APPLICATIONS OF A GRAPH GRAMMAR FOR PROGRAM CONTROL FLOW ANALYSIS

Ken Kennedy*
Linda Zucconi*

Dept. of Mathematical Sciences
Rice University
Houston, Texas 77001

I. Introduction

A standard approach to the analysis of program structure for the purpose of code optimization is to construct the "control flow graph" which models the possible execution paths through the program. Various graph algorithms can be applied to the control flow graph to produce data flow information, possible optimizations, etc. [A1,A2,AC,AU2,AU3,CS,HU1,HU2,HU3,Ke1,Ke2,Ke3,Ke4,Sc,U]. Studies of the form of typical control flow graphs indicate that such graphs tend to fall into a restricted subclass of general graphs. For example, empirical investigations have shown that the vast majority of program graphs have no multiple-entry loops [AC,HU2,HU3,Kn1].

The recent work on "structured programming" has suggested that "good" programs fall into an even more restricted subclass. In fact, purists recommend that all programs be synthesized from three basic control structures: sequential statements, if-then-else statements, and single-entry single-exit loops [Di,Wi].

Formal language theory [HoU] has given us a practical way to specify the set of strings which comprise a given language: via a grammar. It is then a natural idea to extend grammars from the strings to graphs in hopes of getting the same power of expression. Several researchers have used this approach [FKZ,J2,Ro].

In this paper we study the applicability of a grammatical approach to describing the set of control flow graphs which arise from "good" programs in the sense proposed by many programming practitioners. The resulting flow graph language contains all those programs constructed according to the purists' rules and also admits programs with multiple-exit loops if such loops are constructed sensibly. The grammar we use is the "semi-structured flow graph" grammar G_{SSFG} which was studied originally in [FKZ]. There are several appealing properties of this grammar; perhaps the most important, from the point-of-view of a compiler-writer, is the existence of a linear-time parsing algorithm which leads directly to a linear-time data flow analysis method [FKZ].

In the present work we summarize the results from [FKZ] and address several new questions. First, how often do programs written by people with no knowledge of the SSFG rules fall into the language defined by G_{SSFG}? In other words, is the language a natural one for programming? Second, once a program has been parsed according to G_{SSFG} do benefits other than fast data flow analysis accrue?

The paper is organized into three main sections. Section II introduces G_{SSFG} and the parsing algorithm from [FKZ]. Section III is devoted to an empirical study conducted by the authors in an attempt to answer the question of naturalness, described above. Section IV discusses several applications of the graph parse in a "graph attribute grammar" framework. The summary at the end of the paper includes suggestions for further investigation.

II. The Grammar

The semi-structured flow graph grammar G_{SSFG} consists of the nine rules depicted in Figure 1.

* Work supported by the National Science Foundation, Division of Computer Research, Grant DCR73-03365-A01, Department of Mathematical Sciences, Rice University

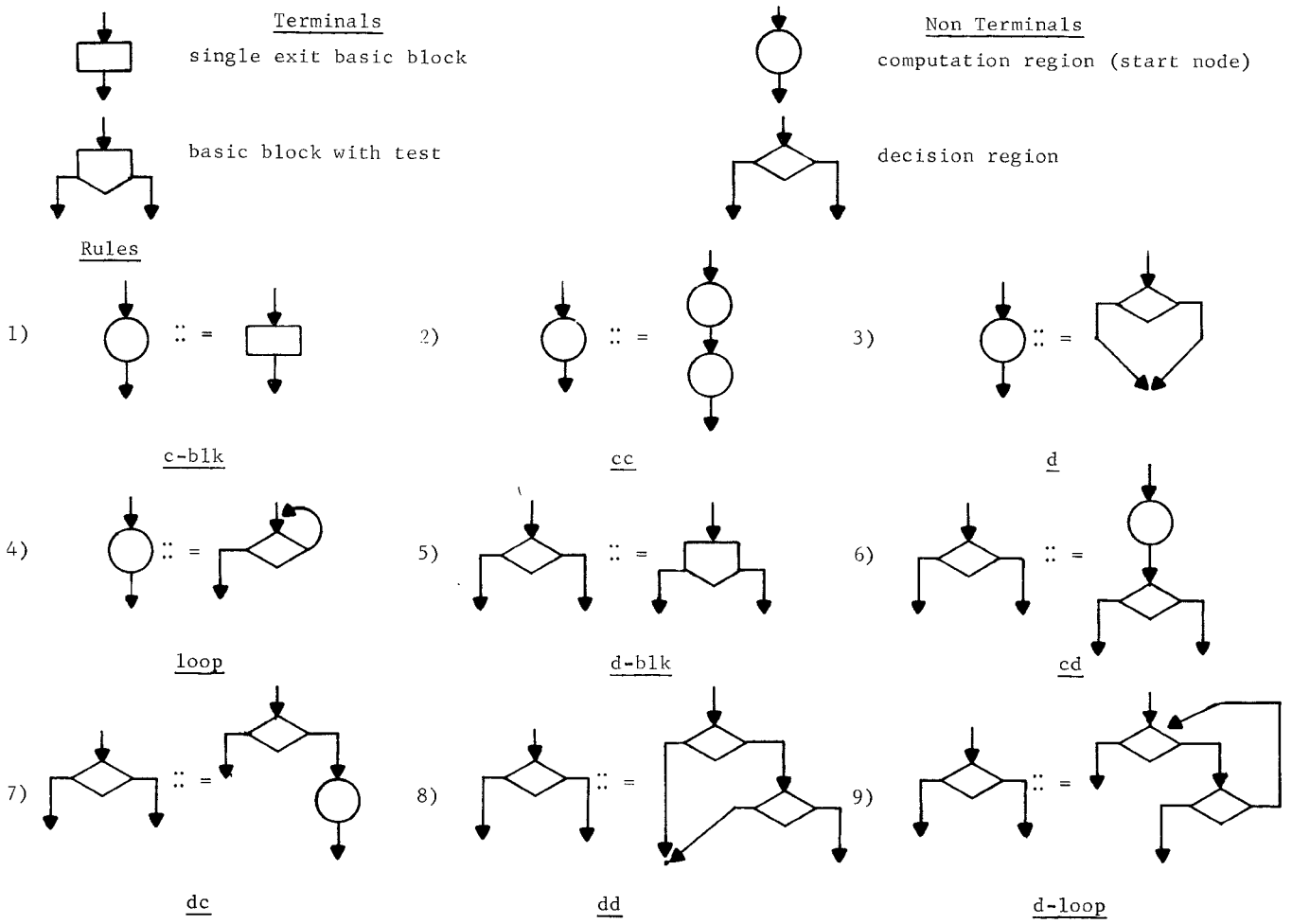


Figure 1. G_{SSFG}

Note that the two terminal symbols represent straight-line blocks of code with and without a conditional jump at the end. The two non-terminals represent a computation region with one exit and a decision region which may do some computation but also makes a decision at some point about which exit to take.

The family $\mathcal{F}_{SSFG}$ of SSFG flow graphs is the set of graphs which can be generated by applying the rules of G_{SSFG}. Note that rules 7, 8, and 9 allow various mirror images. A graph Γ is said to be SSFG-reducible if $\Gamma \in \mathcal{F}_{SSFG}$. Consider the spiral graph depicted in Figure 2. This is an example graph taken from [Ke2].

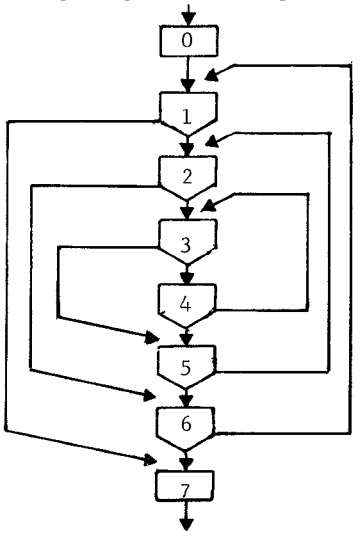


Figure 2. Spiral Graph

A derivation for the spiral graph is shown in Figure 3. Whether or not a particular flow graph is reducible can be determined by applying the parsing algorithm presented in [FKZ]. If the given graph is a member of $\mathcal{F}_{SSFG}$, the parsing algorithm returns the sequence of productions required to reduce it to a single node; otherwise, the algorithm halts with a report of failure.

For completeness, we include an outline of the parse algorithm:

Algorithm P. SSFG Parse (outline).

Input: 1) A graph Γ

2) A list L_u of unvisited nodes in straight order (the reverse of the order of last visit by depth-first search).

Output: If $\Gamma \in \mathcal{F}_{SSFG}$ a parse P_Γ otherwise, the answer "no."

Method:

```
begin  a: = entry node of Γ; P_Γ = ∅;  remove a from L_u;
visit: while a ≠ null do
          apply all reductions possible with a as header, set success to true if at least one
          reduction takes place, false otherwise; add reductions to P_Γ ; make a the unique
          linked predecessor of all its unvisited successors:
          if success and a is linked to predecessor b
             then a: = b
             else if L_u = ∅ then a = null
                  else a: = next unvisited node from L_u;
                       remove a from L_u
                  fi
          fi
          od
       if Γ is now a single computation node then return P_Γ else return "no" fi
end
```

Algorithm P, properly implemented, parses arbitrary graphs in time linear in the size of the graph. Furthermore, the parse leads to a linear-time algorithm for data flow analysis on members of $\mathcal{F}_{SSFG}$, [FKZ].

It was shown in [FKZ] that the family $\mathcal{F}_{SSFG}$ admits many graphs produced through the use of control structures suggested as extensions for "structured programming." Examples are Zahn's control structure, [Za,Kn] Martin's "natural" set of control structures [Ma] and the counterexample of Ashcroft and Manna [AM]. From those results we were led to ponder the question: How many programs, written with or without structured concepts, fall naturally into $\mathcal{F}_{SSFG}$? That question led us to the investigation described in the next section.

III. The Empirical Study

For our study of the structure of typical application programs we decided to look at FORTRAN programs since there exist numerous scientific applications programmed in that language and since the control structures do not particularly encourage good programming.

The next step was to acquire programs for the study. Members of the Rice faculty generously provided us with an ample supply which we subdivided into the following groups.

1) Sixty FORTRAN programs written by students in an engineering class where "structured programming" was not taught. The sample represented student solutions to two problems:
   a) find the roots of a cubic equation,
   b) compute the time to target and the required angle from the plane for a projectile with given velocity and range.
2) Two hundred twenty-seven subroutines from a large Chemistry Dept. application: to construct the wave function for a compound during molecular scattering using the "stabilization" method. The routines were as much as ten years old and many had been written at other universities.
3) Sixty-two Physics Dept. routines used to calculate eigenvalues and wave functions for certain Hamiltonians in a molecular structure application.
4) Seventy-six Biochemistry Dept. routines comprising a large crystallographic system for analyzing X-ray diffraction data to deduce molecular structure.

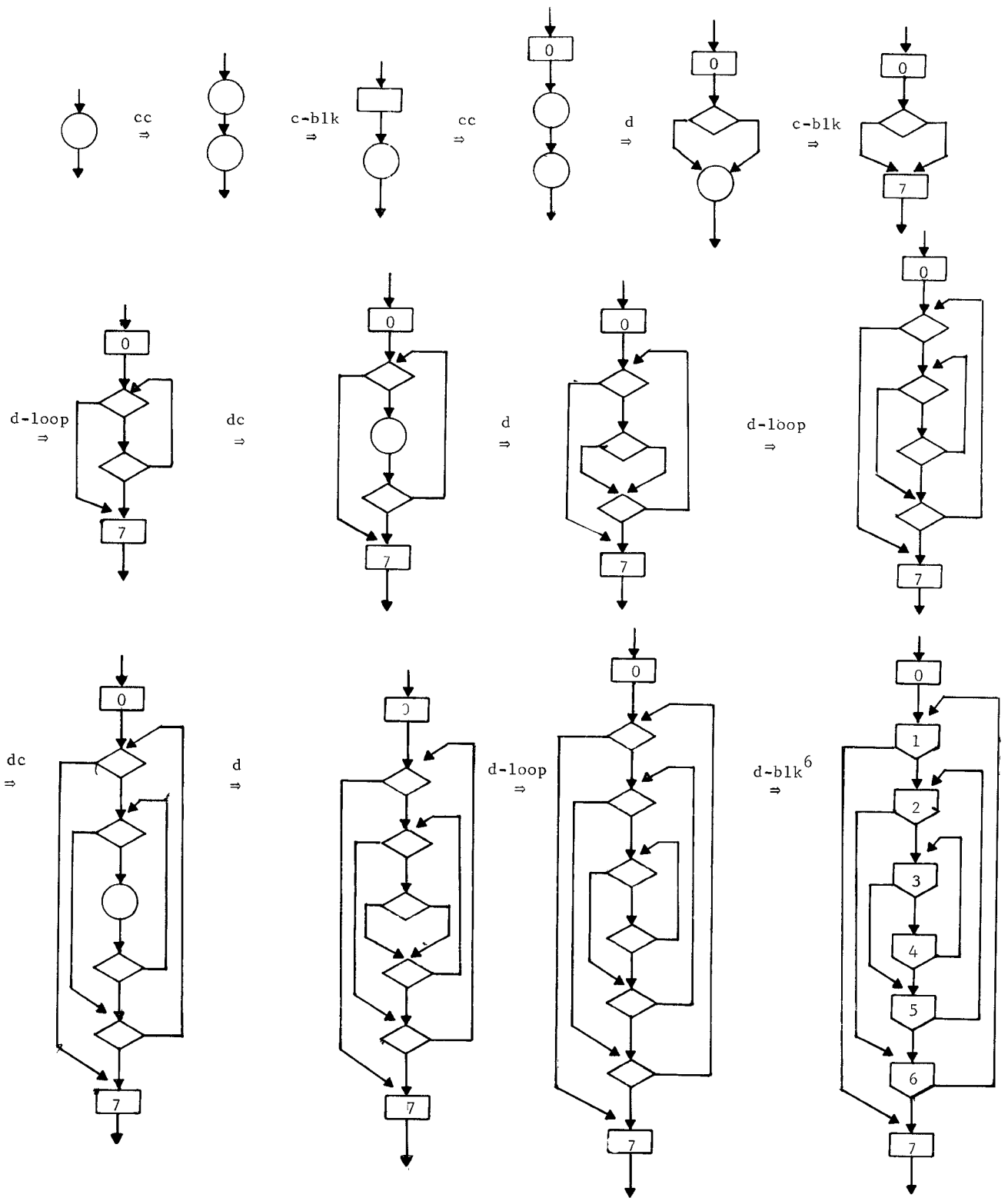


Figure 3. Derivation of Spiral Graph

5) Five routines used by the Chemistry Dept. to determine mean square end-to-end separation of molecules via a Monte Carlo simulation of macro molecular structure.

6) Thirty-six routines comprising two Mathematical Sciences Dept. applications:

   a) a finite element differential equation solver which uses a Galerkin procedure and isoparametric elements.

   b) a collection of H1 procedures for solving quadratic homogenous partial differential equations.

7) Thirty-four routines from two Geology Dept. programs:

   a) a program which solves the free convection problem for a medium with variable viscosity by solving a system of coupled partial differential equations.

   b) a program which simulates the slow flow of liquids under given boundary conditions.

All in all, a total of five hundred routines were analyzed to determine how many were Cocke-Allen reducible and how many were SSFG-reducible. The results are summarized in Table 1 below.

Group	Number of programs	Number of Cocke-Allen reducible	% Cocke-Allen reducible	# SSFG-reducible	% SSFG reducible	% of Cocke-Allen reducible programs SSFG reducible
1	60	58	97	55	92	94.8
2	227	221	97	199	88	94.3
3	62	60	97	56	90	93
4	76	62	81.5	41	54	66
5	5	4	80	0	0	0
6	36	33	92	30	83	91
7	34	32	94	30	88	94
Total	500	470	94	411	82	87.5

Table 1. Reducibility Analysis

The last column, indicating the percentage of Cocke-Allen reducible programs which are also SSFG-reducible is worth comment. Since most fast data flow analysis routines work for Cocke-Allen reducible programs, one should compare the linear-time SSFG method with those for applicability. The last column in Table 1 indicates that the SSFG method would work in nearly 90% of those cases which could be handled by one of the fast non-linear methods [GW,Ke3,AU3].

Overall, more than four out of five of these programs, all written without the benefit of structured programming, were SSFG-reducible. Though we were pleased with this result, we were disturbed by the intractability of groups 4 and 5, without which 88% of the programs would have been SSFG-reducible. We therefore undertook a study of the programs in these groups to determine the reasons they failed to reduce so frequently.

The first step was to perform a static analysis on all the programs to determine average length and frequencies of various statement types. The results of this study appear in Table 2.

Group	Avg. # of statements	Avg. # if-statements	Avg. # do-loops	Avg. # goto's	Avg. # computed goto's	Avg. # labelled statements
1. Student	41	2.6	.4	2	0	5
2. Chemistry	97	9	5	3	.28	13
3. Physics	66.5	5	6	2.5	.25	16
4. Biochem	300	31	11	15	1.5	69
5. Chemistry	320	50	10	20	3	54
6. Math Sci	76	6	5	3	1	11
7. Geology	89	7	7	4	1	15

Table 2. Static Analysis

From this table we can see that the programs in groups 4 and 5 were extremely long and contained many goto-statements. If we compare the statistics for group 4 with those for group 2, which reduced more frequently, we see that group 4 programs were three times as long (on the average) but contained five times as many goto's, five times as many labelled statements and only twice as many do-loops. It seems likely that many of the loops in group 4 were implemented with goto's rather than do's, resulting in a fairly complex control flow structure.

Our next step was to analyze programs in groups 4 and 5 to determine specifically why they failed and to see if a richer grammar might reduce them. For the purposes of this analysis, we defined two new grammars. The first, extended the SSFG grammar by adding two new rules depicted in Figure 4.

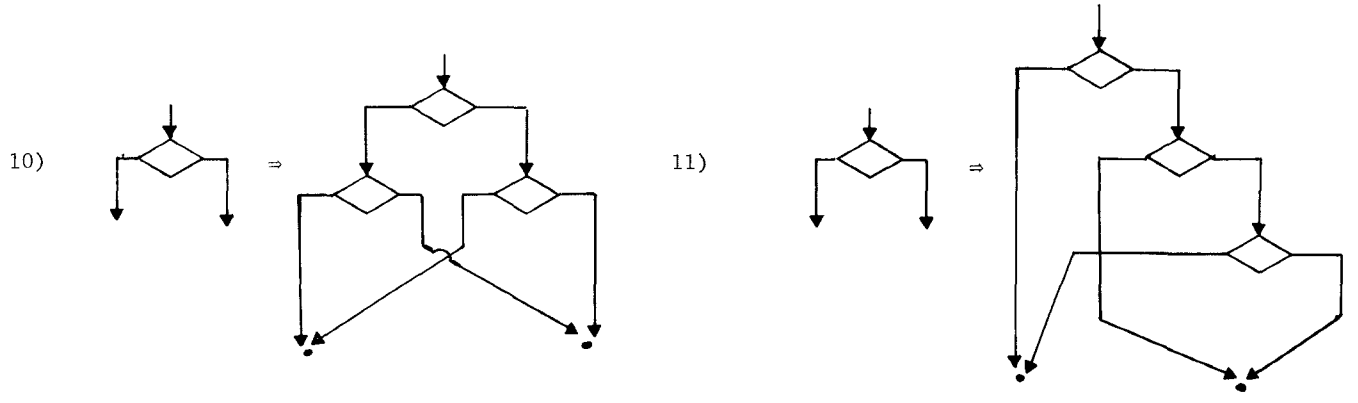


Figure 4. $SSFG_x$ extensions

The resulting grammar is called $SSFG_x$ and an analysis of groups 4 and 5 showed that an additional 7 programs reduced under these rules.

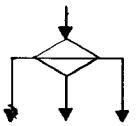
A more ambitious grammar admitting 3-exit regions was also considered. This grammar, named $SSFG_3$, consists of rules 1-7 of SSFG and the nine rules depicted in Figure 5. Groups 4 and 5 were also analyzed for reduction under this grammar. The results of these studies are summarized in Table 3.

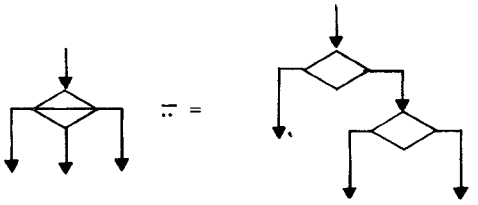
	# of programs	Number of Cocke-Allen reducible	# SSFG reducible	# $SSFG_x$ reducible	% Cocke-Allen $SSFG_x$ reducible	# $SSFG_3$ reducible	% Cocke-Allen which were $SSFG_3$ reducible
4. Biochem	76	62	41	47	76	58	93.5
5. Chemistry	5	5	0	1	25	3	75

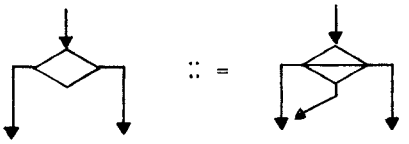
Table 3.

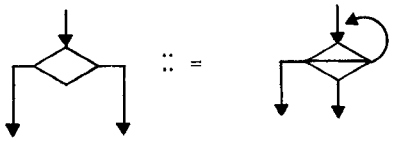
It can be seen that significant improvements in reduction percentages can be achieved by using the larger grammars.

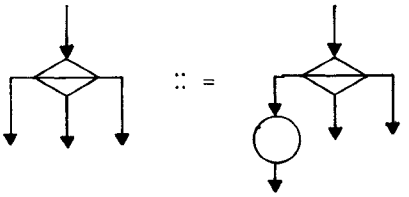
However, it is our belief that the programs in group 4 and 5 have too many complex control structures and that increasing the complexity of our grammar is not an appropriate way to deal with such programs. We are currently investigating a systematic code-copying approach which will be reported on in another paper.

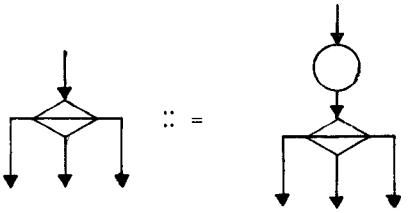
new nonterminal  (three exit region)

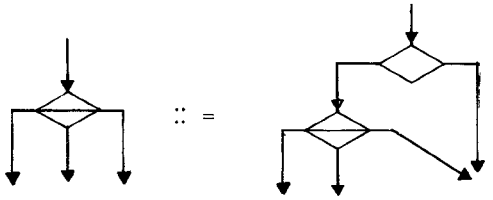
8) 

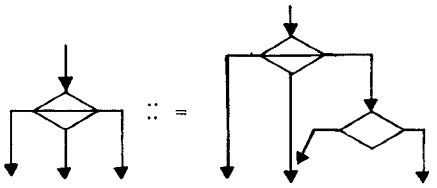
9) 

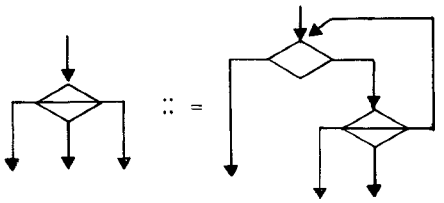
10) 

11) 

12) 

13) 

14) 

15) 

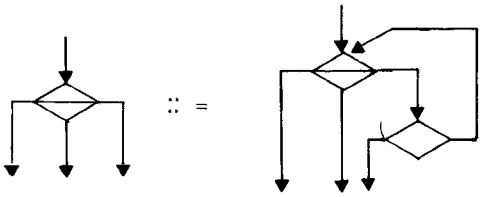
16) 

Figure 5. Additional rules for $SSFG_3$

IV. Applications

The grammatical description of program flow graphs allows a number of applications to be carried over from language theory. In particular, many algorithms on the flow graph can be specified via "attributes" on the graph grammar [Kn3,Kn4]. As an example, we show how one might specify the analysis of "profitability."

Simply stated, the problem is this. Suppose we are given for each branch (x,y) in the program, the probability p(x,y) that that branch will be taken after block x is executed. We wish to determine the expected frequency f(x) of execution of each block x in the program during a single execution of the whole program. Under certain simplifying "Markov assumptions" [CK], the expected frequencies can be expressed as follows:

$$f(n_o) = 1 \qquad n_o, \text{ the program entry node}$$

$$(*)$$

$$f(x) = \sum_{(y,x)\in E} p(y,x)f(y) \qquad \text{all other nodes}$$

A solution-finding method for the system of equations (*) is described by the graph attribute grammar in Figure 6.

There are several points to notice about this graph grammar.

1) The attribute p is a synthesized attribute; that is, its value for composite regions is based on its value for nodes within the region. The values of p for terminal nodes are "given."

2) The attribute f is an inherited attribute; that is, its value for nodes within a region is based upon its value for the region as a whole (and upon the values of p).

3) Because of the dependence of f upon p, we are forced to evaluate p first. Thus the attribute grammar in Figure 8 gives rise to a two-pass algorithm. The first pass moves forward through the parse, computing p for larger and larger regions; the second pass moves backward through the parse, computing f for smaller and smaller regions.

4) The first value of f is the one for the whole program n_o which is 1 by assumption (the whole program is executed once).

Figures 7 and 8 present an example profitability computation, with Figure 8 depicting the reduction pass and associated transition probability computations and Figure 9 depicting the production pass and frequency computations.

From the considerations above and the example, we can see that the attribute grammar specification gives rise to the classical algorithm for profitability [CK]. Furthermore, by applying analogs of techniques in [KW,KR] we can compile these attribute grammars into efficiently-executing finite-state machines which use the parse (or its reverse) as input. The eventual result may be a system to generate graph-based optimization algorithms. By analogy with [FKZ] these algorithms should be linear in the size of the input program.

V. Summary and Conclusions

We have investigated the practicability of a graph grammar for program control flow from two viewpoints: its "naturalness" for describing programs and its applicability to compiler construction.

An empirical study has shown that programmers (even those untutored in "structured programming") tend to write programs which are derivable using G_{SSFG}. We might conclude that a typical "good programmer" would not find rules based upon G_{SSFG} too restrictive.

We have also shown a promising new way for specifying global flow algorithms via an attributed version of G_{SSFG}. This could lead to a system for the automatic generation of compiler optimizers.

Acknowledgements

We are particularly grateful to our colleagues at Rice who provided us with sample programs:

Biochemistry Dept: Professor Florante Quiocho and George Phillips.
Chemistry Dept: Professor Edward F. Hayes, Professor Frederick T. Wall, Richard Liedtke, and John Chen.
Geology Dept: Professor Jean-Claude De Bremaecker.
Physics Dept. Professor Neal F. Lane, Tom Winter, and Allen Haggard.
Mathematical Sciences Dept: Professor Mary F. Wheeler, Bruce Darlow, and Naresh Garg.
Civil Engineering Dept: Professor Edward C. Holt, Jr.

We also appreciate helpful comments given by Barry Rosen of IBM Research and Rodney Farrow of Rice.

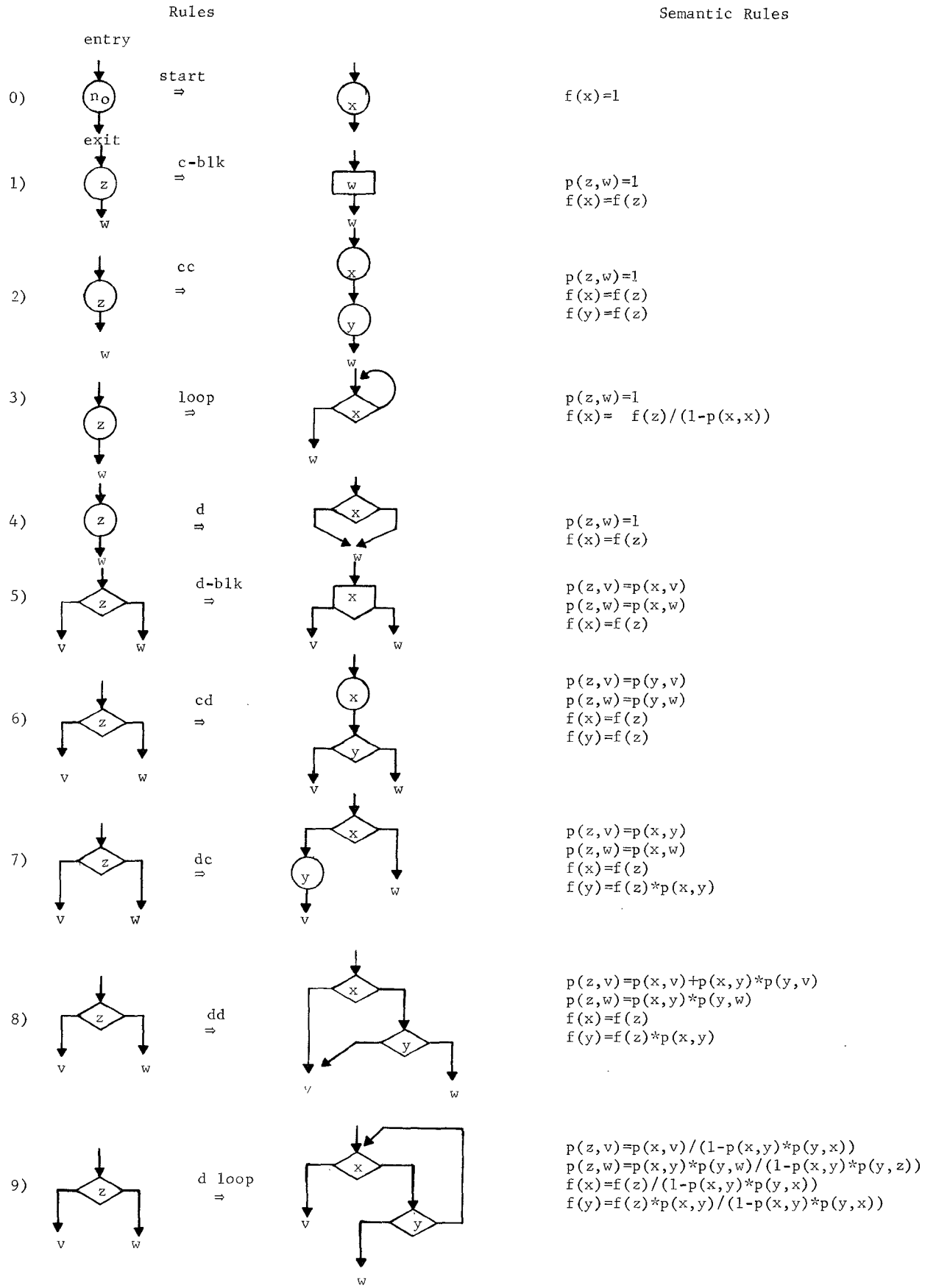


Figure 6. Attributed G_{SSFG} for "Profitability"

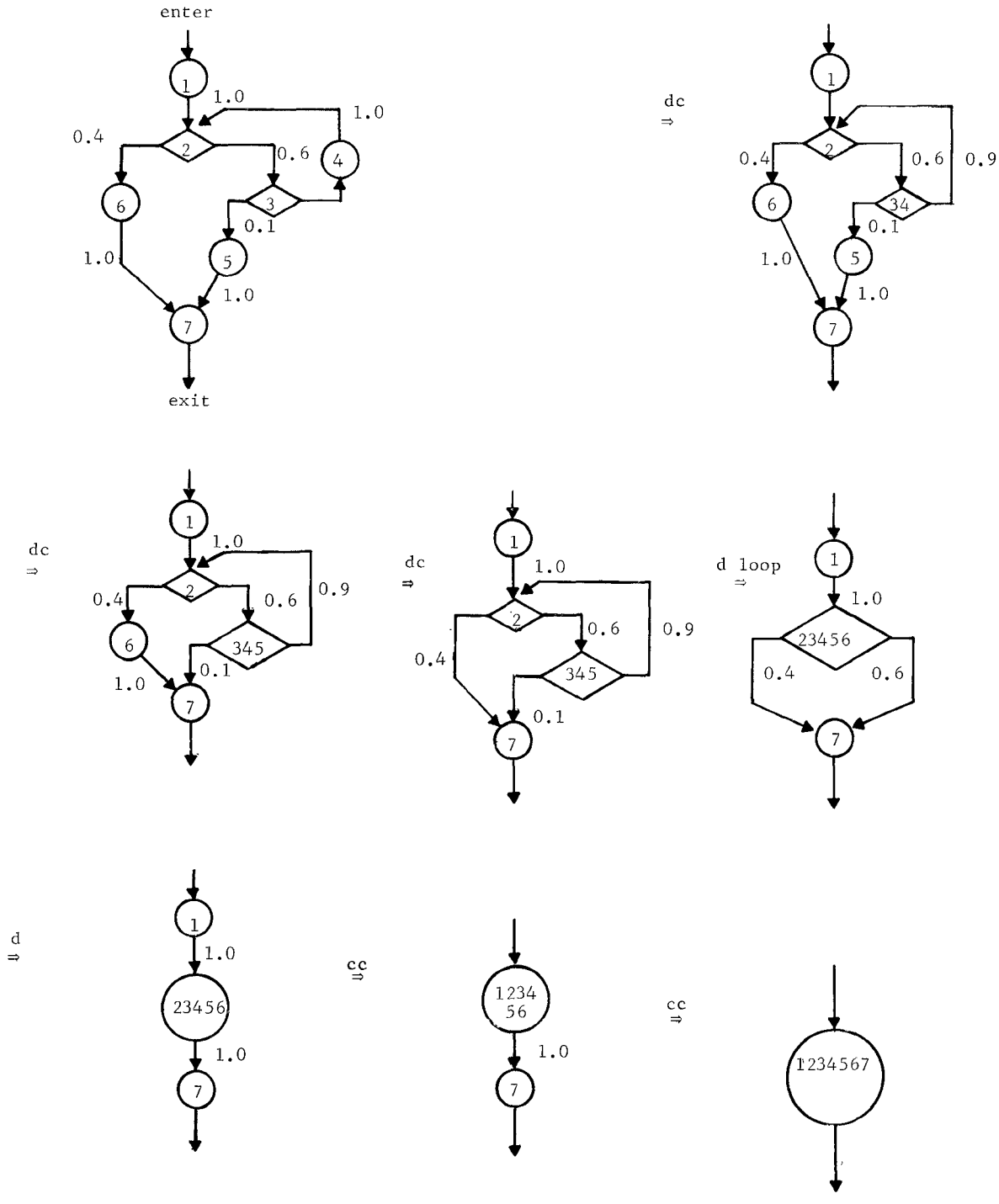


Figure 7. A Profitability Reduction Sequence

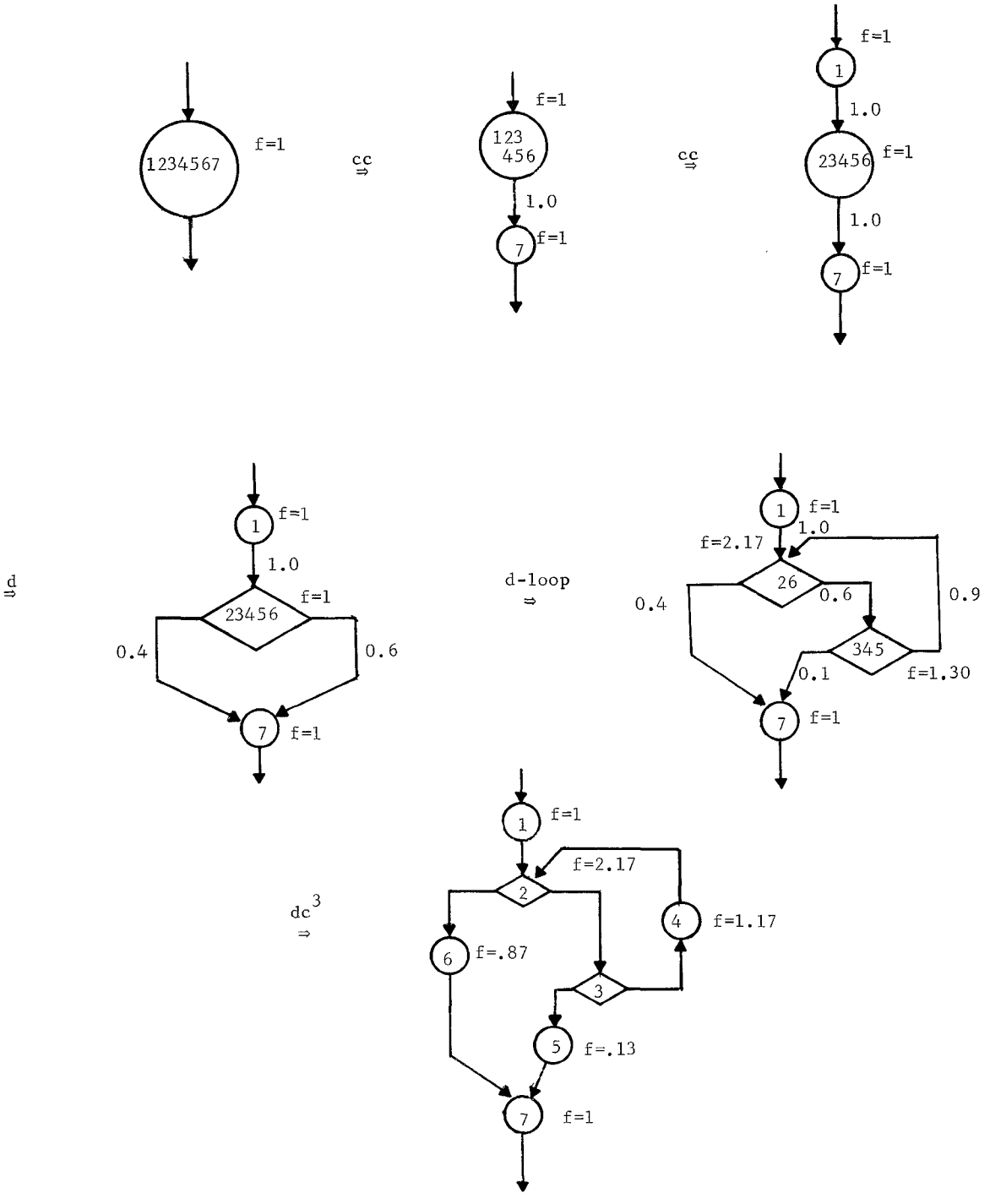


Figure 8. Profitability Production Sequence

REFERENCES

[ASU] Aho, A.V., Sethi, R. and Ullmann, J.D., "Code Optimization and Finite Church-Rosser Systems" in Design and Optimization of Compilers, (R. Rustin, ed.) Prentice-Hall, Englewood Cliffs, N.J., 1972.

[AU1] Aho, A.V. and Ullmann, J.D., The Theory of Parsing, Translation and Compiling, Vol. I:Parsing, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[AU2] Aho, A.V. and Ullman, J.D., The Theory of Parsing, Translation and Compiling, Vol. II: Compiling, Prentice-Hall, Englewood Cliffs, N.J., 1973.

[AU3] Aho, A.V. and Ullman, J.D., "Node Listings for Reducible Flow Graphs," Proc. Seventh Annual ACM Symposium on Theory of Computing, Albuquerque, N.M., pp. 177-185 (May, 1975).

[A1] Allen, F.E., "Control Flow Analysis," SIGPLAN Notices, Vol. 5, No. 7, pp. 1-19, July 1970.

[A2] Allen, F.E., "A Basis for Program Optimization," Proc. IFIP Conf. 71, North-Holland Publishing Co. Amsterdam, 1971.

[AC] Allen, F.E. and Cocke, J., "Graph Theoretic Constructs for Program Flow Analysis," IBM Research Report RC 3923, T.J. Watson Research Center, Yorktown Heights, N.Y., July 1972.

[AM] Ashcroft, E. and Manna, Z., "The Translation of 'Goto' Programs Into 'While' Programs," Proc. IFIP Conf. 71, North Holland Publishing Co., Amsterdam, 1971.

[Be] Beatty, J.C., "An Algorithm for Tracing Live Variables Based On A Straightened Program Graph," IBM Technical Report TR 00.2503, IBM Poughkeepsie Laboratory, December 1973.

[BJ] Böhm, C. and Jacopini, G. "Flow Diagrams, Turing Machines, and Languages With Only Two Formation Rules," CACM, Vol. 19, No. 5, May 1966.

[CK] Cocke, J. and Kennedy, K., "Profitability Computations on Program Flow Graphs," IBM Research Report RC 5123, T.J. Watson Research Center, Yorktown Heights, N.Y., November 1974.

[CS] Cocke, J. and Schwartz, J.T., Programming Languages and Their Compilers, New York University, New York, 1969.

[Di] Dijkstra, E.W., "Notes on Structured Programming," in Dahl, Dijkstra and Hoare, Structured Programming, Academic Press (1972).

[EBA] Earnest, C.P., Balke, K.G. and Anderson, J., "Analysis of Graphs by Ordering of Nodes," JACM, Vol. 19, No. 1, Jan. 1972, pp. 23-42.

[FKZ] Farrow, R., Kennedy, K., and Zucconi, L., "Graph Grammars and Global Program Data Flow Analysis," to appear in Proceedings of the Seventeenth Annual IEEE Symposium on the Foundations of Computer Science, October 1976.

[FS] Friedman, D.P. and Shapiro, S.C., "A Case For While-Until," SIGPLAN Notices, Vol. 9, No. 7, July 1974.

[GW] Graham, S.L. and Wegman, M., "A Fast and Usually Linear Algorithm for Global Flow Analysis," Conf. Record of the Second ACM Symposium on Principles of Programming Languages, Palo Alto, California (Jan. 1975) pp. 22-34.

[HU1] Hecht, M.S., and Ullman, J.D., "Analysis of a Simple Algorithm for Global Data Flow Problems," Proc. ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages, Boston, Mass., Oct. 1973.

[HU2] Hecht, M.S., and Ullman, J.D., "Flow Graph Reducibility," SIAM J. Computing, Vol. 1, No. 2, pp. 188-202, June 1972.

[HU3] Hecht, M.S. and Ullman, J.D., "Characterizations of Reducible Flow Graphs," JACM, Vol. 21, No. 3, pp. 367-375, July 1974.

[J1] Jazayeri, M., "On Attribute Grammars and the Semantic Specification of Programming Languages," Ph.D. Thesis, Computing and Information Sciences Dept., Case Western Reserve University, (October 1974).

[J2] Jazayeri, M., "Live Variable Analysis, Attribute Grammars, and Program Optimization," draft, Dept. of Computer Science, University of North Carolina, Chapel Hill, N.C. March 1975.

[ka] Karpinski, R.H., "An Unstructured View of Structured Programming," SIGPLAN Notices, Vol. 9, No. 3, March 1974.

[Ke1] Kennedy, K., "A Global Flow Analysis Algorithm," International J. Computer Math., Section A, Vol. 3, pp. 5-15, Dec. 1971.

[Ke2] Kennedy, K., "A Comparison of Algorithms for Global Flow Analysis," Technical Report 476-093-1, Dept. of Mathematical Sciences, Rice University, Houston, Texas, Feb. 1974.

[Ke3] Kennedy, K., "Node Listings Applied to Data Flow Analysis," Conf. Record of the Second ACM Symposium on Principles of Programming Languages, Palo Alto, California, Jan. 1975.

[Ke4] Kennedy, K., "Use-definition Chains with Applications," Rice Technical Report 476-093-9, Dept. of Mathematical Sciences, Rice University, Houston, Texas, April 1975.

[KR] Kennedy, K. and Ramanathan, J., "A Deterministic Attribute Grammar Evaluator Based on Dynamic Sequencing," Technical Report 476-093-12, Dept. of Mathematical Sciences, Rice University, Houston, Texas, Oct. 1975.

[KW] Kennedy, K. and Warren, S.K., "Automatic Generation of Efficient Evaluators for Attribute Grammars," to appear in Conf. Record of the Third ACM Symposium on Principles of Programming Languages, Atlanta, Ga., Jan. 1976.

[KP1] Kernighan, B.W. and Plauger, P.J., The Elements of Programming Style, McGraw-Hill Book Co., New York, 1974.

[KP2] Kernighan, B.W. and Plauger, P.J., "Programming Style: Examples and Counterexamples," Computing Surveys, Vol. 6, No. 4, Dec. 1974.

[Ki] Kildall, G.A., "A Unified Approach to Global Program Optimization," Proc. ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages, Boston, Mass., Oct. 1973.

[Kn1] Knuth, Donald E., "An Empirical Study of Fortran Programs," Software Practice and Experience, Vol. 1, No. 2, 1971, pp. 105-134.

[Kn2] Knuth, Donald E., "Structured Programming with GOTO Statements," Computing Surveys, Vol. 6, No. 4, Dec. 1974.

[Kn3] Knuth, Donald E., "Semantics of Context-Free Languages," Math. Systems Theory J. 2, pp. 127-145 (1968).

[Kn4] Knuth, Donald E., "Semantics of Context-Free Languages: Correction," Math Systems Theory J 5, p. 95 (1971).

[LC] Lee, R.C.T. and Chang, S.K., "Structured Programming and Automatic Program Synthesis," SIGPLAN Notices, Vol. 9, No. 4, April 1974.

[MT] Markowsky, G. and Tarjan, R.E., "Lower Bounds on the Lengths of Node Sequences in Directed Graphs," IBM Research Report RC 5477, T.J. Watson Research Center, Yorktown Heights, N.Y., July 1975.

[Ma] Martin, J.J., "The 'Natural' Set of Basic Control Structures," SIGPLAN Notices, Vol. 8, No. 12, Dec. 1973.

[Ro] Rosen, B., "Tree Manipulating Systems and Church-Rosser Theorems," JACM, Vol. 20, No. 1, Jan. 1973.

[Sa] Sanfield, S., "The Scope of Variable Concept: the Key to Structured Programming?," SIGPLAN Notices, Vol. 9, No. 7, July 1974.

[Sc] Schaefer, M., A Mathematical Theory of Global Program Optimization, Prentice-Hall, Englewood Cliffs, N.J., 1973.

[Se] Sethi, Ravi, "Testing for the Church-Rosser Property," JACM, Vol. 21, No. 4, Oct. 1974.

[Sh] Shneiderman, B., "The Chemistry of Control Structures," SIGPLAN Notices, Vol. 9, No. 12, Dec. 1974.

[T] Tarjan, R.E., "Depth-First Search and Linear Graph Algorithms," SIAM J. Computing, Vol. 1, No. 2, June 1972.

[U] Ullman, J.D., "Fast Algorithms for the Elimination of Common Subexpressions," Acta Informatica, Vol. 2, pp. 191-213, 1973.

[We] Wegner, E., "Control Structures for Programming Languages," SIGPLAN Notices, Vol. 10, No. 2, Feb. 1975.

[WR] Weiderman, N.H. and Rawson, B.M., "Flowcharting Loops Without Cycles," SIGPLAN Notices, Vol. 10, No. 4, April 1975.

[Wi] Wirth, N., "On the Composition of Well-Structured Programs," Computing Surveys, Vol. 6, No. 4, Dec. 1974.

[WRH] Wulf, W.A., Russell, D.B. and Haberman, A.N., "BLISS: A Language for Systems Programming," CACM, Vol. 14, No. 12, Dec. 1971.

[Z] Zahn, C.T., "Structured Control in Programming Languages," Proceedings of the 1975 NCC, AFIPS Conference Proceedings, Vol. 44, AFIPS Press (1975).