1978-05

Analysis of Real-Time Systems by Data Flowgraphs

Kodres, Uno R.

IEEE Transactions on Software Engineering, Vol. SE-4, No. 3, May 1978
http://hdl.handle.net/10945/41300

Analysis of Real-Time Systems by Data Flowgraphs

UNO R. KODRES

Abstract—The concept of a data flowgraph is formalized as a bipartite directed graph. Each execution sequence of a computer program has a corresponding data flowgraph which describes functionally what happens to the data if that execution sequence is followed.

The data flowgraph gives information which is useful in analyzing parallel processing, test case preparation, error analysis, and program verification.

An attack aircraft tactical system is used to illustrate how the concept of data flowgraphs is applied to analyze real-time systems.

Index Terms—Bipartite graph, control complexity, data flowgraph, discrete systems, execution sequence, independence, parallel processes, program analysis.

I. Introduction

THE basic inadequacy in program documentation in both flowchart and programming languages form is that both forms concentrate on how a problem is being solved rather than what the problem is.

If we design a multiplier either in hardware, firmware or software, it is essential to know what the problem is. In proving correctness of programs we must first state what the problem is before we can verify that our method of solving it is correct.

There are three major ways in which we state what the problem is:

1) by the use of formulas;
2) by drawing boxes and joining them by lines;
3) by the use of decision tables.

Formulas have the advantage that we have learned to manipulate them in order to derive equivalent expressions which serve to simplify the problems. Also, formulas can directly and easily be communicated to computers if compilers are available.

Formulas have disadvantages when they extend over several lines or several pages, or when a large collection of formulas are used to describe a problem. Computer designers have used both formulas and drawings to document the designs. IBM is a major user of drawings. Design documents which describe computer hardware are documented with box/line drawings.

Box/line drawings have both advantages and disadvantages. The major disadvantages of these documents are that we must relearn to manipulate the drawings in order to simplify them and that we have no compilers for graphics. Generally human transcribers are used to translate graphic pictures into notation understandable to the computer. The computer redraws the pictures usually losing positional integrity for both boxes and lines and thus losing some possibly important information.

Manuscript received December 22, 1977; revised February 8, 1978.

The author is with the Department of Computer Science, Naval Postgraduate School, Monterey, CA 93940.

The major advantages of box/line drawings are that data flow can be explicitly shown across formulas, information is not constrained to lines (one-dimensional) but can easily extend into pages (two-dimensional), and levels of abstraction are conveniently achieved by functionally labeling boxes and lines.

This paper formalizes the concept of box/line drawings. Such drawings form a bipartite directed graph, bi-digraph, which we shall also call data flowgraphs.

A program flowchart also corresponds to a graph construct called a flowgraph. To each execution sequence in the flowgraph corresponds a data flowgraph which describes at a chosen level of abstraction what happens to the data.

A similar data flow analysis is carried out by Allen and Cocke [1], although both their control flow and data flow are differently conceived and applied.

A control complexity analysis which makes use of a control flowgraph and introduces a complexity measure, the cyclomatic number of the control flowgraph, by McCabe [9], has some similarities to our concept of the flowgraph. The flowgraph in this paper is abstractly the same as graphs used in circuit theory, discrete systems analysis, and cost-oriented network flow problems.

Shneiderman *et al.* [10] showed experimentally that flowcharting has little value in increasing programmer productivity. The value we see in flowcharts or flowgraphs is related to automated computer analysis of algorithms, both the execution time analysis and the data flow analysis.

In Section II the flowgraph corresponding to the flowchart is defined and shown to be abstractly identical to similar graphs encountered in discrete systems analysis. In Section III the data flowgraph corresponding to an execution sequence in the flowgraph is defined. Section IV illustrates the usefulness of the concept by applying it to the analysis of an airborne real-time tactical system (A6-E). Section V is a summary of what the paper contains.

II. Flowcharts and Flowgraphs

The language of flowcharts has been intimately tied to computer programming from the very beginning. However, there is no universal agreement about what a flowchart represents. Much of the literature in computer science conceives of flowcharts as composed of actions, represented by vertices of a directed graph, and possible successor actions, represented

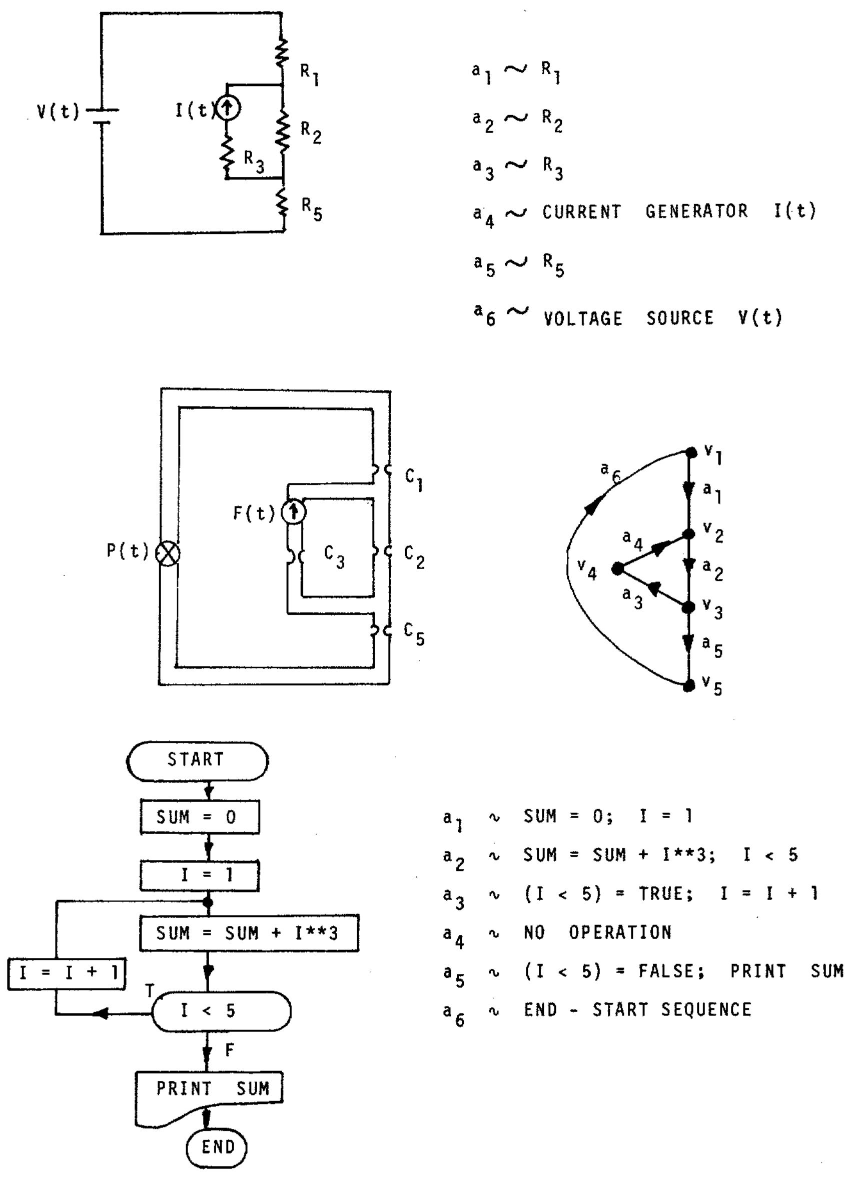


Fig. 1. Abstract similarity of three problems.

by vertices, which are joined by arcs indicating that for the given action there is one or more possible successor actions.

An alternate conception of flowcharts associates vertices with control points in the flowchart and arcs with actions and control transitions. This view is abstractly identical to discrete systems analysis of two terminal elements [6]. This view also is abstractly identical to network flow problems in which unit transportation costs are associated with arcs [4]. It is this view of flowcharts which we shall use in what follows. For further information and a formal presentation of this view of flowcharts, see [7]. Fig. 1 illustrates the abstract similarity of an electrical network, a hydraulic network, and a flowchart. All three problems are representable by the directed graph. The dual set of variables in electrical networks are currents and voltages; in hydraulics, flows and pressures; in flowcharts, number of executions of a program sequence and total execution time of that sequence.

The execution time analysis of a program can be carried out in a similar way in which electrical network problems are solved. The system of equations solved reduces to a linear system of ordinary equations instead of a linear system of differential equations which is typical of electrical discrete systems.

III. Data Flowgraphs

Although the flowchart analysis gives us an effective way of determining execution times and the complexity of programs, it does not give much information on the independence of processes or how processes can be executed simultaneously without interfering with each other. The concept of data flowgraphs helps to graphically display what happens to data, how data are transformed, and how one can partition the process into subprocesses with a minimal need of data transfers.

We illustrate the ideas first before we formalize the concepts. Referring back to Fig. 1, consider the graph consisting of arcs a_i and vertices v_i. Each arc of this graph corresponds to an instruction sequence which carries out a computation on some input data. For example, arc a_1 corresponds to a typical

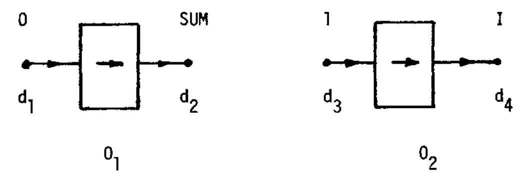


Fig. 2. Two components of the graph corresponding to the flowgraph arc a_1.

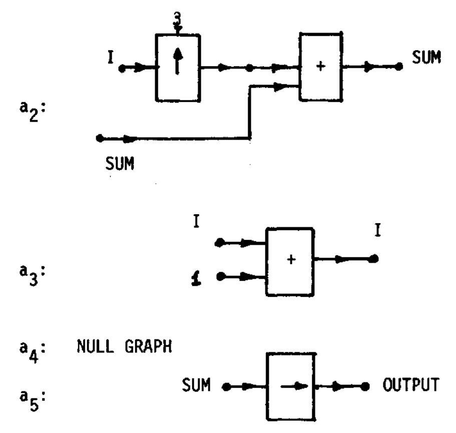


Fig. 3. Components of the data graph.

instruction sequence

```
LDA 0

STA SUM

LDA 1

STA I
```

We associate with each data item a vertex d_i (Fig. 2) and with each operation another vertex O_j, in a manner used by digital circuit designers ever since computers were first designed and manufactured. We connect the data item to the operation which uses the data item as an input by an arc directed into the operation vertex. The output data item or items are connected to the operation by arcs directed away from the operation vertex. The graphs resulting from carrying out this association with flowgraph arcs are graphs which contain two types of vertices where arcs connect data vertices to operation vertices and where operation vertices in turn are only connected to data vertices. Such graphs (Fig. 3) are known as bipartite directed graphs, or bi-digraphs [8].

For each arc in the flowgraph we construct a corresponding bi-digraph. We note that control operations such as branch or halt instructions do not alter any data and hence do not require a correspondent bi-digraph.

We next determine an execution sequence or all execution sequences for a given flowgraph.

We are now ready to formalize the definition of a data flowgraph.

Definition 1: A data flowgraph corresponds to an execution sequence in a flowgraph as follows. Let

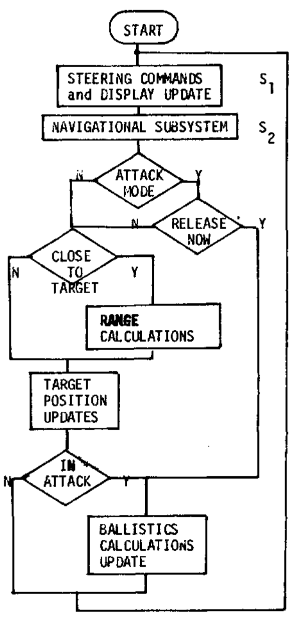
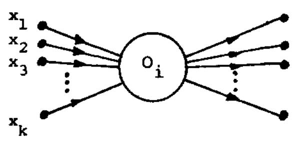


Fig. 4. A6-E tactical program flowchart.

$$S = (a_1, a_2, \cdots, a_n)$$

be an execution sequence in a flowgraph. To each arc a_i, there corresponds a mapping of input variables $x_1, x_2, \cdots, x_k$ into output variables $y_1, y_2, \cdots, y_m$. This functional relationship is denoted by some operation O_i and indicated as a subgraph



If one or more of the input variables of O_i is identical to an output variable of some other operation O_j, then we identify such variables by the same vertex in the data flowgraph. Similarly, if there is a common input or output variable to one or more operations O_j, we identify such variables by the same vertex in the data flowgraph. If this procedure is successively carried out for each arc in the execution sequence S, the resulting graph is a data flowgraph of S.

We have given so far very simple examples to illustrate the idea of a data flowgraph. It is easy to see that in a complex program there are large numbers of execution sequences and hence data flowgraphs. Therefore this method of analysis would become so complex that it becomes useless.

Fortunately the idea of a data flowgraph lends itself very naturally to levels of abstraction. We can illustrate this with the example which comes from the A6-E tactical system.

IV. An Illustrative Real-Time System

A. Data Flowgraph Construction

An attack aircraft (A6-E) tactical system is used to illustrate the flowchart and data flowgraph analysis techniques.

Fig. 4 illustrates the top level flowchart which describes the systems operation. After a hardware checkout and initialization programs have been executed, the program goes into the infinite loop described by the flowchart. In this system

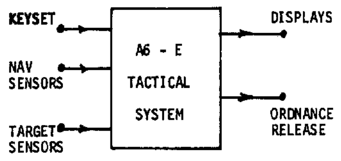


Fig. 5. Data flowgraph of the A6-E tactical system.

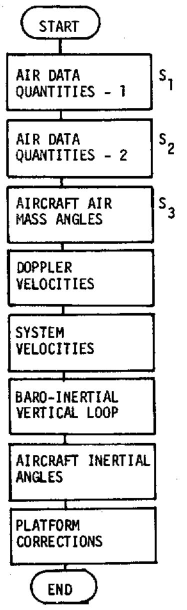


Fig. 6. Flowchart of the navigational subsystem.

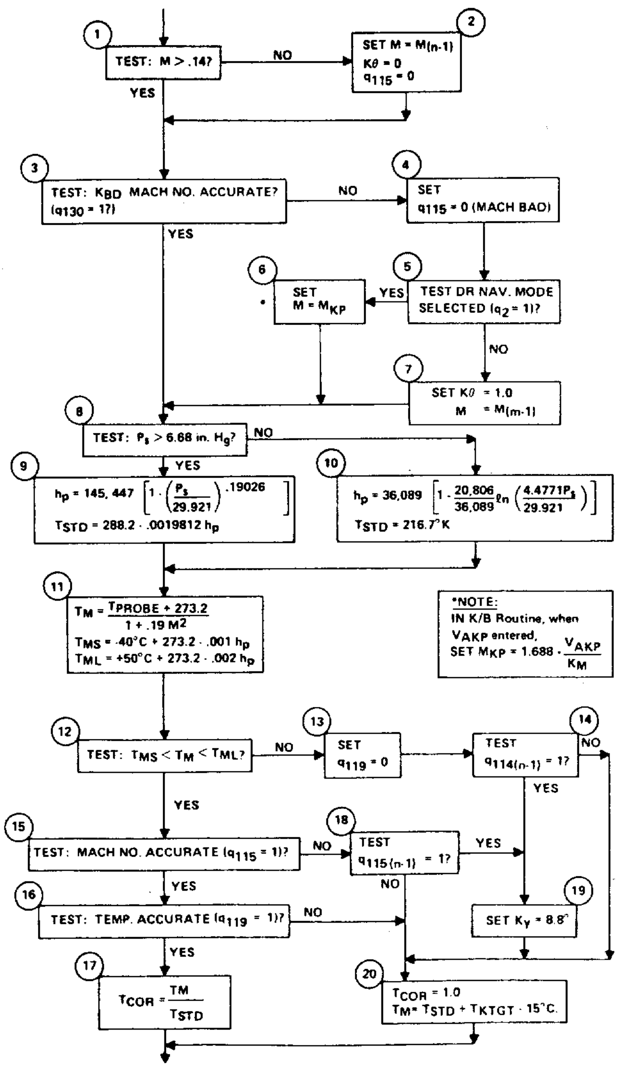

TEST: M > .14?
NO
YES
SET M = M(n-1) Kθ = 0 q115 = 0
TEST: KBD MACH NO. ACCURATE? (q130 = 1?)
SET q115 = 0 (MACH BAD)
TEST DR NAV. MODE SELECTED (q2 = 1)?
SET M = MKP
SET Kθ = 1.0 M = M(m-1)
TEST: Ps > 6.68 in. Hg?
hp = 145, 447 [1 - (Ps/29.921).19026]
TSTD = 288.2 - .0019812 hp
hp = 36,089 [1 - 20,806/36,089 ln (4.4771Ps/29.921)]
TSTD = 216.7°K
TM = (TPROBE + 273.2)/(1 + .19 M2)
TMS = -40°C + 273.2 - .001 hp
TML = +50°C + 273.2 - .002 hp
*NOTE: IN K/B Routine, when VAKP entered, SET MKP = 1.688 · VAKP/KM
TEST: TMS < TM < TML?
SET q119 = 0
TEST q114(n-1) = 1?
TEST: MACH NO. ACCURATE (q115 = 1)?
TEST q115(n-1) = 1?
TEST: TEMP. ACCURATE (q119 = 1)?
SET Ky = 8.8°
TCOR = TM/TSTD
TCOR = 1.0 TM = TSTD + TKTGT - 15°C.

Fig. 7. Flowchart of air data quantities-1.

the executive program is very simple: a sequence of tasks is executed without a set of priorities. The task is bypassed whenever there is no need to execute it. The analog inputs from the sensors are sampled periodically based on a real-time system's clock.

On execution of the infinite loop, that is, the transition from the control point above "Steering Commands · · ·" to the control point below "Ballistics Calculations · · ·" in Fig. 4, may be regarded either as a set of all possible execution sequences, or as a single transition of control. If we regard it as a single transition of control, then we can represent what happens to the data with a single data flowgraph as shown in Fig. 5. This representation corresponds to the overall view of what the program does. Each data set vertex represents data items which serve as inputs or outputs of the system, whereas the operation carried out by the system is represented by a single vertex.

If we wish to consider a more detailed view of the operation under the same transition of control, then each distinct execution sequence gives rise to a data flowgraph and the set of all such data flowgraphs describe in more detail what the single flowgraph expresses in Fig. 5.

The Navigational Subsystem consists of eight sequential steps in Fig. 6. The first step is entitled Air Data Quantities-1 in Fig. 7. This flowchart gives the finest level of detail and enables a programmer to proceed directly to writing a higher level language or an assembly language program.

Corresponding to the flowchart in Fig. 7 we construct a flowgraph in Fig. 8. In the flowgraph we have distinguished between the vertices from which exactly one arc issues and the vertices from which more than one arc issues. The latter vertices are symbolized by a small diamond indicating that several execution sequences emanate from that vertex.

We also distinguish between two types of arcs: one (symbolized by a square) corresponds to a statement which permanently alters a data value; the other (symbolized by an arrow) corresponds to a transition in control which does not permanently alter any data values.

The heavy arrows constitute a spanning tree of the flowgraph which is of significance in execution time analysis as well as control complexity analysis.

Further simplification of the analysis can be obtained by subdividing the flowgraph into so-called control segments, which are segments of a program with a single entry and single exit. The control segment from v_0 to v_5 in Fig. 8 is shown in the flowchart form in Fig. 9, and in the flowgraph form in Fig. 10(a).

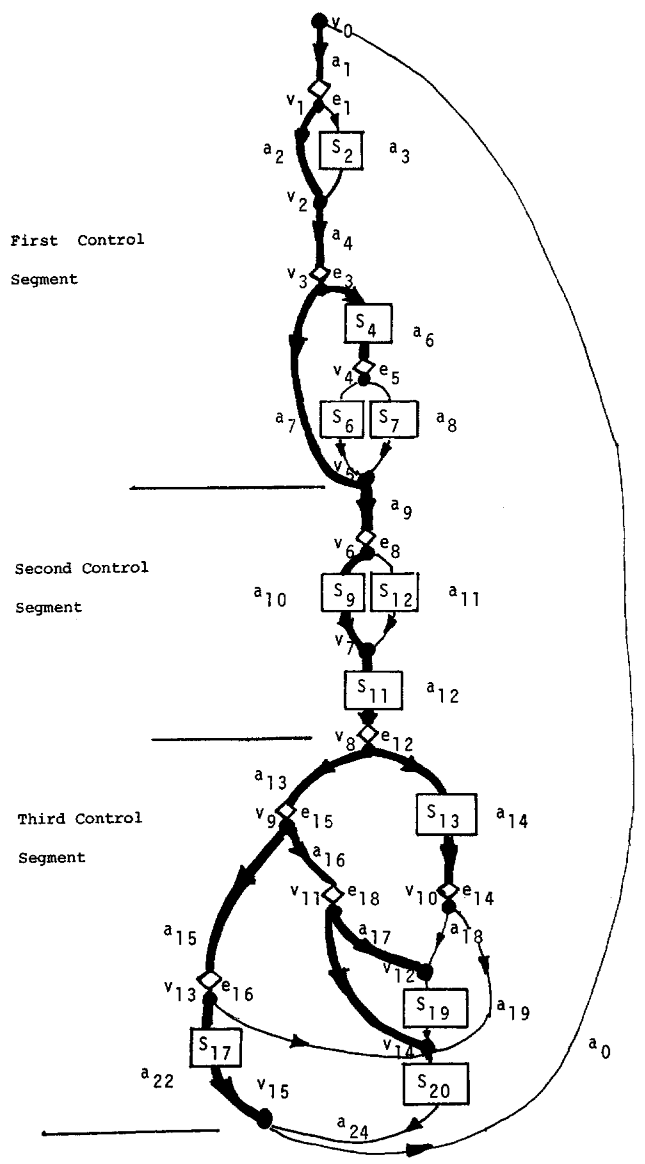


Fig. 8. Flowgraph of air data quantities-1.

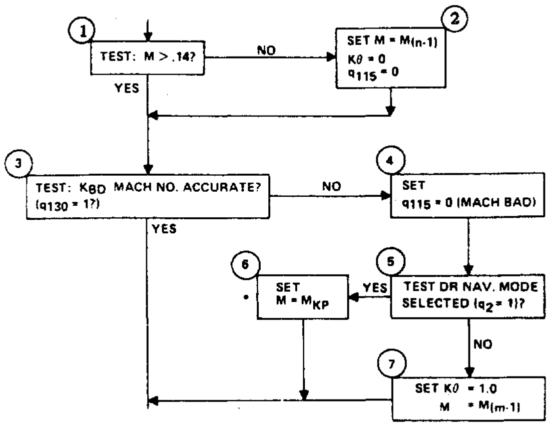


Fig. 9. First control segment of air data quantities-1.

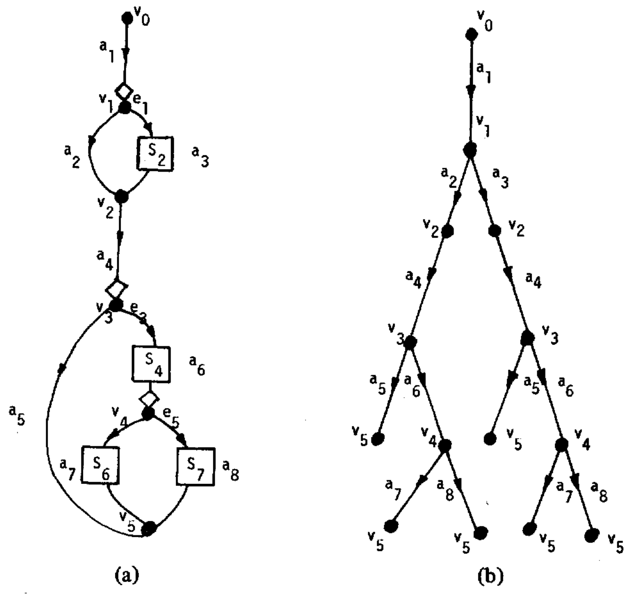


Fig. 10. Flowgraph and execution sequence tree.

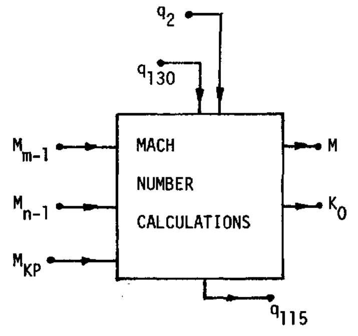


Fig. 11. Data flowgraph of control segment in Fig. 9.

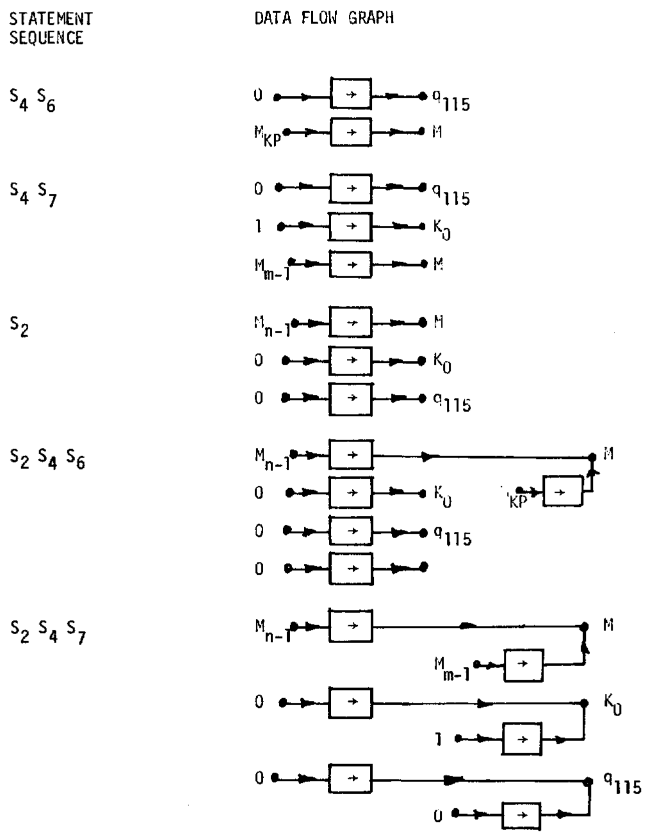


Fig. 12. Data flowgraphs corresponding to five statement sequences.

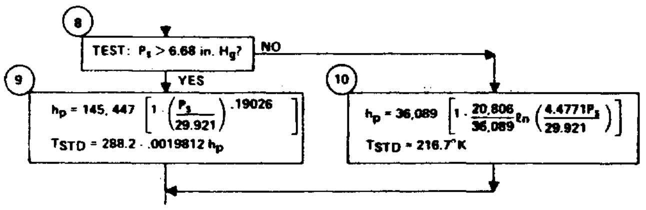

TEST: $P_s > 6.68$ in. H_g?
NO
YES
$h_p = 145,447\left[1 - \left(\frac{P_s}{29.921}\right)^{.19026}\right]$
$T_{STD} = 288.2 - .0019812\, h_p$
$h_p = 36,089\left[1 - \frac{20,806}{36,089}\ln\left(\frac{4.4771 P_s}{29.921}\right)\right]$
$T_{STD} = 216.7°K$

Fig. 13. Flowchart of second control segment.

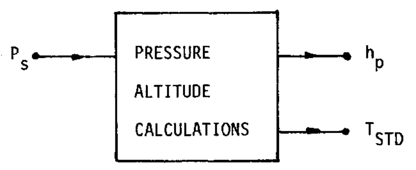


Fig. 14. Overview data flowgraph of Fig. 13.

As before, we can generate a data flowgraph which describes an overview of what takes place in the control segment, as shown in Fig. 11. If we are interested in more detail, then we look at all possible execution sequences which are described as a rooted tree in Fig. 10(b). Corresponding to the six possible execution sequences, there are five distinct statement sequences and their corresponding data flowgraphs in Fig. 12.

The control segment v_5 to v_7 can similarly be described in flowchart form in Fig. 13, as an overview in Fig. 14, or in complete detail as in Fig. 15(a) and 15(b).

The control segment from v_7 to v_{15} is shown in flowchart form in Fig. 16 and as an overall data flowgraph in Fig. 17. The overall view of the entire page and how the control segments fit together is displayed in Fig. 18.

If a program contains a loop, then the corresponding flowgraph is a repetition of flowgraphs each of which describes the data flow for a single execution sequence.

We wish to note that in the A6-E tactical program there are about sixty pages of flowcharts—some less complex, some more complex. The page used to illustrate the data flowgraph concept is thus a small but sufficiently complex example to illustrate the usefulness in:

1) analyzing parallel processing,
2) test case preparation,
3) error analysis,
4) program verification.

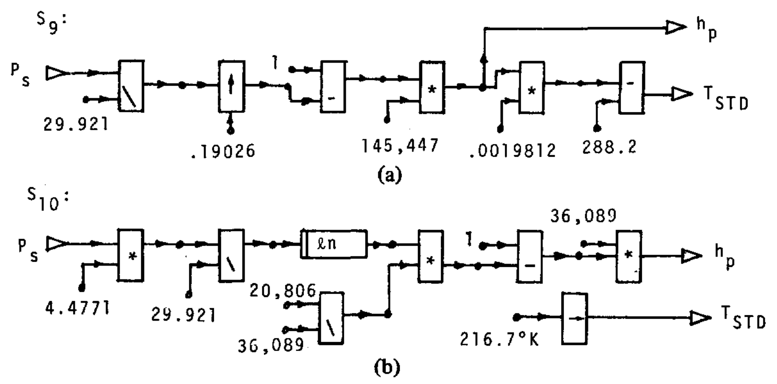
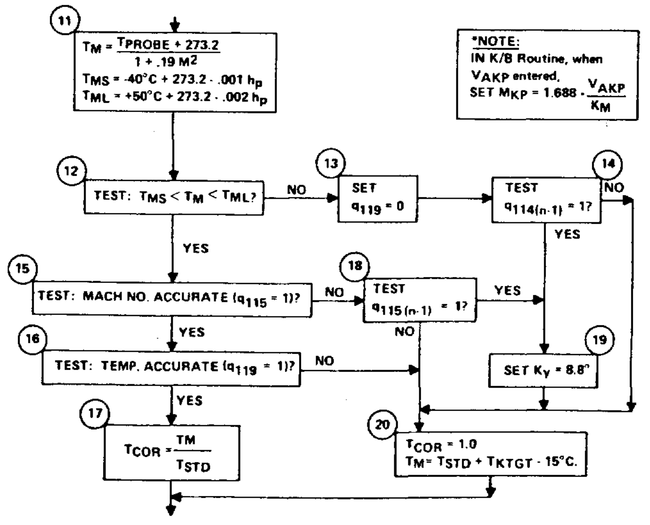


Fig. 15. Detailed data flowgraph of Fig. 13.



11
TM = (TPROBE + 273.2) / (1 + .19 M2)
TMS = -40°C + 273.2 - .001 hp
TML = +50°C + 273.2 - .002 hp
*NOTE:
IN K/B Routine, when VAKP entered, SET MKP = 1.688 · VAKP / KM
12
TEST: TMS < TM < TML?
NO
13
SET q119 = 0
14
TEST q114(n-1) = 1?
NO
YES
15
TEST: MACH NO. ACCURATE (q115 = 1)?
NO
18
TEST q115(n-1) = 1?
YES
NO
16
TEST: TEMP. ACCURATE (q119 = 1)?
NO
19
SET Ky = 8.8°
17
TCOR = TM / TSTD
20
TCOR = 1.0
TM = TSTD + TKTGT - 15°C.

Fig. 16. Flowchart of control segment three.

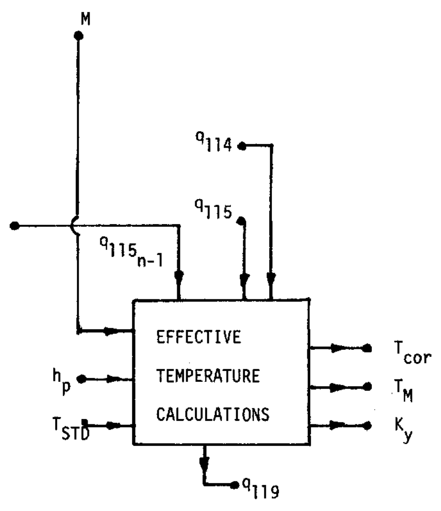


Fig. 17. Overview data flowgraph of Fig. 16.

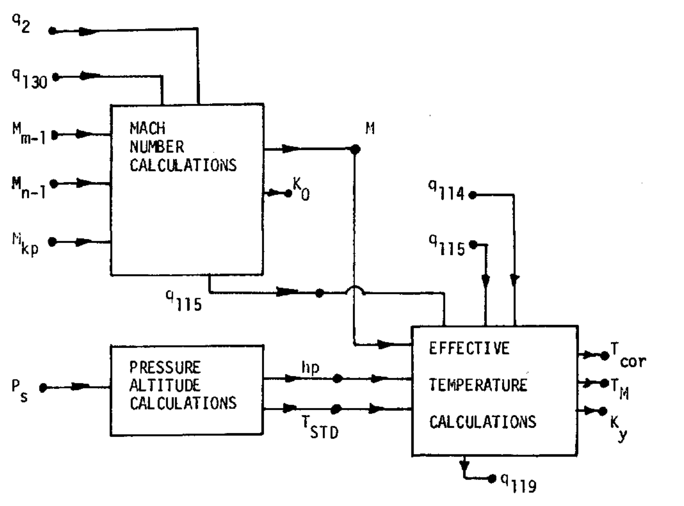


Fig. 18. Data flowgraph describing three control segments of air data quantities-1.

B. Analysis of Parallel Processing

The main concerns in distributing a process among several computers is the resulting inefficiencies introduced by the distribution:

1) greater communication problems between computers;

2) duplication of data and programs;

3) imbalance in execution times and program size among the processors.

The data flowgraphs allow us to explicitly determine the number of data elements which must be communicated to other processes.

In Fig. 18 the data flowgraph shows that the Pressure Altitude Calculations have no data values in common with the Mach Number Calculations. We can conclude that these processes could be carried out in different computers completely independently without creating communications problems. The same graph also shows that the Effective Temperature Calculations use two data values generated by Mach Number Calculations and two data values from Pressure Altitude Calculations.

If a single computer must solve the problem, then eight data values are used as inputs and five values are used as outputs. If two computers are to solve the problem, and if Mach Number Calculations are done in one computer and the Pressure Altitude and Effective Temperature Calculations are done in the other, then the communication problems are increased by the two data values which must be communicated. Effective Temperature Calculations may not be started after Mach Number Calculations are ready. This causes timing problems in addition to communication problems; however, data flowgraphs allow the analysis of both problems.

Data flowgraphs reduce the distributed system design problem to a graph partitioning problem with side constraints. Several effective algorithms exist for solving that problem. Reference [2] reviews the published literature on graph partitioning. The side constraints may be taken to be program size and maximal execution time. The problem then becomes: partition the graph into a minimal number of partition elements so that the number of arcs cut is minimized, and the bounds on program size and maximal execution times are satisfied. The problem is completely analogous to the hardware partitioning problem for creating mother-boards in computer design. The mother-board design problem had to satisfy input–output constraints which permitted only a certain maximum number of input–output connections for each mother-board, as well as a certain maximum number of daughter-cards on each mother-board.

C. Test Case Preparation

As all of us who write computer programs know, a program is seldom correct when first executed. Therefore it is safe to assume that programs contain errors, and we would like to generate test cases which are:

1) exhaustive enough to discover all errors;

2) small enough in number so that we can effectively test the cases;

3) suggestive of what must be done to make corrections.

Computer programming is in many ways analogous to a lengthy derivation in solving calculus problems. The way in which one gains confidence in a lengthy derivation is by looking in the answer section to see if the result obtained agrees with the one given in the book. If the results agree, then we usually are satisfied. If not, then we try to establish equivalence. Failing that, we check algebraic manipulations, signs, differentiations, integrations, etc., until an error is discovered. If we work on an even-numbered problem which does not have an answer in the back of the book, we resort to different schemes. If we trust a friend we call him to find out what result he obtained. If the results do not agree, we check back to see when the results first disagreed.

In programming we use similar techniques. We write a program in some higher level language such as Fortran. If we write a special-purpose routine to evaluate a trigonometric function, we compare our results with the corresponding library routine. How many test values do we use? If we have a polynomial of degree n, then theory tells us that $n + 1$ points are sufficient to establish identity. By using a random number generator to generate a set of $n + 1$ random numbers in the interval of interest and by finding that the results agree with sufficient accuracy, we become confident that no further errors exist. We remain confident until someone discovers that for a particular case the results are incorrect. We fix the program and add these particular cases to our repertoire of test cases. Usually the endpoints of finite intervals, zero value, or undetected underflows and overflows are a cause of trouble. How does the data flowgraph concept help us generate test cases?

We observed in the previous segment that the control segment v_0 to v_5 had six different execution sequences and five different statement sequences. Each statement sequence gives rise to a different function. Therefore we must prepare at least five different data sets, one for each function. As shown in Fig. 12 each function is a constant function, and, hence, theoretically, one data value for each data set would be sufficient to check identity to a previously computed constant function.

If one considers the flowgraph in Fig. 8 in its entirety, then the total number of execution sequences between v_0 and v_{15} is 72. The number of execution sequences may be calculated by a vertex labeling process which labels each vertex by the number of distinct access paths from the entry point.

Because several execution sequences give rise to the same statement sequence, only 50 distinct statement sequences exist. If each statement sequence yielded a unique function, we would have to construct at least 50 different data sets, each data set containing at least one set of values. In the A6-E tactical program there are 60 pages of flowcharts of control complexity with about an average of 20 statement sequences each. Therefore, there are approximately $(20)^{60}$ statement sequences in the entire program. If each statement sequence gave rise to a different function, we would have to generate $(20)^{60}$ data sets, each containing at least one set of data values. Clearly testing the program would become impossible.

The data flowgraph corresponding to the control segment (i.e., a program segment with a single entry and a single exit)

v_0 to v_5 in Fig. 11 is disconnected from the data flowgraph for the next control segment. This means that the two functions are independent and can be tested independently from each other.

Theoretically, the first control segment requires five data sets, the second two data sets. Altogether 5 + 2 data sets are necessary instead of $5 \cdot 2 = 10$.

The third control segment is a rational function containing five statement sequences. Hence five data sets would need to be constructed to test out that function. In order that two rational functions be identical, it is sufficient to form the product polynomial.

If

$$R_1 Q_2 \equiv Q_1 R_2$$

then

$$\frac{R_1}{R_1} \equiv \frac{Q_1}{Q_2}$$

for all x for which $R_2(x) \neq 0$ and $Q_2(x) \neq 0$.

In the above example, four data values per data set would be sufficient. Thus $5 + 2 + 5 = 12$ data sets are sufficient to test the identity of the functions calculated in the flowgraph in Fig. 8 instead of the 50 data sets estimated on the basis of the statement sequences in the flowgraph.

The flowchart page used in this example has average complexity. If we assume that the complexity of each page is on the average similar to this page and the data flowgraphs exhibit the same degree of independence as they do on this page, then only

$$60 \cdot 12 = 720$$

data sets need to be constructed instead of $(20)^{60}$. Clearly 720 data sets would be a reasonable number to construct even if each data set contained ten sets of data values.

D. Error Analysis

The data flowgraphs lend themselves to numerical error analysis. McCracken and Dorn [3] present an elementary but complete treatment of roundoff error propagation for both fixed-point and floating-point arithmetic. They use data flowgraphs (or process graphs in their terminology) to develop the error-bound estimates for each function. We shall not repeat their presentation here but refer the reader to their book.

In addition to numerical error-bound analysis, data flowgraphs give strong indications of possible trouble spots or errors in the program. For example, in the statement sequence $S_2\ S_4\ S_7$ in Fig. 12, variable M is set to M_{n-1} in S_2 and reset to M_{m-1} in $S_4\ S_7$. Although the function may be correctly calculated, it is done clearly in an inefficient fashion. It is not hard to reconstruct the program to eliminate this inefficiency and still compute the same function.

E. Program Verification

For real time tactical systems, program verification is particularly difficult to accomplish. In order to verify that a program does what is intended, there must be an unambiguous statement of intention. Frequently the statement of intention is in a flowchart form. Because flowcharts imply a sequence of statements and a sequence of controls, the statement of intention not only specifies what is wanted, but also in what sequence it should occur. This overspecifies the program, causes inefficiency, and removes useful flexibility.

Program verification, as described by Hantler and King [5], consists of:

1) stating formally what assumptions are made about input variables before entry into a procedure;

2) asserting algebraically or by logical statements what the output variables will contain when the procedure is completed;

3) symbolically executing the program as described in some programming language to show that the results agree with the specification.

The data flowgraph is a graphic expression of one or more algebraic statements. Therefore algebraic statements can be interpreted into data flowgraphs. This allows us to construct a data flowgraph or a set of data flowgraphs for the specification statement.

The symbolic execution of a program is equivalent to the process of constructing a data flowgraph for each execution sequence in the program. The verification process is the process of checking whether or not the data flowgraphs corresponding to the specification are equivalent to the data flowgraphs obtained from the execution sequences.

An automated process of showing equivalence is by no means an easy process to construct. Conceptually, however, data flowgraphs will enhance program verification.

V. Summary

This paper introduces the data flowgraph concept as a useful tool in the analysis of real-time systems. Although there are more applications, this paper has focused attention on four applications which are particularly useful in systems analysis:

1) process partitioning into subprocesses for distributed processing;

2) generating test cases to establish that the program under design is identical to a known, correctly functioning test program;

3) numerical error analysis and the analysis of inefficiencies occurring in the program;

4) program verification.

The data flowgraph is useful conceptually as well as in the practical domain of implementation of algorithms to carry out the analysis tasks.

References

[1] F. E. Allen and J. Cocke, "A program data flow analysis procedure," *Commun. Ass. Comput. Mach.*, vol. 19, pp. 137-147, Mar. 1976.

[2] M. A. Breuer, Ed., *Design Automation of Digital Systems: Theory and Techniques*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

[3] D. D. McCracken and W. S. Dorn, *Numerical Methods and Fortran Programming*. New York: Wiley, 1964.

[4] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.

[5] S. L. Hantler and J. C. King, "An introduction to proving cor-

rectness of programs," *Computing Surveys*, vol. 8, pp. 331-353, Sept. 1976.
[6] H. E. Koenig, Y. Tokad, and H. K. Kesavan, *Analysis of Discrete Physical Systems*. New York: McGraw-Hill, 1967.
[7] U. R. Kodres, "Discrete systems and flowcharts," *IEEE Trans. Software Eng.*, to be published.
[8] —, "Logic circuit layout," in *Proc. Joint Conf. on Mathematical and Computer Aids to Design*, Oct. 1969, pp. 165-191.
[9] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. SE-2, pp. 308-320, Dec. 1976.
[10] B. Shneiderman, R. Mayer, J. McKay, and P. Heller, "Experimental investigation of the utility of detailed flowcharts in programming," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 373-381, June 1977.

Uno R. Kodres received the B. A. degree in 1954 from Wartbug College, Waverly, IA, and the M.A. and Ph.D. degrees from Iowa State University, Ames, in 1956 and 1958, respectively.

While at Iowa State University he had a teaching fellowship in mathematics. He joined IBM in 1958 on the Design Automation Project at Poughkeepsie, NY. In 1963 he joined the Naval Postgraduate School, Monterey, CA, where he is presently an Associate Professor in the Department of Computer Science. In 1967 he was a consultant to IBM in Menlo Park on the Advanced Computational Systems Project. Most recently his interest is in homogeneous, distributed computer architectures making use of large-scale integrated technology. He is a coauthor of a book on design automation, has lectured at the UCLA sponsored short courses, and was invited as a Fulbright Scholar to Yugoslavia to lecture on automated design.

Dr. Kodres is a member of the Association for Computing Machinery, the Society for Industrial and Applied Mathematics, the Mathematical Association of America, and Sigma Xi.

Testing Software Design Modeled by Finite-State Machines

TSUN S. CHOW

Abstract—We propose a method of testing the correctness of control structures that can be modeled by a finite-state machine. Test results derived from the design are evaluated against the specification. No "executable" prototype is required. The method is based on a result in automata theory and can be applied to software testing. Its error-detecting capability is compared with that of other approaches. Application experience is summarized.

Index Terms—Control structure, finite-state machines, reliability, software testing, test covers, validity.

Introduction

ONE of the most difficult problems in testing is finding a test data selection strategy that is both valid and reliable [4]. The general problem is not solvable [13]. However, by restricting ourselves to look at the "control structure" of software systems only, and only those that can be modeled by a finite-state machine, we do find a testing strategy that is both valid and reliable.

This paper presents a new testing strategy that we call "automata theoretic." It has the following characteristics:

1) only the control structure of the design is checked;
2) it does not require an "executable" specification;
3) test sequences are guaranteed to reveal any errors in the control structure, provided that some reasonable assumptions are satisfied.

As far as testing the correctness of the control structure in a design is concerned, our method is superior to those existing testing strategies that are based on the structure of the design. By means of examples, we will show that there are errors that are detected by our method, while going undetected by other testing strategies.

In addition, we will define a new hierarchy consisting of "*n*-switch set covers," a generalization of the modified "switch cover." Although we prefer the automata theoretic method to *n*-switch set covers, we are able to specify *analytically* the classes of errors that can be detected by an *n*-switch set cover. Up until now, the best one could do was to obtain *empirical* data concerning the reliability of a testing strategy [13]. Of course, we remind the reader that here we are only dealing with the verification of the control structures at the design level, and only those that can be modeled by finite-state machines.

Next, our discussion proceeds to an examination of the

Manuscript received December 7, 1977; revised February 3, 1978.
The author is with Bell Laboratories, Naperville, IL 60540.