Flowchart Schemata and the Problem of Nomenclature

M. H. Williams
Computer Science Department, Heriot-Watt University, 79 Grassmarket, Edinburgh EH1 2HJ, UK

The problem of nomenclature (different names being used to refer to the same object or idea, and the same name being used to refer to different objects and ideas) plagues many areas of Computer Science. The theory of flowchart schemata is no exception to this. This paper attempts to clarify some of the issues concerned with reducibility and structuredness in flowchart schemata.

1. INTRODUCTION

The theory of flowchart schemata\(^1,2\) (or program schemata\(^3\) or flow chart schemas\(^4\) or program schemas\(^5\)) is concerned with the study of flowcharts\(^1\) (or flow charts\(^6\) or charts\(^4\) or flow diagrams\(^7\)). A flowchart is a collection of interconnected nodes\(^5\) (or boxes\(^1,6\) or primitives\(^8\)). Some authors distinguish between four separate types of node:

(a) an operation node or process box\(^2,6\) or action\(^7\)
(b) a decision node\(^5\) or decision box\(^2,6\) or predicate\(^7\)
(c) an entry node\(^7\) or START box\(^6\) or START node\(^5\)
(d) an exit node\(^7\) or FINISH box\(^6\) or HALT node\(^5\)

Others use only the first two types but choose one particular node as the entry node and one as the exit node of the flowchart; others, again, made no distinction at all between different types of node.

This brief introductory paragraph is evidence of the way in which different names are used to refer to the same thing. Even worse is the problem of the same name being used by different authors to denote different objects or ideas.

Within the theory of flowchart schemata the term ‘reducibility’ is a particularly overworked term and is (in some cases) closely related to ‘collapsibility’. ‘Structuredness’ is another closely related concept. Since some confusion has arisen out of the differences in interpretation given by different authors, this paper attempts to clarify some of the issues surrounding these terms.

2. REDUCIBILITY AND COLLAPSIBILITY

2.1. Reducibility (1)

The concept of reducibility as defined by Kosaraju, based on work by Knuth and Floyd,\(^8\) is a form of equivalence relationship. One flowchart $G$ is said to be strongly reducible to another flowchart $H (G \leq_s H)$ if they are equivalent to within certain simple restructuring rules, such as node splitting. More formally, $G$ is strongly reducible to $H$ if and only if every node of $H$ is a node of $G$ and for every interpretation (execution) and every input the computational histories are identical for the two flowcharts. Similarly $G$ is weakly reducible to $H$ (written $G \leq_w H$) if and only if every node of $H$ is a node of $G$ and for every interpretation and every input the outputs are equal (or both do not terminate) for both flowcharts.

To illustrate this, consider the portions of flowcharts shown in Fig. 1. The portion in Fig. 1(b) is both strongly and weakly reducible to that in Fig. 1(a). The same is true of the portions shown in Fig. 1(d) and Fig. 1(c). On the other hand the portion in Fig. 1(f) is weakly reducible to that in Fig. 1(e) but not strongly reducible to it.

Similarly, one class of flowcharts $C_1$ is reducible (strongly or weakly) to another class $C_2$ if and only if every flowchart of $C_1$ is reducible to a flowchart of $C_2$ (written $C_1 \leq_s C_2$ or $C_1 \leq_w C_2$). If $C_1$ is reducible to $C_2$ but $C_2$ is not reducible to $C_1$, this is written as $C_1 <_s C_2$ or $C_1 <_w C_2$. If $C_1 \leq_w C_2$ and $C_2 \leq_w C_1$ then $C_1 \equiv_w C_2$.

Kosaraju then defined certain classes of flowcharts which he studied in detail. Four interesting classes are:

1. The class of D-charts which is defined as:
   (a) Any process box (action) is a D-chart.
   (b) If $G_1$ and $G_2$ are D-charts and $p$ is a predicate then $\text{SEQUENCE}(G_1, G_2)$, IF $p$ THEN $G_1$ ELSE $G_2$ and WHILE $q$ DO $G_1$ (where $q \in \{p, \neg p\}$) are D-charts.

In other words a D-chart is a flowchart composed only of simple sequences, simple (2-way) selection and while-loops.

2. The class of WR-charts (WR $\equiv$ While and Repeat) which is defined as:
   (a) Any process box is a WR-chart.
   (b) If $G_1$ and $G_2$ are WR-charts and $p$ is a predicate then $\text{SEQUENCE}(G_1, G_2)$, IF $p$ THEN $G_1$ ELSE $G_2$, WHILE $q$ DO $G_1$ and REPEAT $G_1$ UNTIL $q$ (where $q \in \{p, \neg p\}$) are WR-charts.

In other words a WR-chart is a flowchart composed only of simple sequences, simple selection, while-loops and repeat-loops.

3. The class of TD$_1$-charts (TD$_1$ $\equiv$ Top-Down) which is defined as:
   (a) Any process box is a TD$_1$-chart.
   (b) Any decision box is an IP$_1$-chart.
   (c) Any flowchart constructed from any number of TD$_1$-charts and at most one IP$_1$-chart is a TD$_1$-chart if it contains a single exit path and an IP$_1$-chart if it contains two exit paths. The class of TD$_1$-charts thus includes while-loops and $n + \frac{1}{2}$ loops.\(^9\)
(4) The class of BJ*-charts (BJ ≡ Böhm and Jacopini\(^{10}\)) which is defined as:
(a) Any process box is a BJ*-chart.
(b) If \(G_1\) and \(G_2\) are BJ*-charts and \(p\) is a predicate then \(\text{SEQUENCE}(G_1, G_2)\) and IF \(p\) THEN \(G_1\), ELSE \(G_2\) are BJ*-charts.
(c) For \(i \leq n\) if \(G_1, G_2, \ldots, G_i\) are BJ*-charts and \(p_1, p_2, \ldots, p_i\) are predicates then the \(i\)-exit loop shown in Fig. 2 is a BJ*-chart.

Thus, the class of BJ*-charts includes any loops of the form shown in Fig. 2 with up to \(i\) exits.

Kosaraju proved various properties of these classes such as:
(a) The class of BJ₀-charts is equivalent to the class of D-charts.
(b) Any flowchart is reducible to a D-chart provided that it does not contain a reachable loop with more than one exit.

and so on.

2.2. Reducibility (2)

Cowell, Gillies and Kapsi\(^{11}\) introduced a notion of reducibility which is based on flowgraphs rather than flowcharts. Following their definition a flowgraph is a graphical representation in which operation nodes are ignored. Each node in a flowgraph corresponds to either the exit node or a decision node in the original flowchart. Thus a flowgraph is a representation of the underlying structure of a flowchart. If one flowchart \(G\) is strongly reducible (by the definition in the previous section) to another flowchart \(H\) and they both have the same number of decision nodes then \(G\) and \(H\) both have the same flowgraph.

If the class of flowgraphs corresponding to some class of flowcharts \(X\) is written as \(\text{FGRAPH}(X)\) then from the definition of the previous section:

\[
\text{FGRAPH(D-charts)} \equiv \text{FGRAPH(WR-charts)} \\
\equiv \text{FGRAPH(TD₁-charts)}
\]

The only type of flowgraph which does not fall into this class is one which contains at least one loop with more than one exit.

![Figure 2. An \(i\)-exit loop.](https://academic.oup.com/comjnl/article-abstract/26/3/270/464433/fig2)
A flowgraph is said to be reducible if it has a proper subgraph (i.e. a set of nodes with one entry and one exit) and irreducible if it does not have such a subgraph. In this case reducibility is not concerned with the equivalence of two flowcharts but with whether or not the flowgraph corresponding to a given flowchart can be subdivided into smaller units.

If a flowgraph is irreducible and consists of more than one node then:
(a) It cannot be a member of FGRAPH(D-charts), FGRAPH(TD1-charts), etc.
(b) It must contain at least one multi-exit loop.

2.3. Reducibility (3)

Prather and Giulieri use the term reducible/irreducible in a similar way to Cowell, Gillies and Kaposi except that they apply it directly to the flowcharts rather than to the flowgraphs, viz. a flowchart is reducible if it has a proper subflowchart (set of nodes with one entry and one exit). In this case, if a flowchart is irreducible and consists of more than one node, it must contain at least one of the unstructured elements outlined in Section 3.

2.4. Reducibility (4)

Another interpretation of the terms reducible/irreducible is that given by Tarjan and subsequently referred to by various authors, including Aho, Hopcroft and Ullman and Hecht. Following their definitions, an elementary irreducible flowchart is a flowchart with more than one node in which no pair of nodes X, Y can be found such that the only paths leading to X arise from Y. The simplest example of an elementary irreducible flowchart is given in Fig. 3.

Figure 3. Simplest elementary irreducible flowchart (Tarjan).

In this connection the notion of an interval is important. Given a flowchart G and some node of the flowchart n, the interval with header n, I(n), is the set of nodes consisting of n and all other nodes j such that the immediate predecessors of j are in I(n) (i.e. all arcs entering node j arise from nodes in I(n)). The term reduction by intervals is used to refer to the process of partitioning a flowchart into intervals. An irreducible flowchart is any flowchart which, if reduced by intervals, will produce an elementary irreducible flowchart. A reducible flowchart on the other hand does not contain any elementary irreducible flowcharts.

2.5. Collapsibility

A concept which is closely related to reducibility is collapsibility, as proposed by Hecht. This is defined in terms of two transformations:

(a) Transformation T1: If e is an edge (or arc) of some flowchart which emanates from some node X and leads back to the same node, then T1 is the removal of this edge.

(b) Transformation T2: If y is a node of some flowchart other than the entry node and x is the only direct predecessor of y then T2 is the replacement of the two nodes x and y and any edges between them by a single node z. The predecessors of x become the predecessors of z and the successors of y become the successors of z.

A flowchart is said to be collapsible if and only if it can be transformed into the trivial graph (consisting of a single node) by repeated application of T1 and T2. Hecht has shown that collapsibility is equivalent to the notion of reducibility in Tarjan's sense.

Figure 4 illustrates the concept. Transformation T2 can be applied at several points in Fig. 4(a) (T2(A,a), T2(A,b), T2(B,d), T2(e,C), T2(D,f) to produce the flowchart given in Fig. 4(b). Transformation T1 can then be applied at node C to produce the flowchart in Fig. 4(c). Finally T2 can be applied once more (T2(C,D)) to produce the flowchart in Fig. 4(d). At this stage no further transformations can be applied. Hence none of these flowcharts is collapsible.

2.6. Structured collapsibility

A variation on the idea of collapsibility is that of structured collapsibility. The principle behind structured collapsibility is to remove any operation node which is linked directly to another node (operation, decision, entry or exit) without any incoming flowline separating them. Any simple selections or single-exit loops (while, repeat or n + j) may be removed at the same time. If each operation node is written in the form

\[ l: (p_1, \ldots, p_n)x(s_1, s_2) \]

and each decision node as

\[ l: (p_1, \ldots, p_n)x(s_1, s_2) \]

where l denotes the number or label of the statement, \( p_1, \ldots, p_n \) denotes the predecessors of statement l (i.e. \( p_1 \rightarrow l \), \( \ldots, p_n \rightarrow l \)) and \( s_1, s_2 \) the successors of statement l (i.e. \( l \rightarrow s_1, l \rightarrow s_2 \)) then the transformations associated with structured collapsibility are as follows:

1. \( l: (\alpha)l(l) \rightarrow \) (entry node is operation node) remove node l
2. \( n: (l)l(n + 1) \rightarrow \) (exit node is operation node) remove node n
3. \( i: (\alpha)(\beta, \gamma) \rightarrow i: (\alpha)l(l) \rightarrow i: (\alpha)i(i) \)
4. \( i: (\alpha)(\beta, \gamma) \rightarrow i: (\alpha)l(l) \rightarrow i: (\alpha)l(l) \)
5. \( i: (\alpha)(\beta, \gamma) \rightarrow i: (\alpha)l(l) \rightarrow i: (\alpha)l(l) \)
6. \( i: (\alpha)(\beta, \gamma) \rightarrow i: (\alpha)l(l) \rightarrow i: (\alpha)l(l) \)

\[ i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \]

7. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
8. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)

\[ i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \]

9. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
10. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)

\[ i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \]

11. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
12. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)

\[ i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \]

13. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
14. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)

\[ i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \]

15. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
16. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)

\[ i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \]

17. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
18. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)

\[ i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \]

19. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
20. \( i: (\alpha)A(x, y) \rightarrow i: (\alpha)A(x, k) \)
FLOWCHART SCHEMATA AND THE PROBLEM OF NOMENCLATURE

Figure 4. Applying Hecht’s transformations to a flowchart to test for collapsibility.

(7) $i: (\alpha)A(i,x)$ or $i: (\alpha)A(x,i) \rightarrow (\text{loop})$ replace $i$ by $i: (\alpha)A(x)$
(8) $i: (\alpha)A(j,i), j \neq i \rightarrow (\text{condition})$ replace $i$ by $i: (\alpha)A(j)$

This process is illustrated in Fig. 5. Fig. 5(a) contains a flowchart before applying these transformations. Fig. 5(b) shows the flowchart after these transformations have been applied. Essentially what this process does is to remove each basic TD, subflowchart replacing it with an operation node, and remove each operation node which is adjacent to another node without any incoming edge separating them.

If this process is combined with Prather and Giuliani’s reduction process or that described in Ref. 2, the structure shown in Fig. 5(c) would be identified and removed from the flowchart and the remaining nodes would collapse until the flowchart was reduced to a single node. Thus the structure in Fig. 5(c) represents the only unstructured component—as will be discussed in the next section.

Note that the algorithm for structured collapsibility could be modified to remove only D-subflowcharts or WR-subflowcharts although this is more complicated to do.

3. STRUCTUREDNESS

One very important idea which is connected to reducibility and collapsibility is the concept of structuredness. This idea was introduced initially by Böhm and Jacopini.

Figure 5. Illustration of structured collapsibility.
who showed how general flowcharts can be decomposed into a small set of base diagrams. However, not all flowcharts can be decomposed into 'a finite number of given base diagrams'. To overcome this problem they proposed two normalization methods for converting flowcharts which would not be decomposed into base diagrams from some finite set into equivalent flowcharts which can be decomposed thus. This idea, viewed from the point of view of synthesis instead of analysis, namely that flowcharts should be constructed initially from a small set of base diagrams has been termed structured programming.

Arising from their work, a structured flowchart is one which is composed from (or can be decomposed into) a small set of base diagrams, an unstructured flowchart is one which cannot be decomposed into a small set of base diagrams without altering it in some way (normalizing it). As has been noted in the previous section, there are various possible choices for this set of base diagrams. To distinguish between these, a flowchart will be referred to as D-structured if it can be decomposed into D-charts, WR-structured if it can be decomposed into WR-charts and TD₁-structured if it can be decomposed into TD₁-charts. Similarly a flowchart may be D-unstructured, WR-unstructured or TD₁-unstructured.

With the recognition of the importance of structured programming, there has been considerable interest in identifying what makes a flowchart structured and how an unstructured flowchart can be transformed into a structured one. To understand the constituents of unstructured flowcharts, it is helpful to draw an analogy with the world of matter. Using this three levels can be distinguished:

(a) At the lowest level are ‘unstructured particles’. These are the building blocks of the next level.
(b) At the next level one has ‘unstructured elements’. These are flowcharts constructed from structured and unstructured particles, but which may not exist in isolation if they have unsatisfied bonds (more than one entry or exit point).
(c) Unstructured elements may be combined into ‘unstructured compounds’ by coupling corresponding unsatisfied bonds. An unstructured compound is a ‘stable’ flowchart with a single entry point and a single exit point.

It must be stressed that this is a very crude analogy but one that does help to give perspective to the constituents of unstructuredness.

In a study of the complexity of programs and flowcharts, McCabe defined a flowchart as structured if it is a D-chart (i.e. D-structureness) and identified four fundamental characteristics which characterize a D-unstructured flowchart. In terms of the above analogy these would be unstructured particles of D-structureness. They are:

(a) branch out of a loop
(b) branch into a loop
(c) branch out of a selection
(d) branch into a selection.

Unfortunately he overlooked the fact that neither a repeat-loop nor an $n + rac{1}{2}$-loop are D-structures, and any flowchart which contains either of these two types of loop will not be a D-chart. Thus the full catalogue of unstructured particles which characterize D-structureness must also include:

(e) $n + rac{1}{2}$ loop
(f) repeat loop.

Obviously the unstructured particles which characterize WR-unstructureness are (a) to (e) of the above list, whereas those which characterize TD₁-unstructureness consist of those in McCabe’s original list (i.e. (a)–(d)).

When these unstructured particles are combined with structured particles (simple sequence, simple selection, etc.) this gives rise to unstructured elements. To illustrate these McCabe gave a diagrammatic representation of the four simplest elements. These are shown in Fig. 6. He also noticed that none of these constructs can exist in isolation in a flowchart (at least, none of the unstructured elements of TD₁-unstructureness can exist in isolation) since each has more than one entry point or more than one exit point. Consequently he postulated a set of four basic unstructured compounds which can arise out of combining these elements; these are shown in Fig. 7(a)–(d). These same four compounds were recognized by Williams in a paper submitted over a year before McCabe’s paper. He also recognized another basic unstructured compound obtained when two multi-exit loops are combined into a pair of parallel loops as shown in Fig. 7(g).

Oulsnam has pointed out that the construct in Fig. 6(a) can be regarded as two separate constructs depending on which exit path is taken as the main exit and which the auxiliary exit. Likewise, the construct in Fig. 6(b) can be viewed as two separate constructs depending on which entry path is taken as the main entry path and which the

![Figure 6. The simplest unstructured elements of TD₁-unstructureness.](https://academic.oup.com/comjnl/article-abstract/26/3/270/464433/9)
auxiliary. However, when each of these elements is paired with another element (as in Fig. 7), the distinction disappears. Oulsnam has added two other basic unstructured compounds to the list given by McCabe. These are shown in Fig. 7(e) and (f).

Of course, these are not the only unstructured elements and compounds which are possible. Williams and Williams and Oshser have dealt with loops with more than two exit points or more than two entry points or indeed with $n$ entry points and $m$ exit points ($n \geq 2, m \geq 2$). Again such loops do not exist in isolation but occur in combination with other unstructured elements. Thus the number of combinations is endless.

Note that the combination of the concept of structured collapsibility together with the reduction process described by Prather and Giuliani as mentioned in Section 2.6 results in a process for identifying unstructured compounds. The isolation of unstructured elements or particles is more difficult and a typical method for doing this is that given by Williams and Oshser.

It follows that any method which aims at transforming an unstructured flowchart into a structured one must deal with unstructured particles or elements rather than unstructured compounds. Various methods have been proposed, e.g. Ashcroft and Manna, Wulf, Williams and Oshser, Oulsnam, etc. Differences arise in the identification of the unstructured particles or elements, the transformations for converting them to structured form and in the structured form produced (D-structured or TD$_1$-structured).

4. CONCLUSIONS

Some interpretations of the related concepts of reducibility, collapsibility and structuredness as they apply to flowcharts have been presented in order to clarify the current position. The intention is not to provide a complete catalogue of interpretations but to study certain important ones.

It is hoped that the reader will have gained some insight from this into the similarities and differences between the interpretations and approaches adopted by different authors. In particular attention is drawn to the following:

(a) The similarity/difference between reducibility as defined by Knuth and Floyd and Kosaraju and the flowgraphs used by Cowell, Gillies and Kaposi.

(b) The similarity/difference between reducibility as defined by Cowell, Gillies and Kaposi and reducibility as used by Prather and Giuliani.

(c) In the case of flowgraphs (Cowell, Gillies and Kaposi) all operation nodes are removed, thereby camouflaging certain unstructured compounds (except those involving multi-exit loops). The process of collapsibility isolates elementary unstructured flow-
charts; when applied to the structures in Figs 6 and 7, only Figs 6(b) and 7(b) are not collapsible (these involve multi-entry loops). Structured collapsibility removes unnecessary operation nodes and basic structured forms or base diagrams, retaining only those nodes which preserve the unstructuredness of the flowchart.

(d) A distinction is drawn between unstructured particles, unstructured elements and unstructured compounds and between different types of structuredness (D-structuredness, TD1-structuredness, etc.). The sets of unstructured particles/elements corresponding to different types of unstructuredness are slightly different.

REFERENCES


Received November 1982