Program Structures: Some New Characterisations*

N. E. Fenton

Centre for Software Reliability, City University,
Northampton Square, London EC1V 0HB, United Kingdom

AND

R. W. Whitty

Centre for Systems and Software Engineering, Department of Computing and Mathematics,
South Bank Polytechnic, Borough Road, London SE1 0AA, United Kingdom

Received March 30, 1987; revised March 8, 1991

Irreducible program flowgraphs are important in the study of program structuredness. In this paper a partial order is defined on the set of irreducible flowgraphs. This allows us to characterise those programs which can be re-structured by node splitting, where structuredness is defined in terms of any "subgraph-closed" set of irreducible flowgraphs. This includes all sets of structural components available in modern programming languages such as Ada, Modula-2, and C. In particular, we apply our results to the study of multiple-exit and multi-level exit control structures, giving a new characterisation of the EJ, charts of Kosaraju and a sharpening of two results of Peterson, Kasami, and Tokura. © 1991 Academic Press, Inc.

1. INTRODUCTION

One of the main aims of the mathematical study of structured programming is to discover how difficult it is to turn unstructured programs into structured programs. We want our programs to be built out of a well-defined set of allowable programming structures. When a program fails to satisfy this structuredness requirement we want to transform it into a functionally equivalent program which is structured. Mathematical structured programming studies how much the program must change in order to become structured. This is the same as asking what is the expressive power of our set of allowable programming structures. In more abstract terms, given a class S of control structures, mathematical structured programming seeks to characterise those program P whose control flow is structured in terms of S, up to preservation of the instructions and tests used by P. In other words, we want to decide when there is a program which is S-structured and which computes the same function as P using the same instructions and tests.

* Work supported in part by the U.K. government under grant ALV/PRJ/SE069.
One approach to this problem (e.g. [12, 14]) is to derive such characterisations directly. This can be done by explicitly constructing a certain program $P$, for which every $S$-structured equivalent uses a different set of instructions and tests from $P$. Or it may be possible to demonstrate that no such program can exist. Unfortunately, this approach seems to rely on very complex and ad hoc proof techniques, even when the strong connections this subject has with formal language theory are exploited (as is the case in [15]). An alternative approach is to study purely structural operations upon program control flow, such as node splitting. In some cases these may be sufficiently powerful to transform any program to an $S$-structured, functionally equivalent program, with the same instructions and tests, whenever some such equivalent program exists. The relationship of these operations to $S$-structuredness can then be studied in isolation, for example using graph theory.

In [5], flowgraphs were used to model program control flow and an "unfolding" operation was defined, by which a large class of flowgraphs (the "folded $S$-graphs") could be transformed to $S$-structured form under preservation of instructions and tests. For technical reasons, it proved most convenient to work with the sets $S = S_n$ of all "irreducible" flowgraphs with at most $n$ tests, with the set $S_1$ corresponding to the $D$-structures of the traditional structured programming literature. One of the main results in [5] characterised the folded $S_n$-graphs in term of forbidden subgraphs; the chief purpose of the present paper is to generalise this characterisation to apply to any set of commonly occurring program structures, for instance, those in Ada, Modula-2, and C.

Of course, characterising $S$-structuredness in this way is ultimately of theoretical interest only in so far as the unfolding operation, or any similar structural operation, can be guaranteed to produce $S$-structuredness, under functional equivalence and preservation of instructions and tests, whenever it exists. Although some notable progress in this area has been made recently [4, 7] such guarantees seem to be very difficult to achieve and we do not address this problem in the present paper. However, unfolding, which is a special type of node splitting, seems to be sufficiently general for most practically occurring types of structuredness.

In the next section we review some definitions and notation, including the role played in our theory by irreducible flowgraphs and by the fundamental operations of unfolding and node splitting. A key step towards obtaining our characterisation theorems is to define a partial order on the irreducible flowgraphs and prove that this partial order possesses a least element. This property is used in Section 3, where the main result of the paper generalises the characterisation of folded $S$-graphs from $S_n$ to arbitrary sets $S$ which are "subgraph-closed," a condition which is natural from a programming point of view. In Section 4, we apply the characterisation theorem to study programs structured in terms of loops with single-level and multi-level exists. We give a new characterisation of the $BJ_n$-charts of Kosaraju [14] and compare it to the characterisation of Kohoutková-Nováková [13], and we give slight sharpenings of two well-known results of Peterson et al. [18]. Finally in Section 5 we summarise the main issues in the paper and suggest what we feel to be the most profitable directions for further research in this area.
Proofs have been omitted from the paper except where they are very short and help to illustrate the methods we are using. A longer version of the paper [6], containing proofs of all results, was a formally reviewed deliverable of the UK Alvey project “Structure-Based Software Metrics.” It was also reviewed by independent referees of *Journal of Computer and System Sciences* and is available upon request.

2. **Mathematical Background**

In order to keep the present paper as self-contained as possible, we review the basic definitions from [5] in this section, as well as introduce some new concepts.

We require some basic terminology from graph theory. All graphs G considered are finite and directed. An edge from vertex x to vertex y is denoted by the ordered pair \((x, y)\). *Multiple edges* between x and y are allowed, in which case \((x, y)\) denotes an arbitrarily chosen representative of the set of these edges and the cardinality of this set is called the *multiplicity* of \((x, y)\). *Self-loops* of the form \((x, x)\) are also allowed.

For any vertex x of G, we use \(d_G^-(x)\) to denote the *indegree* of x in G and \(d_G^+(x)\) to denote the *outdegree* of x in G.

A *walk* in G is a sequence of edges

\[(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n)\]

of G. The sequence is called a *cycle* if \(x_0 = x_n\), and no other vertices are repeated.

2.1. **Theory of Flowgraphs**

**Definition 2.1.** A flowgraph \(F\) is a triple \((G, a, z)\) consisting of a graph G, together with distinguished vertices a, the *start vertex* and z, the *stop vertex* of G having the property that for every vertex v of G there is some walk from a to z which passes v. Further, the stop vertex has outdegree zero: \(d_G^+(z) = 0\).

Note that, except for the stop vertex, vertices of a flowgraph may have arbitrary outdegree, in order to take account of *Case* statements, computed *Goto* statements etc.

We refer to G as the *graph* of F. The vertices of G of outdegree 1 are called *process* vertices; the remaining vertices of G (excluding the stop vertex z) are called *control* vertices.

We denote by \(\mathcal{F}\), the family of all flowgraphs. From time to time we want to focus attention on the more traditional family of flowgraphs, in which all control vertices have outdegree precisely 2. Thus we use \(\mathcal{F}_2\) to denote the restriction of \(\mathcal{F}\) to this family.

Flowgraphs having no process vertices are called *DD-graphs* [17] (the DD denoting “decision to decision”). It is useful to associate with any flowgraph \(F = (G, a, z)\) a unique DD-graph \(DD(F) = (DD(G), DD(a), z)\), obtained by
collapsing every process vertex onto its unique successor vertex. Formally, $DD(a)$ is the unique nonprocess vertex nearest to $a$ (possibly $DD(a) = z$), and $DD(G)$ is obtained according to the following definition:

**Definition 2.2.** Let $G$ be any graph with vertex set $V$. Define $DD(G)$ to be the graph with vertex set

$$V' = \{ x \mid x \in V \text{ and } d_G^+(x) \neq 1 \},$$

and, for $x, y \in V'$, $(x, y)$ is an edge of $DD(G)$ with multiplicity $k$ if and only if $x$ has precisely $k$ distinct paths to $y$ in $G$ in which no vertices except $x$ and $y$ lie in $V'$.

The construction of Definition 2.2 plays an important role in later sections. An example is given in Fig 1.

Flowgraphs are analysed chiefly in terms of their subflowgraphs:

**Definition 2.3.** Let $F = (G, a, z)$ and $F' = (G', a', z')$ be flowgraphs, with $G$ and $G'$ having vertex sets $V$ and $V'$, respectively. $F'$ is called a subflowgraph of $F$ if $G'$ is a subgraph of $G$ and

1. either $a' = a$ or $d_{G'}^-(a') < d_G^-(a')$.
2. $d_{G'}^+(x) = d_G^+(x)$ for all $x \in V' - z'$.

In other words, if $V'' = V' - z'$ then $(V - V'', V'')$ contains an edge to $a'$, if $a' \neq a$, and all edges in $(V'', V - V'')$ go to $z'$.

A flowgraph having no nontrivial proper subflowgraphs is called an irreducible flowgraph or, more simply, an irreducible, where a flowgraph is called trivial if it has fewer than two edges (see Fig. 2).

The following characterisation of irreducible flowgraphs [6, 7] is frequently useful when analysing their properties:

![Fig. 1. Deriving the associated DD-graph.](image-url)
LEMMA 2.4. Let $F = (G, a, z)$ be a flowgraph. Then $F$ is an irreducible if and only if

1. $F_0$ is not a proper subflowgraph of $F$ (see Fig. 3).
2. For every control vertex $x$ of $G$, $x$ has two paths to $z$ which have no common vertices except $x$ and $z$.
3. Every control vertex of $G$ has a path to every other vertex of $G$.

Finally we introduce some sets of irreducibles which are prominent in later sections. The set of irreducibles having at most $n$ control vertices, for $n = 0, 1, \ldots$, is denoted by $S_n$. The set $S_0$ comprises precisely the trivial flowgraphs and the flowgraph $F_0$ (Figs. 2 and 3). In [5], attention was restricted to the family $\mathcal{F}_2$, and $S_1$ corresponded to the traditional $D$-structures. However, in the context of $\mathcal{F}$, this correspondence no longer holds, since $S_1$ contains flowgraphs, such as that shown in Fig. 4, which characteristically have no matching control statements in structured programming languages like Pascal.

As in [5], we define $C_{(k)}$, $k = 1, 2, \ldots$, to be the graph consisting of $k + 1$ vertices $x_0, \ldots, x_{k-1}$ and $z$, and edges $(x_i, z)$ and $(x_i, x_{(i+1) \mod k})$, for $i = 0, 1, \ldots, k - 1$ (see Fig. 5, for example).
We define the set $BJ_n$, $n = 1, 2, \ldots$, to represent the control structures sequence, loops with up to $n$ exists, and the CASE statement:

$$BJ_n = \{ F = (G, a, z) \in \mathcal{F} \mid F \in S_0, \text{ or } DD(G) = C_k, 1 \leq k \leq n, \text{ or } DD(G) \text{ consists of an edge } (a, z) \text{ of arbitrary multiplicity and no other edges} \}.$$ 

By convention we set $BJ_0 = S_0$. It is apparent that the $D$-structures (with CASE) are precisely the members of $BJ_1$. The sets $BJ_n$ were identified by Kosaraju [14] because of their theoretical interest. The set

$$BJ_\infty = \bigcup_{k=1}^{\infty} BJ_k,$$

representing the multiple-exit loop structures, is more natural in programming terms.

2.2 A Partial Ordering of the Irreducibles

In this section we define a partial order on the set of irreducibles (and incidentally on the restriction of this set to $\mathcal{F}_2$). Our main result is that any two irreducibles have an upper and lower bound under this partial order. In so far as the irreducibles may be regarded as the fundamental components of structured programming, this structure theorem is of interest in its own right, although it
is only the existence of the lower bound which is required in Section 3, when we characterise the folded S-graphs.

Let $G$, $G'$ be any graphs. We say that $G'$ is generated from $G$ if $G'$ can be derived from $G$ by a single application of one of the following operations:

1. (g1): subdivide an edge of $G$ by inserting a new vertex.
2. (g2): choose a vertex $x$ of $G$ and add a new edge from $x$ to itself or to any other vertex of $G$.
3. (g3): subdivide an edge of $G$ by inserting a new vertex $x$ and add a new edge from $x$ to itself or to any other vertex of $G$.

We refer to the new vertex or edge as being generated by (g1), (g2), or (g3). We define a partial order $\leq$ on the set of all irreducibles by:

$$F = (G, a, z) \leq F' = (G', a', z') \text{ if and only if } F = F' \text{ or there exists a sequence } G = G^{(0)}, G^{(1)}, \ldots, G^{(k)} = G' \text{ of graphs of irreducibles, with } k \geq 1,$$

such that $G^{(i+1)}$ is generated from $G^{(i)}$, for $0 \leq i \leq k - 1$.

A natural question to ask about any partially ordered set is whether each pair of elements of the set possesses a lower and upper bound with respect to the partial order. Consider first, the restriction of $\leq$ to the DD-irreducibles of outdegree at most 2, i.e., the DD-irreducibles in $\mathcal{F}_2$. The fact that any two elements of this restriction have a lower bound, follows from a result proved in [11]:

**Theorem 2.5.** Let $G$ be the graph of a DD-irreducible in $\mathcal{F}_2$, having $n \geq 3$ vertices. Then $G$ is generated from the graph of a DD-irreducible with $n - 1$ vertices by operation (g3).

To prove the existence of lower bounds in the general partial order $\leq$ over $\mathcal{F}$, we extend Theorem 2.5 to irreducibles of arbitrary outdegree:

**Proposition 2.6 [6].** Let $G$ be the graph of an irreducible with at least two edges. Then $G$ is generated from the graph of some irreducible by one of (g1)–(g3).

From this, we prove in [6]:

**Theorem 2.7.** For any two irreducibles $F_1 = (G_1, a_1, z_1)$ and $F_2 = (G_2, a_2, z_2)$, there exist irreducibles $F_{12}^{\text{lower}}$ and $F_{12}^{\text{upper}}$ such that

$$F_{12}^{\text{lower}} \leq F_1, \quad F_2 \leq F_{12}^{\text{upper}}.$$

2.3. Node Splitting and Unfolding

Suppose we have a set of flowgraphs $S \subseteq \mathcal{F}$. The members of $S$ are called the basic S-graphs. The class of S-graphs (the S-structured flowgraphs, representing S-structured programs) is the smallest class of flowgraphs closed under the usual nesting operation, called composition [5]; see Fig. 6. In some cases, a program...
which is not $S$-structured may be transformed into an $S$-structured program, by duplicating some parts of its code. Traditionally, the corresponding flowgraph operation used to transform flowgraphs into $S$-graphs, has been node splitting. Node splitting has gained wide acceptance as a means of increasing structuredness, particularly in the context of the $D$-structures [2, 14, 18], and has been studied for more general sets in an influential paper of Peterson et al. [18] and, more recently, by Fučík and Král [7]. In [5, 8] a very restricted form of node splitting, called unfolding, was studied. Hence the folded $S$-graphs are those which can be transformed into $S$-graphs by a finite sequence of applications of this unfolding operation.

Corresponding to the operations of node splitting and unfolding, we may identify two classes of flowgraphs: those which are $S$-structured up to node splitting (i.e., may be transformed into $S$-graphs by a finite sequence of applications of the node-splitting operation) and those which are $S$-structured up to unfolding (i.e., the folded $S$-graphs). These classes are of fundamental importance and we have provided mathematically precise definitions in [6]. We offer a less formal treatment here since the definitions are quite complex; unfortunately, this complexity seems to be unavoidable, if the definitions are to be mathematically precise.

Figure 7 illustrates node splitting applied to a subgraph $H$ in an arbitrary graph.
G: each entry edge of \( H \) is expanded to a copy of the subgraph of \( H \) reachable from that edge.

When we apply node splitting to flowgraphs, we must be careful to take into account the fact that the start vertex \( a \) is a legitimate entry point of a subgraph.

Unfolding acts in the same way as unfolding, but is only applied in the special case where the subgraph is a subflowgraph; see Fig. 8.

It is clear that node splitting is a far more general operation than unfolding, since it applies to arbitrary subgraphs. This makes it very difficult to work with, except in the restricted case of the set \( BJ_1 \) (the \( D \)-structures). In particular, we cannot take an arbitrary flowgraph and repeatedly apply node splitting to test whether it is \( S \)-structured up to node splitting. In the first place, the process may not terminate; in the second place, if two different sequences of node splittings are applied to a flowgraph, then one may lead to an \( S \)-graph while the other may not. Repeated unfolding, on the other hand, must always terminate, because it will eventually reduce any flowgraph to one in which all subflowgraphs have a single entry vertex; moreover, if any two sequences of unfoldings are applied to a flowgraph then either both will lead to an \( S \)-graph, or neither will. More formally, we say that unfolding over \( \mathcal{F} \) (or \( \mathcal{F}_2 \)) satisfies the Finite Church Rosser property under equivalence of \( S \)-graphs [5, Theorem 5.7].

Unfolding clearly enjoys certain advantages over node splitting as a way of achieving \( S \)-structuredness. However, there are flowgraphs which are \( S \)-structured up to node splitting, but which fail to be \( S \)-structured up to unfolding (i.e., which fail to be folded \( S \)-graphs); see [6] for example. The fact that the set of flowgraphs which are \( S \)-structured up to node splitting may properly contain the set of folded \( S \)-graphs, might seem to be an argument for retaining the operation of node splitting. However, for those instances of \( S \) that are of practical or theoretical interest, these two sets turn out to be identical. Specifically, in [6] we proved:

![Diagram](image.png)

**Fig. 8.** Unfolding the subflowgraph \( F \)' at entry points \( a' \) and \( a'' \).
PROPOSITION 2.8. Let $S$ be one of the classes $S_n$ or $BJ_n$, $n = 0, 1, \ldots$, or $BJ_\infty$. Let $F = (G, a, z)$ be a flowgraph which is $S$-structured up to node splitting. Then $F$ is a folded $S$-graph.

3. THE CHARACTERISATION THEOREMS

One of the main results of [5] characterised the folded $S_n$-graphs over $\mathcal{F}_2$:

THEOREM 3.1 [5, Theorem 5.12]. Let $F = (G, a, z)$ be a flowgraph. Let $S_n^2$ denote the restriction of $S_n$ to $\mathcal{F}_2$ and let $I_m^2$ be the set of all graphs of DD-irreducibles in $\mathcal{F}_2$ with precisely $m$ control vertices. The following are equivalent:

1. $F$ is a folded $S_n^2$-graph.
2. $G$ has no subgraph $H$ containing $z$, such that $DD(H) \in I_m^2$.

Theorem 3.1 characterises the folded $S_n^2$-graphs in terms of forbidden subgraphs, where these subgraphs are precisely the graphs of the smallest DD-irreducibles not in $S_n^2$, i.e., those with $n + 1$ control vertices, excluding the stop vertex. The main result of this section generalises Theorem 3.1 by replacing "smallest with respect to number of control vertices" with "smallest with respect to $\leq". This allows us to characterise folded $S$-graphs for any set $S$ of irreducibles, subject to one restriction which can be motivated fairly intuitively in programming terms.

We begin our process of generalisation with a definition from [5]:

DEFINITION 3.2 [5]. Let $S$ be a set of irreducibles. We say that $S$ is complete if $S$ satisfies:

1. $F_0 \in S$ (see Fig. 3).
2. for any irreducible $F$, if $DD(F) \in S$, then $F \in S$ also.

By concentrating on complete sets $S$ of irreducibles, we have the following very general sufficient condition for a flowgraph to be a folded $S$-graph.

THEOREM 3.3. Let $S$ be any complete set of irreducibles. Let $F = (G, a, z)$ be any flowgraph and suppose that $F$ is not a folded $S$-graph. Then $G$ has a subgraph $H$ containing $z$, such that $DD(H)$ is the graph of a DD-irreducible not lying in $S$.

Theorem 3.3 proves useful for showing that all members of some given class of flowgraphs are folded $S$-graphs: it is sufficient to prove that no member of the class can "contain" a forbidden flowgraph, i.e., one not lying in $S$. The theorem does not, however, offer an effective means of checking that a specific flowgraph is a folded $S$-graph, since there will be infinitely many such forbidden subgraphs to check. The key role of the lower bound result of Theorem 2.7 for $\leq$ is in overcoming this difficulty, as shown in the next two results:
Lemma 3.4 [6]. Let \( F = (G, a, z) \) and \( F' = (G', a', z') \) be irreducibles, such that \( F \leq F' \). Then \( G' \) has a subgraph \( H \) containing \( z \), such that \( DD(H) \) is isomorphic to \( DD(G) \).

Proposition 3.5. Let \( S \) be any complete set of irreducibles containing the least element of \( \leq \) (the 2-vertex trivial flowgraph of Fig. 2). Let \( \tilde{S} \) consist of the graphs of the smallest (with respect to \( \leq \)) \( DD \)-irreducibles not in \( S \). Let \( F = (G, a, z) \) be any flowgraph, such that \( F \) is not a folded \( S \)-graph. Then some subgraph \( H \) of \( G \) containing \( z \) satisfies \( DD(H) \in \tilde{S} \).

Proof. By Theorem 3.3, there exists a \( DD \)-irreducible \( F' = (G', a', z') \) not lying in \( S \), such that \( G' = DD(H') \) for some subgraph \( H' \) of \( G \) containing \( z \). Now the least element of \( \leq \) is in \( S \), so Theorem 2.7 applies, whence \( F'' \leq F' \) for some \( F'' = (G'', a'', z'') \), with \( G'' \in \tilde{S} \). Now, by Lemma 3.4, \( H = G'' \) is the required subgraph of \( G \).

Proposition 3.5 gives a sufficient condition for a specific flowgraph \( F \) to be a folded \( S \)-graph, for any complete set \( S \) containing the least element of \( \leq \): we construct the (finite) set \( \tilde{S} \) and check that no members of \( \tilde{S} \) are contained in \( F \). We would like this to be a necessary condition as well. However, merely requiring that \( S \) contain the least element of \( \leq \) is not a strong enough restriction on \( S \) to ensure this—\( F \) may contain a subgraph \( H \), satisfying the conclusion of Proposition 3.5, and yet be a folded \( S \)-graph. We must choose \( S \) so that such “harmless” subgraphs do not appear in the set \( \tilde{S} \). This motivates the restriction which we place on our sets \( S \) of irreducibles:

Definition 3.6. Let \( S \) be a set of irreducibles. We say that \( S \) is subgraph-closed if, whenever \( F_1 = (G_1, a_1, z_1) \in S \), and \( F_2 = (G_2, a_2, z_2) \) is an irreducible, such that \( G_2 = DD(H_1) \) for some subgraph \( H_1 \) of \( G_1 \) containing \( z_1 \), then \( F_2 \in S \) also. As in Proposition 3.5, we denote by \( \tilde{S} \) the set consisting of the graphs of the least \( DD \)-irreducibles not in \( S \).

Requiring a set \( S \) of irreducibles to be subgraph-closed can be seen as a quite natural restriction to impose. Essentially, it says that if we have chosen some basic control structures from which to build our programs, then it makes no sense to disallow any substructures of these control structures. In any case, such substructures could always be manufactured artificially by a programmer, by making a suitable interpretation of the control and process vertices in the enclosing control structure.

Although fairly simple and natural, subgraph-closure is sufficiently powerful to provide us with the converse to Proposition 3.5. Moreover, it subsumes the completeness property so that we can still invoke the previous results in this section:

Lemma 3.7. Any subgraph-closed set of irreducibles is complete.
Proof. Let $S$ be a subgraph-closed set of irreducibles. Since $DD(F_0)$ consists of the single control vertex which is the stop vertex of $F_0$, $DD(F_0)$ lies in any member of $S$. Therefore, since $S$ is subgraph-closed, $F_0 \in S$. To see that property of Definition 3.2(2) holds for $S$, let $F = (G, a, z)$ be any irreducible such that $DD(F) \in S$. Then $F \in S$ by setting $F_1 = DD(F)$ and $H_1 = G_1$ in Definition 3.6. □

We are now ready to give a necessary and sufficient condition for a flowgraph to be a folded $S$-graph, thus generalising Theorem 3.1.

**Theorem 3.8.** Let $S$ be any subgraph-closed set of irreducibles and let $\mathcal{S}$ be as in Definition 3.6. Let $F = (G, a, z)$ be any flowgraph. The following are equivalent:

1. $F$ is a folded $S$-graph.
2. $G$ has no subgraph $H$ containing $z$, such that $DD(H) \in \mathcal{S}$.

We have already established (Proposition 3.5) that $\Rightarrow (1)$ in Theorem 3.8. The proof of the other implication, $(1) \Rightarrow (2)$, more or less mirrors the proof in [5] of Theorem 3.1, and is given in [6].

**Remark 3.9.** 1. By virtue of Theorem 2.5, we may restrict both $S$ and $\mathcal{S}$ in Theorem 3.8 to $\mathcal{F}_2$.

2. By applying Theorem 3.8 to the set $S = S_n$, $n = 0, 1, \ldots$, we obtain Theorem 3.1, since we may restrict attention to $\mathcal{F}_2$ by Remark 3.9(1).

4. **Applications to Multiple Exit and Multi-level Exit Loop Structures**

4.1. **Multiple Exit Loops**

In this section we are concerned with sets of irreducibles which arise naturally in programming practice, namely those which represent structures having escape exits, as provided by, for example, the Ada EXIT WHEN statement. We begin with the best-known example, that of the $BJ_n$-graphs (traditionally called $BJ_n$-charts). The $BJ_n$-graphs have been characterised by Kohoutková-Nováková [13] as follows:

**Theorem 4.1** (Kohoutková-Nováková). Let $F = (G, a, z)$ be a flowgraph. Then $F$ is a $BJ_n$-graph if and only if

1. for any vertex $x$ of $G$, if there are two paths from $a$ to $x$, ending in distinct edges $(y, x)$ and $(y', x)$, $y \neq y'$, then the immediate postdominator of the immediate predominator of $x$ is $x$ itself.

2. for any strongly connected subgraph $H$ of $G$, with distinct vertices $y_1, \ldots, y_k$, of $H$, having paths to $z$ passing no other vertices of $H$, we have

   (a) $k \leq n$.  


(b) \( y_i \) has edges to at most two distinct vertices, \( i = 1, \ldots, k \)

(c) For some vertex \( x \) of \( G \), there exist edges \( (y_i, x), i = 1, \ldots, k \).

(In [13], a further condition, viz

(d) there exists a path from \( a \) to some \( y_i \) passing no other vertices of \( H \),

was imposed. We do not require this condition, since we allow a process vertex to be an entry vertex of a subflowgraph, as is the case for a \texttt{REPEAT UNTIL} loop, for example).

In Theorem 4.1, condition 1 ensures that all subflowgraphs have a single-entry vertex; condition 2, together with 1, ensures that all subflowgraphs are \( BJ_n \)-graphs. Now Theorem 3.8 applies to the set \( BJ_n \) and we use this fact to derive an alternative characterisation of the \( BJ_n \)-graphs, in terms of forbidden subgraphs. To construct the set \( \tilde{BJ}_n \) we take every graph, which may be generated by a single application of operations (g1)-(g3) to the graphs in \( BJ_n \), and which are the graphs of \( DD \)-irreducibles not in \( BJ_n \); deleting any graph greater (with respect to \( \leq \)) than some other graph in this set, yields the set \( \tilde{BJ}_n \). The resulting set is shown in Fig. 9.

Now, from Theorem 3.8, we have:

**Corollary 4.2.** A flowgraph \( F = (G, a, z) \) is a \( BJ_n \)-graph, if and only if

1. every subflowgraph of \( F \) has a single-entry vertex.
2. \( G \) has no subgraph \( H \) containing \( z \), such that \( DD(H) \in \tilde{BJ}_n \).

Corollary 4.2(2) characterises the folded \( BJ_n \)-graphs, and also, by Proposition 2.8, the flowgraphs \( BJ_n \)-structured up to node splitting. Although Corollary 4.2(1) and (2) are not precisely analogous to (1) and (2) in Theorem 4.1, we may note the following:

1. \( DD(H) = C_{(n+1)} \) in Corollary 4.2 corresponds to failure of a strongly connected subgraph to satisfy Theorem 4.1(2a).

\[ \begin{align*}
\{ C_{(n+1)} \} & \quad \text{(i)} \quad \text{(ii)} \\
\text{(iii)} & \quad \text{(iv)} \quad \text{(v)}
\end{align*} \]

*Fig. 9.* The set \( BJ_n \) for \( n \geq 1 \). Graphs (iv) and (v) are present only for \( n \geq 2 \).
2. \( DD(H) = \) one of (i)-(iii) of Fig. 9 corresponds to failure to satisfy Theorem 4.1(2b).

3. \( DD(H) = \) one of (iv), (v) of Fig. 9 corresponds to failure to satisfy Theorem 4.1(2c).

We observe that it would probably be possible to derive Theorem 4.1 as a direct corollary of Theorem 3.8, although it is not clear that anything would be gained by doing so.

Obviously, Corollary 4.2 precludes the possibility of every flowgraph being a folded \( BJ_n \)-graph. This remains true even if any number of loop exits are allowed, i.e., we consider \( BJ_\infty \)-graphs. This was first proved by Peterson, Kasami, and Tokura [18, Theorem 3]:

**Theorem 4.3 (Peterson et al.).** Not every flowgraph is \( BJ_\infty \)-structured up to node splitting.

To prove Theorem 4.3, Peterson et al. exhibited a counter-example which had four control vertices. Now suppose we consider only flowgraphs in \( F_2 \). By definition of \( BJ_\infty \) and Remark 3.9(1), the forbidden subgraphs which remain in Corollary 4.2 are the graphs (iv) and (v) of Fig. 9. Flowgraphs whose graphs are either of these cannot be folded \( BJ_\infty \)-graphs. Thus, appealing to Proposition 2.8, these graphs, which have three control vertices provide sharper counter-examples for Theorem 4.3.

### 4.2. Multi-level Exit Loops

We now turn to the question of multi-level exits from loops. Consider the case of a language, such as Ada, which offers a multi-level exit mechanism. Any loop may contain exit statements which conditionally transfer control to a fixed statement outside the loop. These exit statements may be either in the loop itself, or in some loop nested within it. Similarly, conditional structures (IF THEN ELSE and CASE) may contain exits which conditionally transfer control out of the structure, either directly, or from some nested substructure. In general we refer to the above conditional exit mechanisms as multi-level exits (see [1] for a good overview of this area).

Multi-level exits may be modelled in flowgraph terms as follows: define the set \( ML_\infty \) to consist of all irreducibles which can be obtained by zero or more applications of the following operation, defined on folded \( BJ_1 \)-graphs \( F = (G, a, z) \):

if \( F' = (G', a', z') \) is a subflowgraph of \( F \) (possibly with \( F' = F \)), then apply (g3) (Section 2.2) to \( G' \), directing the new edge to \( z' \).

Peterson et al. proved the following result [15, Theorem 4].

**Theorem 4.4 (Peterson et al.).** Every flowgraph in \( F_2 \) is \( ML_\infty \)-structured up to node splitting.
Using Theorem 3.8, we proved a sharper result in [6]:

**Corollary 4.5.** Every flowgraph in $\mathcal{F}_2$ is a folded $ML_\infty$-graph.

As a last example to demonstrate the flexibility and simplicity of Theorem 3.8, in [6] a characterization is given of another important family of structured flowgraphs. Specifically, let the set $L$ be defined by

$$L = BJ_2 \cup \{ F = (G, a, z) \mid DD(G) = L_k, k = 3, 4, \ldots \},$$

where $L_k$ has $k + 1$ vertices $x_0, \ldots, x_{k-1}$ and $z$, and edges $(x_i, x_{i+1})$ and $(x_{i+1}, x_i)$, for $0 \leq i \leq k - 2$, and $(x_0, z)$ and $(x_{k-1}, z)$. In [6] it is shown that Proposition 2.8 can be extended to the class $L$. Thus following the method of Theorem 3.8, we find that $L = BJ_2 - \{ L_3 \}$, and we have:

**Corollary 4.6.** A flowgraph $F = (G, a, z)$ is $L$-structured up to node splitting if and only if $G$ has no subgraph $H$ containing $z$ such that $DD(H) \in BJ_2 - \{ L_3 \}$.

5. **Concluding Remarks**

Restricted GOTO mechanisms, as provided by the multi-level exit facility in Ada, are now widely accepted as a necessary evil in programming methodology. It has become desirable to study structuredness, not just in the traditional sense of the word, focussing on the $D$-structures, but in the context of any set $S$ of basic flowgraphs that may be chosen as the building blocks of structured programming. In [5] it was argued on technical and intuitive grounds that only complete sets $S$ of irreducibles need be studied. We have proposed a further natural restriction on $S$—that it be subgraph-closed—and have given a characterization of those flowgraphs which are structured, in terms of any subgraph-closed set $S$ of irreducibles, up to unfolding. The motivation for studying these folded $S$-graphs is that they form a large subclass of those flowgraphs which are $S$-structured up to preservation of instructions and tests. We have provided further motivation by showing that, in this respect, unfolding is just as powerful as the well-known operation of node splitting, for a variety of important instances of $S$. Moreover, unfolding is considerably more amenable to mathematical and algorithmic analysis than node splitting.

In the light of the methods we have discussed, the irreducibles assume a fundamental importance in the theory of structured programming. We have shown how earlier results on generating all graphs of $DD$-irreducibles may be extended to impose a partial order structure on the set of all irreducibles. The main property of this partial ordering is that every pair of elements has a lower and upper bound. This property is of interest for any partial order (for instance, it is necessarily satisfied by lattices), so the result would seem to be important in its own right. In
addition, we use the lower bound result in deriving our characterisation of the folded \( S \)-graphs.

It is worth noting that, once we require our sets \( S \) to be subgraph closed, the main technical reason for requiring \( S \) to contain only irreducibles disappears (see the remarks following Algorithm 1 in [5]). It would be interesting to see how far our results extend to arbitrary sets \( S \) that are subgraph-closed, although this has no intuitive justification in programming terms.

A more crucial line of further research is to study more closely the relationship between unfolding and general program transformations which preserve sets of instructions and tests. We might tentatively suggest that unfolding was as powerful as any such transformation, wherever it is as powerful as node splitting. Whether node splitting is, without restriction, a powerful as any such transformation, also seems to be an open question, which might be resolved with closer attention to the links between structured programming and finite automata theory.

The theory we have discussed in this paper has essentially been directed at software written in second and third generation languages (in particular, we have illustrated its applicability to sequential Ada and Pascal code). There is a pressing need for such theories to be extended to address fourth generation languages, specification languages and concurrent processes (as exemplified by Occam, of Ada tasking). A very promising framework within which to tackle this need in the latter instance, has been established by Ginzburg and Yoeli [9], who have shown how the basic concepts of generalised structured programming may be applied to parallel control flow.

REFERENCES

6. N. E. Fenton and R. W. Whitty, Characterisations of structured programs, in “Structure-Based Software Metrics, Collected Papers” (J. J. Elliott et al., Eds.), CSSE, South Bank Polytechnic, Borough Road, London.