APPENDIX B

GRAPH GENERATION AND COMPLEXITY MEASURES

This appendix is devoted to the development of a systematic method that can be used to build control-flow structures and to divide them into levels of complexity. First the terminology used in this appendix will be introduced. Then we will discuss the graph generation method. We will conclude with a discussion of some related papers.

B.1. Terminology

(1) A *control-flow graph* (*c.f.g.*) is defined here as a flow diagram (also: flow chart, see figure B.1) from which all statement 'boxes' have been deleted, and all junction points of control-flow-lines are indicated with circles, as indicated in figure B.2.

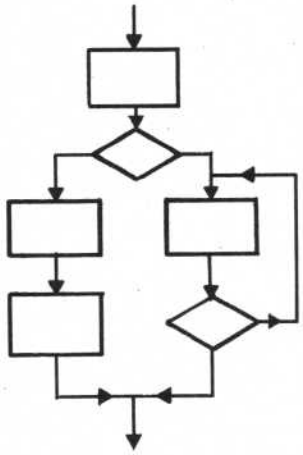


Figure B.1.
Flow Diagram

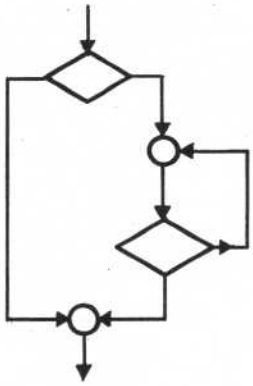


Figure B.2.
Control-Flow Graph

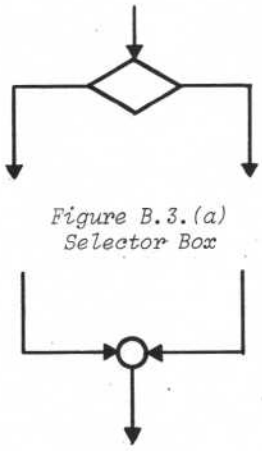


Figure B.3.(a)
Selector Box

Figure B.3.(b)
Collector (Junctor) Box

(2) A c.f.g. is called *proper* if it has precisely one entry-point and precisely one exit-point. (The c.f.g. in figure B.2 is a proper c.f.g.)
(3) Two or more c.f.g.'s can be combined to form larger structures, either by simple concatenation, or by nestings. Combinations of this type are called *structured combinations*.
(4) Any c.f.g. which can be obtained by the structured combination of smaller proper c.f.g.'s is named *decomposable*. (The c.f.g. in figure B.2 is decomposable.)
(5) Any c.f.g. which is not decomposable, and cannot be transformed into an

'equivalent' c.f.g. which is decomposable, is an *elementary control-flow graph (e.c.f.g.)*[1].
(Below we will discuss the notions of 'graph equivalence' and 'graph transformation' in more detail.)

(6) The *complexity level* of an e.c.f.g. is defined as the number of selector boxes it contains (see figure B.3).

(7) The *complexity level* of a c.f.g. is defined as the maximum of the complexity levels of the e.c.f.g.'s into which it can be decomposed.

(8) Each c.f.g. is related to a *family of c.f.g.'s* of the same complexity level. All c.f.g.'s which can be constructed by combining the c.f.g. considered with c.f.g.'s of the same or of a lower complexity level, belong to this family. The subset of the family which consists of c.f.g.'s which can be obtained by a structured combination of proper e.c.f.g.'s is a *proper family*.

(9) The set of proper e.c.f.g.'s with a complexity level of no more than one, are said to belong to the *Minimal Set* or *Dijkstra Set*.

(10) The family of c.f.g.'s which is related to the members of the Dijkstra Set is called *the structured family of c.f.g.'s*.

Observe that each c.f.g. corresponds to a finite set of (uninterpreted) flow diagrams. In each arc of the c.f.g. we can insert a statement box. If the c.f.g. has N arcs we can insert between 1 and N such statement-boxes. Thus each c.f.g. corresponds to:

$$\sum_{m=1}^{N} \binom{N}{m}$$

specific, uninterpreted flow diagrams. (No arc needs to contain more than one statement box.)
In the next section we will show that the Dijkstra Set contains three e.c.f.g.'s. These e.c.f.g.'s correspond to concatenation structures, conditional selection, and iteration. It will also be shown that there is no more than one proper e.c.f.g. with a complexity level of two. That e.c.f.g. corresponds to a simple loop-exit construct.
The structured family of c.f.g.'s corresponds (at least qua structure) to all programs that can be drawn in the form of a structogramme.

B.2. Method

We discuss a systematic way to generate all proper e.c.f.g.'s. As a starting point we use the trivial e.c.f.g. of complexity level zero, indicated in figure B.4. We call this e.c.f.g. the '0^{th} generation' or 'common ancestor'.
Observe that any (e.)c.f.g. should contain at least one control path (arc).
The trivial e.c.f.g. from figure B.4 is therefore the only proper e.c.f.g. with a complexity level of zero.
Observe that the common ancestor represents all concatenation structures.

To generate a next generation of proper e.c.f.g.'s (in this case the 1^{st} generation of e.c.f.g.'s of complexity level one), we must insert the two building blocks represented in figure B.3 in the ancestor-graphs.
Each proper e.c.f.g. may have only one entry point and one exit point, so the building blocks of figure B.3 can only be inserted in pairs, as indicated in figure B.5.
The common ancestor has only one insertion point (one arc) and the pair of figure B.5 can be inserted in only two different ways. Either we use arc 1 as the entry point of the substructure, join arcs 2 and 3, and use arc 4 as the exit point; or we use arc 3 as the entry point, join arcs 1 and 4, and use arc 2 as the exit point. The result of the insertions is indicated in figure B.6.

— — — — —

(1) Notice the analogy between e.c.f.g.'s and prime numbers.

Figure B.4.
Trivial Graph of 0th Generation

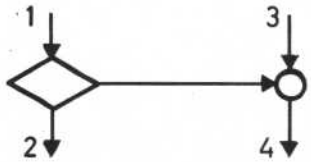
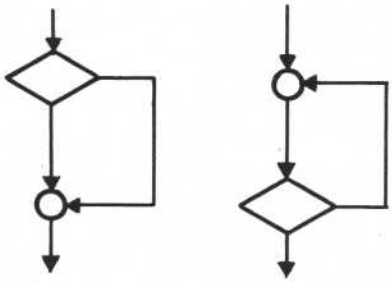


Figure B.5.
Inserted Combination

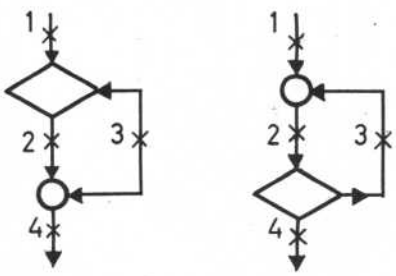


(a) *(b)*

Figure B.6.
First Generation

The e.c.f.g. in figure B.6.a corresponds to selection structures, the other e.c.f.g. corresponds to iteration structures.

To generate all proper e.c.f.g.'s with a complexity level of two (the 2^{nd} generation) we must insert another pair (figure B.5) in each of the two e.c.f.g.'s of the first generation. In figure B.7 we have indicated all possible insertion points. In each point we can insert either the selector box or the junctor box, but not both as that would yield a decomposable graph and not an e.c.f.g.



(a) *(b)*

Figure B.7.
Insertion Points for the Second Generation

The insertion yields 5 different non-decomposable c.f.g.'s, as indicated in figure B.8. Four of these contain adjacent junctor boxes. These c.f.g.'s can be transformed into *equivalent* c.f.g.'s which are decomposable. The four c.f.g.'s considered in figure B.8 are therefore no true e.c.f.g.'s and do not belong to the '2^{nd} generation of proper e.c.f.g.'s'.
We will first show how the transformations can be made, and what type of equivalence we have in mind.

All the c.f.g.'s considered have adjacent junctor boxes. We distinguish between two cases:

(1) either the arc between the two junctor boxes does contain a statement box in the corresponding flow diagrams which are considered relevant,
(2) or it does not.

Those flow diagrams which do contain a statement box in the arc considered can trivially be transformed into flow diagrams which do not. (The statement box is replaced by two equivalent (qua function) statement boxes in the arcs directed at the junctor box which precedes the statement box considered (all related to the 'control-flow-direction').) The statement box is, in other words, 'pushed' through the collector node in the opposite direction of the flow-of-control. Note that this transformation does not affect the structure of the c.f.g.
If the arc between the collector boxes contains no (more) statement boxes in the flow diagrams which are relevant, the order of the collector boxes

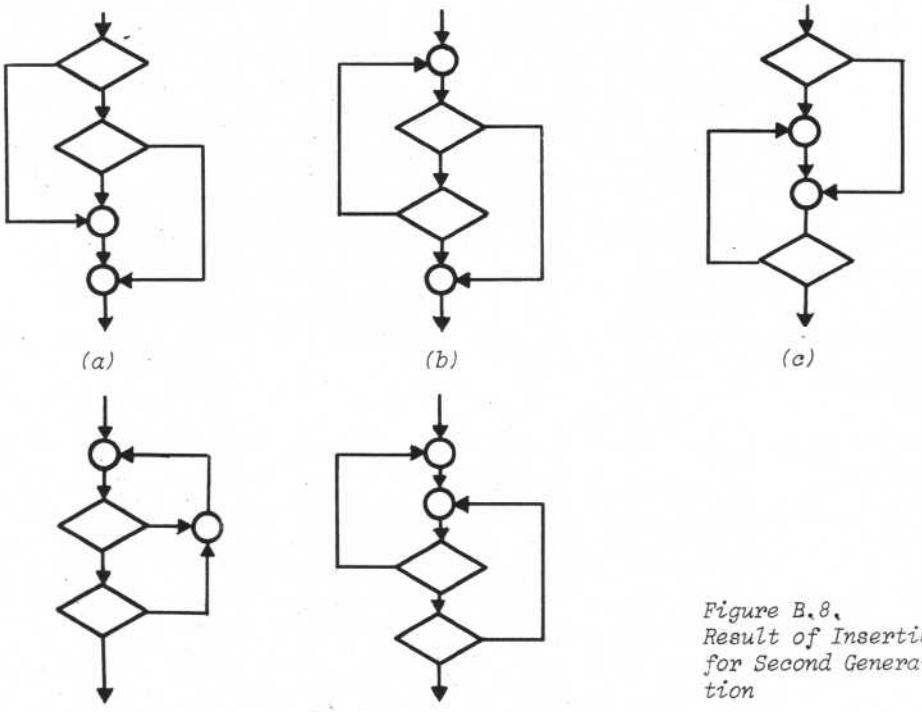


Figure B.8. Result of Insertion for Second Generation

becomes irrelevant. The boxes can be rearranged in any order that seems convenient, without disturbing the flow-of-control. Rearrangement of the collector boxes in the four c.f.g.'s considered, yields the three c.f.g.'s indicated in figure B.9. Observe that the two c.f.g.'s indicated in figure B.8.d and B.8.e are both equivalent to the same c.f.g. indicated in figure B.9.c. The c.f.g.'s in figure B.9 are clearly not e.c.f.g.'s as they are decomposable. The four c.f.g.'s considered in figure B.8 (B.8.a, B.8.c, B.8.d, B.8.e) are therefore also not e.c.f.g.'s, as they are equivalent to decomposable c.f.g.'s.

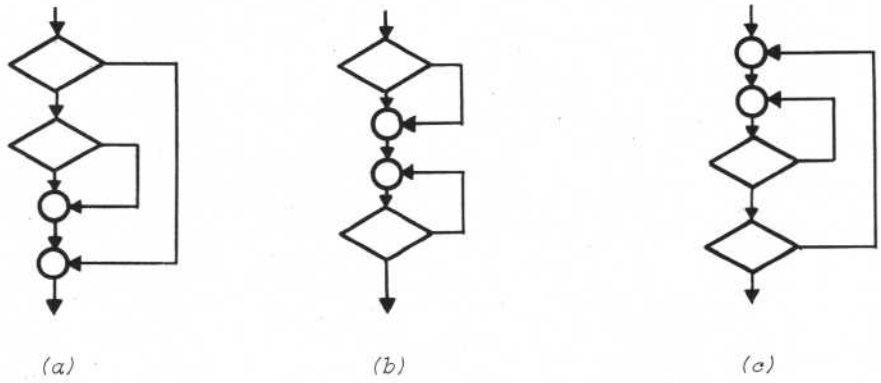


Figure B.9.: Graph 9a is equivalent to 8a, 9b to 8c, and 9c to 8d & 8e.

The only proper e.c.f.g. with a complexity level of 2 appears to be the one indicated in figure B.8.b. We recognize this structure as a single-level loop-exit, although one may also interpret it as a backward goto-jump from a selection structure.

If we continue the generating process we find 12 non-decomposable c.f.g.'s with a complexity level of 3. In figure B.10 we have given the insertion points for this new generation.
From the 12 structures of complexity level 3, only 2 seem to be relevant to normal programming practice. These two structures are represented in figure B.11, below. They correspond to a 'two-level loopexit' and a duplicated 'single-level loop exit', as would be feasible with, for instance, the REPEAT/EXITIF construct discussed in section 3.5.
The number of structures found in still higher generations increases quite rapidly with the complexity level. The practical relevance of these structures, however, decreases equally rapid. We therefore stop the generating process at this point.
Observe that the loop-exit structures found in the 3rd generation of e.c.f.g.'s include the flow graph presented by **Bohm & Jacopini '66** as an example of a non-decomposable structure (see fig. B.11.b; cf. Mills **'72, pg. 27**). This flow graph is copied in figure B.12.

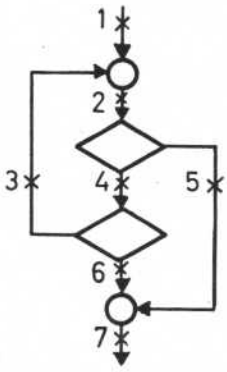
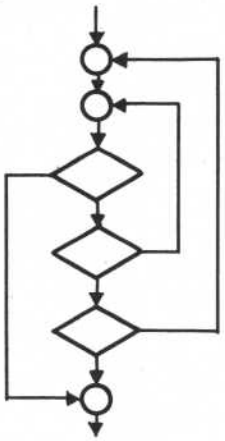
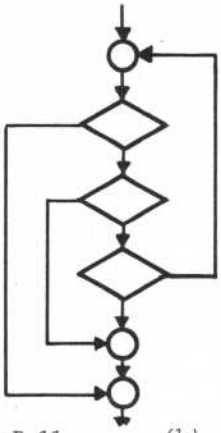


Figure B.10.
Insertion Points for the 3rd Generation.



(a)



(b)

Figure B.11.
Two e.c.f.g.'s from the 3rd Generation.

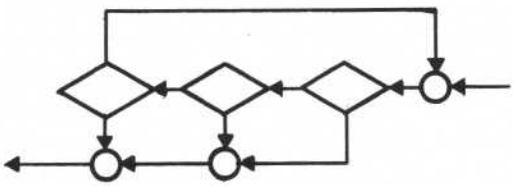


Figure B.12.
Non-Decomposable Flow Graph Discussed by Bohm & Jacopini.

Summarizing: we have seen that the 'Dijkstra Set' contains precisely 3 members, corresponding to *concatenations*, *conditional selections*, and *iterations*. We have also seen that there is only 1 proper e.c.f.g. with a complexity level of two. The generating method described here allowed us to construct a hierarchy of control structures, ordered by their complexity.

B.3. Discussion

We will discuss two articles in relation to the methods described here: **McCabe '76** and **Martin '73.**

McCabe introduced the 'cyclomatic number' from graph theories as a measure for the complexity of a flow graph. The cyclomatic number is defined as 'the maximum of linearly independent paths in the graph' **(McCabe '76, pg. 309).** If a is the number of arcs, n is the number of vertices, and p the number of connected components in the graph considered, then the cyclomatic number equals

$$(a - n + p).$$

(See **Berge '73.**)

McCabe was able to prove that:

> "... the cyclomatic complexity of a structured program equals the number of predicates plus one, ..."
> **(McCabe '76, pg. 314).**

This complexity measure, however, ignores the hierarchical structure of well-structured programs. The complexity level defined in this appendix is defined in terms of e.c.f.g.'s into which a c.f.g. can be decomposed, and thus accounts explicitly for their structure.
McCabe realized the difficulty noted here, and attempted to solve it, rather unconvincingly, by stating that:

> "... the essential complexity of a structured program is one."
> **(McCabe '76, pg. 318).**

Martin '73 discussed another method for generating control-flow structures, while using selector boxes and collector boxes as independent building blocks. Martin made a distinction between D-type structures and C-type structures (respectively with one entry and two exits, or two entries and one exit). Flow charts which can be decomposed into such D-type or C-type structures are, however, not necessarily simpler than flow charts which cannot. Using D-type structures independent of C-type structures further implies an unrestricted use of goto-statements. Martin did however confirm that:

> "... every 'legitimate' use of the goto would lead to a control structure more complex than the rules of good programming permit."
> **(Martin '73).**

Martin did not discuss a systematic method for generating control-flow structures. In fact, the structures which Martin did consider are of little relevance to (structured) programming.
The 'natural set' of control-flow structures derived by Martin is rather awkward (see figure B.13). Martin aptly remarked that:

> "They lack structure and they cause ambiguous decompositions."
> **(Martin '73).**

The attempted solution of Martin was to eliminate some of the impractical structures from this 'natural set', with the aid of two *ad hoc* rules.

Summarizing: The structures which were derived by Martin and which are not included in the 'Dijkstra Set' derived here are either irrelevant (figure B.13, structures f, g, h, and i) or unusable as they contain inherent, and unrestricted goto-jumps (figure B.13, structures d and e). The only relevant extension of the three basic control-flow structures from structured programming (i.e. the structures in the 'Dijkstra Set'), namely the loop-exit shown in figure B.8.b, was not found by Martin.

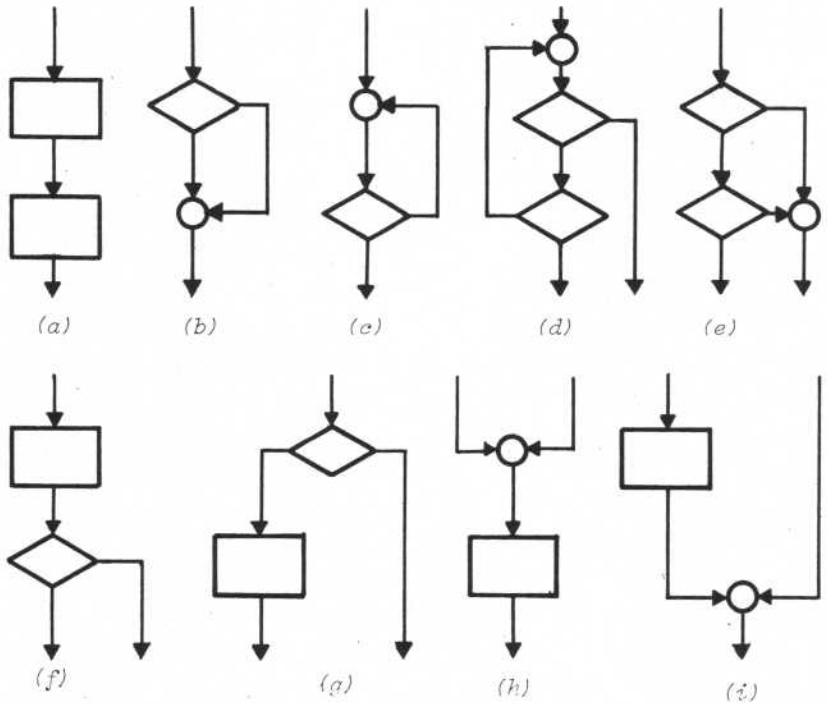


Figure B.13.
'Natural Set' Derived by Martin

B.4. References

Berge, C. (1973), *Graphs and hypergraphs*, North-Holland Publishing Co., Amsterdam, 1973.

Bohm, C. & Jacopini, G. (1966), *Flow diagrams, Turing machines and languages with only two formation rules*, Comm. ACM, Vol. 9, No. 5, 366-371, (May 1966).

Martin, J.J. (1973), *The natural set of basic control structures*, Sigplan Notices, Vol. 8, No. 12, 1973, 5-14.

McCabe, T.J. (1976), *A complexity measure*, IEEE Trans. on Software Eng., Vol. SE-2, No. 4, 1976, 308-321.

Mills, H.D. (1972), *Mathematical foundations for structured programming*, IBM-FSD-1972, Gaithersburg, Maryland, Febr. 1972, 62 pgs.