorandum 9806, Silver Spring, Maryland: Naval Ordnance Laboratory, 1949.
34. O.-J. Dahl, E.W. Dijkstra, and C.A.R. Hoare. *Structured Programming*. Academic Press, London - New York - San Francisco, 1972.
35. M. Davis. *Computability and Unsolvability*. McGraw-Hill, 1958.
36. M. Davis. Mathematical logic and the origin of modern computers. In R. Herken, editor, *The Universal Turing Machine - A Half-Century Survey*. Oxford University Press, 1988. Originally in: Studies in the History of Mathematics. Mathematical Association of America, 1987, pages 137-165.
37. M. Davis. *The Universal Computer: The Road from Leibniz to Turing*. Norton, 1st edition, 2000.
38. M. Davis. Universality is ubiquitous. YouTube: www.youtube.com/watch?v=ZVTgtODX0Nc, May 2012. Presentation at the Turing Centennial Celebration at Princeton, 10–12 May 2012.
39. E.G. Daylight. Dijkstra's rallying cry for generalization: the advent of the recursive procedure, late 1950s – early 1960s. *The Computer Journal*, March 2011. doi: 10.1093/comjnl/bxr002.
40. E.G. Daylight. *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains*. Lonely Scholar, October 2011. www.lonelyscholar.com.

41. E.G. Daylight. *The Dawn of Software Engineering: from Turing to Dijkstra*. Lonely Scholar, 2012. www.lonelyscholar.com, ISBN 9789491386022.
42. E.G. Daylight. A Hard Look at George Dyson's Book "Turing's Cathedral: the Origins of the Digital Universe". In *Turing in Context II*, 2012. Lecture available on video: www.dijkstrascry.com/presentations.
43. E.G. Daylight. On Mahoney's Accounts of Turing. Blog post: www.dijkstrascry.com/Mahoney, 20 August 2013.
44. E.G. Daylight. Strachey vs. Dijkstra :: Infinite vs. Finite. Blog post: www.dijkstrascry.com/infinite, 29 March 2013.
45. E.G. Daylight. Turing's 1936 Paper and the First Dutch Computers. CACM blog post: http://cacm.acm.org/blogs/blog-cacm/167012-turings-1936-paper-and-the-first-dutch-computers/fulltext, 19 August 2013.
46. E.G. Daylight. De kern van de zaak. In *De Geest van de Computer*. To appear. English translation to appear.
47. E.W. Dijkstra. An attempt to unify constituent concepts of serial program execution. In *Proceedings of the Symposium Symbolic Languages in Data Processing*, pages 237–251, New York and London, 1962. Gordon and Breach Science Publishers.
48. G. Dyson. *Turing's Cathedral: The Origins of the Digital Universe*. Penguin Books, 2012.
49. U. Erlingsson, Y. Younan, and F. Piessens. Low-level software security by example. In *Handbook of Information and Communication Security*, pages 633–658. Springer, 2010.
50. S. Ginsburg and H. Gordon Rice. Two families of languages related to ALGOL. pages 350–371, 1961.
51. V.E. Giuliano and A.G. Oettinger. Research on automatic translation at the Harvard Computation Laboratory. In *Information processing: proceedings of the International Conference on Information Processing*. Unesco, Paris, 1959.
52. R. Goodman, editor. *Annual Review in Automatic Programming I: Papers read at the Working Conference on Automatic Programming of Digital Computers held at Brighton, 1–3 April 1959*. Pergamon Press, 1960.
53. S. Gorn. Planning universal semi-automatic coding. In *Symposium on Automatic Programming for Digital Computers*, pages 74–83, Washington D.C., May 1954. Office of Naval Research, Department of the Navy.
54. S. Gorn. Real solutions of numerical equations by high speed machines. Technical Report 966, Ballistic Research Laboratories, October 1955. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection).
55. S. Gorn. Standardized programming methods and universal coding. *Journal of the ACM*, July 1957. Received in December 1956.
56. S. Gorn. Common programming language task, final report: Report of the work in the period 1 May 1958 to 30 June 1959. Technical Report AD59UR1, July 31 1959. Available from the "Saul Gorn Papers" from the University of Pennsylvania Archives (unprocessed collection): UPT 50 G671 Box 39.
57. S. Gorn and W. Manheimer. *The electronic brain and what it can do*. Science Research Associates, Inc., 1956.
58. E.M. Grabbe, S. Ramo, and D.E. Wooldridge, editors. *Handbook of Automation, Computation, and Control: Computers and Data Processing*, volume 2. John Wiley and Sons, Inc., 1959.
59. A. Hodges. *ALAN TURING: The Enigma*. Burnett Books, 1983.

60. A. Hodges. Alan Turing: the logical and physical basis of computing. *British Computer Society*, 2007. http://www.bcs.org/ewics.
61. Yu. I. Ianov. On the equivalence and transformation of program schemes. *Doklady Aka. Nauk S.S.S.R.*, 113:39–42, 1957.
62. D. Kahn. *The Codebreakers: The Story of Secret Writing.* The Macmillan Company, 1967.
63. L.V. Kantorovich. On a mathematical symbolism convenient for performing machine calculations. *Doklady Aka. Nauk S.S.S.R.*, 113:738–741, 1957.
64. S.C. Kleene. *Introduction to Metamathematics.* Van Nostrand, 1952.
65. D.E. Knuth. *Selected Papers on Computer Languages.* CSLI, 2003.
66. S. Lavington, editor. *Alan Turing and his Contemporaries: Building the world's first computers.* bcs, 2012.
67. D. Leavitt. *The Man Who Knew Too Much: Alan Turing and the Invention of the Computer.* Atlas Books, 2006.
68. D. Leavitt. *Alan Turing, l'homme qui inventa l'informatique.* Dunod, 2007.
69. J.A.N. Lee. *Computer pioneers.* IEEE Computer Society Press, 1995.
70. W.N. Locke and A.D. Booth, editors. *Machine Translation of Languages.* The Technology Press of The Massachusetts Institute of Technology and John Wiley & Sons, Inc., New York, 1955.
71. M.S. Mahoney. *Histories of Computing.* Harvard University Press, 2011.
72. A.A. Markov. *Theory of Algorithms.* Academy of Sciences of the USSR, 1954.
73. J. Martin-Nielsen. 'This war for men's minds': the birth of a human science in Cold War America. *History of the Human Sciences*, 23.(5):131–155, 2010.
74. W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. math. Biophys.*, pages 115–133, 1943.
75. L. De Mol. Doing mathematics on the ENIAC. Von Neumann's and Lehmer's different Visions. In E. Wilhelmus and I. Witzke, editors, *Mathematical practice and development throughout History*, pages 149–186. Logos Verlag, Berlin.
76. L. De Mol and M. Bullynck. A short history of small machines. In *CiE 2012 - How the World Computes*, 2012.
77. L. De Mol, M. Bullynck, and M. Carle. 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, 2010.
78. E.F. Moore. A simplified universal Turing machine. *Proc. ACM*, September 1952.
79. P. Mounier-Kuhn. Comment l'informatique devint une science. *La Recherche*, (465):92–94, June 2012.
80. P. Mounier-Kuhn. Computer Science in French Universities: Early Entrants and Latecomers. *Information & Culture: A Journal of History*, 47(4):414–456, November – December 2012.
81. P. Mounier-Kuhn. Logic and Computing in France: A Late Convergence. In *AISB/IACAP World Congress 2012 — History and Philosophy of Programming*, 2012.
82. P. Mounier-Kuhn. Algol in France: From Universal Project to Embedded Culture. *IEEE Annals of the History of Computing*, To Appear.
83. D. Nofre. Unraveling Algol: US, Europe, and the Creation of a Programming Language. *IEEE Annals of the History of Computing*, 32(2):58–68, 2010.
84. D. Nofre. Alan Jay Perlis, 2012. Short bio of Perlis, available on the official ACM website: amturing.acm.org/award_winners.
85. D. Nofre, M. Priestley, and G. Alberts. When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950–1960.

Technology and Culture, 2013. Re-drafted final submission. To appear with perhaps some minor modifications.
86. A.G. Oettinger. Programming a digital computer to learn. *Philosophical Magazine*, 43(347):1243–1263, 1952.
87. A.G. Oettinger. Account identification for Automatic Data Processing. *Journal of the ACM*, 4(3):245–253, 1957.
88. A.G. Oettinger. *Automatic Language Translation.* Harvard University Press, 1960.
89. A.G. Oettinger. Automatic syntactic analysis and the pushdown store. In *Proc. of Symposia in App. Math., Amer. Math. Soc. 12*, 1961.
90. A.G. Oettinger. Reminiscences of the boss. In *Makin' Numbers: Howard Aiken and the Computer*, pages 203–214. MIT Press, 1999.
91. Office of Naval Research, Department of the Navy. *Symposium on Automatic Programming for Digital Computers*, Washington D.C., May 1954.
92. A. Olley. Existence precedes essence — meaning of the stored-program concept. In *IFIP Advances in Information and Communication Technology*, pages 169–178. Springer, 2010.
93. A.J. Perlis. Announcement. *Communications of the ACM*, 1(1), January 1958.
94. M. Priestley. *A Science of Operations: Machines, Logic and the Invention of Programming.* Springer, 2011.
95. P.M. Priestley. *Logic and the Development of Programming Languages, 1930-1975.* University College London, May 2008. PhD thesis.
96. M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of research and development*, 3:115–125, 1959.
97. B. Randell, editor. *The Origins of Digital Computers: Selected Papers.* Springer-Verlag, 1973.
98. S.N. Razumovskii. On the question of automatization of programming of problems of translation from one language to another. *Doklady Aka. Nauk S.S.S.R.*, 113:760–762, 1957.
99. J.A. Robinson. Logic, computers, Turing, and von Neumann. In K. Furukawa, D. Michie, and S. Muggleton, editors, *Machine Intelligence 13: Machine Intelligence and Inductive Learning.* Clarendon Press, 1994.
100. P. Rosenbloom. *Elements of Mathematical Logic.* Dover, 1950.
101. H. Rutishauser. The use of recursive procedures. In R. Goodman, editor, *Annual Review in Automatic Programming 3.* Pergamon Press, New York, 1963.
102. J.G. Sanderson. On simple low redundancy languages. *Communications of the ACM*, 8(10), October 1965. Letters to the Editor.
103. A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2nd series*, 42:230–265, 1936.
104. A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proceedings of the London Mathematical Society, 2nd series*, 43, 1937.
105. A.M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.
106. W.L. van der Poel. A simple electronic digital computer. *Appl. sci. Res.*, 2:367–399, 1952.
107. W.L. van der Poel. *The Logical Principles of Some Simple Computers.* PhD thesis, Universiteit van Amsterdam, February 1956.
108. C.J. van Rijsbergen. Turing and the origins of digital computers. In *Aslib Proceedings*, volume 37, pages 281–285. Emerald Backfiles, June/July 1985. Paper presented at an Aslib Evening Meeting, Aslib, Information House, 27 March 1985.

109. A. van Wijngaarden. Generalized ALGOL. In R. Goodman, editor, *Annual Review Automatic Programming*, volume 3, pages 17–26. Pergamon Press, 1963.
110. M.Y. Vardi. Who begat computing? *Communications of the ACM*, 56(1):5, 2013.
111. R.L. Wexelblat, editor. *History of Programming Languages.* Academic Press, 1981.
112. M.V. Wilkes. Can machines think? *Spectator*, 6424:177–178, 1951.
113. M.V. Wilkes. *Memoirs of a Computer Pioneer.* MIT Press, 1985.