Article

On Folk Theorems

David Harel
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

Criteria are suggested for determining if a statement is a folk theorem. The ideas are then illustrated with a detailed example from the theory of programming.

Key Words and Phrases: flowchart, folk theorem, structured progamming, while-programs
CR Categories: 2.1, 4.2, 5.24

1. Introduction

Folklore: The traditional beliefs, legends and customs, current among common people.
—*The Oxford English Dictionary*

Theorem: A general conclusion which has been proved.
—*Mathematics Dictionary*, Van Nostrand Reinhold

In view of these quotations one might be tempted to conclude simply that a folk theorem is a general conclusion which has been proved and which is a traditional belief, legend, or custom current among common people. The purpose of this paper is to refine this definition somewhat, adapting it to the purposes of the research community in computer science. Accordingly, we shall attempt to provide a reasonable definition of or, rather, criteria for folk theorems, followed by a detailed example illustrating the ideas. The latter endeavor might take one of two possible forms. We could take a piece of folklore and show that it is a theorem, or take a theorem and show that it is folklore. As an example of the first form we could have shown that the statement $P \neq NP$, which is folklore, is also a theorem. However, since we have resolved to introduce no new technical material in this paper, and moreover, since researchers in our community seem to be less familiar with folklore than with theorems, we prefer and will adopt the second approach. Accordingly, in Section 3 we present strong evidence to the effect that a particular theorem about flowcharts is a folk theorem.

Author's address as of August 15: D. Harel, Dept. of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel.

© 1980 ACM 0001-0782/80/0700-0379 $00.75.

2. What is a Folk Theorem?

We would like the reader to try and recall the last time he received a referee's report in which the referee dismissed his latest achievement as being "a piece of folklore which has been around for ten years," or, alternatively, the last time the reader himself drastically reduced a student's overflowing enthusiasm about a result, labeling it "a good try but unfortunately a folk theorem." The high frequency of such rather unhappy occasions calls for extra investigation. What makes a statement a folk theorem? What facts prompted the careful reviewer to make his devastating comment, or by virtue of what did the reader himself dare turn a self-confident student or colleague into a temporarily crushed individual?

Let us quote three more definitions of folk objects:

Folk song: A song made and handed down among the common people; folk songs are usually of anonymous authorship and often have many versions.

Folk tale (or *folk story*): A story usually of anonymous authorship and legendary or mythical elements, made and handed down orally among the common people.
—*Webster's New World Dictionary of the American Language*

Folk tale: A popular story handed down orally from past generations.
—*The Oxford Advanced Learners Dictionary of Current English*

We will not try our hand at providing a similar clear-cut definition of a folk theorem suited for a dictionary, though it is somewhat amusing to see what happens when the word "theorem" is substituted for "song" or "story" in the above. In particular, we are certain that the (by now annoyed) reader would require an explanation of the phrase "common people" in our context. On closer inspection, however, it seems that there are three central properties which we can abstract from these definitions: *popularity*, *anonymous authorship*, and *age*. Note that these phenomena are also implicit in the definition of *folklore* quoted at the start. "current among common people" implies popularity, "beliefs, legends and customs" indicates anonymous authorship, and "traditional" adds the dimension of time.

When trying to clarify these three aspects and transfer them from the realms of telling tales and singing songs to that of proving theorems, it should be noted that neither one of them is to be taken in its strict literal sense. The popularity of a theorem need not necessarily be established by a head count of researchers who have used or quoted it, though (as illustrated in the next section) statistics can be helpful. Rather, it usually shows up when one tries asking a handful of colleagues whether the statement is true. With the "I've heard something like that before" look on his face, the person questioned will usually be able to give the correct answer and most often he will be able to accompany it with a proof too.

Often the theorem is quoted by virtue of it being very similar to another statement which is being proved. This brings us to the "often have many versions" phenomenon, which is closely related to popularity. Surely, with the size of our research community being what it is, slick, compact statements of theorems tend to be remembered even by people not working in the actual area of research relevant to those theorems. The statement "**while**-programs compute everything" is much more catchy and appealing than its obvious rigorous counterpart. Stating such a theorem in this form greatly increases its popularity and with it the chances of its becoming a "legend" or simply a folk theorem. The vagueness in a theorem's statement can thus give rise to more than one (legal) version of it, and again it becomes more fun and more folkish when each version requires a seemingly different proof. (The reader is advised not to confuse such folk theorems with statements which are generic, i.e., statements that are expressed deliberately in a form from which many real theorems are given as instances, like "parallel time equals space." See, e.g., chapter 2 of [36]. This we do not want to regard as being folk.)

Anonymous authorship should also not be taken completely literally. Indeed, since the periods of time we are talking about are not so great (see below), it is rarely the case that no one knows with whom the theorem, or its rigorously stated various versions, originated. Rather, what tends to happen is that the effort involved in tracing back such a theorem is far greater than that involved in reproving it; proofs (or the ideas behind the proofs) of folk theorems tend to be "handed down" together with their statements. (This fact, incidentally, implies that a deep theorem with a complicated proof will usually not qualify as a folk theorem). The roots of a folk theorem might be buried in some obscure lecture notes, in a letter to an editor, or worse, in a "private communication." It is usually possible, if one is willing to put on his Alex Haley cap, to undig those roots (at least those recorded in print), though the ideas behind the proof might not have been completely original with the author of the earliest such root. Indeed, and here is where the age of the theorem comes in, some of the ideas behind the best known folk theorems in computer science go back to Turing, Kleene, or von Neumann. (The usual saying, if one cannot find an explicit mention in their works, is "It was *known* to Turing . . . .") One must remember though that ten or even five years is an enormous time span in our field, and that furthermore the dizzying rate of development within the field, together with frequent changes in notation and terminology, can render such root-finding endeavors quite formidable. Sometimes the theorem is the result of a number of researchers' successive observations on previous work and any of these might be later quoted as the origin of the theorem. A "real" folk theorem will be reproved over the years again and again, an activity which in view of these remarks is completely justified and should be regarded as part of the culture. Our amateur historian may thus become easily confused, despair, and switch to a less depressing hobby such as stamp collecting.

Hence, while the pop-auth-age properties seem to be necessary and sufficient conditions for a theorem to be folklore, the ways in which they appear and can be established are by no means clear-cut. On this note let us now turn to a specific example of theorem.

3. A Folk Theorem

In the interest of making this section readable for as large an audience as possible, we have collected its more technical parts in the Appendix, which the technically oriented reader is strongly urged to regard as part of the text itself. However, since the reader would presumably not be reading this journal otherwise, we can safely assume that he is familiar with such general terms as *computer*, *program*, and *flowchart*, and also with more specialized ones such as *variable*, *program-counter*, and **while-do**. Without further ado, let us now introduce the subject of this folk section.

FOLK THEOREM. *Every flowchart is equivalent to a* **while**-*program with one occurrence of* **while-do**, *provided additional variables are allowed*

Facts pointing to its folkishness are apparent at the very start of an investigation of this theorem. Specifically, most people's first reaction is to attribute the theorem to the 1966 paper of Böhm and Jacopini [15] which appeared in this journal.[3] In fact, some of the published appearances of the theorem, which we mention later, are attributed by their authors to [15], although, as we shall see, it is not this theorem which Böhm and Jacopini proved.

(Let us remark here that this and other phenomena discussed in the sequel are to be taken as rather amusing manifestations of the lore and legend of our young field, and by no means are they criticism of the research

[3] Actually, it is the first part of [15], due to Jacopini, which is of interest here.

involved. We will do our best to convey this spirit to the reader as we go along.)

Returning to our topic, we note that the theorem proved in the first part of [15] asserts that, with additional Boolean variables[2], every flowchart is equivalent to a **while**-program (with, in general, more than one occurrence of **while-do**). Having proved the first "structuring result" of its kind, Böhm and Jacopini's work has become immortalized as containing "the mathematical justification for structured programming," to paraphrase many authors on the subject. The resulting universal popularity of [15], the fact that due to its rather technical style it is apparently more often cited than read in detail, and, of course, the similarity of the theorem proved in [15] to the one we are interested in seem to be some of the reasons for this common reaction.

Böhm and Jacopini prove their result by providing "local" transformations on the various kinds of flowcharts possible, leading to the final structured **while**-form, which involves only sequencing (;), conditionals (**if-then-else**), and iteration (**while-do**). Put another way, their proof is by induction on the structure of a flowchart. The Boolean variables are used to "mark" paths taken by the computation, in order to remember them later.

We can point to three subsequent papers in which similar proofs of the Böhm and Jacopini result appear; namely, Mills [67], and Culik [24, pp. 17–19; 25, pp. 11–14]. Indeed, it seems that H.D. Mills who, in his unpublished lecture notes, termed the result of [15] "The Structure Theorem" was one of the driving forces behind the glorification of Böhm and Jacopini's result, which he reproved and discussed in highly attended lectures and seminars in the following years.

It is interesting that precisely what is needed in order to extend this very proof to one proving our Folk Theorem can be found in Mirkowska's 1972 thesis in Polish about algorithmic logic [70].[3] The "normal form theorem" of [70] states that with additional Boolean variables (of the kind allowed in [15]) every **while**-program is equivalent to one in which **while-do** occurs only once. The proof, which is also by induction, provides local transformations which serve to eliminate nested and neighboring **while**'s and to distribute **while-do** over **if-then-else**. A sample transformation appears in the Appendix. To Mirkowska's proof of what we might call the second half of our Theorem, a complete paper by Perkowska [77, p. 441] was devoted two years later. A third exposition of this proof appears in a paper by Kreczmar [55, pp. 23–24] which deals mainly with interesting but completely different issues.

Thus, we have found that Böhm and Jacopini [15] are not to be detached from our investigation, since besides being the first to prove a "structuring result" and the first, it seems, to illustrate some of the power of Boolean variables, their work together with that of Mirkowska [70] can be interpreted as a complete proof of our own Theorem. If one prefers symbols over words and is willing, for the moment, to overlook priorities in favor of documenting proofs appearing in print, one might associate this proof of the Folk Theorem with

$$([15] \vee [67] \vee [24] \vee [25]) \wedge ([70] \vee [77] \vee [55]),$$

where "$\vee$" and "$\wedge$" stand, respectively, for "or" and "and."

Now, all this sounds like complicated mathematical research concerning a rather deep result for whose proof the efforts and partial results of at least two researchers, working separately and, indeed, in different countries, were needed. The real fun in investigating this Theorem starts when one realizes that there is a completely different, almost trivial way of proving the Theorem and that, furthermore, it is with this proof that the names of Böhm and Jacopini are often erroneously associated. This "global" proof, given more rigorously in the Appendix, involves constructing a simple one-loop program which, starting with the first "box" in the original flowchart, executes one such box each time around the loop. Upon completing the execution of a box, a variable is set to the index of the following box in the original flowchart. (For a test box, an **if-then-else** accomplishes this for the two possible outcomes.) Control then returns to the beginning of the loop. The body of the loop, which is executed until the index of the original STOP box is detected, starts with a nested set of conditionals which test the value of this one new variable and branch to the appropriate box accordingly. Since any flowchart contains only a finite number of boxes, the new (numerical) variable can be simulated by a finite number of Boolean ones, giving the Theorem.

It is interesting to note that both the local transformation of Böhm and Jacopini [15] and Mirkowska [70], and the aforementioned global one fail, by their very nature, to preserve the structure of the original flowchart. Ironically, though, they are proofs of a so-called "structuring result."

But let us continue our investigation, being curious as to whether we ourselves, by virtue of having just presented the global proof, are to play a central role in this little tale. We shall clearly be deprived of this if the proof has appeared in print earlier. "Appeared," did we say? Yes, it has appeared. But that would seem to be a rather mild word to associate with the situation we now set out to describe.

Textbooks are a good place to start. And, indeed, if one browses through the books published in the general area of structured programming, one finds two which contain the above proof: McGowan and Kelly in 1975 [63, pp. 62–64] and more recently, in 1979, Linger, Mills, and Witt [60, pp. 118–120]. A similar scan of books of a more theoretical nature reveals two more occurrences of the proof: Greibach's 1975 monograph on program schemes [38, pp. 4.52–4.53] and Clark and Cowell's 1976

[2] These can take on one of two distinct values, distinguishable by a test.

[3] Not having [70] in front of us, we rely here on personal communication with Mirkowska and on the reference in [55].

introductory text [20, pp. 61–63]. It is noteworthy that in all four cases the theorem stated is the Böhm and Jacopini one (and hence its attribution in [63] and [38] to [15]), but the global proof makes use of only one **while-do** and so proves our stronger Folk Theorem.

Since no pointers to the origin of the proof are given in these references, our book search has been only partly successful. The idea of the global proof seems to be simple and popular enough to have been essentially reinvented by the authors of these books. Reinvented, that is, if we can find it occurring in print even earlier.

Leaping from books to newsletters, a reader leafing through the February 1975 issue of *SIGPLAN Notices* notices a two-page paper by Plum [79, pp. 32–33], in which the author, terming the use of the Böhm and Jacopini [15] method "mathematical overkill," presents a "simple proof of this simple result"—the global proof above. One finds it hard to believe, though, that it took almost nine years for this observation to find its way to the printer.

We must turn, it seems, to one of computer science's standard oracles: D. Knuth's scholarly survey paper published in 1974 [52], besides containing much stimulating material concerning the (non-) value of results such as our Theorem, confirms our suspicion by reducing the nine years to slightly over one; in [52, p. 274] we find our sixth occurrence of this proof, but here it is attributed primarily to "the comments made by Cooper in 1967" in his letter [22] to the editor of this journal. Indeed, D.C. Cooper [22] (eight years, we might add, before the almost identical [79]!) notes that "a stronger reduction [than that of Böhm and Jacopini] is possible," and proceeds to prove our Folk Theorem the global way.

And so, we are exposed to what seem to be the first stages in the evolution of an authentic folk theorem: As a reaction to the journal publication of a proof of half the theorem, an almost trivial proof of the whole is suggested in a letter to the editor, and many subsequent authors, not having seen the aforementioned letter (indeed, one does not expect them to have seen it), then proceed to reinvent the proof themselves.

This situation is enhanced by the following additional five papers, all of which present Cooper's proof: Ashcroft and Manna's 1971 IFIPS paper on replacing **goto**'s by **while-do** ([2, p. 148]; see also [3, p. 140]), the similar but independent 1972 paper of Bruno and Steiglitz [17, p. 521], Wulf's 1972 "case against the **goto**" [87, pp. 66–67], Mill's 1975 journal version of the Structure Theorem [68, p. 45], and the appendix to the 1977 paper on reliable programs of Linger and Mills [59, pp. 136–139]. Of these, [2, 68, 59] credit Cooper with the proof, [17] gives the proof without credit but the theorem stated is the weaker Böhm and Jacopini version with credit to them, and [87] in a way typical of the folkishness of the situation credits [15] with the *stronger* version.

Similarly typical, by the way, is Denning's September 1975 complaint [27] about Plum's February 1975 paper [79]: "Precisely this construction was suggested by Bruno and Stieglitz in 1972." Denning's argument can actually be made five years stronger by referring to Cooper!

It is also worth remarking that in [2, p. 149; 3, pp. 141–142] Ashcroft and Manna, being interested in structuring results which preserve some of the structure of the original flowchart, provide in addition a slightly more efficient variant of the global proof, which they attribute also to Cooper, in which what one might call a "maximal-loop-free" component of the flowchart (as opposed to a single box) is executed each time around the loop.

Now, even if we were to end our story here, there is no doubt that we could quite safely term this result a folk theorem. As it turns out, though, our little game[4] is far from being over.

At this point we venture our opinion that the global proof, and hence our Theorem itself, is actually rooted in the early work of John von Neumann on the structure of digital computers. In Section 6 of his 1946 (!) joint paper with Burks and Goldstine [18], the idea of executing a program on a computer by means of a large loop which carries out an instruction at a time is described. A program counter is updated to contain the address of the next instruction before returning to the start of the loop. Admittedly, this is not a proof of a theorem, but the idea is certainly there, prompting one to wonder whether Cooper's observation should not be taken simply as providing an operational semantics for flowcharts, rather than being the proof of a theorem.

As if all this were not enough, this same proof appears in a number of additional papers, in which its detection is considerably harder. In the first, Meyer and Ritchie [65, Section 6], one can find the idea in a proof of the fact that primitive recursive functions of certain time complexity can be written with some fixed depth of bounded loops. The proof involves one potentially unbounded loop (corresponding to our single **while-do**), the body of which executes one step of the computation at a time with the aid of numerical program counters.

Next, the idea can be found in Bjorner [11] as the "Canonical Program" on page 438, which simulates any given "flowchart machine" using a next-state function.

A third such occurrence is in Brainerd and Landweber [16, pp. 122–123] where the idea is incorporated into the proof that labeled Markov algorithms compute partial recursive functions. There, too, numerical program counters are used.

Another occurrence can be extracted from the proof of theorem 16 in Pratt's 1976 paper on dynamic logic [81, pp. 119–120] in which a universal counter machine is simulated by a one-loop regular program which employs a numerical program counter.

A most interesting and subtle occurrence of this proof can be found in the fifth paper, Elgot [32], also from 1976, which deals with the language of multientry mul-

[4] For an enlightening example of a similar but much "larger" game, in which the period spanned is not measured in years but in centuries, the reader is referred to Robert K. Merton's delightful book on the "Dwarf-Giant" aphorism, [64].

tiexit flowcharts. Theorem 3.1 of [32, p. 44] states that every such flowchart is equivalent to one in the sublanguage analogous to conventional **while**-programs, but with no additional variables at all! In view of the negative results of Knuth and Floyd [53], Ashcroft and Manna [2, 3], Peterson, Kasami, and Tokura [78], Indermark [47], Kosaraju [54], and Kasai [49], from which it follows that for conventional flowcharts this statement is false, a careful reading of Elgot's proof becomes necessary. The evidence of the folkishness of our Theorem is considerably reinforced when one discovers that this proof is also essentially the global one in disguise! Boolean variables are cleverly replaced by what are called "trivial schemes" in [32], which we might call "cables," in which only one "wire" of each is connected. Such a cable forces control to proceed to one designated box for execution next time around the loop. Some details and an illustration can be found in the Appendix.

Digging slightly deeper into the earlier papers in algebraic semantics from which Elgot's paper [32] on structured programming grew, one discovers some interesting facts. A normal form result for certain kinds of operations on certain kinds of uninterpreted algebras occurs in various papers differing quite drastically from one another in notation and terminology. Thus, in a 1973 paper of Elgot [31, pp. 213–222] (published in 1975) we find a theorem about the "normal description of a morphism over an iterative theory." In Wand's 1972 paper [85, p. 335] (published in 1973) we find a normal form theorem for "μ-clones of operations over a lattice algebra." In a 1978 paper by Tiuryn [83, pp. 21–22] we find a normal form theorem for "regular polynomials over regular algebras." Finally, it was as early as 1969 when Bekić [9, pp. 12–15] proved a normal form theorem for "definable operations in general algebras."

All these four results can be shown to give rise to versions of our theorem for the more general case of "multiwire" recursive program schemes (as opposed to flowcharts). Without attempting to describe the technical details of these papers, which are all far beyond the scope and intention of this paper, we remark that in our terminology the proofs in Bekić [9] and Elgot [31] are local and in Tiuryn [83] and Wand [85] global. Furthermore, the proofs in [9, 31] start with schemes which are analogous to **while**-programs, and all four proofs use, essentially, the "trivial scheme" mechanism for mimicking Boolean variables. These observations make the task of fitting these proofs into our framework somewhat easier. Accordingly, our decisions on these were to classify Bekić [9] and Elgot [31] as additional proofs of the Mirkowska part [70] of the local proof, and to add Wand [85] and Tiuryn [83] to the list of Cooper-like global proofs. Nevertheless, as the algebraic approach is so different from that of most of the other proofs we have mentioned, we choose to assign priority credit for the second part of the local proof to Bekić and Mirkowska independently, although the former precedes the latter by three years.

Returning to the global proof, the learned reader can probably see now where these findings are leading us. The search for occurrences of this proof in the (obviously relevant) literature on structured programming and flowcharts led to Cooper's operational version [22], which, in turn, served to attract our attention to Burks, Goldstine, and von Neumann [18]. But now we have gradually been lured into considering theorems in algebraic semantics (e.g., [83, 85]) and in recursive function theory (e.g., [16, 65]), which, at first sight, seem quite unrelated to our naive flowcharts. And so, there seems to be no escape from considering the grand common ancestor of all such results—Kleene's 1936 normal form theorem for partial recursive functions [51]!

Indeed, Kleene [51, p. 736] (see also Kleene [50, p. 288] and Rogers [82, pp. 29–30]) showed that every partial recursive function f can be described as the application of a primitive recursive function g to μh, where h is primitive recursive and μ, the "minimization" operator, acts, in essence, like a **while** loop. The "body" of that loop, h, can be loosely described as simulating one step in the computation of f using coordinates for the "current value" of the function and the "label" of the next step. The function g simply isolates the final value of the computation by projecting on the appropriate coordinate.

Now, Kleene's theorem appears repeatedly in countless papers on recursive function theory. However, since we did, after all, start out with flowcharts, and since the line must be drawn somewhere, we have decided to draw it right here; no attempt shall be made to search for all occurrences of Kleene's theorem, and the ones we have mentioned [50, 82] will not qualify as proofs of our Theorem.

To summarize this part of our tale, the global proof has been traced down two orthogonal paths, each of which has led to a pioneer—J. von Neumann in his 1946 work on designing computers [18] and S.C. Kleene in his 1936 work in recursive function theory [51]. Although Cooper [22] provided the first explicit proof of our Theorem as stated, we feel it is reasonable to make the modest assumption that he, as well as all subsequent provers, was either directly or indirectly influenced by [18] and [51]. Consequently, we assign credits for the global proof of our Theorem to Kleene [51] and Burks, Goldstine, and von Neumann [18] independently. So, in fact, the Theorem *was* "known" to Kleene and von Neumann.

Again, blurring credits in the interest of enumeration and begging the reader's pardon for inserting an annoying self-reference to the present paper [40] in which, no doubt, the proof appears again,[5] we associate the global proof of the Theorem with

[5] This is the "referring to as many of one's own papers as possible" syndrome, taken to a new extreme: a reference to the very paper being read! In the absence of any other notable contributions of the present paper, and in view of the amount of work that went into writing it, we cannot resist the temptation to claim priority for this first.

$([51] \vee [18]) \wedge ([65] \vee [22] \vee [11] \vee [2] \vee [17] \vee$
$[87] \vee [85] \vee [52] \vee [16] \vee [68] \vee [79] \vee$
$[38] \vee [63] \vee [32] \vee [20] \vee [81] \vee [59] \vee$
$[83] \vee [60] \vee [40])$.

As mentioned earlier, statistics are perhaps not crucial for the classification of a theorem as folklore, but they are impressive nevertheless. The version of the Theorem we have talked about so far[6] has essentially two proofs. Of the local proof we have found $4 \wedge 5$ occurrences (four of the first part, five of the second, none of both) and of the global proof $2 \wedge 20$. While it is not clear whether $4 \wedge 5$ should evaluate to 4, 5, 9, or 20, there are certainly *many* occurrences.

In the process of exposing these facts we have also found numerous references to the Theorem or its Böhm and Jacopini part, which do not contain proofs, mainly in connection with the issue of structured programming. For a sample of fifty see Arsac [1, p. 3], Baker [4, p. 99], Baker and Kosaraju [5, p. 555], Banachowski [6, p. 115], Banachowski et al. [7, p. 22], Bates [8, p. 196], Benson [10, pp. 145–146], Bloom and Tindell [12, p. 271], Boehm [13, p. 113], Bohl [14, p. 140], Chandra [19, p. 1], Cohen and Levi [21, pp. 209, 225], Culik [26, p. 54], Denning [27, p. 10; 28, p. 216], Dijkstra [29, p. 148], Donaldson [30, p. 53], Engelfriet [34, p. 204], Fischer and Fischer [35, p. 46], Goodman and Hedetniemi [37, pp. 19–20], Harel [39, p. 89], Harel, Norvig, Rood, and To [41, p. 218], Hopkins [44, p. 59], Hughes [45, p. 59], Hughes and Michtom [46, p. 61], Jensen and Tonies [48, pp. 228, 236], Kasai [49, p. 177], Knuth and Floyd [53, p. 31], Kosaraju [54, p. 252], Leavenworth [56, p. 55], Ledgard and Marcotty [57, p. 632], Lee and Chang [58, p. 65], Martin [61, p. 5], McGowan [62, p. 25], Miller and Lindamood [66, p. 56], Mills [69, p. 90], Myers [71, p. 110; 72, p. 5], Nassi and Shneiderman [73, pp. 13–14], Neely [74, p. 120], Nicholls [75, pp. 409–410], Partch [76, p. 1282], Peterson, Kasami, and Tokura [78, p. 511], Prather [80, pp. 159, 170], Van Gelder [84, pp. 3, 5], Wise, Friedman, Shapiro, and Wand [86, p. 441], Yourdon [88, pp. 146–147; 89, p. 107], Yourdon and Constantine [90, p. 66], and Zelkowitz, Shaw, and Gannon [91, p. 60].

Many of these references contain interesting relevant material, but in order to keep this investigation from getting out of hand we will not describe them all. Let us just remark that Bohl [14] informs us that Böhm and Jacopini [15] was "initially published in Italian in 1965," that Cohen and Levi [21] extend the Theorem to parallel programs, and that Prather [80] proves a similar result[7] for Turing machines.

[6] Alert reader: yes, there is another version to come!
[7] This is a syntactic result for "structured Turing machines," not a Kleene-type normal form theorem for functions.

Our story is given a final and quite unexpected folkish twist by the existence of a second version of the Theorem. In preparation for this, we were careful to choose the adjective "additional" rather than "Boolean" in its statement. In this version, auxiliary rather than Boolean variables are allowed, which are simply new variables of the types used in the original flowchart. Although not guaranteed the ability to distinguish at will between two distinct values as in the Boolean version, we are able to freeze current values in new variables, and by retesting them later are able to "remember" the outcome. (See the sample transformation in the Appendix.)

The reader is urged to try his hand at proving this version, which is apparently harder than the first. In fact, we are not aware of any single global proof of it. Curiously (or should we say, folkishly), a full proof of this version does not seem to be available in any single paper and, as we shall see, can be found only by juxtaposing three papers (published, incidentally, on three different continents . . .).

In 1971, Cooper [23, pp. 47–49] and Engeler [33, pp. 93–94] independently pointed out that any flowchart can be put into a "block form," which is basically a treelike flowchart in which branches are allowed to bend backward, but to ancestors only. In the same year, Ashcroft and Manna showed that for any flowchart in this form there is an equivalent **while**-program with additional auxiliary variables. (This result is stated in [2, p. 148], relying on a proof appearing in a preceding technical report. The full proof appears in their final version [3, pp. 135–138], and also, with reference to [2], in Greibach's monograph [38, pp. 4.53–4.54].) Being uninterested in ruining the structure of the original program, Ashcroft and Manna make no attempt to push their techniques any further.

However, in a paper published in 1972, Hirose and Oya [42, pp. 369–370; 43, pp. 65–68], apparently without having seen [2], lay down the final brick by proving that every **while**-program can be transformed into one with a single **while-do**, using auxiliary variables. (In [43], the proof starts from arbitrary flowcharts but the part leading to **while**-programs is rather sketchy compared to [2], whereas the transformation to single **while-do** form is detailed and precise.)

All three components of this proof are "local," i.e., proceed by induction on the structure of the program. A sample transformation from [42] appears in the Appendix. Here, too, enumeration (modulu different versions of the same paper) gives $([23] \vee [33]) \wedge ([2] \vee [38]) \wedge [42]$.

And so our folk tale ends and barring unexpected complications, our Folk Theorem has good chances of living happily ever after. We have found that it has two versions, the first of which has two different proofs, one of these essentially going back to the cave-dwelling days of Kleene (1936) and von Neumann (1946). The credits are illustrated in Figure 1. Besides over fifty references

Fig. 1. The Roots of the Theorem.

Boolean version	local proof	flowcharts $\xrightarrow{\text{(Bohm and Jacopini)}}$ **while**-programs $\xrightarrow{\text{(Bekič} \vee \text{Mirkowska)}}$ single **while-do**
	global proof	flowcharts $\xrightarrow{\text{(Kleene} \vee \text{(Burks, Goldstine, and von Neumann))}}$ single **while-do**
auxiliary version	local proof	flowcharts $\xrightarrow{\text{(Engeler} \vee \text{Cooper)}}$ block-form $\xrightarrow{\text{(Ashcroft and Manna)}}$ **while**-programs $\xrightarrow{\text{(Hirose and Oya)}}$ single **while-do**

to the Theorem or its parts, we have found $(4 \wedge 5) + (2 \wedge 20) + (2 \wedge 2 \wedge 1)$ printed occurrences of its proof,[8] which evaluates to 27, 36, or 64, depending, respectively, upon whether "$\wedge$" is interpreted as "max", "+", or "$\times$". These proofs, which span forty-four years, also span the complete spectrum of recognized scientific literature: textbooks, monographs, survey articles, journal papers, conference proceedings, newsletters, theses, technical reports, lecture notes, letters to editors, and self-referential folk tales.

Quite a folk theorem, it seems....

4. The Future

We have postulated the appropriately adapted properties of popularity, age, and anonymous authorship as criteria for determining if a statement is a folk theorem, and have illustrated by exhibiting a theorem, the folkishness of which seems to be beyond doubt.

As for the future, we envision three possible directions for further research:

(1) Compiling an encyclopedic list of folk theorems in computer science.
(2) Investigating the related concepts of *folk definition* and *folk technique* (that is, proof-technique).
(3) Showing that folk facts such as $P \neq NP$ are folk theorems.

5. Appendix

First, some definitions. A *flowchart* is a finite directed graph with nodes labeled either with an assignment of the form $x \leftarrow f(\bar{y})$ for variable x, function symbol f, and vector of variables $\bar{y}$, or with a test which is a Boolean-combination of expressions of the form $p(\bar{y})$ for predicate symbol p and vector $\bar{y}$. Assignment nodes have one outgoing edge and test nodes have two, labeled 1 and 0. There is one START node with no incoming edges and one outgoing edge, and at least one STOP node with no outgoing edges. An *interpretation* I of a flowchart F consists of a set of domains, an appropriate association of domains with the variables of F, and an appropriate association of functions and predicates over various cross-products of the domains with the function and predicate symbols of F. Given an interpretation I and initial values for the variables appearing in F, the way in which F proceeds to compute its values is straightforward and is assumed to be known to the reader. Denote by F_I the function, associating with each set of input values for F its output values, or a special "undefined" symbol if F does not terminate. Also, let *asn*(F) and *test*(F) be the sets of assignments and tests appearing in F.

Next, we define an appropriate set of **while**-programs, relative to sets of assignments and tests A and T. Define WH(A, T) as the least set such that:

(1) $A \subset WH(A, T)$
(2) if $W_1, W_2 \in WH(A, T)$ and $P \in T$, then
  (i) $(W_1; W_2) \in WH(A, T)$,
  (ii) **if** P **then** W_1 **else** $W_2 \in WH(A, T)$, and
  (iii) **while** P **do** $W_1 \in WH(A, T)$.

Again, given an appropriate interpretation and initial values for the variables appearing in a **while**-program W, the standard method of defining the computation of W is assumed to be known. We use W_I similarly as with flowcharts. Let $WH_1(A, T)$ be the set of those elements of WH(A, T) which contain at most one occurrence of **while-do**.

Now, to be able to rigorously state the Boolean version of our Theorem, we let

$Basn(F) = asn(F) \cup \{p_i \leftarrow \textbf{true}, p_i \leftarrow \textbf{false}\}, 1 \leq i < \infty,$

and

Btest = the set of Boolean combinations of tests in *test*(F) and expressions of the form $p_i?$, for $1 \leq i < \infty$,

[8] We would greatly appreciate pointers to those occurrences of proofs we might have missed in our ignorance, in our haste, or otherwise.

Fig. 2. Cooper's Global Proof.

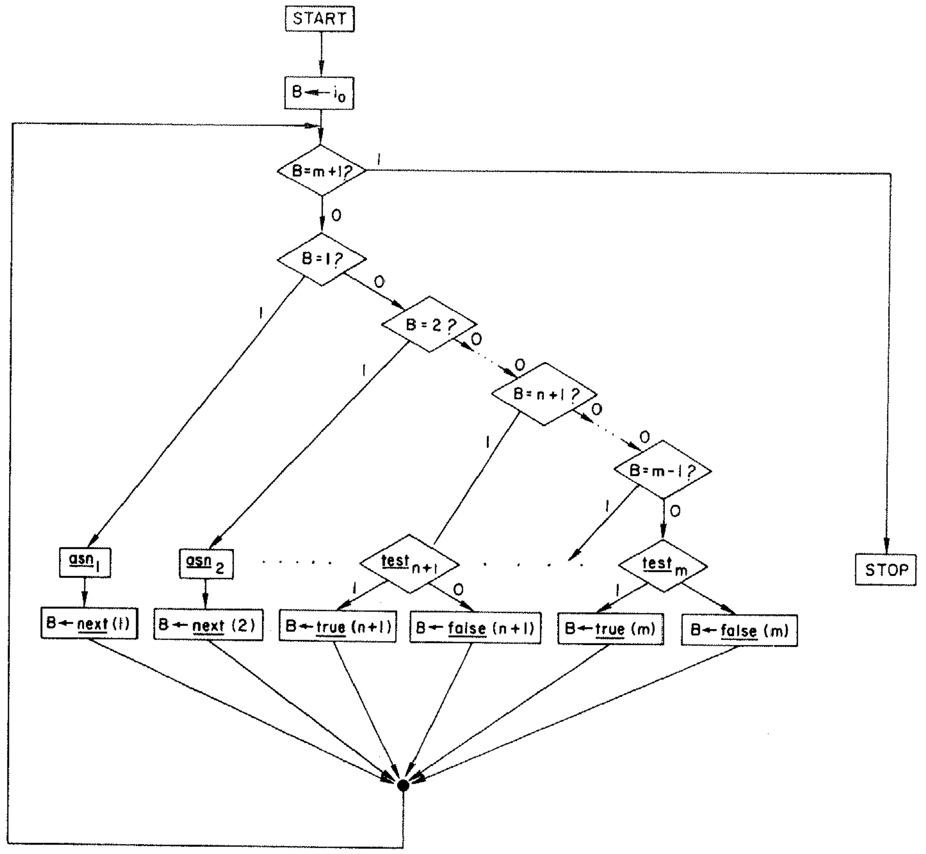
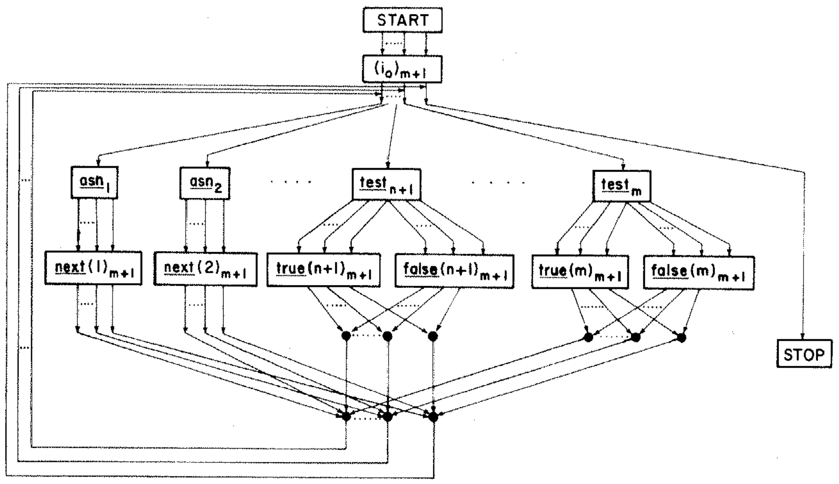

START
B ← i_0
B = m+1?
1
0
B = 1?
B = 2?
B = n+1?
B = m−1?
$\underline{asn}_1$
$\underline{asn}_2$
$\underline{test}_{n+1}$
$\underline{test}_m$
STOP
B ← $\underline{next}$ (1)
B ← $\underline{next}$ (2)
B ← $\underline{true}$ (n+1)
B ← $\underline{false}$ (n+1)
B ← $\underline{true}$ (m)
B ← $\underline{false}$ (m)

Fig. 3. Elgot's Proof.


START
$(i_0)_{m+1}$
$\underline{asn}_1$
$\underline{asn}_2$
$\underline{test}_{n+1}$
$\underline{test}_m$
$\underline{next}(1)_{m+1}$
$\underline{next}(2)_{m+1}$
$\underline{true}(n+1)_{m+1}$
$\underline{false}(n+1)_{m+1}$
$\underline{true}(m)_{m+1}$
$\underline{false}(m)_{m+1}$
STOP

where p_1, p_2 ... are new variables and F is an arbitrary flowchart. An interpretation is called *nice* if the p_i range over the domain {**true**, **false**} and the test p_i? is true (i.e., evaluates to 1) iff the value of p_i is **true**. Denote the set of nice interpretations by NICE. Then the Boolean version of our Folk Theorem states:

(*) $(\forall F)\ (\exists W \in WH_1\ (\mathit{Basn}(F), \mathit{Btest}(F)))$
$(\forall I \in \text{NICE})\ (F_I = W_I)$.

In order to illustrate the Böhm and Jacopini [15] ∧ Mirkowska [70] "local" proof, we provide one of the transformations of Mirkowska [70]:

while P_1 **do** (W_1; **while** P_2 **do** W_2)

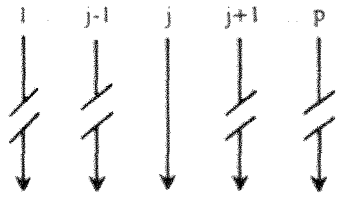
$$\Big\downarrow$$

"$p_1 \leftarrow P_1$"; $p_2 \leftarrow$ **false**; **while** ($p_1 \vee p_2$) **do** (**if** p_2 **then** W_2 **else** W_1; "$p_2 \leftarrow P_2$"; "$p_1 \leftarrow P_1$"), where, e.g., "$p_1 \leftarrow P_1$" stands for **if** P_1 **then** $p_1 \leftarrow$ **true else** $p_1 \leftarrow$ **false**.

Cooper's global proof [22] proceeds as follows: Given a flowchart F, denote the assignment nodes of F in some arbitrary, but fixed order by $asn_1, \ldots, asn_n$, and the test nodes similarly by $test_{n+1}, \ldots, test_m$. Think of all the STOP nodes as being numbered m + 1. For each $1 \leq i \leq n$, let $next(i)$ be the subscript corresponding to the node adjacent to asn_i (by its outgoing edge), and for each $n + 1 \leq i \leq m$ let $true(i)$ and $false(i)$ be the subscripts corresponding to the nodes adjacent to $test_i$ (by the 1 and 0 outgoing edges, respectively). Let i_0 be the subscript corresponding to the node adjacent to the START node. Assume now that a new variable B ranges over the set $\{1, 2, \ldots, m + 1\}$, and that we have assignments $B \leftarrow i$ and tests $B = i$? for each i in that set, with the meaning obvious. It should be clear that B can be replaced by $\lceil \log m \rceil + 1$ Boolean variables encoding the value of B, with $B \leftarrow i$ and $B = i$? being replaced by the appropriate sequences of assignments and conjunctions of tests.

The **while**-program $W \in WH_1(Basn(F), Btest(F))$ which is equivalent to F in the sense of (*), is given (using the obvious translation into **while**-form) by the flowchart in Figure 2.

To illustrate Elgot's proof [32], we will not provide a rigorous definition of his multiwire flowcharts, but trust that the reader will be able to gather as much as is needed from the following. For positive integers j and p where $j \leq p$, the "trivial scheme j_p" of [32] is simply the "cable" of width p with all but the j'th component disconnected:



Elgot's proof is illustrated by the multiwire flowchart of Figure 3. In it, e.g., $next(i)_{m+1}$, which follows asn_i, is a cable of width m + 1 in which control, so to speak, may only flow through the $next(i)$'th wire, which is ultimately connected to $asn_{next(i)}$, as expected. In other words, Cooper's variable B is "hard wired" into the multiwire cables of Elgot's program.

In order to state the second version of the Theorem rigorously, we let $Aasn(F)$ be the set of assignments of the form $x \leftarrow f(\bar{y})$, where x and the components of $\bar{y}$ come from the union of $\{u_1, u_2, \ldots\}$ and the set of variables appearing in F. Here the u_i are new variables, and f is a function symbol appearing in F. $Atest(F)$ is defined as the set of Boolean combinations of tests of the form $p(\bar{y})$, for $\bar{y}$ as above and p in F. The "auxiliary" version of the Folk Theorem states:

$$(**) \qquad (\forall F)\,(\exists W \in WH_1(Aasn(F), Atest(F))\,(\forall I)(F_I \equiv W_I).$$

As an illustration of the ([33] $\vee$ [23]) $\wedge$ ([2] $\vee$ [38]) $\wedge$ [42] proof of this version, we provide the transformation of Hirose and Oya [42] analogous to that of Mirkowska [70] given above:

while P_1 **do** (W_1; **while** P_2 **do** W_2)

$$\big\downarrow$$

$\bar{u} \leftarrow \bar{x}$; **if** P_1 **then** W_1; **while** P_1' **do if** P_2 **then** W_2 **else** ($\bar{u} \leftarrow \bar{x}$; **if** P_1 **then** W_1).

where $\bar{x}$ is the vector of all variables appearing in the original F, $\bar{u}$ is a vector of new variables, and P_1' is the result of substituting the u's for the x's in P_1. Here, for clarity, the clause **else** $\bar{x} \leftarrow \bar{x}$ has been omitted.

Received 10/79; revised and accepted 2/80

Acknowledgments. We are indebted to N. Pippenger for helpful discussions about folk theorems in general and about the one described in Section 3 in particular. Also, thanks go to all the people we have pestered recently with questions about folk theorems and translations from flowcharts to **while**'s; may this work help erase the puzzled looks we often saw on their faces as a result.

References
1. Arsac, J. Un langage de programmation sans branchements. *Rev. Franc. d'Autom. Inf. Rech. Oper.* (June 1972), 3–34.
2. Ashcroft, E., and Manna, Z. The translation of "go to" programs to "while" programs. In *Inform. Proc. 71*, 1971, pp. 147–152.
3. Ashcroft, E., and Manna, Z. Translating program schemes to while-schemes. *SIAM J. Comptng. 4*, 2 (June 1975), 125–146.
4. Baker, B.S. An algorithm for structuring flowcharts. *J. ACM 24*, 1 (Jan. 1977), 98–120.
5. Baker, B.S., and Kosaraju, S.R. A comparison of multilevel break and next statements. *J. ACM 26*, 3 (July 1979), 555–566.
6. Banachowski, L. Investigations of properties of programs by means of the extended algorithmic logic I. *Annal. Soc. Math. Pol.*, Series IV; *Fundamenta Informaticae I*, 1 (1977), 93–119.
7. Banachowski, L., et al. An introduction to alogrithmic logic. In *Mathematical Foundations of Computer Science*, Mazurkiewitcz and Pawlak, Eds., Banach Ctr. Publications, Warsaw, Poland, 1977, pp. 7–99.
8. Bates, J.L. A logic for correct program development. Ph.D. th., Cornell Univ., Aug. 1979.
9. Bekić, H. Definable operations in general algebras, and the theory of automata and flowcharts. Tech. rep., IBM Lab., Vienna, Dec. 1969.
10. Benson, J.P. Structured programming techniques. Proc. IEEE Symp. on Comptr. Software Reliability, New York, April 1973, pp. 143–147.
11. Bjorner, D. Flowchart machines. *BIT 10* (1970), 415–442.
12. Bloom, S.L., and Tindell, R. Algebraic and graph theoretic characterizations of structured flowchart schemes. *Theoretical Comptr. Sci.* 9 (1979), 265–286.
13. Boehm, B.W. Software design and structuring. In *Practical Strategies for Developing Large Software Systems*, E. Horowitz, Ed., Addison-Wesley, Reading, Mass., pp. 103–128.
14. Bohl, M. *A Guide for Programmers*. Prentice-Hall, Englewood Cliffs, N.J., 1978.
15. Böhm, C., and Jacopini, G. Flow diagrams, Turing machines, and languages with only two formation rules. *Comm. ACM 9*, 5 (May 1966), 366–371.
16. Brainerd, W.S., and Landweber, L.H. *Theory of Computation*. John Wiley and Sons, New York, 1974.
17. Bruno, J., and Steiglitz, K. The expression of algorithms by charts. *J. ACM 19*, 3 (July 1972), 517–525.

18. Burks, A.W., Goldstine, H.H., and von Neumann, J. Preliminary discussion of the logical design of an electronic computing instrument, 1946. In *J. von Neumann, Collected Works*, Vol. V, A.H. Taub, Ed., MacMillan, New York, 1963, pp. 34–79.
19. Chandra, A.K. Degrees of translatability and canonical forms in program schemas. Proc. 6th ACM(SIGACT) Symp. on Theory of Comptng., 1974, pp. 1–12.
20. Clark, K., and Cowell, D. *Programs, Machines and Computations: An Introduction to the Theory of Computing*. McGraw-Hill, New York, 1976.
21. Cohen, A.T., and Levi, L.S. Structured flowcharts for multiprocessing. *Comptr. Languages 3*, 4 (1978), 209–226.
22. Cooper, D.C. Böhm and Jacopini's reduction of flow charts. *Comm. ACM 10*, 8 (Aug. 1967), 463, 473.
23. Cooper, D.C. Programs for mechanical program verification. *Machine Intell. 6*, Edinburgh Univ. Press, 1971, pp. 43–59.
24. Culik, K. Structured algorithms and structured programming. Rep. CS-79-40, Dept. of Comptr. Sci., Penn. State Univ., Aug. 1979.
25. Culik, K. Entry strong components and their applications (in computer science). Rep. TR Nov. 79-01, Dept. of Comptr. Sci., Wayne State Univ., Nov. 1979.
26. Culik, K. What is a flowchart loop and about structured programming. *SIGPLAN Notices* (ACM) *15*, 1 (Jan. 1980), 45–57.
27. Denning, P.J. Comments on mathematical overkill. *SIGPLAN Notices* (ACM) *10*, 9 (Sept. 1975), 10–11.
28. Denning, P.J. Two misconceptions about structured programming. Proc. Ann. ACM Conf., Minneapolis, Minn., Oct. 1975, pp. 214–215.
29. Dijkstra, E.W. Go to statement considered harmful. *Comm. ACM 11*, 3 (March 1968), 147–148.
30. Donaldson, J.R. Structured programming. *Datamation 19*, 12 (Dec. 1973), 52–54.
31. Elgot, C.C. Monadic computation and iterative algebraic theories. In *Logic Colloquium '73*, North-Holland Pub. Co., Amsterdam, 1975, pp. 175–230.
32. Elgot, C.C. Structured programming with and without go to statements. *IEEE Trans. Software Eng. SE-2*, 1 (March 1976), 41–54.
33. Engeler, E. Structure and meaning of elementary programs. Proc. Symp. Semantics of Algorithmic Languages, *Lecture Notes in Math.*, Vol. 188, Springer-Verlag, New York, 1971, pp. 89–101.
34. Engelfriet, J. *Simple Program Schemes and Formal Languages. Lecture Notes in Comptr. Sci.*, Vol. 20, Springer-Verlag, New York, 1974.
35. Fischer, B., and Fischer, H. *Structured Programming in PL/I and PL/C*. Marcel Dekker, Inc., New York and Basel, 1976.
36. Goldschlager, L.M. Synchronous parallel computation. Ph.D. th., TR-114, Univ. of Toronto, Dec. 1977.
37. Goodman, S.E., and Hedetniemi, S.T. *Introduction to the Design and Analysis of Algorithms*. McGraw-Hill, New York, 1977.
38. Greibach, S.A. *Theory of Program Structures: Schemes, Semantics, Verification. Lecture Notes in Comptr. Sci.*, Vol. 36, Springer-Verlag, New York, 1975.
39. Harel, D. And/or programs: A new approach to structured programming. Proc. IEEE Specifications for Reliable Software Conf., Cambridge, Mass., April 1979, pp. 80–90.
40. Harel, D. On folk theorems. *Comm. ACM 23*, 7 (July 1980).
41. Harel, D., Norvig, P., Rood, J., and To, T. A universal flowcharter. In Proc. AIAA/IEEE/ACM/NASA Comptrs. in Aerospace Conf. II, Los Angeles, Calif., Oct. 1979, pp. 218–224.
42. Hirose, K., and Oya, M. Some results in general theory of flow charts. Proc. of the First USA-Japan Comptr. Conf., Sponsored by AFIPS, Tokyo, Japan, Oct. 1972, pp. 367–371.
43. Hirose, K., and Oya, M. General theory of flow charts. *Comment. Math. Univ. St. Pauli*, XXI-2 (1972), 55–71.
44. Hopkins, M.E. A case for goto. *SIGPLAN Notices* (ACM) *7*, 11 (Nov. 1972), 59–62; Proc. ACM Ann. Conf., Boston, 1972.
45. Hughes, J.K. *PL/I Structured Programming*. John Wiley and Sons, New York, 1973.
46. Hughes, J.K., and Michtom, J.I. *A Structured Approach to Programming*. Prentice-Hall, Englewood Cliffs, N.J., 1977.
47. Indermark, K. On a class of schematic languages. Technical rep. 82, Inst. for Res. and Programming, Gesellschaft fur Mathematik und Datenverarbeitung mbH, Bonn, Germany, Nov. 1974.
48. Jensen, R.W., and Tonies, C.C. *Software Engineering*. Prentice-Hall, Engelwood Cliffs, N.J., 1979.
49. Kasai, T. Translatability of flowcharts into while programs. *J. of Comptr. and Syst. Sciences 9*, 2 (Oct. 1974), 177–195.
50. Kleene, S.C. *Introduction to Metamathematics*. Van Nostrand Co., New York, 1952.
51. Kleene, S.C. General recursive functions of natural numbers. *Math. Annalen 112* (1936), 727–742.
52. Knuth, D.E. Structured programming with go to statements. *Comptng. Surv. 6*, 4 (Dec. 1974), 261–301.
53. Knuth, D.E., and Floyd, R.W. Notes on avoiding "go to" statements. *Inform. Processing Letters 1*, (1971), 23–31.
54. Kosaraju, S.R. Analysis of structured programs. *J. of Comptr. and Syst. Sciences 9*, (1974), 232–255.
55. Kreczmar, A. Effectivity problems of algorithmic logic. In *Annal. Soc. Math. Pol. Series IV; Fundamenta Informaticae 1*, 1 (1977), 19–32.
56. Leavenworth, B.M. Programming with(out) the goto. *SIGPLAN Notices* (ACM) *7*, 11 (Nov. 1972), 54–58; Proc. ACM Ann. Conf., Boston, 1972.
57. Ledgard, H.F., and Marcotty, M. A genealogy of control structures. *Comm. ACM 18*, 11 (Nov. 1975), 629–639.
58. Lee, R.C.T., and Chang, S.K. Structured programming and automatic program synthesis. *SIGPLAN Notices* (ACM) *9*, 4 (April 1974), 60–70; Proc. Symp. on Very High Level Languages, Santa Monica, Calif., March 1974.
59. Linger, R.C., and Mills, H.D. On the development of large reliable programs. In *Current Trends in Programming Methodology*, Vol. 1, R.T. Yeh, Ed., Prentice-Hall, Englewood, Cliffs, N.J., 1977, pp. 120–139.
60. Linger, R.C., Mills, H.D., and Witt, B.I. *Structured Programming: Theory and Practice*. Addison-Wesley, Reading, Mass., 1979.
61. Martin, J.J. The "natural" set of basic control structures. *SIGPLAN Notices* (ACM) *8*, 12 (Dec. 1973), 5–14.
62. McGowan, C. Structured programming: A review of some practical concepts. *Computer 8*, 6 (1975) 25–30.
63. McGowan, C.L., and Kelly, J.R. *Top-Down Structured Programming Techniques*. Petrocelli/Charter, New York, 1975.
64. Merton, R.K. *On the Shoulders of Giants: A Shandean Postscript*. Harcourt, Brace and World, New York, 1965.
65. Meyer, A.R., and Ritchie, D.M. The complexity of loop programs. IBM Res. Rep. RC-1817, 1966.
66. Miller, E.F., Jr., and Lindamood, G.E. Structured programming: Top-down approach. *Datamation 19*, 12 (Dec. 1973), 55–57.
67. Mills, H.D. Mathematical foundations for structured programming. IBM rep. FSC 72-6012, Fed. Syst. Div., Gaithersburg, Md., 1972.
68. Mills, H.D. The new math of computer programming. *Comm. ACM 18*, 1 (Jan. 1975), 43–48.
69. Mills, H.D. How to write correct programs and know it. Proc. IEEE Tutorial on Structured Programming, Washington, D.C., Sept. 1975, pp. 84–91.
70. Mirkowska, G. Algorithmic logic and its applications. Doctoral diss., Univ. of Warsaw, 1972 (In Polish).
71. Myers, G.J. *Reliable Software through Composite Design*. Van Nostrand Reinhold Co., New York, 1975.
72. Myers, G.J. *Composite/Structured Design*. Van Nostrand Reinhold Co., New York, 1978.
73. Nassi, I., and Shneiderman, B. Flowchart techniques for structured programming. *SIGPLAN Notices* (ACM) *8*, 8 (Aug. 1973), 12–26.
74. Neely, P.M. On program control structure. Proc. Ann. ACM Conf., Atlanta, Ga., 1973, pp. 119–125.
75. Nicholls, J.E. *The Structure and Design of Programming Languages*. Addison-Wesley, Reading, Mass., 1975.
76. Partch, B. Improved technology for application development management overview. Proc. SHARE XLI, Miami, Florida, Aug. 1973, pp. 1281–1300.
77. Perkowska, E. Theorem on the normal form of a program. *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys. 22*, 4 (1974), 439–442.
78. Peterson, W.W., Kasami, T., and Tokura, N. On the capabilities of while, repeat, and exit statements. *Comm. ACM 16*, 8 (Aug. 1973), 503–512.
79. Plum, T. W-S. Mathematical overkill and the structure theorem. *SIGPLAN Notices* (ACM) *10*, 2 (Feb. 1975), 32–33.
80. Prather, R.E. Structured Turing machines. *Inform. and Control 35* (1977), 159–171.
81. Pratt, V.R. Semantical considerations on Floyd-Hoare logic. Proc. 17th Symp. on Foundations of Comptr. Sci., Houston, Texas, Oct. 1976, pp. 109–129.
82. Rogers, H., Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967.

83. Tiuryn, J. Fixed-points and algebras with infinitely long expressions, Part II. μ-clones of regular algebras. Technical rep. 311, Inst. of Comptr. Sci., Polish Acad. of Sci., 1978.
84. Van Gelder, A. Structured programming in COBOL: An approach for application programmers. *Comm. ACM 20*, 1 (Jan. 1977), 2–12.
85. Wand, M. A concrete approach to abstract recursive definitions. In *Automata, Languages and Programming*, M. Nivat, Ed., North-Holland Pub. Co., Amsterdam, 1973, pp. 331–341.
86. Wise, D.S., Friedman, D.P., Shapiro, S.C., and Wand, M. Boolean-valued loops. *BIT 15* (1975), 431–451.
87. Wulf, W.A. A case against the goto. *SIGPLAN Notices* (ACM) 7, 11 (Nov. 1972), 63–69; Proc. ACM Ann. Conf., Boston, 1972.
88. Yourdon, E. *Techniques of Program Structure and Design*. Prentice-Hall, Englewood Cliffs, N.J., 1975.
89. Yourdon, E. *Managing the Structured Techniques*. Second edition, Yourdon Press, New York, 1979. (First edition, *How to Manage Structured Programming*, 1976.)
90. Yourdon, E., and Constantine, L.L. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Designs*. Yourdon Press, New York, 1978.
91. Zelkowitz, M.V., Shaw, A.C., and Gannon, J.D. *Principles of Software Engineering and Design*. Prentice-Hall, Englewood Cliffs, N.J., 1979.

Management Science/ Operations Research
D.F. Shanno
Editor

Gamma Variate Generators with Increased Shape Parameter Range

R.C.H. Cheng and G.M. Feast
University of Wales Institute of Science and Technology

Recently developed gamma generators are compact, easily programmed, and possess a uniform speed over the operating range, but are only valid for $\alpha > 1$. A transformation of variable together with a technique suggested by Kinderman and Monahan of generating random variates using the ratio of uniform variates are combined to produce a family of generators valid for all $\alpha > 1/n$ where n is an arbitrary integer. Thus if n is greater than unity, variates with α less than unity can be sampled. The cases $n = 2$ and $n = 4$ are considered explicitly and are shown to retain the features of compactness, ease of programming, and uniform speed.

Key Words and Phrases: gamma variates, random numbers, simulation
CR Categories: 5.5, 8.1

1. Introduction

Since the publication of the review paper of Atkinson and Pearce [2], there have appeared a large number of gamma variate generators covering the case $\alpha > 1$ where α is the shape parameter. One of the more noteworthy methods is based on the general technique of generating random variates using the ratio of uniform variates proposed originally by Kinderman and Monahan [6]. Kinderman and Monahan [7] and Cheng and Feast [5] have suggested algorithms using this technique to generate gamma variates. The former paper also contains some sound criteria for the choice of an optimal algo-

Authors' present address: R.C.H. Cheng and G.M. Feast, Department of Mathematics, UWIST, King Edward VII Ave., Cardiff CF1 3NU, UK.