Flow diagrams, assertions, and formal methods*

Mark Priestley

The National Museum of Computing, Bletchley Park, UK

Abstract. This paper examines the early history of the flow diagram notation developed by Herman Goldstine and John von Neumann in the mid-1940s. It emphasizes the motivation for the notation's mathematical aspects and the provision made for formally checking the consistency of diagrams. Goldstine and von Neumann's introduction of assertion boxes is considered in detail. The practical use of flow diagrams is considered briefly, and the paper then reads Turing's 1949 essay on "Checking a large routine" in the light of his likely knowledge of the Goldstine/von Neumann notation. In particular, his different use of the term "assertion" is considered, and related to the earlier work.

Keywords: Flow diagrams · Assertions · Formal methods

1 Introduction

Flowcharts are one of the defining visual representations of modern computing. Introduced in 1947 by Herman Goldstine and John von Neumann as part of a comprehensive methodology for what they called the "planning and coding of problems", they were a ubiquitous aid to the development of computer programs for at least the next two decades. A wide variety of notations was used, but all forms of the diagrams contained boxes representing operations and decision points, linked by directed line segments representing the flow of control [18].

Despite this ubiquity, historians have questioned the role of flowcharts. Rather than being a significant part of the development process, they were criticized as being burdensome and misleading documentation produced only at the behest of bureaucratically-minded project managers. Ensmenger [5] describes them as *boundary objects*, whose value lay in their ability to mediate between managers and developers, while meaning something different to the two groups.

Given this, it comes as something of a surprise to realize that, for Goldstine and von Neumann, flow diagrams provided not only a graphic representation of program structure but also a sophisticated mathematical notation. They defined a number of formal conditions, akin to what we would now call proof rules, for demonstrating the consistency of a diagram. It is not unreasonable, if slightly anachronistic, to describe the original diagrams not simply as a design notation but as an early attempt to define a formal method for software development.

* I thank Troy Astarte and Brian Randell for the invitation to give the talk at the HFM 2019 workshop on which this paper is based.

Computer scientist Cliff Jones [17, 16] cited the 1947 diagrams as a precursor of Alan Turing's 1949 paper on "checking a large routine" [21]. Jones focused on the term "assertion": Turing defined a number of assertions to be checked by the programmer, and the Goldstine/von Neumann notation included a feature called "assertion boxes" which appeared to allow arbitrary logical formulas to be inserted into the diagram. This appears, in turn, to look forward to the work of Robert Floyd in the mid-1960s: in a paper widely regarded as a milestone in the development of formal methods, Floyd attached propositions to the line segments in flowcharts to provide a basis for constructing formal proofs about the correctness of the program represented by the diagram [6].

This gap of almost 20 years should make us pause and wonder whether the resemblance between early and later work is simply a superficial similarity, or whether there are deeper and more meaningful connections. This paper focuses on the development of the Goldstine/von Neumann notation and examines the motivation for its development and the problems it was supposed to solve. The notation is then used to analyze Turing's 1949 flow diagram, highlighting the similarities and the differences between the two approaches.

2 Block Diagrams

Von Neumann's collaboration with Goldstine began in 1944, when the latter was the US Army's representative on the ENIAC project [12]. Along with ENIAC's designers Presper Eckert and John Mauchly, the pair worked on designs for a successor machine, EDVAC, the first so-called "stored-program" computer. By 1946, however, the team had split up and Goldstine followed von Neumann to the Institute for Advanced Study to work on the electronic computer project there. Flow diagrams were first described in a 1947 project report [11], and Goldstine later gave an outline history of their development:

> In the spring of [1946] von Neumann and I evolved an exceedingly crude sort of geometrical drawing to indicate in rough fashion the iterative nature of an induction. At first this was intended as a sort of tentative aid to use in programming. Then that summer I became convinced that this type of *flow diagram*, as we named it, could be used as a logically complete and precise notation for expressing a mathematical problem, and indeed that this was essential to the task of programming. Accordingly, I developed a first, incomplete version and began work on the paper called *Planning and Coding* [...] Out of this was to grow not just a geometrical notation but a carefully thought out analysis of programming as a discipline. [9, pp. 266-7]

As far as the surviving evidence allows us to judge, this account is quite accurate. A "first, incomplete" version of the notation appears in an unpublished draft of the *Planning and Coding* reports [10]. At this stage, the notations were called *block diagrams* but by the time the first report was published in

1947, significant syntactic and semantic modifications had taken place, and the terminology had changed [11].

They were not the first graphical representations of computer programs. A wide range of notations had been used to document ENIAC programs, including "master programmer diagrams" [12]. Named for the machine's high-level control unit, these box-and-arrow diagrams represented the "steppers", devices which counted loop iterations and controlled the execution of straight-line blocks of operations shown as simple boxes. The diagrams therefore presented the high-level organization of a program, and were capable of showing complex structures of nested loops and conditional branches.

So-called "flow sheets" had been in use since the late nineteenth century to show the flow of materials in processes in industries such as milling [22], and in the 1920s more general "process charts" were proposed as part of a methodology for describing and improving industrial and commercial processes [7]. A 1947 standard [1] distinguished operation from flow process charts, the latter showing the events affecting some material in an industrial process. Ensmenger [5] notes that flow diagrams are sometimes said to have come to computing through this route, thanks to von Neumann's undergraduate studies in chemical engineering, though similar "flow charts" had been independently used on the ENIAC project to describe the processing of decks of punched cards [19].

However, the fact that the diagrams were originally called "block diagrams" suggests an alternative source, namely the use of block diagrams in electronics. These provided a high-level view of the structure of a circuit, and Goldstine and von Neumann's block diagrams similarly presented a high-level view of the problem-specific organization of a computer's memory. As there was no physical flow of material between processes being illustrated, the use of the term "flow" may not have immediately suggested itself (the metaphor of a "flow of control" seems to postdate the adoption of the diagrams). The change in terminology may reflect the evolution, described below, from a kind of memory map to a more abstract representation of the structure of a computational process. In 1946, Haskell Curry and Willa Wyatt [3] had drawn what they called a "flowchart" to describe the structure of an ENIAC program, in which electronic pulses did flow through the machine's wires to control the order of processing, a usage which may have helped legitimize the term "flow diagram".

The ultimate aim of the diagrams was to effect a division of labour in the process of preparing problems for automatic computation:

> We have attempted [...] to standardize upon a graphical notation for a problem in the hope that this symbolism would be sufficiently explicit to make quite clear to a relatively unskilled operator the general outline of the procedure. We further hope that from such a *block-diagram* the operator will be able with ease to carry out a complete coding of a problem. [10]

The process described in the final report was much more complex, but the aim was the same: to bring the work to a point from which the code could

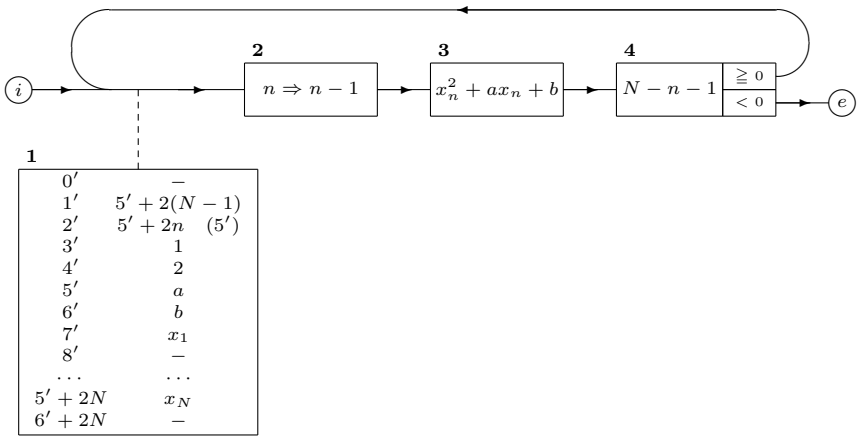

Fig. 1. Block diagram describing the computation of N values of a polynomial [10]. x_n is found in location $5' + 2n$ and the value of $x_n^2 + ax_n + b$ is stored in location $6' + 2n$.

be generated in a straightforward way. The reports contain a range of example problems that are described and coded in detail with the aid of the new diagrams.

In the summer of 1945, von Neumann coded a number of problems to test the code he was designing for the EDVAC. One of these survives, namely a routine to merge two sequences of data [19]. Von Neumann used the familiar technique of "definition by cases" to describe the general outline of the procedure, including alternative courses of action. The introduction of diagrams in 1946 might have been motivated by the belief that diagrams were intrinsically clearer than text for complex problems or that they would be more accessible to "relatively unskilled operators". It may not be a coincidence that ENIAC's first operators recalled using the block diagrams of the machine's electronics as a way coming to understand it well enough to program it.

Fig. 1 shows the first diagram from the draft report, a program to calculate and store N values of the polynomial $x_n^2 + ax_n + b$. Each box in the diagram represents a contiguous area of memory. Box 1 is a *storage box* describing the data manipulated by the program: $0', 1', \ldots$ are symbolic addresses of memory locations whose contents are described using the mathematical vocabulary in which the problem is stated. Boxes 2, 3, and 4 represent instructions whose effect is specified by the expressions in the boxes. The program consists of a simple loop controlled by what Goldstine and von Neumann referred to as the "induction variable" n. Box 2 represents the code that increments the value of n, box 3 represents the calculation of the polynomial's value for the current value of n (but doesn't state where that value is to be stored), and the *alternative box* 4 represents the test for loop termination. The initial value $n = 0$ must be inferred by comparing the initial $(5')$ and the general $(5' + 2n)$ values given in box 1 for storage location $2'$. The expression $n \Rightarrow n - 1$ in box 2 denotes the change in the value of n from one iteration to the next.

Address	Order	Result	Comment
2.1	$2'$	A : $5' + 2(n-1)$	Clear A and add
2.2	$4'$ h	A : $5' + 2n$	Add to A
2.3	$2'$ S	$2'$: $5' + 2n$	Store
2.4	3.1 Sp	3.1 : $(5' + 2n)$ R	Store address field
2.5	3.2 Sp	3.2 : $(5' + 2n)$ ×	Store address field
2.6	3.4 Sp	3.4 : $(5' + 2n)$ R	Store address field
2.7	$3'$ h	A : $6' + 2n$	Clear A and add
2.8	3.8 Sp	3.8 : $(6' + 2n)$ S	Store address field
3.1	$5' + 2n$ R	R : x_n	Load register
3.2	$5' + 2n$ ×	A : x_n^2	Multiply
3.3	$0'$ S	$0'$: x_n^2	Store
3.4	$5' + 2n$ R	R : x_n	Load register
3.5	$5'$ ×	A : ax_n	Multiply
3.6	$0'$ h	A : $x_n^2 + ax_n$	Add to A
3.7	$6'$ h	A : $x_n^2 + ax_n + b$	Add to A
3.8	$6' + 2n$ S	$6' + 2n$: $x_n^2 + ax_n + b$	Store
4.1	$2'$ −	A : $-(5' + 2n)$	Clear A and subtract
4.2	$1'$ h	A : $2N - 2 - 2n$	Add to A
4.3	2.1 Cc	$N - n - 1 \geqq 0$	Conditional transfer
4.4	e C		Jump to next order e

Fig. 2. Symbolic code for the polynomial program (after [10]). Three blocks of memory contain the instructions corresponding to boxes 2, 3, and 4 in Fig. 1. The instruction code is defined in [2]. Each order contains a memory reference (in some cases to a location in the table itself) and a code symbol. The result column shows the effect of transfer and arithmetic orders by giving the updated contents of the accumulator (A), the register (R), or a particular memory location, as appropriate. "Sp" orders copy data from the accumulator to the address field of the specified location.

Fig. 2 shows the code corresponding to the operation boxes 2, 3, and 4 in Fig. 1. Von Neumann thought of computer memory as a symbolic space consisting of addressable locations in which *words* were stored. *Number words* held coded numbers and *order words* held coded instructions. Most instructions included a numeric field, the address of the memory location on which the instruction was to operate. The purpose of executing instructions was to bring about changes in the contents of memory, a process that von Neumann described as a kind of substitution. Newly calculated numbers could replace the entire contents of a number word or the address field within an order word. The code also seems to allow for the substitution of entire order words, but none of Goldstine and von Neumann's examples use this capability, and the block and flow diagrams provided no way to represent its effect. This capability would prove to be crucial in the automation of coding through such tools as assemblers but, apart from a rather unconvincing discussion of subroutine relocatability, the *Planning and Coding* reports did not cover this topic.

A block diagram can therefore be interpreted in two very different ways. At one level, it is an abstract map of part of a computer's memory. Each block in the diagram corresponds to an area of memory and the directed lines joining them represent what Goldstine called the "itinerary" of the control organ as it executes the program. In this respect, the diagrams are abstract representations of machine-specific hardware, just as ENIAC's master programmer diagrams were. But at the same time the new diagrams aspired to be, in Goldstine's words, "a logically complete and precise notation for expressing a mathematical problem". An important theme in the evolution of the notation was trying to find a way to reconcile these rather different aims.

3 Describing iterative processes

In Goldstine's account, the use of diagrams began as an attempt to "indicate [...] the iterative nature of an induction". The most problematic aspect of this was finding a way to describe the changing value of the inductive variable; as this section will explain, Goldstine and von Neumann reached for the concept of substitution to manage this relationship, but it proved less than straightforward to devise a way to make this work.

The loop in Fig. 1 is controlled by the induction variable n. The value of n is not explicitly stored, however, and it only appears in the definition of the contents of location $2'$, namely the address of the location storing x_n. The code in Fig. 2 corresponding to box 2, then, must first increment the value in $2'$ by 2 (instructions 2.1 to 2.3), and then write this new value into the address fields of the instructions which retrieve x_n (3.1, 3.2, and 3.4) and the instruction which stores the new value of the polynomial (3.8). So while the diagram shows a loop controlled by a simple induction variable, the code corresponding to the mathematical operation of incrementing that variable performs a range of quite different tasks.

The annotations given in the code help us understand the way in which the substitution is expressed. The variable n, where $1 \leq n \leq N$, defines the current iteration of the loop. At the point where the dashed line attaches the storage box to the control flow line, then, the values held in storage correspond to the value $n-1$, as they have not yet been updated by the box 2 code. At this point, the value in $2'$ is $5'+2(n-1)$, recorded as the accumulator contents after instruction 2.1. Adding 2 to this gives the required value of $5' + 2n$ and the substitution $n \Rightarrow n-1$ describes the algebraic change.

The notation is rather unfortunate, however, in that the diagram suggests that location $2'$ holds the value $5' + 2n$ at the point of attachment, i.e. before box 2, which appears to increment the value of n. In a hand-written insertion on the typescript, von Neumann commented as follows:

> An alternative procedure would be this: Attach the storage box in its initial form, i.e. with a $5'$ opposite the memory location number $2'$, outside the n-induction loop, i.e. between $\textcircled{i}$ and the first $\longrightarrow$. Attach at the

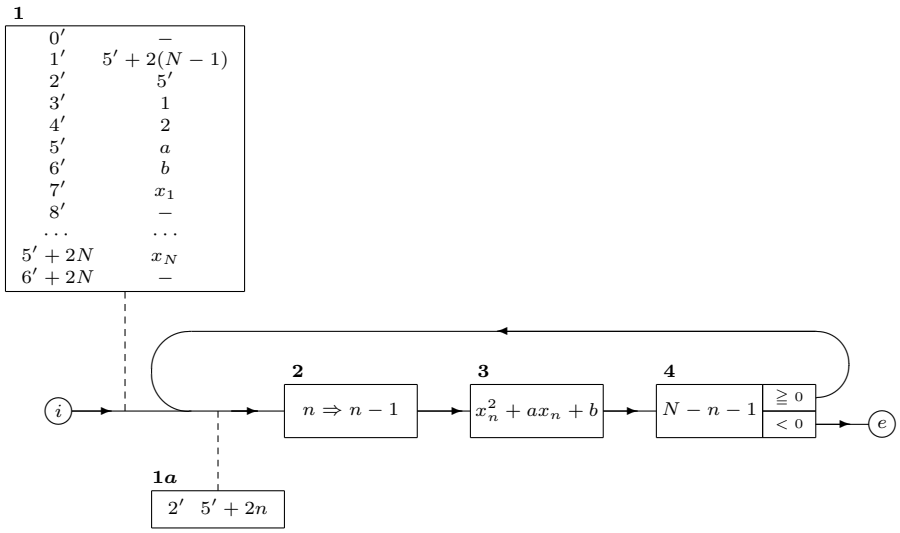

Fig. 3. Block diagram with distributed storage boxes

storage box's present location, i.e. within the n-induction loop, a small storage box which indicates only the change that takes place during the induction: $\boxed{2' \ : \ 5' + 2n}$. At present the simpler notation of the text will be used, however there are more complicated situations (e.g. multiple inductions) where only a notation of the above type is unambiguous. [10]

The block diagram with these changes is shown in Fig. 3. This diagram clearly distinguishes the initial value of $2'$ from the more general value given in terms of the inductive variable that it has while the loop is executing.

It looks as if we should be able to do more: the value stored in $2'$ is changed when n is incremented, and it is tempting to insert another storage box after box 2 describing the updated contents of $2'$ as $5' + 2(n+1)$. However, this would make the diagram inconsistent: neither box 3 nor box 4 changes the value of n, so when the loop reenters just before box 2, $2'$ must still hold the value $5'+2(n+1)$. But at this point, box 1a states that its value is $5' + 2n$.

The root of the problem is an ambiguity in the treatment of n. On the one hand it is the inductive variable, recording the ordinal number of the current loop iteration, but on the other hand, it helps define a stored quantity which is updated at a particular point within the loop. It is therefore unclear exactly when n is incremented, and it seems to be impossible to reconcile these two aspects and to indicate consistently and usefully the point at which the mathematical variable is incremented.

In the flow diagram notation, Goldstine and von Neumann addressed this problem by making a cleaner separation between mathematics and code, and

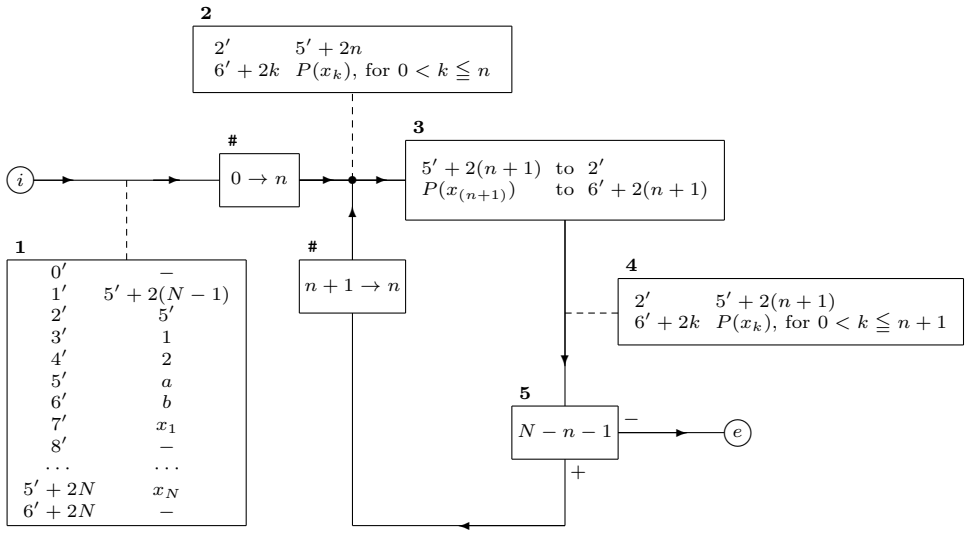

Fig. 4. Flow diagram for the polynomial calculation ($P(x) = x^2 + ax + b$)

altering and clarifying the semantics of substitution. The flow diagram in Fig. 4 shows one way of representing the polynomial program in the later notation.

The boxes in the diagram now fall into two different categories. As in the block diagrams, operation, alternative and storage boxes correspond to areas of storage containing number or order words, as appropriate. Operation boxes now explicitly show the memory location that a calculated value will be stored in and alternative boxes have a new syntax. However, substitution boxes, marked with **#**, no longer represent areas of storage. Thus in Fig. 4, the substitution box containing the formula $n+1 \to n$ does not represent the coded instructions that will increment the value of n; these are now carried out as part of box 3.

If substitution boxes no longer stand for coded words, what is their meaning? The best way to approach this question is by considering the general conditions that Goldstine and von Neumann defined for checking the consistency of a flow diagram. The aim of these conditions was to show that the values recorded in storage boxes were consistent with the operations described in the diagram.

For example, box 4 in Fig. 4 states that location $2'$ holds the value $5'+2(n+1)$. The preceding box, operation box 3, calculates that very value and stores in it $2'$, so in this respect the diagram is consistent. The general form of a consistency condition for this situation is shown graphically on the left-hand side of Fig. 5, and was expressed by Goldstine and von Neumann as follows (a "constancy interval" can be taken to be a region of a diagram containing a storage box):

> The interval in question is immediately preceded by an operation box with an expression in it that is referred "to ..." this field: The field contains the expression in question ... [11]

Fig. 5. Conditions on storage boxes preceded by operation (left) and substitution (right) boxes. The configuration on the right must satisfy the condition $P'[f \to i] = P$

A similar consistency condition was given for the situation where a storage box was immediately preceded by a substitution box. The right-hand side of Fig. 5 shows the general case and the condition was expressed as follows:

> Replace in the expression of the field every occurrence of every such i by its f. This must produce the expression which is valid in the field of the same storage position at the constancy interval immediately preceding this substitution box. [11]

(It is striking that Goldstine and von Neumann's rule applies the substitution to the expression in the storage box following the substitution, a move formally related to the later notion of a weakest precondition.)

For example, consider location $2'$ in box 2 in Fig. 4. Box 2 is preceded by the substitution $0 \to n$. Substituting 0 for n in the expression $5' + 2n$ in box 2 gives the expression $5'$ as the preceding value of $2'$, as recorded in box 1. Box 2 is also preceded, along a different path in the flow diagram, by the substitution $n + 1 \to n$: substituting $n + 1$ for n in $5' + 2n$ gives $5' + 2(n + 1)$, the expression recorded for $2'$ in storage box 4. The consistency of the diagram at this point follows from these two observations and the following structural rule:

> If the interval in question contains a merger (of several branches of the flow diagram), so that it is immediately preceded by several boxes [...], then the corresponding conditions [...] must hold with respect to each box. [11]

4 Assertions

The previous section showed how, by using storage boxes and substitutions, Goldstine and von Neumann found a way of describing in mathematical terms the step-by-step operation of computations, and in particular the behaviour of typical iterative loops. The flow diagram notation also included *assertion boxes*, however, and these have been seen as foreshadowing later uses of assertions in formal methods [16]. To evaluate this claim, it is useful to look at the role of assertion boxes in flow diagrams and the reasons for their introduction.

Assertion boxes were a late addition to the notation, introduced to solve a problem that arose in describing the result of computing $\sqrt{x}$ by the Newton-Raphson method. In the draft report this was coded in a form which limited the

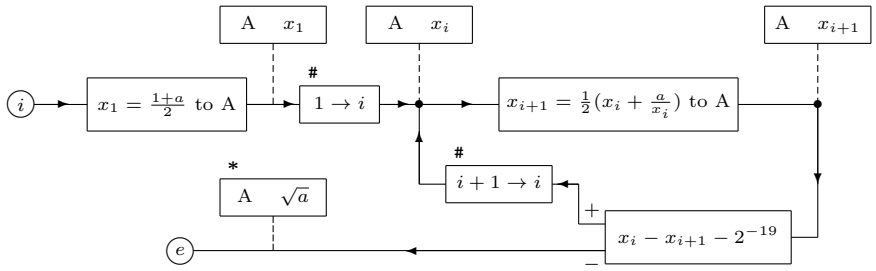

Fig. 6. Von Neumann's initial flow diagram for the calculation of $\sqrt{x}$

process to three iterations, and in early 1947 von Neumann was attempting to develop a more general approach where the loop terminated when the difference between two successive approximations became sufficiently small.

He initially drew a flow diagram similar to the one shown in Fig. 6 [23]. The storage box $*$ at the end of the diagram records where the computed value of $\sqrt{a}$ is stored. However, Goldstine pointed out that this conflicted with the previous storage box that gave the value stored in A as x_{i+1}. Von Neumann then proposed adding the substitution box shown in Fig. 7, where i_0 is defined as the first value of i for which $x_i - x_{i+1} < 2^{-19}$, giving x_{i_0} as the desired approximation to $\sqrt{a}$. He hoped that this would allow the two storage boxes in Fig. 7 to be "reconciled", but soon realized that this solution would not work:

> I must have been feeble minded when I wrote this: $\boxed{i_0 \to i}$ will reconcile $\boxed{A \quad \sqrt{a}}$ with a *succeeding* $\boxed{A \quad x_i}$, but not with a *preceding* one. I.e. one needs something new.
> One might play with new entities like $\boxed{i \to i_0}$, but I think that the best modus procedendi is this:
> Let us introduce a new type of box, called *assertion box*. It can be inserted anywhere into the flow diagram, and it contains one or more relations of any kind. It expresses the knowledge that whenever C gets there, those relations are certainly valid. It calls for *no* operations. It reconciles a storage box immediately after it with one immediately before it (and referring to the same storage position), if the expressions in these are equal by virtue of its relations. It is best to mark assertion boxes, say with a cross #.

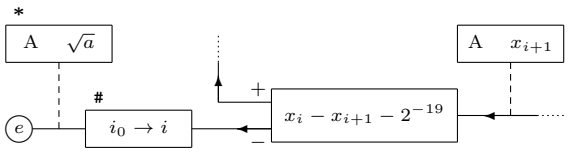

Fig. 7. Von Neumann's "feeble-minded" attempt, using a substitution box

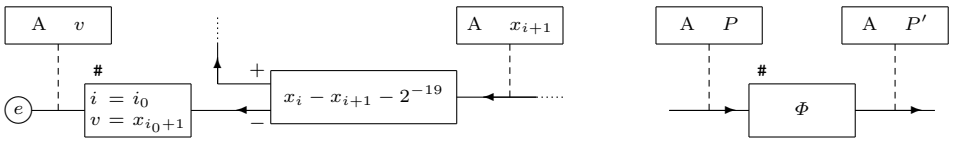

Fig. 8. The final version of the diagram with an assertion box (left-hand side). The general configuration on the right must satisfy the condition $\Phi \Rightarrow P' = P$.

Fig. 8 shows the corresponding portion of the diagram from the published report. The two equations in the assertion box imply that $x_{i+1} = v$ and hence reconcile the expressions defining the contents of A in the two storage boxes. The fact that v is the required approximation to $\sqrt{a}$ was stated in the preamble.

The formulas written in assertion boxes were not meant to be proved. They often represent an injection of knowledge into the diagram (such as the fact that at the end of a Newton-Raphson iteration the value computed is $\sqrt{a}$) or allow the introduction of new symbols with given properties. As with substitution boxes, their structural role in the notation is to reconcile preceding and succeeding storage boxes, according to the general schema given in Fig. 8. Goldstine and von Neumann expressed this condition as follows:

> It must be demonstrable, that the expression of the field [i.e., A] is, by virtue of the relations that are validated by this assertion box, equal to the expression which is valid in the field of the same storage position at the constancy interval immediately preceding this assertion box. [11]

In some cases there was no succeeding storage box, in which case the assertion box has a purely documentary role.

5 Flow diagrams in practice

It is outside the scope of this paper to analyze in detail the corpus of flow diagrams surviving from the years following the publication of the first *Planning and Coding* report. The overall picture is one of great notational diversity, united only by the use of a directed graph to depict the "flow of control" between boxes representing operations of various kinds. At the same time, the text in the boxes became increasingly informal. This section briefly describes the fate of the more formal aspects of the notation in three significant areas.

5.1 The *Planning and Coding* reports

The three *Planning and Coding* reports contain examples of the application of flow diagrams to a variety of problems. It is striking that, despite the very general way in which they were described, substitution and assertion boxes are sparingly used, and for a rather limited range of purposes.

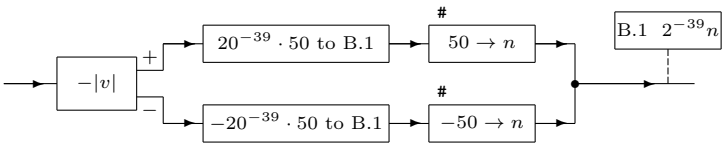

Fig. 9. Substitution boxes uniting different values of a variable

Substitution boxes were almost exclusively used to record the changing values of the induction variables in loops, as shown in Fig. 4. Occasionally they were used in straight-line code to assign a new value to a variable. Fig. 9 shows an application of this where n is given different values on the two branches of a conditional structure.

Assertion boxes were most commonly used at the end of loops. If termination was controlled by an explicit test of the inductive variable, the assertion restated and possibly strengthened the loop termination condition. For example, if an alternative box terminating a loop contained the formula $j - J \geqq 0$, it might be followed by a box asserting $j = J$. If the test terminating a loop did not involve the inductive variable, however, an assertion box might introduce a new variable to denote its final value, as in Fig. 8 above.

Assertions boxes were also used with variable remote connections. In one example, an assertion before the connector stated the conditions under which a variable had one of three possible values, and in another case assertions stated properties of certain variables just after a remote connection.

5.2 The Monte Carlo flow diagrams

The flow diagram notation was put to the test in the development of the first program to use the Monte Carlo method, a simulation of neutron diffusion in fissile material run on ENIAC in April and May, 1948 [12]. This project took place in the same timeframe as, and most likely inspired, plans to use ENIAC as an interpreter for an EDVAC-style code. Accordingly, the program design graduated from a generic computing plan to large and complex flow diagrams drawn using the notation of the *Planning and Coding* reports. Two complete diagrams have been preserved. The first was drawn up by von Neumann himself in the summer or early autumn of 1947, and the second dates from December of the same year.

Von Neumann's diagram consisted of about 70 operation boxes, 25 storage boxes, 20 substitution boxes, and no assertion boxes. There were six loops, including a nested pair of loops defining the large-scale structure of the program. Storage boxes were not included every time a new value was stored in memory. The program was divided into 10 sections, and storage boxes were typically, though not exclusively, placed at the end of a section to record the location of a significant new value calculated in that section. Many storage boxes immediately followed an operation box assigning a value to the location of interest, perhaps

with an intervening substitution box to reconcile notation or introduce a new variable in an obvious way. As a result, the conditions that would need to be checked to be assured that the diagram was well adjusted were largely trivial.

Von Neumann's original design went through an extensive series of changes, but by December it had stabilized and was documented by Adele Goldstine in a second complete flow diagram. This had basically the same structure as von Neumann's, although a couple of sections had been rewritten with alternative algorithms. Storage boxes were used in much the same way as in von Neumann's diagram but notated slightly differently, while the use of substitution boxes in the annotation of loops was rather different.

In this project, we can watch the flow diagram notation evolving in practice. Diagrams were not fully annotated, and became increasingly informal under the twin pressures of application to a large and complex problem and adaptation to the needs of a variety of users. There is no evidence that the conditions for well-adjustedness were recorded or formally checked anywhere. In most cases, these were so trivial that this may not have been felt to be necessary.

5.3 Flow diagrams cross the Atlantic

Two British mathematicians were in direct contact with Goldstine and von Neumann as the flow diagram notation was being developed. In January 1947, Alan Turing represented the National Physical Laboratory at a computing symposium at Harvard and then spent a couple of weeks with Goldstine and von Neumann. On his return to the UK he noted that "[t]he Princeton group seems to me to be much the most clear headed and far sighted of these American organizations, and I shall try to keep in touch with them" [20]. There are no records of the discussions, but it is likely that one topic would have been the approaches to programming being considered at Princeton and the NPL.

The mathematical physicist Douglas Hartree had a long-standing interest in computing machinery, and had visited Philadelphia in 1946 to run a problem on ENIAC. He kept in touch with Goldstine, and was sent a copy of the first *Planning and Coding* report soon after its publication. He took it on a family holiday in the west of England, but unexpectedly good weather left him with little time for reading, as he explained to Goldstine:

> So although I was very glad to get your report with von Neumann on coding, and have looked at it rather superficially, I haven't studied it seriously yet. My first impression was that you had made it all seem very difficult, and I wondered if it was really as difficult as all that?! [13]

Goldstine's reply was rather waspish:

> You suggest that possibly our report on coding seems very difficult. Of course it is very hard for me to be objective about it, but I thought, on the contrary, it was fairly simple. Van Wijngaarden, who is now here with us, spent three days studying the text and was then able to code

> problems with a reasonable degree of proficiency. I hope that after you have had a chance to look at the report in more detail you will agree with his opinion. [8]

Hartree's reservations persisted, and he exhibited a continuing preference for ENIAC-style notations. His 1949 book [14] on computing machines, based on lectures given the previous year, included a single flow diagram presented in parallel with an ENIAC master programmer diagram for the same problem. The flow diagram incorporated a number of modifications to the *Planning and Coding* notation. In a 1952 textbook [15], he even described something that looked very similar to an ENIAC diagram as a flow diagram.

This was typical of British uses of the notation, which seemed to treat it more as a vehicle for exploration than a finished product. At a conference in Cambridge in 1949, five papers presented flow diagrams of one form or another, but about all they had in common was the use of a directed graph. In particular, the mechanism of storage boxes, substitutions, and assertions that enabled the consistency of a diagram to be checked was almost universally ignored. The sole exception was a paper by Turing himself on "Checking a large routine" [21].

6 Checking a routine

As we have seen, Goldstine and von Neumann defined a number of conditions that had to be checked to ensure that a diagram was consistent. They explained the connection between the satisfaction of these conditions and the correctness of the diagram as follows:

> It is difficult to avoid errors or omissions in any but the simplest problems. However, they should not be frequent, and will in most cases signalize themselves by some inner maladjustment of the diagram, which becomes obvious before the diagram is completed. [11]

The worked examples in the *Planning and Coding* reports do not, of course, include any maladjusted diagrams. To get a sense of what this might have meant in practice and how the notation might have worked as a formal method, we need to look at a different example.

Turing's 1949 paper [21] included a flow diagram for computing factorials and discussed how to reason about the correctness of the program. At first sight, Turing's notation is rather different from Goldstine and von Neumann's. Nevertheless, Turing's diagram contains operation and alternative boxes linked by directed line segments, and we will assume that the resulting structure has the same meaning as in the Goldstine/von Neumann notation. For example, in language that could almost have been copied from *Planning and Coding*, Turing writes that "[e]ach 'box' of the flow diagram represents a straight sequence of instructions without changes of control".

Rather than attaching storage boxes at different points around the diagram, Turing presented a single table whose columns were labelled with letters cross-referencing locations on the diagram, a presentation option that had also been

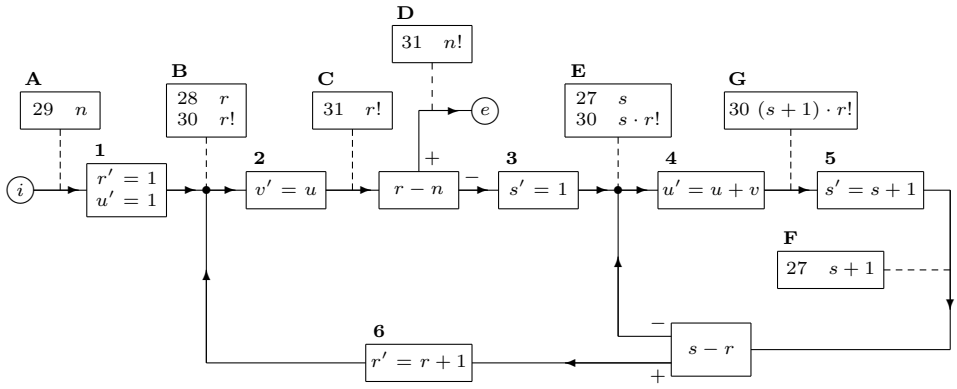

Fig. 10. Turing's flow diagram expressed using the notation of the *Planning and Coding* report. The lettered storage boxes correspond to the sections of Turing's storage table. Operation boxes have been numbered for ease of reference.

described in the *Planning and Coding* report. The table listed the five storage locations used by the program and described their contents using mathematical notation. Turing also associated a unique variable with each storage location:

location 27 : s – inductive variable for inner loop
28 : r – inductive variable for outer loop
29 : n – routine parameter
30 : u – accumulates $(r + 1) \cdot r!$ in inner loop
31 : v – stores $r!$

(The variables u and v do not appear in the storage table.) The sections of the storage table describe the contents of memory just before the boxes to which their labels are attached and so can be represented as storage boxes attached to the flow line immediately before the relevant box. Fig. 10 shows a transcription of Turing's diagram into the Goldstine/von Neumann notation.

Turing's most significant notational deviation was in the operation boxes. Rather than specifying the location where a value is stored, he used primed variable names to indicate what he described as "the value at the end of the process represented by the box". Box 5, for example, contains the expression $s' = s + 1$, indicating that at the end of the box the value of s has increased by 1. Turing does not state when this value is written into the storage location corresponding to s. However, storage box F gives the content of storage location 27 as $s + 1$, implying that the memory update has taken place by the end of the box. The contents of box 5 will therefore be translated as "$s + 1$ to 27" in the Goldstine/von Neumann notation.

As well as storing the new value, Turing's explanation suggests that the value of the variable s has been incremented by the end of box 5. Showing the change of value of a variable, as opposed to a storage location, is the reason Goldstine

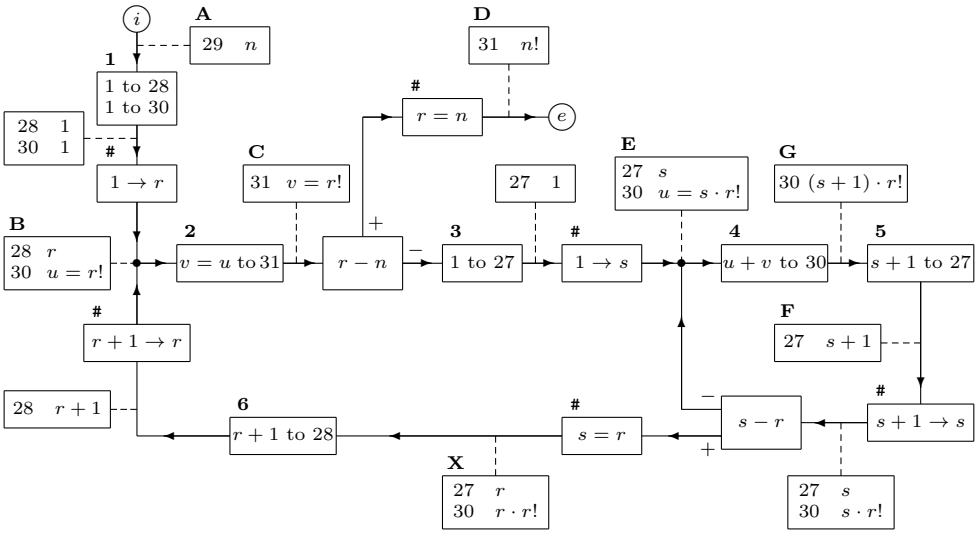

Fig. 11. Turing's diagram with operation boxes rewritten, variables added to storage boxes, and substitution and assertion boxes added. The diagram is "maladjusted", as the substitution $r + 1 \to r$ does not reconcile location 30 in storage boxes X and B.

and von Neumann introduced substitution boxes. Adding these to the diagram, we arrive at the diagram shown in Fig. 11 as a full translation of Turing's flow diagram into the Goldstine/von Neumann notation.

Unfortunately, this diagram is "maladjusted", to use von Neumann's term. At the end of the outer loop, the expression describing the contents of location 30 changes from $r \cdot r!$ to $r!$ in the passage between storage boxes X and B. This change should be reconciled by applying the substitution $r + 1 \to r$ to the expression in box B: however, this gives $(r + 1)!$, which is not equal to the value $r \cdot r!$ given in box X. Morris and Jones [17] describe this as a "discrepancy", commenting that "Turing chooses to regard $[s' = s + 1]$ as having no effect on the values of his variables"; they correct Turing's diagram by changing the expression being tested at the end of the inner loop to $s - 1 - r$, commenting further that Turing appears to give "no clear rule about when the addition of a prime to a letter makes a difference".

This interpretation differs from the natural reading of Turing's explanation adopted above. The root of the problem is that Turing blurred the distinction between storage locations and mathematical variables by associating a variable with each location. As a result, his notation leaves the temporal relationship between updating a variable value and updating a storage location unspecified. We can make this explicit in the Goldstine/von Neumann notation, and Fig. 12 shows an alternative way of making Turing's diagram consistent. Interestingly, separating the operation box that updates location 27 from the substitution

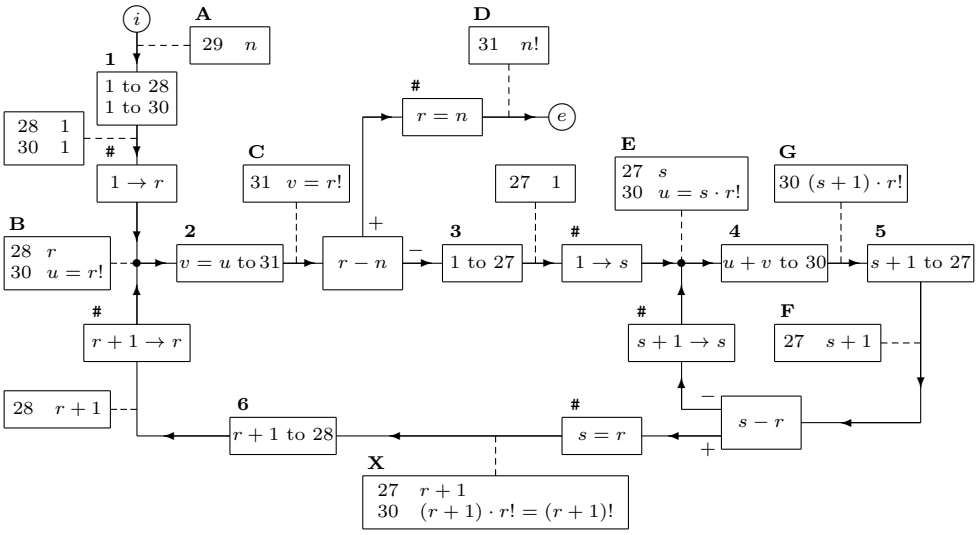

Fig. 12. A "well-adjusted" version of Turing's flow diagram

box that updates the variable s results in a more idiomatic use of the notation, similar in style to the examples in *Planning and Coding*.

Two assertion boxes have been added in Figs. 11 and 12 to make explicit all the conditions necessary to prove the diagram's consistency, but the identities they state were left implicit in Turing's diagram. Turing himself used the term "assertion" in the following, rather different, sense:

> In order to assist the checker, the programmer should make assertions about the various states that the machine can reach.

Unlike assertion boxes, Turing's assertions were not a notational feature. They were written in the columns in the storage table, and their function is to explicitly relate each storage box to its successor, giving additional information about the conditions under which transitions occur and the values of certain variables. For box B, for example, Turing's assertion reads simply "to C", and he paraphrased the argument that the checker would make as follows:

> From the flow diagram we see that after B the box $v' = u$ applies. From the upper part of the column for B we have $u = r!$. Hence $v' = r!$ i.e. the entry for v i.e. for line 31 in C should be $r!$. The other entries are the same as in B.

This is similar in intent to the condition that Goldstine and von Neumann gave for verifying the contents of storage boxes after operation boxes containing "to" (see Fig. 5). The most striking difference is that Turing argues "forwards"

from box B to box C, while Goldstine and von Neumann's condition for substitution boxes works "backwards", as explained above.

Turing's use of "assertion", then, has nothing to do with the assertion boxes of the Goldstine/von Neumann notation. In Turing's usage, assertions are roughly equivalent to the conditions, or proof obligations as we might call them, that the flow diagram imposes on the person checking the routine. The fact that the word "assertion" is used for both is nothing more than a coincidence. One possible explanation for this ambiguity would be that when he wrote the 1949 paper, Turing relied solely on his memory of the discussions that took place in Princeton in January 1947. As we have seen, assertion boxes were introduced after this date, and if Turing had never in fact read the *Planning and Coding* report, he would not have been aware of the later usage.

7 Conclusions

Flow diagrams emerged from a culture of computing that made extensive use of graphical notations. From electronic circuit diagrams to Curry and Wyatt's more abstract flowchart, ENIAC was surrounded by visual representations of the machine and the computations set up on it.

As programs were set up on ENIAC in a very immediate and physical way, by plugging wires and setting switches, diagrams of program structure could also be read as pictures of the machine. Less obviously, the same is true of the diagrams introduced by Goldstine and von Neumann. EDVAC-type machines replaced ENIAC's physical connectivity with more transient connections made in a large multi-purpose memory, and the boxes in a block diagram provide a map of memory usage for a particular problem. By implication, the unmarked white surface of the paper represents the computer's memory, a striking image for the logical space defined by the ambitiously large and functionally undifferentiated storage units planned for the new machines.

Crucially, the new machines also made possible programs that modified their own code. This was a central feature of even the elementary polynomial example described above. Eckert and Mauchly [4] noted that diagrams had been used for "laying out the procedure" for programs on ENIAC but, after making explicit reference to the first *Planning and Coding* report, went on to comment:

> The important point, however, is that [...] the instructions may themselves be altered by other instructions. Therefore the particular program that is chosen may not remain the same during successive traverses of it. Because of this feature, it becomes increasingly difficult to follow the course of more complicated problems unless some systematic procedure is adopted. The flow chart just referred to is the basis for such a procedure.

Flow diagrams, in other words, were a direct response to a new generation of computers with a distinctive architecture demanding a new approach to program planning. The development of the notation was driven by the specific challenge of

adequately describing the behaviour of a simple loop controlled by an inductive variable. Simple counted loops could be modelled perfectly well with existing notations, such as the ENIAC's master programmer diagrams, but even in the polynomial example, the inductive variable n does not just count loop iterations but is involved in updating the address field of a coded instruction in order to specify where the next function value will be stored.

In order to describe this situation precisely, Goldstine and von Neumann developed a formidable formal notation which described a computation on both physical and symbolic levels and provided a way to check the consistency of diagrams. It is striking, then, that users of the notation, themselves included, made little use of its full capabilities. It was either ignored, simplified, or heavily modified for use in particular circumstances. The most faithful user, Turing, applied it not in practical program development but in a conference paper which emphasized precisely its capabilities for checking correctness.

It is beyond the scope of this paper to consider in detail the relationship between the work described here and Floyd's 1967 paper [6]. Floyd, apparently unaware of the earlier work, described an "interpretation" of a flowchart as the association of a proposition with each of its edges. Syntactically, this could be achieved in the Goldstine/von Neumann notation by placing the proposition in an assertion box at the appropriate point in the diagram, though the pragmatics of the two notations are rather different. Floyd intended to give a "semantic definition" of the notation rather than a practical tool for program development, though his definitions would enable proofs of properties of flowcharts to be given. The relationship between such proofs and the consistency conditions put forward in the *Planning and Coding* report remains a topic for future research.

The reasons for the lack of uptake of the formal aspects of the original flow diagram notation remain underexplored. The earliest application of the notation, the evolution of the diagrams for the Monte Carlo program from von Neumann's original diagram to the final, less formal version, provides a good case study. This passage of work is characterized by the fact that a wide range of people with different skills, interests, and responsibilities became involved with the project. The second diagram was drawn by Adele Goldstine and the code produced and later maintained by Klara von Neumann. Neither had a background or training in logic, and the program used address modification only to control the return from a subroutine [12]. In these circumstances, the formal aspects of flow diagrams may have seemed an overhead that added little to solving the problem at hand or, more importantly perhaps, to the stated aim of enabling operators to produce code *from* the diagrams.

References

1. ASME Standard: Operation and flow process charts. American Society of Mechanical Engineers (1947)
2. Burks, A.W., Goldstine, H.H., von Neumann, J.: Preliminary discussion of the logical design of an electronic computing instrument (28 June, 1946), The Institute for Advanced Study

3. Curry, H.B., Wyatt, W.A.: A study of inverse interpolation of the Eniac (1946), Ballistic Research Laboratories Report No. 615. Aberdeen Proving Ground, MD
4. Eckert, J.P., Mauchly, J.: First draft report on the UNIVAC (1947), Electronic Control Company, Philadelphia, PA. Herman Goldstine papers, Hampshire College, box 3
5. Ensmenger, N.: The multiple meanings of a flowchart. Information and Culture **51**(3), 321–351 (2016)
6. Floyd, R.W.: Assigning meanings to programs. In: Schwartz, J.T. (ed.) Mathematical Aspects of Computer Science. Proceedings of Symposia in Applied Mathematics, vol. XIX, pp. 19–32. American Mathematical Society (1967)
7. Gilbreth, F.B., Gilbreth, L.M.: Process Charts. American Society of Mechanical Engineers (1921)
8. Goldstine, H.H.: Letter to Douglas Hartree (16 September, 1947), Herman Goldstine papers, American Philosophical Society, box 3
9. Goldstine, H.H.: The Computer from Pascal to von Neumann. Princeton University Press (1972)
10. Goldstine, H.H., von Neumann, J.: unpublished draft of [11] (1946), John von Neumann papers, Library of Congress, box 33, folder 7
11. Goldstine, H.H., von Neumann, J.: Planning and coding problems for an electronic computing instrument, Part II, Volume 1 (1947), The Institute for Advanced Study
12. Haigh, T., Priestley, M., Rope, C.: ENIAC in Action: Making and Remaking the Modern Computer. MIT Press (2016)
13. Hartree, D.R.: Letter to Herman Goldstine (7 September, 1947), Herman Goldstine papers, American Philosophical Society, box 3
14. Hartree, D.R.: Calculating Instruments and Machines. The University of Illinois Press (1949)
15. Hartree, D.R.: Numerical Analysis. Oxford University Press (1952)
16. Jones, C.B.: The early search for tractable ways of reasoning about programs. IEEE Annals of the History of Computing **25**(2), 26–49 (2003)
17. Morris, F.L., Jones, C.B.: An early program proof by Alan Turing. Annals of the History of Computing **6**(2), 139–143 (1984)
18. Morris, S.J., Gotel, O.C.Z.: Flow diagrams: Rise and fall of the first software engineering notation. In: Barker-Plummer, D., Cox, R., Swoboda, N. (eds.) Diagrammatic Representation and Inference. Lecture Notes in Computer Science, vol. 4045, pp. 130–144. Springer (2006)
19. Priestley, M.: Routines of Substitution: John von Neumann's work on software development, 1945-1948. Springer (2018)
20. Turing, A.M.: Report on visit to U.S.A., January 1st – 20th, 1947 (3 February, 1947), Mathematics Division [NPL]
21. Turing, A.M.: Checking a large routine. In: Report of a Conference on High Speed Automatic Calculating Machines, 22-25 June, 1949. pp. 70–72. University Mathematical Laboratory, Cambridge (1949)
22. Voller, W.R.: Modern Flour Millling (3rd edition). D. Van Nostrand Company (1897)
23. von Neumann, J.: Letter to Herman Goldstine (2 March, 1947), Herman Goldstine papers, American Philosophical Society, box 20