Conversion of unstructured flow diagrams to structured form

M. H. Williams and H. L. Ossher

Department of Computer Science, Rhodes University, Grahamstown 6140, South Africa

Various workers have suggested methods for converting unstructured flow diagrams into equivalent structured ones. Some of these are compared here and a general method for performing such a conversion is discussed.

(Received November 1975, Revised October 1976)

1. Introduction

Because of the desirable properties of structured flow diagrams, the process of converting an unstructured flow diagram to an equivalent structured one has attracted the attention of several workers.

The three basic constituents of structured flow diagrams, known as *base diagrams*, are simple sequence, selection (**if—then—else**) and the **while** loop. Base diagrams can be nested to form complex flow diagrams. A flow diagram which consists entirely of a nest of base diagrams will here be referred to as *structured*, whereas one which does not will be referred to as *unstructured*. (Three other constructs are sometimes also considered basic elements of structured flow diagrams, viz. the **until** loop, the **case** construct and the **loop—while—repeat** construct.)

The intention behind designing a method to structure flow diagrams automatically is not so much to use it for improving unstructured flow diagrams *per se*, as to prove that *any* algorithm can be expressed in a reasonable structured form. However, since some workers involved in automatic programming (Cheatham and Wegbreit, 1972) are looking at the problem of rearranging a program so as to improve it, the techniques for structuring flow diagrams may not go amiss.

In a previous paper (Williams, 1976) unstructuredness in flow diagrams was shown to be due to the presence of one or more of the following:

(*a*) abnormal selection path

(*b*) loop with multiple exit points

(*c*) loop with multiple entry points

(*d*) overlapping loops

(*e*) parallel loops.

These forms are illustrated in **Fig. 1.** It is necessary and sufficient for a general structuring method to be able to eliminate them from any flow diagram.

Several workers have put forward methods for converting unstructured flow diagrams to structured form (Böhm and Jacopini, 1966; Ashcroft and Manna, 1972; Wulf, 1972). A method is described below which is both general and sufficiently well-defined to be performed automatically by computer. It has certain advantages over the other methods mentioned, which should become apparent in the final section where the various methods are compared.

2. Overview of method

An unstructured flow diagram is converted to structured form by systematic structuring of the first three of the forms shown in **Fig. 1**, viz:

(*a*) abnormal selection paths

(*b*) loops with multiple exit points

(*c*) loops with multiple entry points.

The general forms of multiple exit loops and multiple entry loops are shown in **Fig. 2.** In each case, H is the *head* of the loop, R its *range* and K the *back-branch* leading from the end of the range to the head. $x_1, x_2, \ldots, x_m$ $(m \geqslant 2)$ are exits leading to points $X_1, X_2, \ldots, X_m$ outside the loop. Similarly, $e_0, e_1, \ldots, e_n$ $(n \geqslant 1)$ are entries leading from points

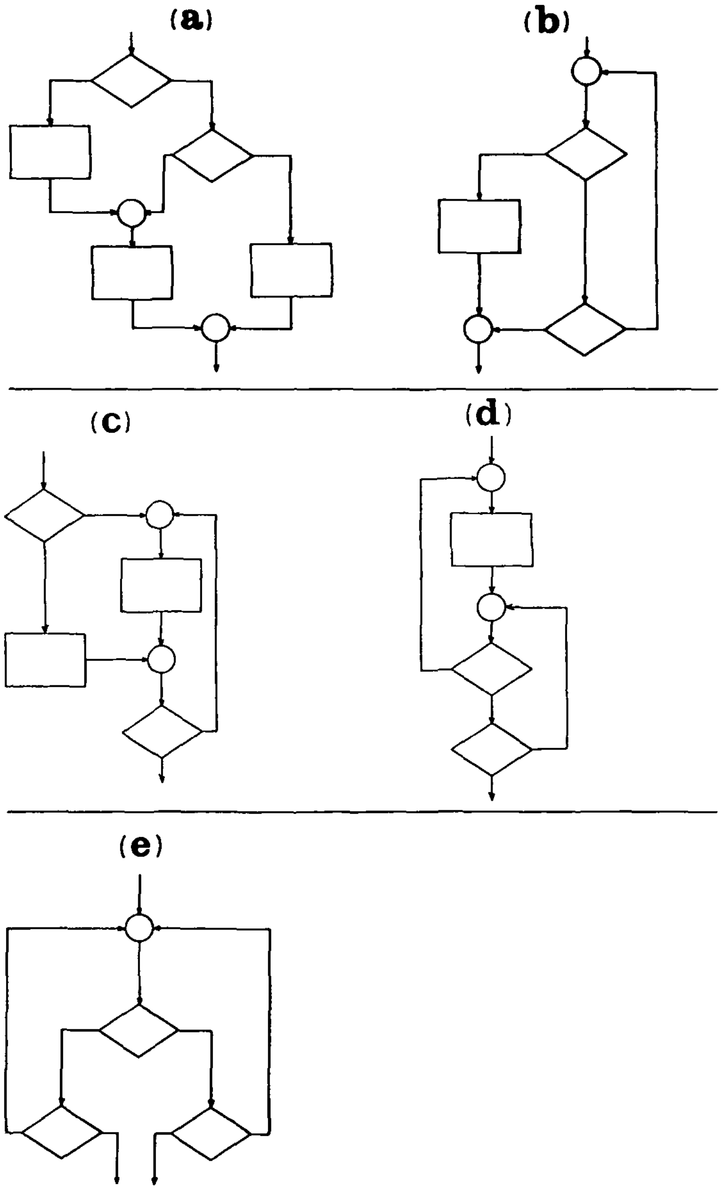


Fig. 1 **The five basic structures which cause unstructuredness in flow diagrams**
(*a*) Abnormal selection path
(*b*) Loop with multiple exit points
(*c*) Loop with multiple entry points
(*d*) Overlapping loops
(*e*) Parallel loops

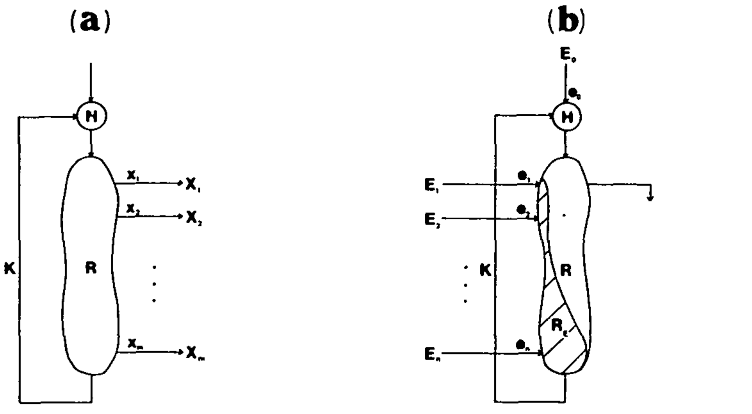


Fig. 2 The general forms of loops
(*a*) Loop with multiple exit points
(*b*) Loop with multiple entry points

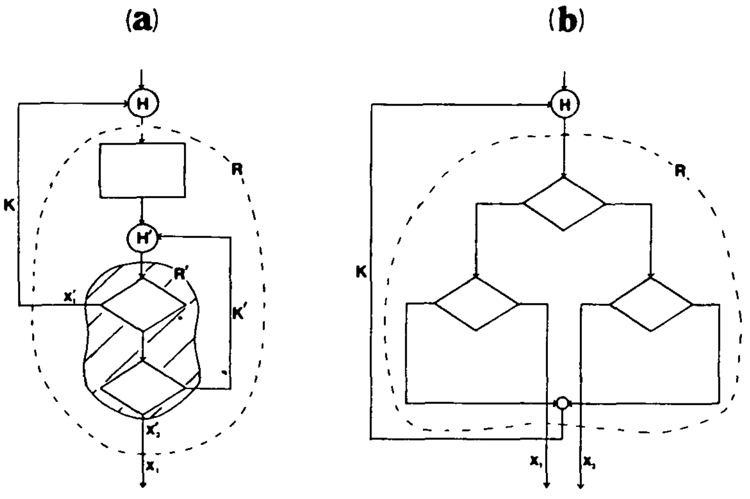


Fig. 3 (*a*) The overlapping loops in Fig. 1(*d*) drawn as nested multiple exit loops
(*b*) The single multiple exit loop corresponding to the parallel loops in Fig. 1(*e*)

$E_0, E_1, \ldots, E_n$ outside the loop to points within it. All entries except e_0 are termed *abnormal*, and R_E consists of every box in R which can be reached from an abnormal entry without branching back to H.

The method owes its generality to the fact that overlapping loops can be interpreted as nested multiple exit loops, and parallel loops can easily be converted to single multiple exit loops. This is illustrated by **Figs.** 3(*a*) and 3(*b*), which show the overlapping and parallel loops in Figs. 1(*d*) and 1(*e*), respectively, redrawn as multiple exit loops with heads, ranges, back-branches and exits clearly shown. The actual boxes of the overlapping loops remain unchanged, and the only change to the parallel loops is that both back-branches have been connected to an additional circle created at the end of the range, which in turn has been connected to the loop head by a single back-branch. This treatment of overlapping and parallel loops is certainly artificial, but has the advantage that it allows loops to be considered individually and virtually independently.

In brief outline, the method is as follows:

(*a*) replace all 'STOP' boxes by a single one, so the flow diagram will have only one exit point

(*b*) insert a circle at each junction point so that circles are the only flow diagram elements having more than one arrow leading to them

(*c*) identify each loop, determining its head, range, exits and entries

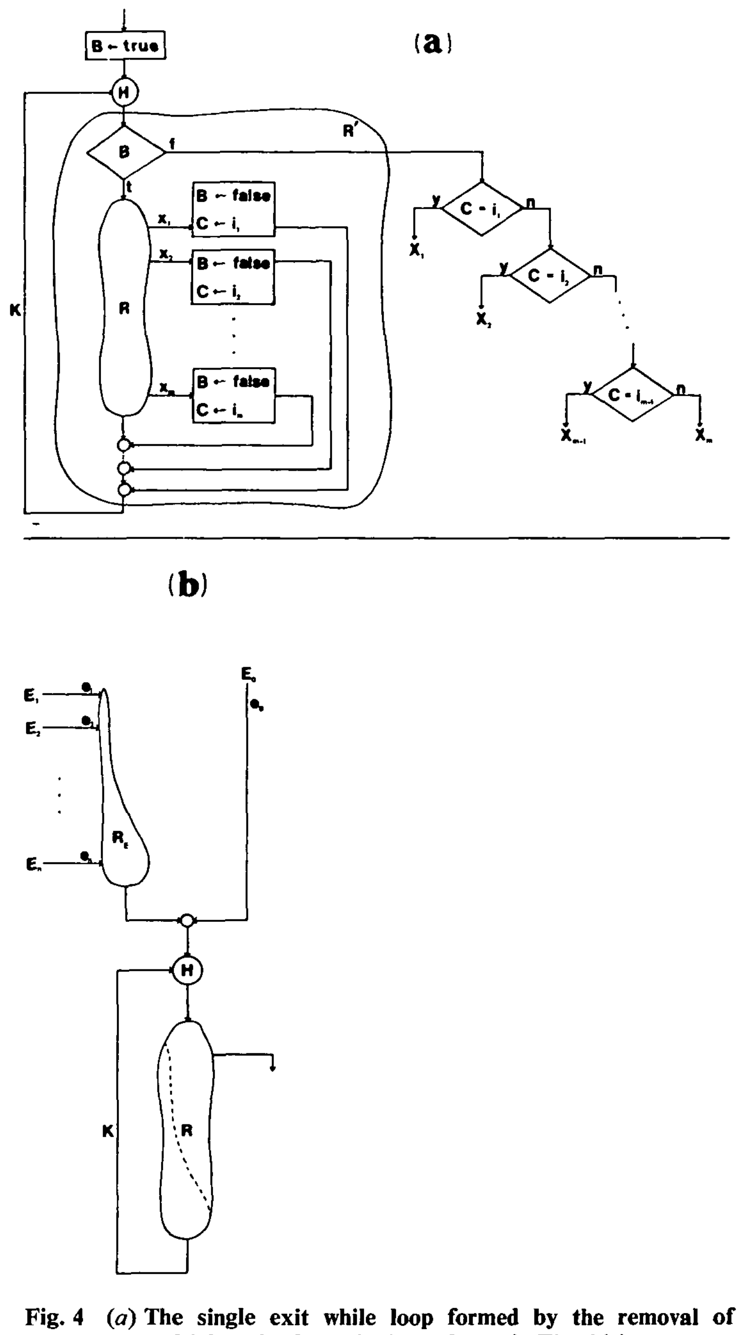


Fig. 4 (*a*) The single exit while loop formed by the removal of multiple exits from the loop shown in Fig. 2(*a*)
(*b*) The structured while loop formed by the removal of multiple entries from the loop shown in Fig. 2(*b*)

(*d*) convert each unstructured loop to structured form by eliminating multiple exits and entries

(*e*) eliminate abnormal selection paths using the well-known method of duplication or node-splitting

(*f*) simplify the resultant structured flow diagram if necessary.

The first two steps are merely preliminary ones, and require no further explanation; nor does node-splitting, which is the accepted technique for eliminating abnormal selection paths. The other steps are considered in detail below. They have been split into several substeps for ease of description, some of which may be combined in practice.

3. Identification of loops

To facilitate the systematic conversion of loops to structured form, it is necessary to identify the heads, ranges, exits and entries of each loop in the flow diagram. Although this can often be done very easily by inspection, the algorithm for performing it automatically is by no means trivial. The following is an outline of it:

(*a*) find the heads of all loops and convert parallel loops to single multiple exit loops. A procedure for doing this is given in **Appendix 1.** It is based on the fact that a loop head is a circle having a path leading from it which eventually returns to it, without passing through the head of an outer loop

(*b*) assign level numbers to all loop heads in such a way that heads of inner loops have higher level numbers than heads of outer loops. A procedure for doing this is given in Appendix 1

(*c*) determine the range of each loop and mark every box with an indication of the innermost loop to which it belongs. This can be done using a procedure given in Appendix 1

(*d*) identify all exits from and entries to each loop. An exit from a loop is any branch leading from a box within the loop to one outside it. Similarly, an entry to a loop is any branch leading from a box outside the loop to one within it. All loop exits and entries can therefore be identified by examining each branch in the flow diagram once.

4. Structuring of loops

Once all loops have been identified, the following algorithm can be used to structure them:

(*a*) select one of the most deeply nested loops with abnormal exits and/or abnormal entries. All multiple exits from a loop are abnormal, and a single exit which is not at the top of the range is considered abnormal as well, because, in Section 1, the **while** loop was assumed to be the only form of loop allowed in structured flow diagrams

(*b*) if the loop has abnormal exits, then, ignoring possible abnormal entries, it has the form shown in Fig. 2(*a*) (except that it might have only one exit, which is not at the top of the range). Choose a unique Boolean variable B and distinct integers $i_1, i_2, \ldots, i_m$ (one for each exit), and convert the loop to the single exit **while** loop shown in **Fig.** 4(*a*). A box setting B **true** must be inserted before each entry (normal or abnormal) to the loop. C is a case variable which may be used for all loops

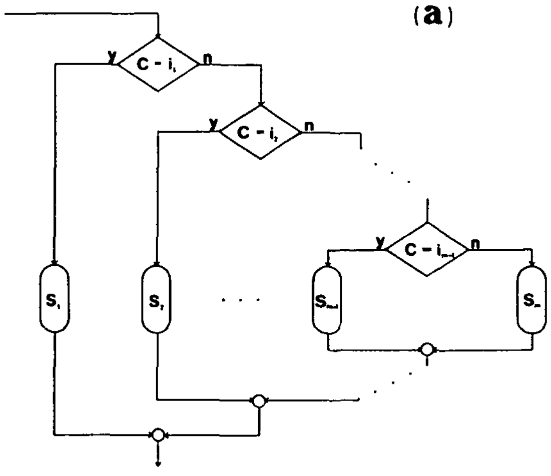
(*c*) if the loop has multiple entries, then, since any possible abnormal exits have already been eliminated, it has precisely the form shown in Fig. 2(*b*), with the exit at the top of the range. (If step (*b*) is applied to the loop, each E_i will be a box setting the unique Boolean variable **true**, and the range R will be the new range R' shown in Fig. 4(*a*).) Convert it to the structured **while** loop shown in Fig. 4(*b*) by duplicating all those boxes in R which can be reached from the abnormal entries (i.e. R_E)

(*d*) if any unstructured loops remain, repeat this entire process.

5. Simplification

Although structured, loops of the form shown in Fig. 4(*a*) are often very clumsy because of the decision boxes testing the case variable at their exit points. These will have been converted to structured selections by node-splitting, and so will have the form shown in **Fig.** 5(*a*). If it makes the loop any simpler, each selection path may be incorporated at the appropriate point in the loop's range, as shown in Fig. 5(*b*). The case variable is no longer required.

This process will not always produce simpler loops, and so should be applied with discretion. Note also that it cannot be applied at all until the flow diagram has been completely structured.



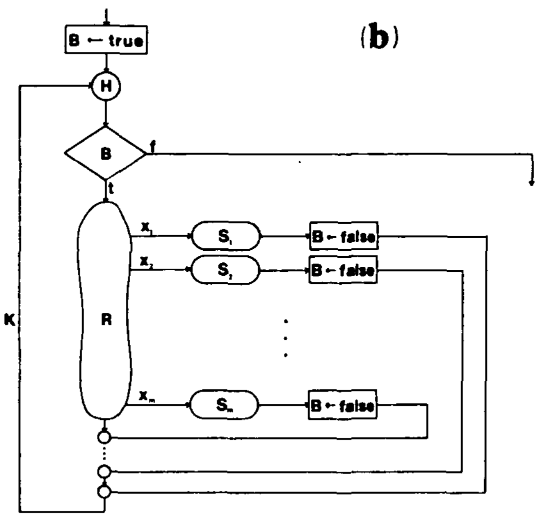


Fig. 5 (*a*) **The selection elements testing the case variable on exit from a loop of the form shown in Fig. 4(*a*)**
(*b*) **The loop with the selection paths incorporated in its range**

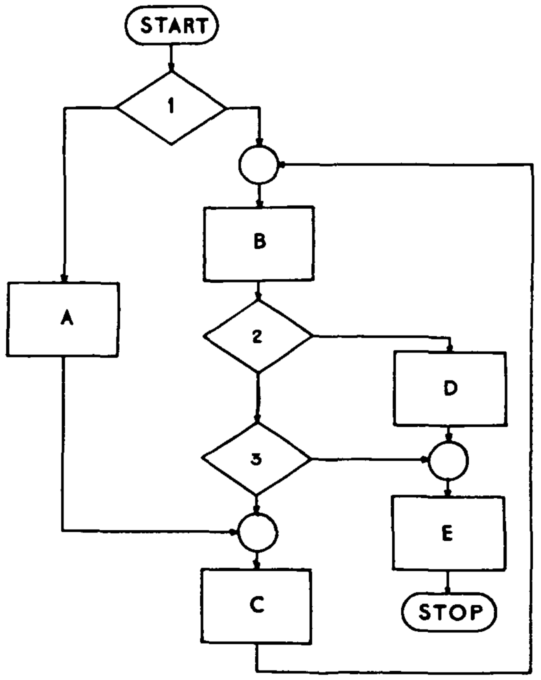
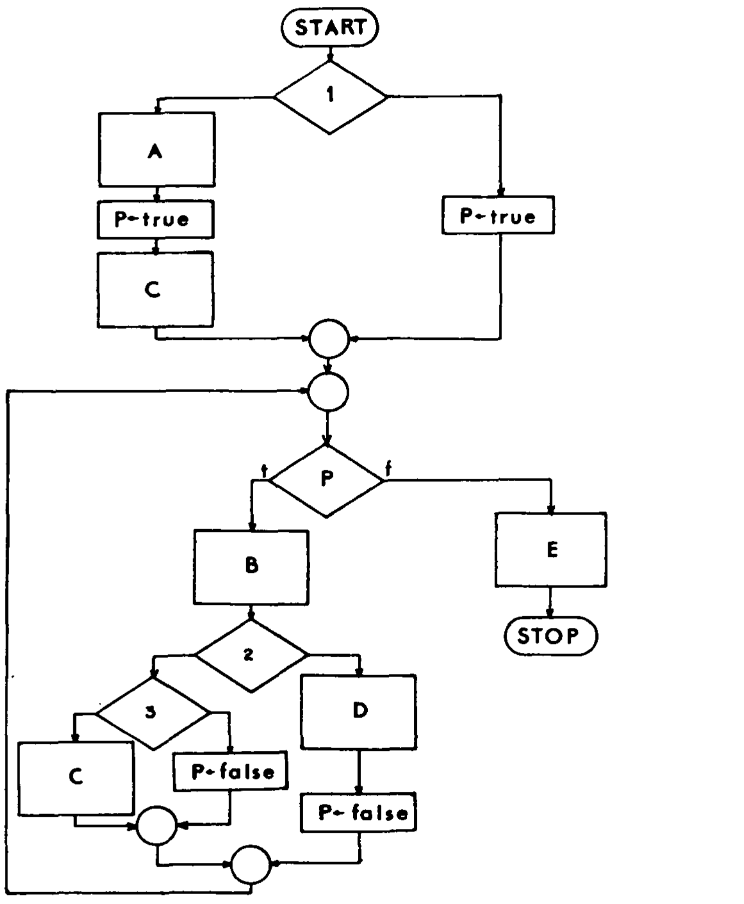
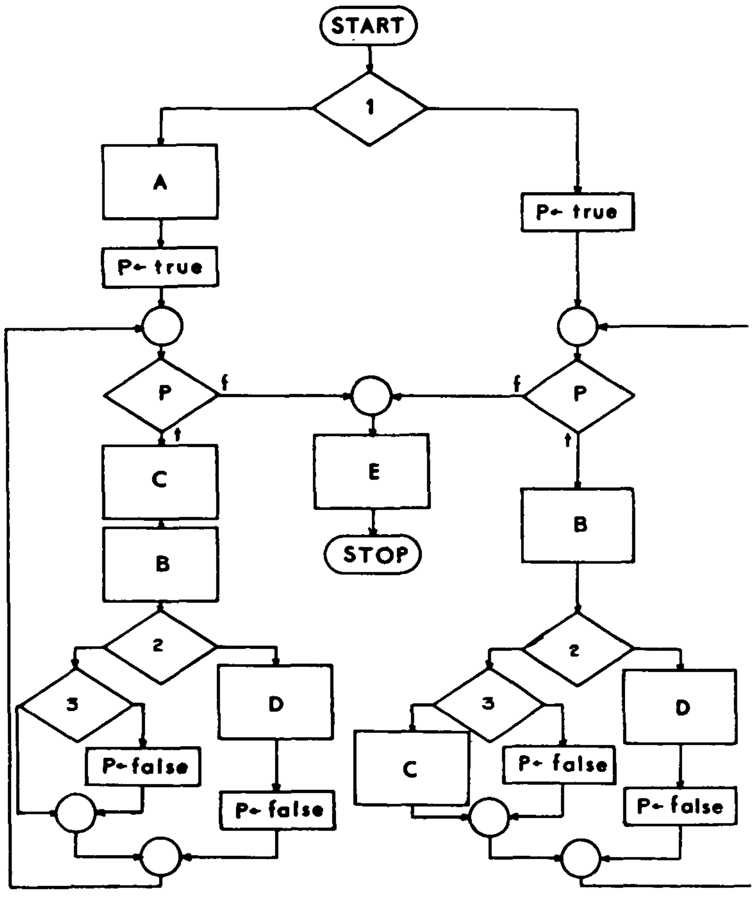


Fig. 6 (*a*) **Flow diagram containing loop with multiple exit and multiple entry points**



(*b*) Equivalent structured form of Fig. 6(*a*), obtained using the current method



(*c*) Equivalent structured form of Fig. 6(*a*), obtained using Ashcroft and Manna's method

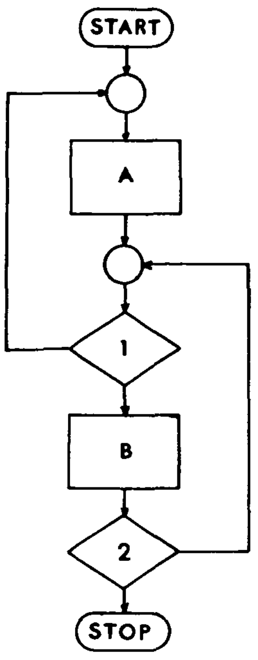
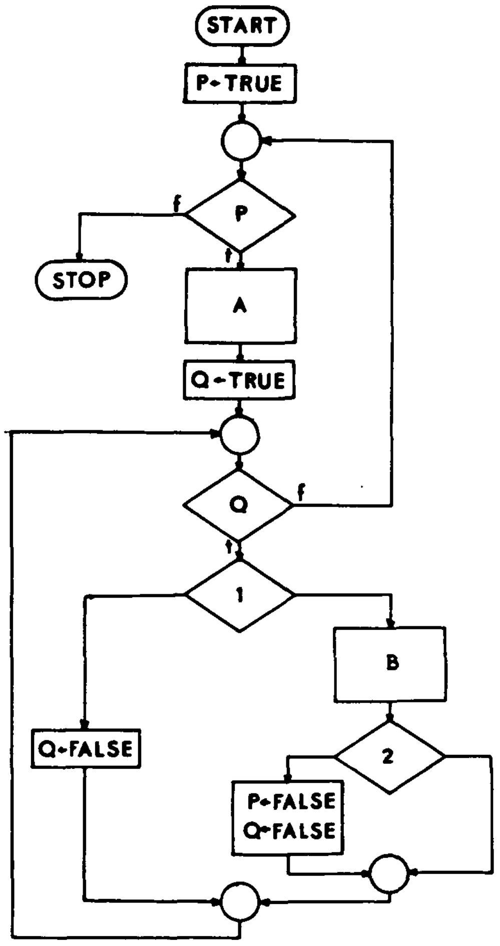


Fig. 7 (*a*) Flow diagram containing overlapping loops



(*b*) Equivalent structured form of Fig. 7(*a*), obtained using the current method

6. Comparison of methods

The method described above and the various other methods mentioned in Section 1 will now be compared by considering their effects on the four forms of unstructured loop shown in Figs. 1(*b*)-(*e*). All methods use node-splitting to handle abnormal selection paths (Fig. 1(*a*)), which are therefore of little interest.

A loop with both multiple exits and multiple entries is shown in **Fig.** 6(*a*), and the result of applying the method described above to it, in Fig. 6(*b*). Ashcroft and Manna's method (1972) can also handle this loop, converting it to the double loop drawn in Fig. 6(*c*).

The other methods do not cater for multiple entry loops at all and so are not completely general. They do handle multiple exit loops, however. Böhm and Jacopini's method (1966) has much the same effect on them as the current method, whereas the effect of Wulf's method is similar to that of the current method without the simplification step.

Fig. 7(*a*) is an example of overlapping loops. The current method converts it to the nested loops shown in Fig. 7(*b*). Ashcroft and Manna's method can also handle this case, though in a slightly different way, but the other methods do not seem able to do so.

Finally, consider the parallel loops drawn in **Fig.** 8(*a*) (Mullins *et al.*, 1974). The equivalent structured form produced by the current method is shown in Fig. 8(*b*). Both Ashcroft and Manna's and Böhm and Jacopini's methods produce similar results. Wulf's method, on the other hand, converts the parallel loops to the **while** loop shown in Fig. 8(*c*), which is very similar to the flow diagram produced by the current method if no simplification is performed.

7. Conclusion

Various methods for converting unstructured flow diagrams to equivalent structured ones have been compared in the light of the five structures which cause unstructuredness and a general conversion method, capable of being performed automatically, has been outlined. A computer program which restructures flow diagrams, based on this technique, has been written and has been applied successfully to a number of simple examples.

Of course, mechanical restructuring of this type does not

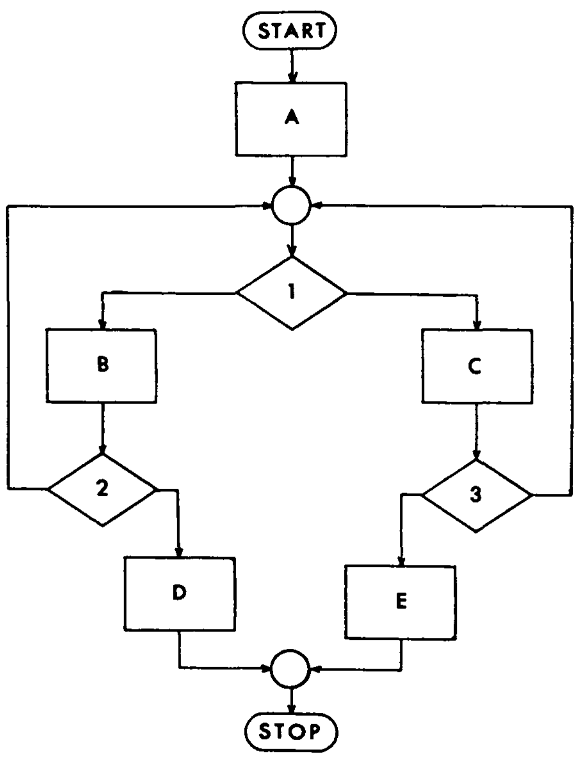
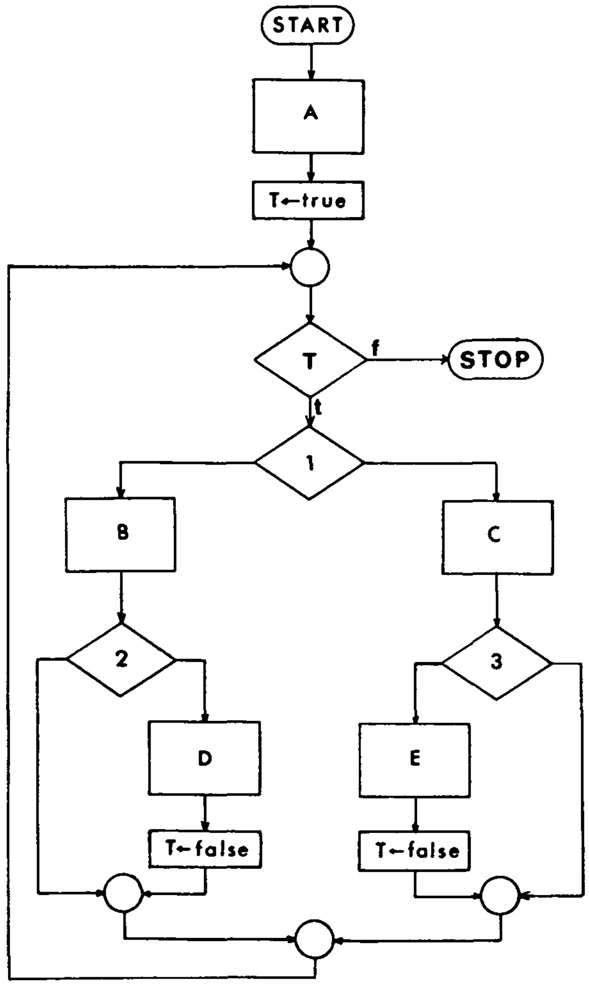
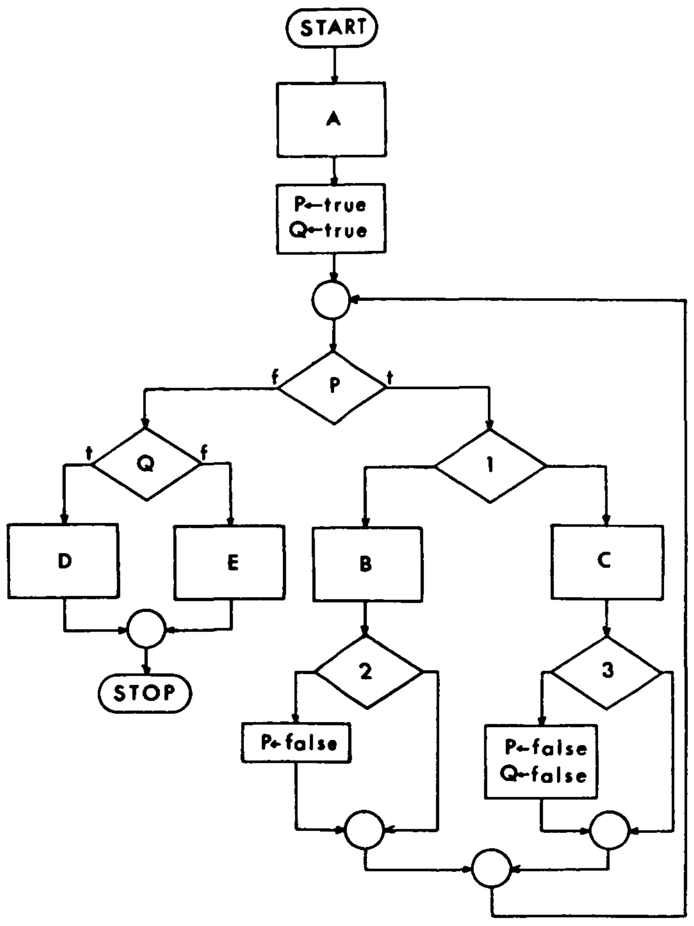


Fig. 8 (*a*) **Flow diagram containing parallel loops (Mullins** *et al*)



(*b*) **Equivalent structured form of Fig.** 8(*a*), **obtained using the current method**



(*c*) **Equivalent structured form of Fig.** 8(*a*), **obtained using Wulf's method** always produce the most efficient or most pleasing solution, since it does not take account of information contained within the flow diagram boxes to reduce the number of Boolean variables required and simplify the equivalent structured flow diagram. This is a much more difficult problem, which in general has not been solved.

Acknowledgements
One of the authors, Mr H. L. Ossher, would like to thank the South African CSIR for a grant to enable this work to be completed.

Appendix 1

The procedures used to identify loops are given below in a semi-formal notation. They operate on flow diagrams represented by linked lists of boxes. Each box has a number of fields, including a CLASS field (specifying whether the box is a process box, decision box, circle or terminal box), and a LOOP field (to hold a pointer to the innermost loop to which the box belongs). Each circle has four additional fields, viz. a TYPE field (specifying the type of the circle, e.g. *loop head* or *selection circle*), a LEVEL field (to contain the level number assigned to the circle), a FLAG field and a PROCESSED field (used by the procedure MARK to indicate whether or not the circle has already been processed).

Three functions, NEXT, RIGHT and LEFT, are used by the procedures for following paths within flow diagrams. NEXT (BOX), where BOX is a process box or a circle, is the box to which the arrow leaving BOX points; RIGHT(BOX) and LEFT(BOX), where BOX is a two-way decision, are the boxes to which the rightmost and leftmost arrows leaving BOX point, respectively:

The procedure FINDHEADS, given below, does three things:

(*a*) classifies circles as either *loop heads* or *selection circles*, and marks them accordingly (by setting their TYPE fields)

(*b*) converts parallel loops to single multiple exit loops; and

(*c*) duplicates circles where necessary so that each circle will have precisely two arrows pointing to it.

It should be called with the actual parameter specifying the first box of the flow diagram.

```
FINDHEADS (BOX) =
  if BOX is a process box then
                         FINDHEADS (NEXT (BOX)) else
  if BOX is a decision box then
                         FINDHEADS (RIGHT (BOX));
                         FINDHEADS (LEFT (BOX)) else*

  if BOX is a circle then
    if BOX is unmarked then
      mark BOX "VISITED ONCE";
      FINDHEADS (NEXT BOX));
      if BOX is still marked "VISITED ONCE"
        then mark BOX "PROVISIONAL SELECTION
               CIRCLE"
        else mark BOX "LOOP HEAD" fi else
    if BOX is marked "VISITED ONCE" then
      mark BOX "PROVISIONAL LOOP HEAD" else
    if BOX is marked "PROVISIONAL SELECTION
           CIRCLE" then
      mark BOX "SELECTION CIRCLE" else
    if BOX is marked "PROVISIONAL LOOP HEAD" then
      CONVERT PARALLEL LOOP else
    if BOX is marked "SELECTION CIRCLE" then
      DUPLICATE SELECTION CIRCLE else
    if BOX is marked "LOOP HEAD" then
      CREATE SELECTION CIRCLE fi fi fi fi fi fi fi fi fi
```

*These may be performed in the reverse order, i.e.
. . . FINDHEADS (LEFT (BOX));
FINDHEADS (RIGHT (BOX)) . . .
in which case the order must be reversed at the relevant points in all the other procedures as well. The effect produced, though equivalent, may be slightly different.

CONVERT PARALLEL LOOP is called when a circle is encountered which has two back-branches leading to it (i.e. which is the head of parallel loops). It converts the parallel loops to a single multiple exit loop as described in Section 2. DUPLICATE SELECTION CIRCLE or CREATE SELECTION CIRCLE is invoked when a circle is encountered which has too many arrows (other than back-branches) leading to it, to create a new selection circle and thereby reduce the number of arrows leading to the original circle.

The procedure ASSIGNLEVELS, given below, assigns level numbers to all circles in such a way that if a path (not including the back-branch of a loop) leads from one circle to another, then the first will have a lower level number than the second. As a result, in the case of nested loops, the head of the inner loop will always have a higher level number than the head of the outer loop. The level numbers assigned to loop heads can thus be used to determine the way in which loops are nested. Those assigned to selection circles, on the other hand, help to perform node-splitting automatically.

When ASSIGNLEVELS is called, the first actual parameter should specify the first box of the flow diagram and the second the level number to be given to the first circle encountered.

```
ASSIGNLEVELS (BOX, LEV) =
  if BOX is a process box then
                    ASSIGNLEVELS (NEXT(BOX), LEV) else
  if BOX is a decision box then
                    ASSIGNLEVELS (RIGHT(BOX), LEV);
                    ASSIGNLEVELS (LEFT(BOX), LEV) else
  if BOX is a circle then
    if BOX is an unflagged loop head then
      BOX[LEVEL]←LEV;
      flag BOX;
      ASSIGNLEVELS (NEXT (BOX), LEV+1) else
    if BOX is a flagged loop head then unflag BOX else
  if BOX is an unflagged selection circle then
    BOX[LEVEL]←LEV;
    flag BOX else
  if BOX is a flagged selection circle then
    BOX[LEVEL]←MAX(BOX[LEVEL], LEV);
    unflag BOX;
    ASSIGNLEVELS (NEXT(BOX), BOX[LEVEL]+1)
                                          fi fi fi fi fi fi fi
```

The procedure MARKRANGES, given below, marks the ranges of all loops by placing, in the LOOP field of each box, a pointer to the head of the innermost loop to which the box belongs. It invokes the procedure MARK, also given, which marks the range of a single loop, and the function POINTERTO, which returns a pointer to the box specified as its parameter. It should be called with the actual parameter specifying the first box of the flow diagram.

```
MARKRANGES (BOX) =
  if BOX is a process box then
                        MARKRANGES (NEXT(BOX)) else
  if BOX is a decision box then
                        MARKRANGES (RIGHT(BOX));
                        MARKRANGES (LEFT(BOX)) else
  if BOX is a circle then
    if BOX is an unflagged loop head then
      flag BOX;
    MARK(NEXT(BOX), POINTERTO(BOX));
    MARKRANGES(NEXT(BOX)) else
  if BOX is an unflagged selection circle then
    flag BOX;
    MARKRANGES (NEXT(BOX)) else
  if BOX is flagged then unflag BOX fi fi fi fi fi fi

MARK(BOX,HPTR) =
  if BOX is a process box then
    MARK (NEXT(BOX), HPTR);
    if NEXT(BOX)[LOOP] = HPTR then
          BOX[LOOP] ← HPTR fi else
  if BOX is a decision box then
    MARK (RIGHT(BOX), HPTR);
    MARK (LEFT(BOX), HPTR);
    if RIGHT(BOX)[LOOP] = HPTR ∨ LEFT
          (BOX)[LOOP] = HPTR then
      BOX[LOOP]←HPTR fi else
  if BOX is a circle then
    if HPTR = POINTERTO (BOX) then
          BOX[LOOP] ←HPTR else
    if BOX is unflagged ∧ BOX[PROCESSED] ≠ HPTR then
      BOX[PROCESSED]←HPTR;
      MARK (NEXT(BOX), HPTR);
      if NEXT(BOX)[LOOP] = HPTR then
            BOX[LOOP] ←HPTR fi fi fi fi fi fi
```

References

Ashcroft, E. and Manna, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71*, North-Holland Publishing Co., pp. 250-255.

Böhm, C. and Jacopini, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Transformation Rules, *CACM*, Vol. 9, No. 5, pp. 366-371.

Cheatham, T. E. and Wegbreit, B. (1972). A Laboratory for the Study of Automatic Programming, *AFIPS Conf. Proc.*, 40, pp. 11-21.

Mullins, J. M., Radue, J. E., and Webb, G. L. (1974). A Tutorial Note on Functional Programming, *Systems/Stelsels*, Vol. 4, No. 8, pp. 13-14.

Williams, M. H. (1976). Generating Structured Flow Diagrams: The Nature of Unstructuredness, *The Computer Journal*, Vol. 20, No. 1, pp. 45-50.

Wulf, W. A. (1972). Programming without the GOTO, in *Proceedings of IFIP Congress 71*, North-Holland Publishing Co., pp. 408-413.

Book reviews

Continued from page 160

is the 'best' book since subjectivity and experience of other methods will bias not only myself but those who are already committed to a particular approach to programming activity. And as the author points out he does not claim infallibility and he recognises the validity of other approaches.

Tausworthe writes easily and it does not take long to become absorbed in what he writes. There are plenty of illustrations and examples to explain further what he wants to put across. Only one note jarred and that is in the earlier sections: on more than one occasion he apologises for moving in a particular direction. Once he gets into his stride, and assumes the reader has decided to go along with him, he drops this mildly irritating posture.

For the 'other camp', that of students and research workers, the book will be less satisfying. Anyone who simply wants a reasonable methodology to apply in programming would be content to read, mark, learn and inwardly digest. Anyone who wants Tausworthe to justify and prove what he writes will not be happy and neither will anyone wanting the state-of-the-art briefing. They should read Yeh's book instead. I fear that some will dismiss Tausworthe also as 'too pragmatic'.

D. R. A. Coan (Manchester)

Simulation of Systems, Proceedings of the 8th AICA Congress, Edited by L. Dekker, 1977; 1122 pages. (*North-Holland*, $91.95)

The proceedings of the 8th AICA Congress takes the now traditional form of preprints together with some short discussion notes and (very valuable) corrections. Major sections of the proceedings cover: System modelling; simulation tools; simulation of specific system, together with examples of current research in the Netherlands presented by demonstration.

Of the theoretical discussions, the paper by Spriet and Vansteenkiste developing Walche functions is likely to be seminal; these functions, the orthogonal binary functions well suited to digital realisation, have proved their worth in several fields now. It is interesting to note that they originate from a highly 'academic' and unpractical thesis in mathematics of the mid-thirties that had to wait for the full development of the digital computer to see practical application. The adaption to the Laplace transform (for linear systems) of the Fast Fourier transform (FFT) is neatly described in a paper by Heinz Waller.

Section Two, simulation tools, has some useful discussion of hybrid computers which will be a source of useful ideas and an attack on defining simulation languages for a range of problems either lying outside conventional engineering or (rather ambitious) in the domain of 'universal' languages. Two other 'in principle' papers are worth mentioning here: the Polish paper (A. Szymański) giving a practical algorithm for a time optimal (Pontryagin) problem with random disturbances and an EAI practical realisation (by Buli and Janác) of an integral equation solution method by analogue computer.

The final section on specific topics is too wide to review in detail but represents a valuable bringing together of practical realisation. The foreword contains Dekker's thoughts on simulation. In the old days we spoke of analogue (or if American, of analog) computers where now we speak of simulating systems. The virtues and vices are still there; limitations in time and equipment require simplifications of the model. These simplifications would be desirable in their own right almost certainly, as a necessity if the experimentation on the model is to be usefully carried out. The art of simulation is no less than the art of analogue programming, for all the availability of 'universal programming languages' and 'hybrid systems with automatic patching'.

The end papers include a short index as well as the panel discussions first on the independence of simulation as a separate discipline, and then on the problem of the credibility of simulation (although the Club of Rome simulations do not have reference here, they are a prime example of the problem of convincing the outsider of credibility).

The proceedings should be available to every technical university, etc.

J. Lewins (London)