Decomposition of flowchart schemata

Ronald E. Prather and Shirla G. Giulieri

Department of Mathematics and Computer Science, University of Denver, Denver, Colorado 80208, USA

A decomposition theory for flowchart schemata is presented, and a series of algorithms for implementing the nested decomposition is discussed. It is suggested that the utilisation of this process as an initial phase of a flowchart structuring routine, will ensure the recognition and preservation of a given flowchart's inherent topology.

(Received October 1979)

1. Introduction

In recent years, there has been a considerable interest in the development of methods for the restructuring of flowcharts. This activity can all be traced to the paper by Bohm and Jacopini (1966), in which it is shown that 'corresponding to each flowchart, there is an equivalent 'structured' flowchart'. Even though the proof of this statement was constructive, in that an algorithm was given for effecting the restructuring, a flood of articles (e.g. Ashcroft and Manna, 1971; Wulf, 1972; Williams, 1974) continued to appear, each claiming a superior structuring technique. A more recent paper (Urschler, 1975) provides a comprehensive summary of these contrasting methods, and offers yet another approach.

In almost every instance where an improved restructuring technique is cited, the authors claim to have preserved or retained the *topology* of the original flowchart. There is no doubt that such an aim is desirable, if we can only decide what it means. In this paper, we seek to lend a more precise understanding to such a claim, and we then describe a flowchart decomposition technique that will ensure that this goal is achieved—at least, by any structuring method that derives this decomposition as its initial phase. In retrospect, our flowchart decomposition theory is seen to be of independent interest, toward obtaining a flowchart *canonical form*, where a program's independent processes are most clearly delineated.

2. Flowchart schemata

Intuitively, a flowchart is a diagram of program flow. In giving uninterpreted abstract names to the flowchart boxes, we speak of a *schema*, and for convenience, we choose to indicate the program flow in an assembly language style. Accordingly, we define a *flowchart schema* F over X and Y to be an n-sequence of labelled statements of the forms

$$i: x(j) \quad \text{or} \quad i: P(j, k)$$

there being one such statement for each i from 1 to n, having

$$1 \leqslant j \quad k \leqslant n + 1 \tag{1}$$

$$x \in X \quad \text{and} \quad P \in Y \tag{2}$$

and, for each label i, at least one path

$$1 \to \cdots \to i \to \cdots \to n + 1 \tag{3}$$

in the relation $\to$ induced by F.

This relation admits $0 \to 1$ to designate the starting point, and relations $i \to j$ for $i: x(j)$, and $i \to j, k$ for $i: P(j, k)$. It is quite common to draw the statements of X as square or rectangular boxes (representing *elementary* operations—of assignment, input, output etc.) and to draw those of Y as diamond-shaped boxes (signifying a program decision); this will be done in the present examples. In the case of the latter, it is also quite common to mark the two outgoing flowlines with distinct symbols; + and − will be used for this purpose. With these conventions, **Fig. 1**(a) shows flowchart schemata with the associated flowchart [Fig. 1(b)]. Note the use of lower case letters for the operations of X and upper case letters for the decisions of Y, another convention that will be followed.

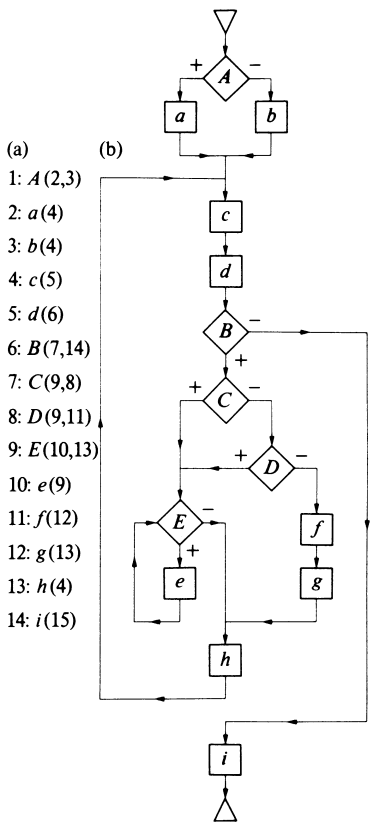

Fig. 1 (a) Flowchart schemata and (b) the associated flowchart

Of primary concern in the sequel are those subsets of labelled statements in a flowchart F, that exhibit some of the same characteristics [i.e. properties (1)–(3) above] as do flowcharts as a whole. Of course, property (2) is already inherited from the overall flowchart. With this much in mind, it can be said that a subset of statements G is a *subflowchart* of the flowchart F if there are distinct labels

$$\nabla \in G \qquad \triangle \notin G$$

and

$$\underset{i \notin G, j \in G}{i \to j \Rightarrow j = \nabla}$$

$$\underset{j \in G, k \notin G}{j \to k \Rightarrow k = \triangle}$$

and also each $j \in G$ lies on at least one path from ∇ to $\triangle$ and we write $G = (\nabla, \triangle)$. Note that except for renumbering, G is a flowchart schema in its own right, and this fact alone

CCC-0010-4620/81/0024-0258 $02.50

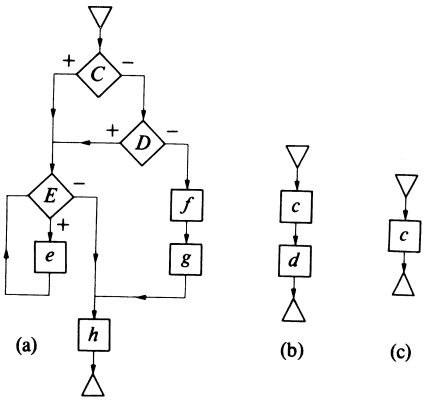

Fig. 2 Three subflowcharts for the flowchart in Fig. 1

should justify our terminology. Moreover, $F = (1, n + 1)$ in viewing F as a subflowchart of itself.

In **Fig. 2**, three subflowcharts are depicted, for the flowchart given in Fig. 1. With the notation just introduced, these would be denoted as (7, 4), (4, 6), and (4, 5), respectively. In reference to the latter, we note that every elementary flowchart box (a member of X) can be viewed as a subflowchart. We call these the *elementary subflowcharts.*

In the case of a general flowchart, it almost seems that there could be little in the way of an underlying theory, something that would be universally applicable. And yet, consider the following relationship among the labelled statements of an arbitrary flowchart (schema) F. We say that i is *dominated* by j (and we write $i \leqslant j$) if every path from i to $n + 1$ includes j. This is easily seen to be a partial order, and the resulting poset is a *tree rooted at* $n + 1$, as can be seen in considering the consequences of the following.

Proposition 1
If F is a flowchart and $i \in F$, then there is a unique maximal chain from i to $n + 1$.

Proof. Since $i < n + 1$ for every $i \in F$, the existence of such a maximal chain follows from finiteness considerations. Uniqueness is shown by induction.

The tree rooted at $n + 1$ so obtained is called the *dominance tree* (Tarjan, 1972) of the given flowchart $F = (1, n + 1)$. For the flowchart of Fig. 1, one obtains the dominance tree of **Fig. 3**, by a method discussed in Section 4. In general, the connection of these trees with the search for subflowcharts is revealed in the following.

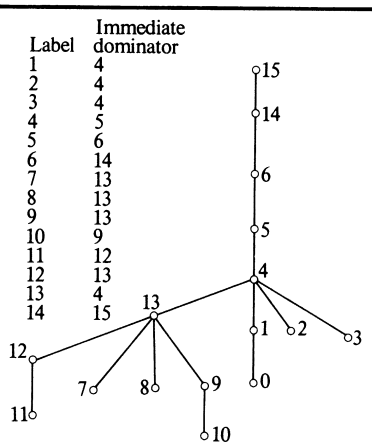

Fig. 3 Dominance tree for Fig. 1

Proposition 2
Let $G = (\nabla, \triangle)$ be a subflowchart of F. Then $\nabla < \triangle$ in the dominance tree of F.

Proof. One must show that every path from ∇ to $n + 1$ includes $\triangle$. Such a path begins in G but ends outside G. According to property (2) in the definition of a subflowchart, the first step in this path to go outside G must be a flowline to $\triangle$; this observation completes the proof.

By way of illustration, note that $7 < 4$, $4 < 6$, and $4 < 5$ in Fig. 3, in accordance with the three subflowcharts of Fig. 2. In order to obtain a partial converse to this proposition, the introduction of the following definition is needed. If $j < k$ in the dominance tree of a flowchart F, the *interval* (j, k) is the collection of labelled statements—excepting that labelled by k itself—lying on paths from j to k. In general, these will not constitute a subflowchart, as is seen by considering the interval (8, 4) in the case of Fig. 1. But the condition that governs such questions is provided in the following proposition, offered without proof.

Proposition 3
Let $j < k$ in the dominance tree of a flowchart F. Then the interval $I = (j, k)$ is a subflowchart of F if and only if

$$h \to i \quad \Rightarrow \quad i = j \qquad (h \notin I,\ i \in I)$$

Note: Write $j \prec k$ in such cases.

3. Decomposition theory

In order to facilitate the discussion of flowchart decomposition, the schemata of Section 2 need to be generalised. Here, a *structured* (flowchart) *schema* S over X and Y is introduced which permits additional labelled statements of the form

$$i: f(j)$$

where the fs are structured schemata, all over again. It is this allowance for recursion that will permit the treatment of the nested decomposition of flowcharts. When interpreting a structured schema as a flowchart F, it is important to remember that the entire flowchart for f is to be substituted at its point of occurrence, with a single entry point i and exit point j. It follows that the flowchart for f is a subflowchart of F.

In general, there will be any number of subflowcharts $f_1, f_2, \ldots f_m$ so appearing in the representation of a structured schema S; this decompositional structure can be denoted

$$F = S(f_1, f_2, \ldots f_m)$$

Of particular interest is the case where the schema takes the form

$$1{:}f_1(2)$$
$$2{:}f_2(3)$$
$$\vdots$$
$$m{:}f_m(m + 1)$$

i.e. that of a *sequential decomposition* into a series of successive subflowcharts. This can be written

$$F = f_1 \circ f_2 \circ \ldots f_m$$

with the occasional use of the alternative form

$$\nabla f_1; f_2; \ldots f_m \triangle$$

in order to call special attention to this important case. Moreover, a flowchart F that has no nontrivial sequential decompositions is said to be *sequentially irreducible.*

Theorem 1
Every flowchart F has a unique sequential decomposition

$$F = f_1 \circ f_2 \circ \ldots f_m$$

into sequentially irreducible subflowcharts.

Proof. For any flowchart $F = (1, n + 1)$, there will exist a unique maximal chain

$$1 = i_0 \prec i_1 \prec \ldots i_m = n + 1$$

with the property that the only flowlines entering (i_k, i_{k+1}) do so in coming from (i_{k-1}, i_k). According to Proposition 3, each of the intervals $f_k = (i_{k-1}, i_k)$ is a subflowchart of F. Moreover, the f_k are sequentially irreducible, or otherwise the maximality of our chain would be contradicted.

With the flowchart of Fig. 1, in relation to the proof of Theorem 1, the maximal chain

$$1 \prec 4 \prec 14 \prec 15$$

is found, and accordingly, the flowchart has the sequential decomposition

$$F = (1, 4) \circ (4, 14) \circ (14, 15)$$

In the alternative form mentioned previously, we thus obtain the structured schema

∇
1: $A(2, 3)$
2: $a(4)$
3: $b(4)$;
1: $c(2)$
2: $d(3)$
3: $B(4, 11)$
4: $C(6, 5)$
5: $D(6, 8)$
6: $E(7, 10)$
7: $e(6)$
8: $f(9)$
9: $g(10)$
10: $h(1)$;
1: $i(2)$
$\triangle$

representing the three sequential subflowcharts of **Fig. 4**.

Note that each of these subflowcharts has the property of being properly contained in a larger subflowchart. Thus, (1, 4) is contained in (1, 14), (4, 14) is contained in (1, 14) and (4, 15), etc. It is evident that a subflowchart of F is *maximal* if it is not properly contained in a larger subflowchart, other than F itself. Moreover, F is *maximally irreducible* if the only maximal subflowcharts are the elementary ones. In general, the components of a sequential decomposition may be decomposed along the following lines.

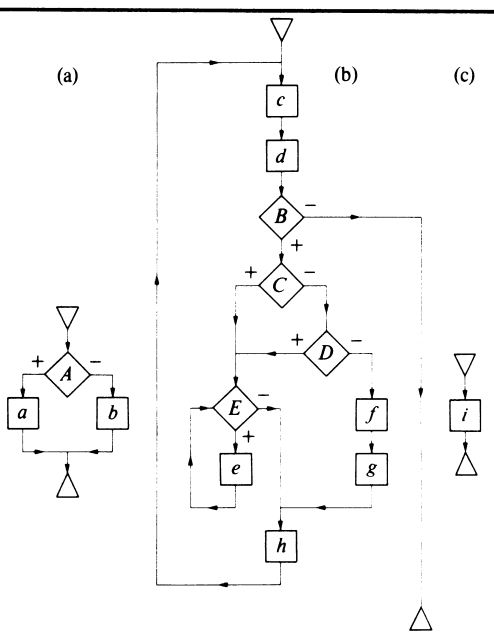

Fig. 4 Three sequential subflowcharts for the maximal chain of Fig. 1

Theorem 2
Let F be any sequentially irreducible flowchart. Then there are unique maximal subflowcharts f_j and a structured schema S, such that

$$F = S(f_1, f_2, \ldots f_n)$$

Proof. The existence of maximal subflowcharts follows from finiteness considerations. Since the elementary subflowcharts are partitioned by the maximal ones, these maximal subflowcharts are unique. Finally, the schema S is obtained by treating the fs as if they were elementary flowchart boxes.

Corollary. Every flowchart F has a unique decomposition

$$F = S_1(f_{11}, f_{12}, \ldots f_{1n_1}) \circ S_2(f_{21}, f_{22}, \ldots f_{2n_2}) \circ \ldots S_m(f_{m1}, f_{m2}, \ldots f_{mn_m})$$

with each f_{ij} maximal in S_i. Since the same is true for the subflowcharts f_{ij}, we are led to a unique nested or iterated *sequential-maximal* decomposition of any flowchart F.

In the decomposition of Theorem 2 for the case of Fig. 4(b), it is found that there are two maximal subflowcharts, namely (1, 3) and (4, 1), as shown in **Fig. 5**. Accordingly, there is a schema S corresponding to Fig. 5(a), and an overall structured schema for representing our decomposition (that of Theorem 2) as

1: $c(2)$		1: 1: $c(2)$
2: $d(3)$		2: $d(3)$ (2)
3: $B(4,11)$		2: $B(3,4)$
4: $C(6,5)$		3: 1: $C(3.2)$
5: $D(6,8)$	$\Rightarrow$	2: $D(3.5)$
6: $E(7,10)$		3: $E(4,7)$
7: $e(6)$		4: $e(3)$
8: $f(9)$		5: $f(6)$
9: $g(10)$		6: $g(7)$
10: $h(1)$		7: $h(8)$ (1)

Note that the two maximal subflowcharts, in turn admit a sequential decomposition, as mentioned in the corollary. Consideration of such nested decomposition of the example, however, is left for discussion in the next section.

4. Algorithmic considerations

The automatic and mechanical implementation of the decomposition of the preceding section is a separate but equally important question. It is not our intention to provide all the details here, but only to identify the broad outlines of the various algorithmic processes, relating to the given decomposition theory. From all that has gone before, it is quite apparent that the dominance tree of a flowchart will play a central role in the implementation. Accordingly, its derivation is the goal of the first of three algorithms to be discussed.

Algorithm 1
In the terminology of Proposition 1, the successor of i in the unique maximal chain from i to $n + 1$ is called the *immediate dominator* of $i \in F$. Clearly, the dominance tree of a flowchart F is completely determined by the list of all of the immediate dominators. These are obtained as follows

(*a*) for $i: x(j)$ the immediate dominator of i is j;

(*b*) for $i: P(j, k)$ all cycle-free paths from i to $n + 1$ must be considered, and the first junction common to all of them is the immediate dominator of i.

Note, however, that a more efficient method for handling the

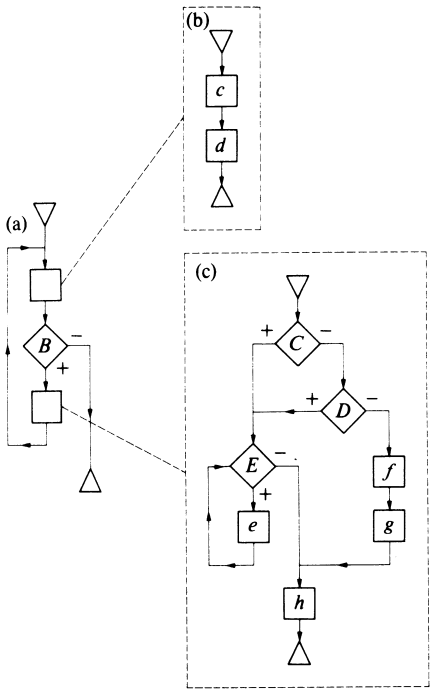

Fig. 5 Two maximal subflowcharts from Fig. 4(b)

latter case has been given elsewhere (Tarjan, 1972). In any event, the list shown at the left of Fig. 3, for our flowchart of Fig. 1, leading to the dominance tree shown, would be obtained.

Algorithm 2

The idea here is to use the dominance tree, together with Propositions 2 and 3, in searching for the sequential subflowcharts f_k to appear in the decomposition of Theorem 1. According to Proposition 2, it is necessary only to find the appropriate subchain

$$1 = i_0 \prec i_1 \prec \ldots i_m = n + 1$$

of the maximal chain

$$1 = j_0 < j_1 < \ldots j_l = n + 1$$

in the dominance tree. Take $i_0 = j_0 = 1$, and assume that i_k has been found, then search upward in the maximal chain, and choose i_{k+1} as the first point with $i_k \prec i_{k+1}$ and the added property that all flowlines entering (i_k, i_{k+1}) do so in coming from (i_{k-1}, i_k). Stopping on reaching $n + 1$, the resulting subchain provides the required series of sequential subflowcharts, in setting $f_k = (i_{k-1}, i_k)$.

Algorithm 3

Again the dominance tree and Propositions 2 and 3 are used, in order to search for the maximal subflowcharts f_j to appear in the decomposition of Theorem 2. According to Proposition 2, only the intervals in the dominance tree need to be considered. Moreover, intervals on the path from 1 to $n + 1$ can be ignored, since we are now dealing with a sequentially irreducible flowchart. From the remaining list of intervals, delete those that fail the condition of Proposition 3. Of those that remain, subflowcharts that are properly contained in others are deleted; finally the maximal subflowcharts remain. In collapsing these to elementary flowchart boxes, we obtain the schema S in the representation of Theorem 2. An alternate procedure for finding the maximal subflowcharts would make use of the fact that the elementary subflowcharts are partitioned by the maximal ones, with the result that every elementary subflowchart is contained in a unique maximal subflowchart.

In order to obtain the full nested sequential-maximal decomposition of a given flowchart, as described in the corollary, Algorithms 2 and 3 must be called recursively, until all subflowcharts are irreducible, both sequentially and maximally. These two conditions of irreducibility are sufficient for the termination of the recursion. By way of illustration, let us return to the sample flowchart of Fig. 1. Recall that we have already demonstrated its sequential decomposition, in connection with Fig. 4. In addition, we have discussed the maximal decomposition of one of the three subflowcharts, as summarised in Fig. 5. The recursive application of Algorithms 2 and 3 will finally yield the structured schema

```
▽
1: A(2, 3)
2: a(4)
3: b(4);
1: ▽
   1:c(2);
   1: d(2)
  △ (2)
2: B(3, 4)
3: ▽
   1: C(3, 2)
   2: D(3, 4)
   3: 1: E(2, 3)
      1: e(1) (5)
   4: ▽
      1:f(2);
      1: g(2)
     △ (5);
   1: h(2)
  △ (1);
1: i(2)
△
```

Note (parenthetically) that the pairs ▽, △ act as **begin–end** brackets, in the sense of an ALGOL or Pascal program. As a matter of fact, that is how these will be translated in the discussions of the next section.

5. Applications to the structuring problem

Most flowchart restructuring algorithms will include some form of preprocessing. Thereafter, the restructuring will generally proceed as a kind of induction on the size of the flowchart. Here, we are recommending a more thorough preprocessing phase, to ensure that any inherent structure in the original flowchart will be recognised by the processes that follow. It is our contention that this structure was quite possibly of the utmost significance to the design and intent of the algorithm, even though the original flowchart was not fully structured in any strict sense.

Almost certainly, any subflowcharts that appear in an algorithm's formulation will have the meaning of some independent processes, perhaps central to our understanding of the algorithm as a whole. As the preprocessing phase, it is suggested that the complete recursive sequential-maximal flowchart decomposition be developed first, as described in the corollary. Thereafter, the actual restructuring can begin on the nested subflowcharts of the decomposition. In whatever way these are restructured, their integrity as subflowcharts will remain intact, all to the advantage of a better understanding of the restructured version of the algorithm.

An additional advantage in this approach should not be overlooked. Most existing restructuring algorithms are quite limited as to the size of flowchart that can be treated. This is perfectly understandable, owing to the inherent complexity of such algorithms. But if the complete flowchart decomposition is first obtained, and only then are any of the structurisers

applied to the embedded subflowcharts in turn, more flowcharts must be considered; however, these are smaller in size. It is expected that this feature will permit the consideration of input flowcharts of a size that might otherwise be prohibitive.

By way of illustration, consider the sample flowchart of Fig. 1 once again. Originally, there was a flowchart of 14 labelled statements. In applying the complete decomposition algorithm, we are left to consider (sub)flowcharts of size at most four, as is seen by reviewing the structured schema concluding the last section. It could be argued that our example has been contrived, in order to provide a dramatic illustration of this point. But a more realistic appraisal would suggest that such reductions are not atypical. If the algorithm was designed in a rational manner, with today's recognition of the importance of structure, we would have to expect a fair density of subflowcharts, and a corresponding reduction in maximum flowchart size, after the decomposition routine had been performed.

Suppose now, that we are operating in the context of a set $\sum$ of *standard schemata*, those at our disposal for restructuring purposes. Typically, we might have

$\sum$ = { **while** *P* **do** . . . , **if** *P* **then** . . . **else** . . . , . . .}

in addition to the sequence construct (**begin** . . . ; . . . ; ; . . . **end**), itself not considered as a member of the set $\sum$, but nevertheless always available. Discounting the notation used here, the members of $\sum$ are thought to be schemata, in the sense already defined. One is thus able to attempt a match between the members of $\sum$ and the various schemata appearing in our decomposition. In fact, in the presence of a given set $\sum$, this search for matchings might be considered as a fourth algorithm in our overall process, to follow the three that have already been discussed. If this algorithm were applied to the structured schema concluding the last section, for the case of the typical set $\sum$ just mentioned, the following would be obtained

```
begin
if A then a else b;
1: begin c; d end (2)
2: B(3, 4)
3: begin
   1: C(3, 2)
   2: D(3, 4)
   3: while E do e (5)
   4: begin f; g end (5);
   h
   end (1);
i
end
```

indicative of the fact that some of the subflowchart schema we had previously obtained were structured (in the sense of $\sum$), while others were not.

Assuming that one has a structuriser, where the two remaining unstructured schemata are restructured as indicated in **Fig. 6**, our final version of the original example of Fig. 1 would appear as

```
begin
if A then a else b;
c; d;
while B do
 begin
 if C then
  while E do e
 else
  if D then
   while E do e
  else
   begin f; g end;
 h; c; d
 end;
i
end
```

after the removal of superfluous **begin–end** pairs. Most importantly, however, it should be noted that any restructuring algorithm will operate only on the two small flowchart schemata represented in Fig. 6. All of the preliminary transformation would have been handled by the algorithms that we have discussed. The result, in any case, is a fully structured flowchart equivalent to the one originally presented, wherein the topology of the original flowchart has been preserved.

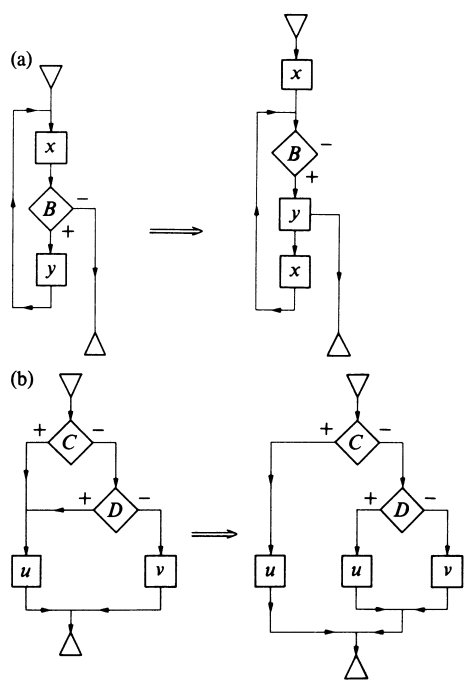

Fig. 6 The two remaining unstructured schemata after restructuring

References

ASHCROFT, E. and MANNA, Z. (1971). The translation of 'GOTO' programs to 'WHILE' programs, in *Proceedings of IFIP Congress* 71 pp. 250–255. North-Holland, Amsterdam.

BOHM, C. and JACOPINI, G. (1966). Flow diagrams, turing machines and languages with only two transformation rules, *Communications of the ACM*, Vol. 9 No. 5, pp. 366–371.

TARJAN, R. (1972). Depth-first search and linear graph algorithms, *SIAM Jour. Comp.*, Vol. 1 No. 2, pp. 146–160.

URSCHLER, G. (1975). Automatic structuring of programs, *IBM Journal of Research and Development*, Vol. 19 No. 3, pp. 181–194.

WILLIAMS, M. (1974). Generating structured flow diagrams: The nature of unstructuredness, *The Computer Journal*, Vol. 20 No. 1, pp. 45–50.

WULF, W. (1972). Programming without the GOTO, in *Proceedings of IFIP Congress* 71, pp. 408–413. North-Holland, Amsterdam.