

The Work of Writing Programs: Logic and Inscriptive Practice in the History of Computing

David E. Dunning
University of Oxford

Abstract—

This article explores the entanglement of logic and computing by focusing on the activity of writing. Though mathematical logic is sometimes cast as the immaterial spirit of the computer's material body, the study of logic also takes place in the physical world through the manipulation of symbols on paper. Already in the nineteenth century, mathematical logic was understood to be related to mechanization, though not as the science behind an as-yet-uninvented technology. Rather, symbolic notations were seen as tools that opened possibilities but required new kinds of work. Turning to early electronic computing in the 1950s, I observe that researchers similarly relied on novel inscriptive techniques to mitigate labor. Finally, considering Charles Hamblin's Reverse Polish Notation, I show how logic was a source of notational invention, emerging as a practical resource for the work of writing programs independently of its role as a plausible theoretical foundation for computer science.

■ **THE PRECISE** nature of the connection between logic and electronic computing has long proved elusive and at times controversial. Already in the earliest attempts to plan ENIAC's successor, a now well-known rift opened between the mathematicians who proudly publicized their plans for the logical design of a new machine and the engineers who insisted that the machine's decisive innovations were rather in the realm of electrical engineering [12, 71–81]. An analogous fault-line later came to organize many influential efforts to think historically about

computing. Michael Mahoney formulated a classic logical–technological bifurcation that remains compelling:

The dual nature of the computer is reflected in its dual origins: hardware in the sequence of devices that stretches from the Pascaline to the ENIAC, software in the series of investigations that reaches from Leibniz's combinatorics to Turing's abstract machines. Until the two strands come together in the computer, they belong to different histories, the electronic calculator to the history

of technology, the logic machine to the history of mathematics [49, 116].

Arguing for a stable and apparently unappreciated separation between technology and mathematics, Mahoney implicitly accorded equal importance to the purportedly distinct histories of hardware and software, physical calculators and abstract logic. Other accounts have approached the same distinction more polemically, arguing for the primary importance of one side over the other.¹

This bifurcation framework has productively resisted anachronism by illuminating the multiplicity of lineages that contributed to the electronic computer, but the hardware–software dualism it assumes is unhelpful when we seek a historical account of how that particular dualist picture of logic’s role in computing became dominant. The present article joins recent efforts to describe a more complex historical relationship between logic and computing technology [16] [15] [37]. There is growing appreciation of the ways that, in the words of Liesbeth De Mol and Giuseppe Primiero, “relations between formal logic, engineering practices and physical machinery characterize some fundamental issues within computer science and its history,” central to the very “nature of the discipline” [16, 202]. Rather than a stable dualism for organizing historical analysis, the shifting relationship between logic and computing has been a source of ongoing tension within the histories we seek to understand. I propose to develop a new perspective on that tension by focusing on the activity of writing.

One major problem with casting mathematical logic as the immaterial spirit of the computer’s material body is that the study and use of logic also takes place in the physical world: typically it has taken place through the manipulation of symbols on paper. How does our image of the relationship between technology and mathematical logic change when we remember that writing too is a technology? At first glance it may seem odd to insist upon the technological status of writing in the contemporary era. Surely the initial *invention* of writing has been a major event in the

history of technology wherever it has occurred, but when we consider a society in which writing has been an established practice for centuries, writing’s technological aspect tends to become invisible. Perhaps sometimes this invisibility is simply a function of a technology operating in such a stable way that it really has become uninteresting; other times this oversight is a consequential blind spot.² It is illuminating to remember that writing is a technology precisely when the question of how people use it reopens. Computer programming, like earlier developments in the history of mathematics and logic, relied on powerful new ways of utilizing written symbols.

The obvious novelty of the digital computer has motivated interpretations, akin to determinist accounts of the printing press, that stress the radical change wrought by a new technology for the material production of writing. For Friedrich Kittler, the rise of computing meant that “the act of writing has vanished,” replaced by the manipulation of illegible electronic signals; the user has access only to “a new hierarchy of programming languages,” while “it remains mathematically impossible, in physical/physiological terms, to have access to [*erreichen*] all these layers [*Schichten*].”³ Indeed the complexity and to most users the opacity of the engineered physical phenomena that underlie the act of writing on a computer—in contrast to, say, writing with a pencil—is unprecedented. But we should also attend to human practices of writing, which underwent no absolute rupture upon the invention of the stored-program computer (or any other landmark in the history of computing). Rather, as Stephanie Dick has observed, “Handwritten practices informed and shaped different employments of computation, and computers displaced and transformed handwritten practices that preceded them. These histories should be studied

¹For an account that prioritizes the role of mathematical logic, see [14]. Mark Priestley has advanced a revisionist argument that logic’s importance to computing was retrospectively constructed rather than causally decisive [55]. I broadly agree with Priestley’s critique of the traditional story about logic’s role in computing, but aim to move beyond that opposition here.

²On the persistent importance of old technologies in modern history, see [22]

³[39, 219, 221]. For a determinist thesis on the revolutionary effects of the printing press, see [23], and for a forceful critique, see [36].

in tandem.”⁴ In this article I present one such tandem history, which I suggest reveals a practical affinity between logic and computing that helped make plausible the articulation of their conceptual linkage.

The use of symbols is an open-ended space of possibilities. If we aim to understand the historical context in which radically new forms of writing and reading emerged, we stand to learn from historians who study other periods of novelty in literate practice. In a classic study of “the uses of literacy” in England between 1066 and 1307, M. T. Clanchy observed that the ways one can interact with, understand, or benefit from the written word are far more varied than a binary distinction between literacy and illiteracy implies [13, 1]. Rather than asking how many people in High Medieval England could read and write, Clanchy focused on the growing production, use, and preservation of written records in secular life. He stressed that even serfs frequently understood that their rights and obligations were recorded in writing, and they came to use charters to transfer property. Thus, “Those who used writing participated in literacy, even if they had not mastered the skills of a clerk” [13, 2]. The written word is a technological system that affords numerous uses premised on varied kinds of understanding. Drawing on this expansive sense of “using literacy,” I understand “inscriptive practices” to encompass a similarly open-ended range of uses for written symbols, whether in mathematics, logic, or computing. In the case of computing, inscriptive practice includes a spectrum of activities that extends from writing in particular formal notations by hand, through knowing what the holes punched on a given subroutine tape do, to the material and social arrangements by which a community manages this diversity of tasks. In mathematical logic and computer programming alike, translating a problem into a formal system, manipulating the ensuing formalism to find a result, and even understanding what *could* be formalized all constitute different kinds of knowl-

edge, all of them uses of formal literacy.

This article describes how logic and then computing separately came to be disciplines centrally concerned with multiple ways of using literacy in formal systems, leading to a situation in which it made sense to view logic as a practical resource for the work of writing programming. I begin by briefly discussing mathematical logic in nineteenth-century Britain with an emphasis on the work of writing; I note in particular that systems of logical symbolism and of mechanization were already understood to be closely related well before 1900. Next I turn to the inscriptive practices of early electronic computer users, examining the technologies of writing presented in the watershed 1951 textbook *The Preparation of Programs for an Electronic Digital Computer* [70]. I situate the book’s central technique—the material, social, and theoretical organization of a subroutine library—as an innovation in the history of writing. As programming developed as a form of literary labor, some practitioners explicitly turned to logic as a resource, as I show in the final section by describing Charles Hamblin’s reinvention of Jan Łukasiewicz’s Parenthesis-Free Symbolism as Reverse Polish Notation. Crucially, the idea Hamblin borrowed was not conceptual but inscriptive: he followed Łukasiewicz in taking seriously the choice of convention for ordering symbols on the page in a system of writing. Mathematical logic increasingly came to be understood as a promising theoretical foundation for the young discipline of computer science, but it was already a model of notational ambition and a source of inscriptive techniques.

TECHNOLOGIES OF FORMAL LITERACY

The confrontation between abstract science and material technology that Mahoney identified in the computer is also visible within theoretical logic itself. As Mahoney observed, the digital computer has important scientific and technological aspects alike. But this bifurcation has a self-similar quality: when we zoom in on the abstract logic of the scientific side, we find that logic too is constituted by scientific and technological aspects. Each specialized notation is a

⁴[17, 88]. Other efforts to situate computing in longer histories of writing include Mark Priestley’s study of flow diagrams [58] and Jean Lasségue and Giuseppe Longo’s reflections on the relationship between the invention of vowels and modern formal languages [44]. The emerging field of critical code studies represents an effort to study code as text hermeneutically; see for example [52].

tool.⁵ The many competing symbolisms through which the study of logic became mathematical in nineteenth-century Europe were technologies for specifying how people would interact with marks on paper.

The first author to spark sustained dialogue around a mathematical approach to logic was the English schoolmaster and self-taught mathematician George Boole (1815–1864), a participant in the English school of symbolical algebra. The mathematical culture that flourished in England in the first half of the nineteenth century embraced unprecedented levels of abstraction in mathematical concepts. The mathematician George Peacock (1791–1858), a leader of this movement, understood algebra to be fundamentally a science of symbols, one that had only incidentally been so far concerned primarily with numbers.⁶

In 1847, Boole applied this way of thinking to the ancient tradition of formal logic. Evidently drawing on Peacock, though not citing him, Boole proposed that calculus and mathematical analysis were fundamentally techniques for using written symbols, usually but not essentially connected to numbers.⁷ He suggested that “the definitive character of a true Calculus [is] that it is a method resting upon the employment of Symbols”; the fact that those symbols had so far tended to represent numbers was a historical accident, not an essential characteristic of mathematical analysis [6, 4]. He proceeded to assign logical rather than quantitative meanings to the symbols of ordinary algebra, with some corresponding modifications to their rules of manipulation. Using existing typographical conventions for representing addition, subtraction, multiplication and division of known values and variables, he constructed a “mathematics of the human intellect” [6, 7]. In 1854 he revised and extended his system in what became his most famous work, *An Investigation of the Laws of Thought* [7].

Mahoney diagnosed a tendency to “prematurely unite [the computer’s] multiple historical sources into a single stream, treating Charles

Babbage’s analytical engine and George Boole’s algebra of thought as if they were conceptually related by something other than 20th century hindsight” [49, 116]. While the specter of unwarranted anachronism is certainly real, Mahoney overstated his case. At the very latest, Boole’s and Babbage’s specific projects were united in the hindsight of the 1860s. As Desmond MacHale and Yvonne Cohen have recently related, the two men met briefly in 1862 at the International Exhibition in London [48, 182–6]. Boole’s longtime friend, a solicitor named Joseph Hickson Hill (1812–1903), described the meeting a few years later in a letter to Boole’s daughter MaryAnn written shortly after George’s death. He reported that Babbage explained and demonstrated one of his Difference Engines at length, “so there was quite a lecture given by Prof. Babbage to Prof. Boole.” It is unclear whether either professor connected Boole’s logic to the mechanization on display just then, but Hill certainly did at the time of his recollection:

As Boole had discovered that the brains of reasoning might be conducted by a mathematical process, and Babbage had invented a machine for the performance of mathematical work, the two great men together seemed to have taken steps towards the construction of that grand prodigy—a calculating machine.⁸

MacHale and Cohen conclude that this letter “refutes for example those historians who believe that Boole was not thinking along the lines of a calculating machine or a computer,” judging this leap to be too much for Hill to have made on his own [48, 184]. Whoever the originator and regardless of Boole’s own interest in the connection, Hill’s letter is evidence that the possible union of Boole’s logic and Babbage’s engine did not elude the Victorian imagination.

Nor indeed did that union remain in the realm of the imaginary. Also in the 1860s, the political economist and philosopher of science William Stanley Jevons (1835–1882) reduced it to practice by designing “an analytical engine of a very simple character,” based on Boole’s logic, “which performs a complete analysis of any logical prob-

⁵Ursula Klein influentially described chemical notations as “paper tools” [40]. For other studies of notations as tools in the history of science, see [38], [42], [32], [19].

⁶See [41], [59], [21], [43].

⁷On Boole’s relationship to his mathematical context, see [54], [20].

⁸Joseph Hickson Hill to MaryAnn Boole, 24 May 1866, in [48, 182–6, at 183–4].

lem impressed upon it” [34, 500]. As his use of the phrase “analytical engine” suggests, Jevons was inspired by Babbage, whose difference engine he admired and whose plans for a more general analytical engine he believed had “shown that material machinery is capable, in theory at least, of rivalling the labours of the most practised mathematicians in all branches of their science” [34, 498]. Jevons drew a contrast between the promising mechanical devices of mathematics and the near total absence of such devices in the study of logic. Boole, he argued, had made it possible at last for logic too to be automated. “The ancient syllogism,” Jevons explained, “was incapable of mechanical performance because of its extreme incompleteness and crudeness, and it is only when we found our system upon the fundamental laws of thought themselves that we arrive at a system of deduction which can be embodied in a machine acting by simple and uniform movements” [34, 499]. By demonstrating “that it was possible, by the aid of a system of mathematical signs, to deduce the conclusions of all these ancient modes of reasoning, and an indefinite number of other conclusions,” he believed Boole had brought such a mechanical system of deduction within reach [34, 499].

Yet Jevons also saw much to criticize in Boole’s symbolic system. To cast logic as algebra was controversial to begin with, and even those who applauded the project disagreed about the notational specifics of how it should work.⁹ Jevons accused Boole of having “shrouded the simplest logical processes in the mysterious operations of a mathematical calculus,” such that the “intricate trains of symbolic transformations ... can be followed only by highly accomplished mathematical minds” [34, 499]. Given that logical deduction is not the exclusive domain of mathematicians, a system of logic should not require any specialized mathematical literacy. For if Boole’s logic “were the true form of logical deduction,” Jevons argued, then “only well-trained mathematicians could ever comprehend” the logical thinking upon which all of humanity’s “existence as superior beings depends” [34, 500]. Notation was the realm of Boole’s decisive con-

tribution but it was also, according to Jevons, the arena for further progress. Jevons therefore designed a formal inscriptive system embodying the same fundamental laws as Boole’s but employing far fewer mathematical symbols and operations in a bid to claim greater accessibility [33].

Inspired by Babbage’s machinery and Boole’s inscriptive technique, Jevons fashioned not just a new written system but several physical machines based on it. In 1865 he began work on “a reasoning machine, or logical abacus, adapted to show the working of Boole’s Logic in a half mechanical manner.” This machine “consist[ed] merely of a number of slips of wood with sets of letters or terms upon them, with little hooks by which they can be readily classified in any order. This classification represents the processes and results of reasoning.”¹⁰ The mechanization was partial insofar as it lay more in formal rules for the human user of the machine than in physical mechanism. Leaving the work of rule-bound classification to be done by hand, the machine mechanized logic in basically the same sense as did a rule-bound symbolic system on paper.

Soon Jevons designed a more ambitious machine that actually mechanized the spatial manipulation of letters and terms. He hired “a young clockmaker in Salford” to build it according to his design, but similarly to many designers of mathematical machines, he failed to trust the skilled craftsman on whom he relied. Matthew Jones, in his study of calculating machines, has emphasized mathematicians’ recurring inability to collaborate productively with skilled artisans [37]. Jevons similarly deemed it “necessary for me to go there almost every day to see that he is getting on right. I find it necessary to have each step of the work done separately in order that I may see whether I have planned every thing rightly.”¹¹ It seems the first machine they produced was not satisfactory, but collaborative difficulties notwithstanding, by the next year Jevons was highly optimistic about their progress on yet another new machine “in appearance like a large accordion or a very small piano, & has 21 keys exactly like white piano

⁹On the often contentious development of competing systems and notations, see [51], [19].

¹⁰W. S. Jevons to Herbert Jevons, 25 May 1865, [35, vol. III, pp. 68–9, at 69].

¹¹W. S. Jevons to Herbert Jevons, 25 September 1867, in [35, vol. III, pp. 157–9, at 157].

keys.”¹² He reported to the Royal Society:

By merely reading down the premises or data of an argument on a key board representing the terms, conjunctions, copula, and stops of a sentence, the machine is caused to make such a comparison of those premises that it becomes capable of returning any answer which may be logically deduced from them. ... The actual process of logical deduction is thus reduced to a purely mechanical form [34, 500].

Jevons would later explain to the logician John Venn (of Venn diagram fame), it was “rather a large & awkward thing” that “will not bear traveling” and was often “in bad order.”¹³ But to Jevons it proved nonetheless that a machine was in principle capable of performing deductions from premises.

Jevons doubted the clockmaker’s ability to implement his logic correctly, but trusted that the finished machine truly performed logical deduction. This pairing of objectification with anthropomorphism maintains the resonance with the designers of mathematical calculating machines: Jones shows “how often such tools have been mistakenly understood to be proxies, things capable of substituting for human beings,” and “likewise ... how often genuine proxies, skilled artisans, have been mistakenly understood to be tools” [37, 15]. For Jones this confusion is emblematic of the dissolution of the concept of skilled making into a tenuous separation of intellectual design from material implementation, and indeed Jevons seems not to have appreciated the intellect involved in physically crafting the machine he had imagined.

Jevons was also subject to the converse criticism, that the purportedly intellectual process for which he had designed a material mechanism was itself already—and more effectively—a material instrument. His chief critic from this angle was Venn, who though seemingly interested in borrowing the machine, was also on record as skeptical of its worth. In the article that introduced his

famous diagrams, Venn avowed, “I have no high estimate myself of the interest or importance of what are sometimes called logical machines” [65, 15]. He challenged the idea that what machines could do actually encompassed much of the work of logic:

It is but a very small part of the entire process which goes to form a piece of reasoning which they are capable of performing. For, if we begin from the beginning, that process would involve four tolerably distinct steps. There is, first, the statement of our data in accurate logical language... Then, secondly, we have to throw these statements into a form fit for the engine to work with—in this case the reduction of each proposition to its elementary denials. ... Thirdly, there is the combination or further treatment of our premises after such reduction. Finally, the results have to be interpreted or read off [65, 15–16].

Only the third of these steps, Venn predicted, could ever be accomplished by a machine. So much of the hard work of doing symbolic logic was the inscriptive labor of writing the problem the right way in the right symbolic system. Once translated, the problem could be solved by a machine, but it could also be solved by Boole’s algebraic methods, or more easily using one of Venn’s diagrams. “So very little trouble is required to sketch out a fresh diagram for ourselves on each occasion,” Venn argued, “that it is really not worth while to get a machine to do any part of the work for us” [65, 16]. A paper-based inscriptive technique was in Venn’s view a logical technology superior to any mechanical device. Whether the symbolic rules were carried out by hand or by mechanism, only a human logician, he contended, could attain the multiple literacies needed to frame a problem in prose, convert prose to symbolic notation, manipulate symbolic expressions to be suitable for mechanical processing, and interpret the results of that processing back into prose. He conceded that a machine could be capable of executing the rules of a given inscriptive system, but the work of

¹²W. S. Jevons to Herbert Jevons, 23 June 1868 [35, vol. III, p. 185].

¹³W. S. Jevons to John Venn, 26 March 1876, C45, John Venn papers, Gonville & Caius College Archive, Cambridge, United Kingdom.

reasoning lay in traffic between systems.¹⁴

The Victorian algebraic logicians perceived and discussed a close relationship between symbolic systems and mechanization, but they did not straightforwardly see logic as the science to be applied to a new technology. Boole, Jevons, and Venn each introduced their own new and supposedly improved symbolic tools for the implementation of logic on paper, and as mathematical logic developed, notations continued to proliferate. Enthusiasts and skeptics of logic machines agreed that writing was central to the work being done; the symbols were already tools and the question was whether a device of wood and metal offered meaningfully more mechanization than what was already accomplished by pen, paper, and the right notational practices.

MAKING CODE COMMONPLACE

In the mid twentieth century, the architects of electronic computing similarly considered the labor of formal writing. Popular lore has it that the difficulty of program preparation came as something of a surprise. The celebrated 1945 *Draft Report* focused almost entirely on the internal workings of the proposed EDVAC machine, reflecting an assumption that writing a program for a given task would be a simple matter [67]. As Mark Priestley has shown, however, the appearance of surprise is exaggerated by the fact that John von Neumann's extensive work on programming in 1945 happened not to be reproduced in the particular document that ended up circulating so widely [57]. The brief, incomplete discussion of code at the end of the *Draft Report* obscures the ways its authors were already thinking deeply about the inscriptive work of preparing programs for a new machine.

While academic computer scientists would eventually construct a prestigious intellectual lineage linking the researches of Boole and his interpreters to the theory of programming, conceptual continuity was accompanied by a more

mundane similarity between kinds of work.¹⁵ I argue that the inscriptive practices of early digital computing, while not always overtly connected to logic, came to resemble mathematical logic in their growing reliance on proliferating symbolic systems. That resemblance likely involved a mixture of convergent evolution and causal influence, but even programmers who did not see their work as related to logic still contributed to the development of a discipline fundamentally concerned with the power of notation, which in turn made appeals to mathematical logic as an intellectual lineage increasingly plausible. By the end of the 1940s it was widely acknowledged that the work of writing programs was considerable, in terms of both tedium and real difficulty. In the well-known words of Cambridge computing leader Maurice Wilkes, "the realization came over me with full force that a good part of the remainder of my life was going to be spent in finding errors in my own programs" [69, 145]. As Venn had observed of Boole's logic, so the early programmers quickly discovered: once equipped with a machine that could perform complex symbolic manipulations, writing the problem in just the right manipulable way demanded serious effort.

Wilkes played a major role in shaping the digital computing community's response to the challenge. In 1951, he, David Wheeler, and Stanley Gill published perhaps the central document of early programming, the textbook *The Preparation of Programs for an Electronic Digital Computer, with Special Reference to the EDSAC and the Use of a Library of Subroutines* (hereafter WWG).¹⁶ Wilkes was not particularly interested in mathematical logic, and logic plays no significant role in WWG. My purpose here is not to locate logic itself in what would be an unlikely source, but rather to chart how computing practice, in the hands of researchers unconcerned with mathematical logic, became increasingly reliant on expansive uses of formal literacy. This investment in novel inscriptive techniques gave computing work a practical resemblance to mathematical logic that other researchers would embrace, as we will see in the following section.

¹⁴Half a century earlier, Babbage had articulated a similar division of mathematical analysis into three steps: translation into "the language of analysis," the operations of analysis itself, and translation of an answer back into ordinary language [2, 22]. But whereas Venn considered the stage of symbolic manipulation to be only a very minor source of labor in logic, Babbage's engines were premised on the idea that in mathematics such operations amounted to a formidable and costly workload.

¹⁵For one such construction of an intellectual lineage, see [14].

¹⁶[70]. On the textbook's significance, see [10], [55, chapter 7].

WWG's purpose was to share "methods of preparing programs ...developed with a view to reducing to a minimum the amount of labor required," and thereby lower the threshold of practicality for using the computer to solve real problems [70, preface]. At the time there was no doubt that solutions to the problem of programming would be local—each electronic computer was unique—though the authors believed local strategies could follow universal structures. A tension between universality and locality was evident already in the textbook's title: it concerned work with computers in general, but the entire text was presented with respect to the specific machine in use at Cambridge, the EDSAC. "The methods are described in terms of the code of orders used in the EDSAC," they noted, "but for the main part they may readily be translated into other order codes" [70, preface]. On the one hand, they assumed each machine will involve its own dialect in the form of a locally specific order code. On the other hand, they took this work of translation to be manageable, and certainly worthwhile in exchange for access to techniques developed elsewhere.

Indeed the heart of the textbook was a set of practices for managing and reducing that work. "The labor of drawing up a program for a particular problem is often reduced," they wrote, "if short, ready-made programs for performing the more common computing operations are available" [70, 1]. The idea of preparing and saving such ready-made programs, called subroutines, was already in circulation, but WWG is widely credited with systematizing and disseminating the method of subroutines in a detailed manner.¹⁷ The authors emphasized that the work of translation, whether from an ordinary mathematical description of a task or from an existing program in another machine's order code, need only be performed once for any given piece of code. Through "the establishment of a library of subroutines and the development of systematic methods for constructing programs with their aid," passages of code performing specific tasks could be quoted endlessly in future programs [70, preface]. The library was as important here as the subroutines:

¹⁷On the early history of subroutine-like techniques, see [31], [57, chapter 6].

to significantly reduce programming labor, a computing community need a large selection of subroutines systematically organized and governed by efficient rules for storage and use. By affording easy access to these excerpts of code in multiple inscriptive modes—succinct descriptions, full textual code, and punched tapes—it would facilitate multiple ways of using them.

The subroutine library was first of all a physical thing: a cabinet full of tapes. At the same time it was a social order, insofar as it regimented a set of expected behaviors around the copying and preserving of excerpts of code. In both of these aspects the library was as much an archival enterprise as a logical or computational one. Subroutine tapes were to be retrieved from the physical library, copied onto a program tape, and returned to storage, rather than ever being run themselves. The library's material design and social organization reflected this practice:

Subroutines in the library are punched on colored tape so that they can easily be distinguished from program tapes, which should be white. Several copies of each subroutine are provided and when not in use each copy is rolled in a small cardboard box. The boxes are filed in serial order in a steel cabinet. The master copy of each subroutine is kept under lock and key and is used only when all existing copies of the subroutine are damaged. The master tape is then used to prepare further copies by means of a duplicator. All copies must be checked against the master, by means of a comparator, before being put into the library for general use [70, 43].

The physical and social aspects of the library were closely related. Color indicated a tape's purpose, reinforcing the norm that subroutines were to be copied onto white program tapes rather than run directly from colored tapes. Keeping multiple copies of subroutines allowed researchers to work simultaneously and independently, while the preservation of restricted master copies protected the library from the threat of disorder arising from such independence.

In addition to its material and social func-

tions, the subroutine library was of course also a celebrated conceptual resource that allowed the preparation of programs to become partly an act of collage. Numerous computational processes that were more complex than the operations of a machine's arithmetic unit were nonetheless sufficiently basic to recur frequently in practical problems, for example taking a cube root or a cosine [70, 150–2]. Subroutines allowed a local community to code such processes for their machine once, check the codes for errors, and then deploy them again indefinitely many times without repeating that labor. With a textbook like WWG, even the initial labor of writing the subroutine need not start from scratch, though the machine-specificity of order codes meant that subroutines could not be copied from the textbook exactly as printed. WWG presented a set of exemplars on which other computing communities might model their own subroutines as needed; those local subroutines would in turn form a new library, in the local code, of excerpts ready to be used without requiring additional work.

Metaphors help situate new technologies in established patterns of work, and the library metaphor that became attached to the use of subroutines was apt: the subroutine library was a social institution organized around the collection and ongoing collective use of texts. Mark Priestley has recently shown how subroutines also resembled the well established genre of mathematical tables in their function as resources that stand at the ready for a (human or electronic) computer to consult or call during a longer calculation [56, 129]. The aspects of the subroutine library I want to emphasize are well captured by a third metaphor: the early modern commonplace book. The technique of copying excerpts from one's reading and organizing them under topical headings, called commonplaces, produced over time an instrument for making readily available those passages one might quote or emulate later.¹⁸ I do not claim a direct genealogical connection in the sense of Wilkes or his colleagues drawing inspiration specifically from early modern excerpting practices, though a more diffuse connection is plausible insofar as they operated

in a learned culture descended from the early modern European world of letters. But I am concerned here not with influence but resemblance: a subroutine library functioned like a commonplace book in that it collected passages of code commonly used—which is to say quoted on a program tape—and organized them for easy location in a catalog. While the library metaphor reminds us that subroutines were collected as part of a collaborative social institution, and the table metaphor situates its use in the operations of mathematical practice, the image of the commonplace book recalls the particular inscriptive practices whereby programmers used techniques of compilation to increase programs' reliability and reduce the labor of writing them.

Like the glossing, commentary, or vernacular translation a writer might include in a commonplace book, the descriptive apparatus of the subroutine library was crucial to its usefulness. WWG recommended that reference material operate in multiple notational registers. Depending on the situation, different levels of detail were useful to a program writer seeking a subroutine:

The library catalog used in the Laboratory is drawn up in two sections. One gives a concise specification of the purpose of each subroutine together with sufficient information to enable a programmer to make use of it; this includes information about the operating time and storage space occupied. The second section gives the orders of each subroutine in full [70, 25].

Unlike a library in the usual sense, which can usefully contain texts understood to make deeply flawed arguments, all subroutines in the library were understood to have been rigorously checked, so in practice a program writer needed only to know what a given subroutine did, having reason to trust that it did so accurately. A program writer could use this exemplary passage of code without necessarily possessing the literacy to inspect it meticulously. But a writer who wanted to know a subroutine's exact content was best served by a layer of literacy nested between the catalog's succinct description and the barely scrutable punched tapes that existed in the library's steel cabinet. Hence the second section provided entire sub-

¹⁸See [5], especially 69–74. On related practices in early modern mathematical study, see [60], [68].

routines in formal notation representing EDSAC's order code. In principle a writer could also have deciphered holes on tape if necessary, but they would likely have been much more comfortably literate in the order code as rendered in symbols on paper. Similarly to mathematical logic, though not due to its influence, the preparation of programs relied on the affordances of multiple notational systems variously suited to describing a mathematical process, or recording a machine-specific subroutine for a human reader, or copying that subroutine to a program tape.

Technologies of collection aid local research communities in claiming mastery over nonlocal domains. The subroutine library, originally a physical cabinet existing at a specific location, was the node at which one could pick a succinct description of a universally relevant mathematical task and find it already translated into a sequence of locally specific order codes, both textually and on punched tape.¹⁹ WWG presented a vision in which every computing laboratory would build up its own particular subroutine library, an idiosyncratically selected and organized collection of exempla akin to an individual commonplace book. The initial work of writing a subroutine, i.e. translating an operation described in ordinary mathematical notation into the EDSAC's order code, was considerable, but it only needed to be performed once for a given operation. These operations were conceptually legible to mathematical workers anywhere. The contents of the library—like those of any commonplace book—were units whose significance was easily recognizable to a much wider intellectual culture, despite being inscribed in a local code that was (perhaps analogously to poor handwriting) largely illegible to outsiders. The catalog also ensured that it was not necessary to decipher and translate a punched tape or even a list of orders in textual form back into ordinary mathematical notation in order to understand a subroutine's purpose and copy it into a larger program.

The analogy between commonplace books and subroutine libraries illuminates the reciprocal relationship between novel technologies and the uses of writing. What Wilkes, Wheeler, and Gill

faced was not an entirely new problem brought on by an unprecedented technology. Like the first readers of print centuries before them, they confronted and in turn shaped new technology, equipped already with adaptable literary and organizational practices. As Ann Blair has shown, the use of commonplace books and related practices to manage a perceived excess of written material predated the printing press and shaped the genre of print reference works that eventually became popular [5]. Readers' and writers' practices do not change only in reaction to technological prime movers such as the press or the computer; literate techniques already in common use have served as resources for grasping and even influencing technological change. The commonplace book was a tool for assembling ideas and information rendered in prose, whether by pen or by type. Subroutine libraries deployed similar organizational practices to collect written excerpts as instruments, facilitating the future use of passages of code in ways the required less direct application of formal literacy. Despite their differences, these technologies of collection both coupled material and social practices to aid human beings in circumventing the labor of reading and writing.

The preparation of programs was fundamentally a kind of writing. The work was not, in WWG's influential formulation, closely linked to logic. As Priestley has observed, and in contrast to Wilkes and his colleagues, some leading researchers during this period did invoke analogies between logic and computing, despite the fact that "machine code programs and the instructions they contain bear little resemblance to the sentences of propositional and predicate logic" [55, 174]. Priestley concludes that the analogy likely depended upon "an understanding of machine code as a formal language" [55, 175]. I propose that this understanding may have emerged as much through pragmatic emulation in tricks of writing as through the positing of an intrinsic connection to logical formal languages. The work involved in programming and the decisions that shaped a given system of order codes were matters of using a new kind of literacy, and it was fairly well known that symbolic logicians had been thinking deeply about the uses of formal literacy for the past century. Even in the absence of

¹⁹On the discourse around translation as "the central metaphor used to make sense of the activity of programming" in the early 1950s, see [53, 47].

articulated links between logic and computing in WWG, we observe nonetheless a practical similarity between endeavors deeply concerned with translating problems in and out of symbolic systems in a context where such systems were proliferating. Logic's theoretical value aside, some computing researchers soon made this practical similarity more concrete, emphasizing the logicians' willingness to rethink and reinvent their ways of writing.

ADDRESSLESS BUT LOCAL

By the second half of the 1950s, many computing practitioners sought to mitigate the work of writing programs by developing more convenient systems of coding.²⁰ My focus here is on Charles Hamblin (1922–1985), an Australian philosopher who explicitly observed that mathematical logicians had been fruitfully innovating symbolic systems for over a century, and considered these inscriptive tools worthy of programmers' consideration. Computing was not the first science to consist fundamentally in the development and use of elaborate new formal literacies; symbolic logic's tradition of notational invention offered a promising opportunity for borrowing.

Hamblin had gained electronics experience as a radar officer for the Royal Australian Air Force during World War II, and through his philosophical training was well versed in formal logic. He was a Lecturer in Philosophy at New South Wales University of Technology when the university installed UTECOM, an English Electric DEUCE computer, in 1956. Hamblin became an enthusiastic early user of UTECOM, though also a critical one.²¹ Hamblin believed formal logic offered the key to making coding systems less cumbersome—not through abstract theory, but through a system of writing.

In 1957, at a conference on data processing and automatic computing machines held at the Weapons Research Establishment in Salisbury, South Australia, Hamblin described a structure for order codes modeled on a modified version of a well known but still rather marginal logical notation, Jan Łukasiewicz's parenthesis-free

symbolism.²² Hamblin's adaptation, the so-called Reverse Polish Notation (RPN), is a case of logic having shaped early computing specifically through notation.

Beyond the significance of the individual case, Hamblin's creation illustrates the practical resemblance I describe between logic and programming. Further research is necessary to determine how common similar cases of specifically notational influence were. But many computer workers were at least familiar with mathematical logic, and I will show that Hamblin was in any case not unique in perceiving the relevance of logical notation.

The sentiment that programming was too difficult and too tedious was already widespread. Hamblin could take as given that programming presented a great amount of work, stating, "I suppose everyone would agree that programming a modern digital computer is a laborious and sometimes extremely complicated task" [27, 121.1]. By 1957 the nascent idea of programming languages presented an increasingly popular answer to the challenge.²³ Hamblin would eventually embrace that idea, but at the time he voiced skepticism about these "so-called 'interpretive' schemes and ... secondary instruction codes," dismissing this approach as "liable to be extremely wasteful of machine time, and even in some cases to waste the programmer's time as well" [27, 121.1]. Instead he held fast to the strategy outlined in WWG, which was sufficiently standard by 1957 to require no citation or explanation. The immediate answer, Hamblin argued, was that "a computing organisation must simply build up its library of programs ... and look to its efficiency in adapting them" [27, 121.1]. And looking beyond the immediately practical techniques of the subroutine library, Hamblin hoped that future machines might reduce programming labor by employing better designed order codes in the first place (rather than relying on programming languages). In this paper he described a coding system that he had implemented as a pseudo-machine on UTECOM. He clarified, however, that it was "rather wasteful of machine time" and

²⁰The most influential such efforts led to the rise of what became known as "programming languages"; see [53].

²¹Biographical data for Hamblin are from [1]. UTECOM is described in [62].

²²The system first appeared in print in [46]; a more systematic presentation is [47].

²³On the emergence of a linguistic conception of machine-independent coding, see [53].

he “would not seriously put it forward in this form for general use” [27, 121.6]. His real question was “How feasible is it to build a machine embodying this kind of code?” and his answer—admittedly suggestive rather than definitive—was that it could probably be done “without much real difficulty” [27, 121.7].

The specific problem Hamblin sought to solve in this paper was one of placement: writing programs involved deciding where in the machine numerous numbers should be stored. Order codes always included references to locations in the machine’s storage. On the EDSAC, for example, one coded “A n” to “Add the number in storage location n into the accumulator,” or “T n” to “Transfer the contents of the accumulator to storage location n and clear the accumulator” [70, 5]. Writing a program required not only breaking the complex process down into many precise steps, but also juggling the placement of all the data and instructions involved in those steps. Assigning storage locations represented an even greater burden on the UTECOM that Hamblin used. As a DEUCE computer, it followed the particularly demanding Pilot ACE design, developed at Britain’s National Physical Laboratory and based on Alan Turing’s ideas about using sophisticated programming to allow for maximally simple hardware.²⁴ In contrast to simple EDSAC orders like “A n” or “T n,” every order on the Pilot ACE was a 32-binary-digit word, with 26 digits broken down into 7 meaningful elements designated “N S D C W T G” (and 6 digits in each instruction word going unused).²⁵ These complicated instructions required a user to specify the location of the next order to be followed (in addition to the usual specifications concerning the storage of data). The capability to do so opened the possibility of “optimum programming” that took into account not only the mathematical tasks to be performed but also the most efficient spatial placement of orders with respect to the timing of the machine’s internal mechanical processes. The programmer gained the power and hence a responsibility to store every order in an optimal location for speedy

retrieval at the moment it would be needed.²⁶

Programming UTECOM was thus especially difficult, but Hamblin framed the problem as one common to programming in general. He lamented the necessity of worrying about the physical location of information at all, let alone with the degree of care required for optimum coding. Even in a simpler order code, managing addresses added another layer of work to the already considerable task of programming a mathematical process. He declared that ordinary mathematical notation was “obviously the most convenient [code] for the programmer,” but, because machines required instructions in different order codes,

The remainder of the task of the present-day programmer is essentially a task of translation from one already adequate language into a radically different one, which is at the same time vastly more redundant in that he must make many decisions regarding storage addresses of both numbers and instructions [27, 121.1].

Whereas existing mathematical notation had the advantages of being already well known and having “evolved by natural selection into an extremely efficient language,” the order codes into which programmers translated were unfamiliar and without pedigree [27, 121.1]. Littered as they were with numerous addresses the writer needed to keep track of, they imposed an aspect of labor without analogy in ordinary mathematical notation. Thus not only was translation demanding, but the work of writing programs consisted in translating out of an efficient language into one that taxed the writer more.

Wilkes, Wheeler, and Gill had mentioned that the handling of addresses was a variable feature of order codes, but concluded that there was little difference between codes employing different numbers of addresses. They briefly noted the so-called three-address system used on some machines, in which an order specified an action to be taken on the numbers in two locations and also dictated a third location in which to store the result. They argued that while such order codes could save space compared with single-

²⁴On Turing’s philosophy of hardware and programming, see [30, 399–403].

²⁵For an introduction to this complex system of instructions, see [71].

²⁶On the differences and tradeoffs between contemporaneous programming schemes in early British computing, see [11].

address systems when coding operations on two arguments, three-address codes quickly lost that efficiency if more arguments were required. Thus they reasoned, “The decision as to whether a machine should have a single-address or a three-address code should rest rather with the designer than with the prospective mathematical user” [70, 11–12]. They also acknowledged the possibility of a four-address code, as in the Pilot ACE, which would allow for optimum coding. Conceding possible gains in efficiency, however, the authors held that this complication “confuses the essentially arithmetical nature of programming stressed this book” [70, 12]. But Wilkes, Wheeler, and Gill had only compared one, three, and four address systems; they did not entertain the possibility—realized by Hamblin’s RPN—of a code that avoided addresses altogether.

Hamblin found the key to eliminating the need for addresses not in the axioms and theorems of logic, but rather in logic’s notations. “In the last hundred years or so,” Hamblin wrote, “a good deal of attention has been devoted by formal logicians to the study and invention of symbol-systems for logical purposes. Not very much of this is immediately relevant here, and work on machine-codes uncovers its own problems and possibilities; but one logical notation in particular I should like to mention” [27, 121.2]. Hamblin did not consider computing to be a straightforward application of theoretical logic and he recognized that writing for computers was its own kind of work. It was logicians’ preoccupation with notation that interested him, in particular an inscriptive innovation associated with the local culture of logic in Warsaw.

Between the World Wars, the capital of newly independent Poland had been home to a vibrant culture of mathematical logic. Among the Warsaw School of Logic’s most recognizable contributions was the so-called parenthesis-free symbolism, a notation invented by one of its leaders, Jan Łukasiewicz (1878–1956). Unlike other logical notations that made liberal use of symbols from mathematics and even farther afield, Łukasiewicz restricted his notation to strings of upper- and lowercase letters. Capitals represented functors and were followed immediately by their arguments, which could be lowercase variables or other functors. For example, Cpq represented

“If p , then q ,” and Apq meant “ p or q .” Functors could combine to produce compound propositions, such as “If p or q , then r ,” written $CApqr$. The system could dispense with parentheses because the order of operations was unambiguous: every functor applied to the arguments immediately following it. Łukasiewicz presented his notation as a universally useful contribution to symbolic logic, but logicians outside Poland tended not to adopt it; instead it became a locally distinct technique of the Warsaw School [18].

In the 1950s, several computing researchers independently looked to Łukasiewicz’s notation for a useful way of writing machine instructions. Researchers affiliated with the University of Michigan and the Burroughs Adding Machine Company in Detroit proposed in 1952 “a new method for the evaluation of truth-functions ... which does seem to be practical for formulas of great length and many variables and which has other features of interest. This new method is based on the Polish notation” [8, 2]. A few years later, colleagues of theirs working for Burroughs in Pennsylvania built a physical device according to their scheme. William Miehle reported on that machine to the Association for Computing Machinery in 1955, explaining,

The formula must be expressed in [Łukasiewicz’s] parenthesis-free notation. In this notation, a formula is written and scanned from right to left, and instead of writing an operator *between* the variables such as $(p \text{ dot } q)$, it is written to the *left* (Kpq). Its advantages are that no parentheses are needed and that it can be mechanized as will be shown [50, 189].

The matter-of-fact prescription to write and scan from right to left is striking: Łukasiewicz had never suggested such a practice, and to evaluate his notation in this direction would not illuminate a proposition’s structure in the way a human logician typically hopes to. But if every symbol is to be mechanically evaluated, it is indeed efficient to do so in the order that, according to Łukasiewicz’s design, is backwards. That these Burroughs workers independently developed a system so similar to Hamblin’s indicates that he was not alone in seeing logic as a notational

resource for the work of writing programs.

Hamblin's paper a few years later independently suggested what amounted to an equivalent inscriptive strategy, and it was his formulation as "Reverse Polish notation" that became well known. The same efficient sequencing as scanning right to left could be attained by writing the symbolism "backwards" in the first place and scanning left to right. "It is perfectly feasible," he wrote, "to use a 'reverse Polish' notation in which the operators follow the operands: i.e. in place of $a + b$ " we can write $ab+$, and in place of $(a + b) \times c$ " we can write $ab + c \times$ " [27, 121.2]. This seemingly simple modification of symbolic convention allowed for a powerful new approach to order codes.

The possibility of avoiding addresses was intrinsic to the very arrangement of the letters of RPN on the page. Hamblin explained:

It is now not very difficult to demonstrate that each symbol of a formula can be regarded as a machine instruction. Let us imagine we have a number of storage locations arranged in a linear order and reserved as working-space in connection with the arithmetic unit. I shall refer to these locations collectively as "the accumulator" and to individual locations as "cells." When a number is required for an arithmetic operation it is placed in the *first vacant cell*, ...and when a diadic operation such as addition or multiplication is carried out it is always on the numbers most recently transferred in, i.e. the numbers the last two occupied cells. ...Now the result of any formula in "reverse Polish" notation, interpreted as a sequence of instructions, will be to calculate the number represented by the formula and leave it in cell 1 [27, 121.2].

The machine thus performed the intended calculation without ever once needing to be told where to find a piece of data. Therefore the writer of the program wasted no effort picking suitable storage locations for values obtained along the way. Rather, at every step, the relevant numbers were always precisely the two most readily available in what Hamblin called the "running accumula-

tor." To formulate the mathematical expression in RPN sufficed to manage all the data relevant to the calculation. This system had "the advantage, implicit in the mathematical symbolism, of permitting intermediate results to be 'held' pending the calculation of addition terms"—held, that is, without the human writer needing to specify an address for their holding [27, 121.3].

Several researchers arrived independently at this design principle of storing symbols on a "last-in-first-out" basis, a structure that came to be known as a "stack." I have focused on Hamblin in order to emphasize how he quite explicitly presented this structure as "implicit in the mathematical symbolism," but the other inventors of the stack concept were similarly well-versed in logic and attentive to the details of symbolic systems. Alan Turing, a towering figure in logic whose most famous famous logical contribution depended fundamentally upon the written representation of machines that write, arguably described the stack concept as early as 1945.²⁷ Perhaps the most influential articulation of the stack concept was Klaus Samelson and Friedrich L. Bauer's 1960 paper proposing the use of what they called a "cellar" in the automatic translation of ALGOL into machine language [61]. Though they did not discuss mathematical logic in the article, Bauer later traced the idea directly to his earlier engagement with Łukasiewicz's notation [4, 27–32]. Robert S. Barton, who spearheaded the use of stack architecture in the design of the Burroughs B5000 from the late 1950s, also claimed to have conceived of the potential application to computing immediately upon his own chance encounter with Łukasiewicz's notation [24, 49], though he cited Samelson and Bauer in the 1961 paper detailing his approach [3, 12].

Together with these independently developed variations, Hamblin's Reverse Polish Notation achieved wider circulation among computer scientists than Łukasiewicz's original had among logicians outside Poland. Despite his skepticism about programming languages, Hamblin prepared a "Programming and Operation Manual" facilitating use of his Reverse Polish pseudo-machine, now dubbed GEORGE (General Order Genera-

²⁷For an analysis of the report in which Turing described a stack-like process of "burying and disinterring" instructions [64, 383], see [9]. Turing's classic paper on his "machines" is [63].

tor), as an “instruction language” for DEUCE [28], which caught on among users at the University of New South Wales [66]. Meanwhile at least one representative of English Electric had been present for Hamblin’s presentation at the Weapons Research Establishment, and the company soon followed his ideas in developing their zero-address KDF-9, available from the early 1960s.²⁸ Reverse Polish Notation went on to find perhaps its largest audience through its adoption as the input system for Hewlett-Packard calculators [25].

WWG had consolidated an image of programming as a fundamentally a difficult kind of writing, one fit to be managed by novel inscriptive practices. Hamblin and others, by turning to mathematical logic’s own long tradition of inscriptive innovation, found new techniques for reducing that inscriptive labor. The spatial relationship of letters and symbols in the so-called “Polish” notation for logic, once reversed, provided a solution to a challenge inherent in the writing of programs for physical machines, namely the efficient retention and recalling of relevant data over the course of a complex computation.

CONCLUSION

From its mid-nineteenth-century inception, mathematical logic depended crucially on novel inscriptive techniques. Logicians reimaged logic as fundamentally a matter of rewriting problems in a symbolic system, and as they worked, such systems proliferated. Moreover, Victorian observers already saw the affordances of symbolic techniques as bearing directly on projects of mechanization.

With the advent of electronic computing, the first programmers faced similar challenges. Mathematical problems needed to be rewritten to be treated by machines, and this rewriting demanded an enormous amount of work. Managing this labor meant leveraging the affordances of different systems of writing to save effort wherever possible, using systems strategically to mediate between local specificity and mathematical universality. Each machine was built around

²⁸[45, 76]. An audience question from a Mr. R. Davis of English Electric Co. is recored at [27, 121-12]. On the KDF-9 see [26].

a particular order code, making the activity of writing programs different at every local site of computing. The method of building, maintaining, and using a subroutine library—like other techniques in the long history of reading and writing practices—enabled a user or community to collect universally legible compositions in locally practical formulations. As in mathematical logic, legibility was a matter of organizing notations to facilitate traffic between the different inscriptive systems appropriate to different uses.

Regardless of whether propositional logic figured in the conceptualization of a given program, the work of writing that program resembled symbolic logic in its focus on translating into, out of, and between formalized inscriptive systems. Computing researchers appreciated the resemblance, and, at least in the case of Hamblin’s RPN, consciously turned to logic as a source of specifically notational creativity. Symbolic logic would increasingly appear to be a plausible theoretical foundation for computer science, but it was also already a practical resource for shaping the work of writing programs.

ACKNOWLEDGMENTS

This work was supported in part by the UK Engineering and Physical Sciences Research Council under grant EP/R03169X/1. I thank Troy Astarte, Liesbeth De Mol, Con Diaz, Michael Gordon, Chris Hollings, Ursula Martin, Máté Szabo, and two referees for insightful comments on previous drafts.

REFERENCES

1. Murray W. Allen, “Charles Hamblin (1922–1985),” *The Australian Computer Journal*, vol. 17, no. 4, pp. 194–5, 1985.
2. Charles Babbage, *On the Influence of Signs in Mathematical Reasoning*, Cambridge: J. Smith, 1826.
3. R. S. Barton, “A new approach to the functional design of a digital computer,” *Annals of the History of Computing*, vol. 9, no. 1, pp. 11–15, 1987.
4. Friedrich L. Bauer, “From the stack principle to ALGOL,” in M. Bray and E. Denert, eds., *Software Pioneers: Contributions to Software Engineering*, Berlin: Springer-Verlag, 2002, pp. 27–42.
5. Ann Blair, *Too Much to Know: Managing Scholarly Information before the Modern Age*, New Haven: Yale University Press, 2010.

6. George Boole, *The Mathematical Analysis of Logic, Being an Essay towards a Calculus of Deductive Reasoning*, Cambridge: Macmillan, Barclay, & Macmillan, 1847.
7. George Boole, *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities*, London: Walton and Maberly, 1854.
8. Arthur W. Burks, Don W. Warren, and Jesse Wright, "Truth-function evaluation using the Polish notation," Project M828, Detroit: Burroughs Adding Machine Co., 1952.
9. B. E. Carpenter and R. W. Doran, "The other Turing machine," *The Computer Journal*, vol. 20, no. 3, pp. 269–79, 1977.
10. Martin Campbell-Kelly, "In praise of 'Wilkes, Wheeler, and Gill,'" *Communications of the ACM*, vol. 54, no. 9, pp. 25–7, 2011.
11. Martin Campbell-Kelly, "The development of computer programming in Britain (1945 to 1955)," *Annals of the History of Computing*, vol. 4, no. 2, pp. 121–39, 1982.
12. Martin Campbell-Kelly, William Aspray, Nathan Ensinger, and Jeffrey R. Yost, *Computer: A History of the Information Machine*, 3rd ed., Boulder: Westview Press, 2014.
13. M. T. Clanchy, *From Memory to Written Record: England 1066–1307*, 3rd ed., Chichester: Wiley-Blackwell, 2013.
14. Martin Davis, *The Universal Computer: The Road from Leibniz to Turing*, New York: W. W. Norton & Company, 2000.
15. Liesbeth De Mol and Maarten Bullynck, "Making the history of computing. The history of computing in the history of technology and the history of mathematics," *Revue de synthèse*, vol. 139, series 7, no. 3–4, 361–80, 2018.
16. Liesbeth De Mol and Giuseppe Primiero, "When logic meets engineering: introduction to logical issues in the history and philosophy of computer science," *History and Philosophy of Logic*, vol. 36, no. 3, pp. 195–204, 2015.
17. Stephanie Dick, "Machines who write," *IEEE Annals of the History of Computing*, vol. 35, no. 2, pp. 85–8, 2013.
18. David E. Dunning, "The logic of the nation: nationalism, formal logic, and interwar Poland," *Studia Historiae Scientiarum*, vol. 17, pp. 207–51, 2018.
19. David E. Dunning, "Writing the Rules of Reason: Notations in Mathematical Logic, 1847–1937," PhD dissertation, Program in History of Science, Princeton University, 2020.
20. Marie-José Durand-Richard, "Logic versus algebra: English debates and Boole's mediation," in *A Boole Anthology: Recent and Classical Studies in the Logic of George Boole*, ed. James Gasser, Dordrecht: Kluwer Academic Publishers, 2000) 139–66.
21. Marie-José Durand-Richard, "Peacock's arithmetic : an attempt to reconcile empiricism to universality," *Indian Journal of History of Science*, vol. 46, no. 2, pp. 251–311, 2011.
22. David Edgerton, *The Shock of the Old: Technology and Global History Since 1900*, Oxford: Oxford University Press, 2007.
23. Elizabeth L. Eisenstein, *The Printing Revolution in Early Modern Europe*, Cambridge: Cambridge University Press, 2012.
24. "Burroughs B 5000 Conference," OH 98, oral history on 6 September 1985, Marina del Ray, California, Charles Babbage Institute, University of Minnesota, Minneapolis, 1986, <https://hdl.handle.net/11299/107105>.
25. G. Goth, "Fans of Hewlett-Packard calculators say 'it all adds up,'" *Computing in Science & Engineering*, vol. 4, no. 2, pp. 5–8, 2002.
26. A. C. D. Haley, 1962. "The KDF.9 computer system," in *Proceedings of the December 4–6, 1962, fall joint computer conference (AFIPS '62 (Fall))*, New York: Association for Computing Machinery, 1962, pp. 108–20.
27. Charles Leonard Hamblin, "An addressless coding scheme based on mathematical notation," *Data Processing and Automatic Computing Machines: Proceedings of a Conference (on Data Processing and Automatic Computing Machines) Held at Weapons Research Establishment, Salisbury, S.A., June 3rd–8th, 1957*, 121.1–121.12, Salisbury, South Australia: Weapons Research Establishment, 1957.
28. Charles Leonard Hamblin, "GEORGE IA and II: A semi-translation programming scheme for DEUCE: Programming and Operation Manual," typescript, University of New South Wales, Kensington, New South Wales, archived at <https://web.archive.org/web/20200404093021/http://members.iinet.net.au/~dgreen/deuce/GEORGEProgrammingManual.pdf>, estimated mid 1958 by David Green, <https://web.archive.org/web/20210105131756/http://members.iinet.net.au/~dgreen/deuce/>.
29. D. R. Hartree, "The Mechanical Integration of Differential Equations," *The Mathematical Gazette* vol. 22, no. 251, pp. 342–64, 1938.
30. Andrew Hodges, *Alan Turing: The Enigma*, Princeton: Princeton University Press, 2014.
31. Grace Murray Hopper, "Keynote address," in Richard L. Wexelblat, ed., *History of Programming Languages*, New York: Academic Press, 1981, pp. 7–20.

32. Evan Hepler-Smith, "Paper Chemistry: François Dagognet and the Chemical Graph," *Ambix*, vol. 65, no. 1, pp. 76–98, 2018.
33. W. Stanley Jevons, *Pure Logic or the Logic of Quality Apart from Quantity: With Remarks on Boole's System and on the Relation of Logic and Mathematics*, London: Edward Stanford, 1864.
34. William Stanley Jevons, "On the mechanical performance of logical inference," *Philosophical Transactions*, vol. 160, pp. 497–518, 1870.
35. William Stanley Jevons, *Papers and Correspondence of William Stanley Jevons*, ed. R. D. Collison Black, 7 vols., London: The Macmillan Press LTD, 1972–1981.
36. Adrian Johns, *The Nature of the Book: Print and Knowledge in the Making*, Chicago: University of Chicago Press, 1998.
37. Matthew L. Jones, *Reckoning with Matter: Calculating Machines, Innovation, and Thinking about Thinking from Pascal to Babbage*, Chicago: Chicago University Press, 2016.
38. David Kaiser, *Drawing Theories Apart: The Dispersion of Feynman Diagrams in Postwar Physics*, Chicago: Chicago University Press, 2005.
39. Friedrich A. Kittler, "There is no software," first published as "Es gibt keine Software," in *Writing/Ecriture/Schrift*, ed. Hans Ulrich Gumbrecht, Munich: Wilhelm Fink, 1993, pp. 367–78, trans. Erik Butler in *The Truth of the Technological World: Essays on the Genealogy of Presence*, Stanford, California: Stanford University Press, 2014, pp. 219–29.
40. Ursula Klein, *Experiments, Models, Paper Tools: Cultures of Organic Chemistry in the Nineteenth Century*, Stanford: Stanford University Press, 2003.
41. Elaine Koppelman, "The calculus of operations and the rise of abstract algebra," *Archive for History of Exact Sciences*, vol. 8, no. 3, pp. 155–242, 1971.
42. Whitney E. Laemmli, "Paper dancers: art as information in twentieth-century America," *Information & Culture: A Journal of History*, vol. 52, no. 1, pp. 1–30, 2017.
43. Kevin Lambert, "A natural history of mathematics: George Peacock and the making of English algebra," *Isis*, vol. 104, no. 2, pp. 278–302, 2013.
44. Jean Lasségue and Giuseppe Longo, "What is Turing's comparison between mechanism and writing worth?" in S. Barry Cooper, Anuj Dawar, and Benedikt Löwe, eds., *How the World Computes: Turing Centenary Conference and 8th Conference on Computability in Europe, CiE 2012, Cambridge, UK, June 18–23, 2012, Proceedings*, Berlin: Springer, 2012, pp. 450–61.
45. Simon Lavington, *Early British Computers: The Story of Vintage Computers and the People Who Built Them*, Manchester: Manchester University Press, 1980.
46. Jan Łukasiewicz, "O znaczeniu i potrzebach logiki matematycznej," *Nauka Polska, jej Potrzeby, Organizacja i Rozwój*, vol. 10, pp. 604–20, 1929.
47. Jan Łukasiewicz, *Elements of Mathematical Logic*, trans. Olgierd Wojtasiewicz, New York: Macmillan, 1963.
48. Desmond MacHale and Yvonne Cohen, *New Light on George Boole*, Cork, Ireland: Cork University Press, 2018.
49. M. S. Mahoney, "The history of computing in the history of technology," *Annals of the History of Computing*, vol. 10, pp. 113–25, 1988.
50. William Miehle, "Burroughs truth function evaluator," *Journal of the ACM*, vol. 4, no. 2, pp. 189–92, 1957.
51. Amirouche Moktefi, "The social shaping of modern logic," in Dov Gabbay, Lorenzo Magnani, Woosuk Park, Athi-Veikko Pietarinen, eds., *Natural Arguments: A Tribute to John Woods*, London: College Publications, 2019, pp. 503–20.
52. Nick Montfort, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample, and Noah Vawter, *10 PRINT CHR\$(205.5+RND(1)); : GOTO 10*, Cambridge, MA: The MIT Press, 2013.
53. David Nofre, Mark Priestley, and Gerard Alberts, "When technology became language: the origins of the linguistic conception of computer programming, 1950–1960," *Technology and Culture*, vol. 55, no. 1, pp. 40–75, 2014.
54. Maria Panteki, "The mathematical background of George Boole's Mathematical Analysis of Logic (1847)," in *A Boole Anthology: Recent and Classical Studies in the Logic of George Boole*, ed. James Gasser, Dordrecht: Kluwer Academic Publishers, 2000, pp. 167–212.
55. Mark Priestley, *A Science of Operations: Machines, Logic and the Invention of Programming*, London: Springer, 2011.
56. Mark Priestley, "The mathematical origins of modern computing," in Sven Ove Hansson, ed., *Technology and Mathematics: Philosophical and Historical Investigations*, Cham: Springer, 2018, pp. 107–35.
57. Mark Priestley, *Routines of Substitution: John von Neumann's Work on Software Development, 1945–1948*, Cham: Springer, 2018.
58. Mark Priestley, "Flow diagrams, assertions, and formal methods," in E. Sekerinski et al., eds., *Formal Methods: FM 2019 International Workshops*, revised selected papers, Part II, Cham: Springer, 2020, pp. 15–34.
59. Helena M. Pycior, "George Peacock and the British

- origins of symbolical algebra," *Historia Mathematica*, vol. 8, no. 1, pp. 23–45, 1981.
60. Renée Raphael, "Galileo's *Two New Sciences* as a model of reading practice," *Journal of the History of Ideas*, vol. 77, no. 4, pp. 539–65, 2016.
 61. K. Samelson and F. L. Bauer, "Sequential formula translation," *Communications of the ACM*, vol. 3, no. 2, pp. 76–83, 1960; reprinted in *Communications of the ACM*, vol. 26, no. 1, 9–13, 1983.
 62. R. G. Smart, "The UTECOM digital computer," *Data Processing and Automatic Computing Machines: Proceedings of a Conference (on Data Processing and Automatic Computing Machines) Held at Weapons Research Establishment, Salisbury, S.A., June 3rd–8th, 1957*, 104.1–104.5, Salisbury, South Australia: Weapons Research Establishment, 1957.
 63. Alan M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–65, 1937.
 64. Alan M. Turing, "Proposed electronic calculator (1945)," in B. Jack Copeland, ed., *Alan Turing's Automatic Computing Engine*, Oxford: Oxford University Press, 2005, pp. 370–454.
 65. John Venn, "On the diagrammatic and mechanical representation of propositions and reasonings," *Philosophical Magazine*, series 5, vol. 10, no. 59, pp. 1–18, 1880.
 66. Robin A. Vowels, "The DEUCE—a user's view," in B. Jack Copeland, ed., *Alan Turing's Automatic Computing Engine: The Master Codebreaker's Struggle to Build the Modern Computer*, Oxford: Oxford University Press, 2005, pp. 297–329.
 67. J. von Neumann, "First draft of a report on the EDVAC," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.
 68. Benjamin Wardhaugh, "Rehearsing in the margins: mathematical print and mathematical learning in the early modern period," in *The Palgrave Handbook of Literature and Mathematics*, eds. Robert Tubbs, Alice Jenkins, and Nina Engelhardt, Cham: Palgrave Macmillan, 2021, pp. 553–67.
 69. Maurice Wilkes, *Memoirs of a Computer Pioneer*, Cambridge, Mass.: MIT Press, 1985.
 70. Maurice V. Wilkes, David J. Wheeler, and Stanley Gill. *The Preparation of Programs for an Electronic Digital Computer, with Special Reference to the EDSAC and the Use of a Library of Subroutines*, Cambridge, Mass.: Addison-Wesley Press, Inc., 1951.
 71. J. H. Wilkinson, "An assessment of the system of opti-

num coding used on the Pilot Automatic Computing Engine at the National Physical Laboratory," *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 248, no. 946, pp. 253–81, 1955.

David E. Dunning is a Postdoctoral Research Associate in History of Mathematics at the University of Oxford. He holds a PhD in History of Science from Princeton University (2020), an MPhil in History and Philosophy of Science from the University of Cambridge (2013), and a BA in Mathematics and English from the University of Pennsylvania (2012). His current book project examines the rise of mathematical logic through the lens of notation, exploring both technical and social aspects of symbolic systems. Contact him at david.dunning@maths.ox.ac.uk.