The Analysis of Structured Programs—Part I: Kirchhoff's Equations

R. J. McEliece
Communications Systems Research Section

In this article we show that the analysis of a structured computer program is in some ways much easier than the analysis of an unstructured one. This is in contrast to the usual argument, which is that the synthesis of a structured program is easier than the synthesis of an unstructured one.

I. Introduction

It is often necessary to make a careful analysis of an existing computer program—for example, in order to obtain precise timing information. In this article we will show that if the program is *structured* (in a sense to be made precise in Section II), an important part of this analysis can be done by merely inspecting the program documentation; but if the program is not structured, the analysis is much harder.

Let us begin with an example.[1] Figure 1, which is taken from Tausworthe (Ref. 2, pp. 5–33), is the flowchart for a simple *unstructured* program.

Each box in Fig. 1 represents a program step, and the letter inside the box represents the number of times the corresponding step is performed during one run of the program. While of course these numbers in general depend on the details of the program and the program's

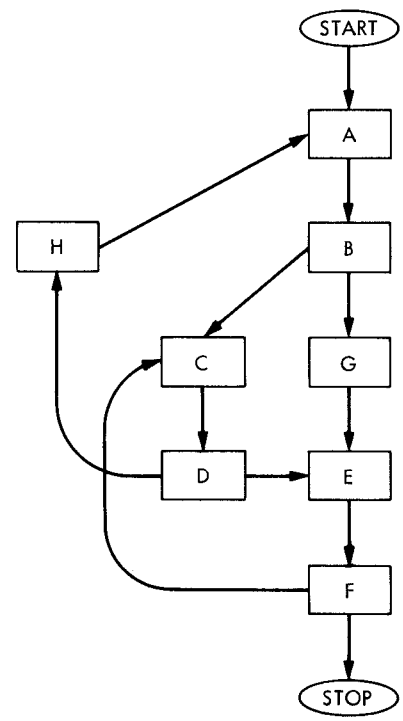


Fig. 1. An unstructured flowchart

[1]The discussion which follows is largely taken from Knuth (Ref. 1, pp. 364–369).

input, there is a considerable amount that can be learned directly from the flowchart. To glean this information, let us draw the flowchart in a more abstract form (Fig. 2), in which the branches are labeled $e_1, e_2, \cdots, e_{12}$.

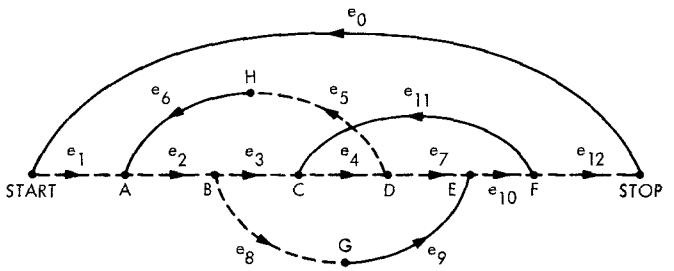


Fig. 2. Flowchart with branches labeled

Let E_j (called a *branch variable*) denote the number of times branch e_j is taken during the program run. Then since the number of times each program step is entered is the same as the number of times it is exited, for each program step we get a linear equation involving the branch variables. These are called Kirchhoff's equations, since if we were considering electrical networks instead of program flowcharts, and if E_j represented the current along the edge e_j, Kirchhoff's current law would yield the same set of equations. Thus we obtain 8 equations in the 12 branch variables:

$$E_1 + E_6 = A = E_2$$

$$E_2 = B = E_3 + E_8$$

$$E_3 + E_{11} = C = E_4$$

$$E_4 = D = E_5 + E_7$$

$$E_7 + E_9 = E = E_{10}$$

$$E_{10} = F = E_{11} + E_{12}$$

$$E_8 = G = E_9$$

$$E_5 = H = E_6.$$

The fictitious edge e_0 has been added so that Kirchhoff's laws will also hold for the START and STOP nodes; i.e.,

$$E_0 = E_1 \qquad \text{(START)}$$

$$E_{12} = E_0 \qquad \text{(STOP)}$$

Since START and STOP are executed exactly once in each program run, we have the boundary condition $E_0 = 1$. From now on, we will focus our attention on the branch variables $E_0, \cdots, E_{12}$ rather than the step variables $A, \cdots, H$.

At this point we could apply classical linear algebra and "solve" our system of equations, but there is another technique which can be used and which takes advantage of the combinatorial structure of the flowchart.

The first step in this procedure is to find a *free subtree* in the flowchart graph. A free subtree is a connected subgraph which contains each node and has no cycles. In our case the dotted branches e_1, e_2, e_3, e_4, e_5, e_7, e_8, e_{10}, e_{12} of Fig. 2 form a free subtree. The remaining branches, e_0, e_6, e_9, e_{11} are called *fundamental branches*. If we add one of the fundamental branches to the free subtree, the resulting graph contains a unique cycle, called a *fundamental cycle*. For example, the fundamental cycle containing e_9 *is* e_9, e_7, e_4, e_3, e_8. However, in this cycle the branches e_7, e_4, e_3 must be traversed in the direction opposite the arrows, so we denote this cycle by $e_9 - e_7 - e_4 - e_3 + e_8$. In this way we list the four fundamental cycles of our flowchart:

$$C_0: + e_0 + e_1 + e_2 + e_3 + e_4 + e_7 + e_{10} + e_{12}$$

$$C_6: + e_2 + e_3 + e_4 + e_5 + e_6$$

$$C_9: - e_3 - e_4 - e_7 - e_8 - e_9$$

$$C_{11}: + e_4 + e_7 + e_{10} + e_{11}$$

The matrix of coefficients of these cycles contains the same information in more compact form:

$$\mathbf{C} = \begin{array}{c} \\ 0 \\ 6 \\ 9 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

It turns out (see Knuth, Ref. 1, p. 366) that the rows of **C** form a basis for the set of solutions to Kirchhoff's equations; i.e., a vector $\mathbf{E} = [E_0, E_1, \cdots, E_{12}]$ satisfies Kirchhoff's equations if and only if **E** is a linear combination of the rows of **C**. Consequently, the four branch variables E_0, E_6, E_9, E_{11}, can be taken as independent, and the remaining 9 can be expressed in terms of them simply by reading off the appropriate columns of **C**:

$$E_1 = E_0$$

$$E_2 = E_0 + E_6$$

$$E_3 = E_0 + E_6 - E_9$$

$$E_4 = E_0 + E_6 - E_9$$

$$E_5 = E_6$$

$$E_7 = E_0 - E_9 + E_{11}$$

$$E_8 = +E_9$$

$$E_{10} = E_0$$

$$E_{12} = E_0$$

Finally, we use our "boundary condition" $E_0 = 1$ and obtain the following values for our original set of unknowns $A, B, \cdots, H$:

$$A = 1 + E_6$$

$$B = 1 + E_6$$

$$C = 1 + E_6 - E_9$$

$$D = 1 + E_6 - E_9$$

$$E = 1 + E_{11}$$

$$F = 1 + E_{11}$$

$$G = +E_9$$

$$H = +E_6$$

This set of equations expresses the number of times each program step is executed in terms of the number of times the three fundamental branches e_6, e_9, e_{11} are traversed. This is as far as Kirchhoff's equations can take us, and to proceed further it is necessary to know more about the program itself. Our point is merely that solving Kirchhoff's equations for an arbitrary (unstructured) flowchart requires a considerable amount of work: the fastest known algorithm for finding a set of fundamental cycles for a directed graph requires $0(n^\gamma)$ arithmetical operations, where n is the number of nodes and γ is an integer between 2 and 3 which depends on the "fine structure" of the graph (Ref. 3, pp. 280–284). Thus the "worst case" performance of this algorithm is $0(n^3)$, which is the same as if we had used ordinary Gaussian elimination on Kirchhoff's equations.

In the next section we show that for structured programs, the situation is much simpler.

II. Structured Programs

In a computer program, it is often helpful to have certain program steps called *subprograms*, and to give details of these subprograms elsewhere. For example, a program flowchart could look like this (Fig. 3):

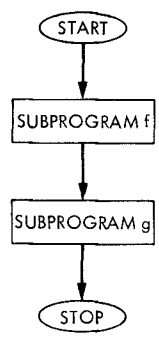


Fig. 3. Program flowchart

and the subprogram flowcharts might look like this (Fig. 4):

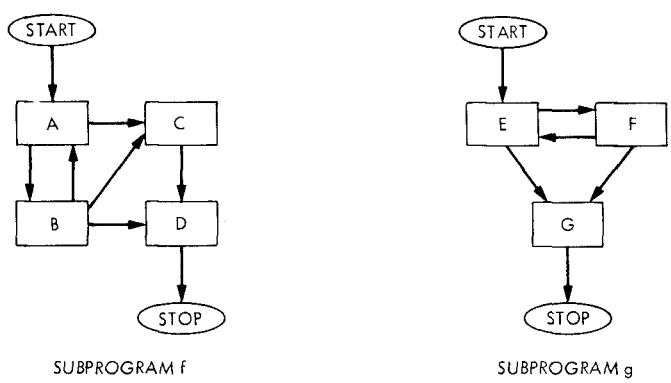


Fig. 4. Subprogram flowchart

This program and its subroutines could be combined into the following flowchart (Fig. 5):

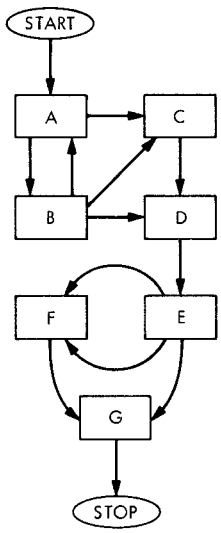


Fig. 5. Program and subroutine flowchart

In general, the subroutines could themselves contain subroutines containing further subroutines, etc. In this way, a very large and complex flowchart can be built up from simple components. The basic idea of *structured programming* is to build up program flowcharts in this way from a small collection of "basic" allowable flowcharts.

Let us now formally define a program *flowchart* as a set of *steps* and a set of *branches*, each branch leading from a step a to a step b, and denoted pictorially by $a \rightarrow b$. This branch is called an *exit* from a and an *entrance* to b. A flowchart must always have two special steps called START and STOP such that there is only one exit from START, and no entrances, and only one entrance to STOP, no exits. Furthermore, a flowchart step with exactly one entrance and exit can be designated a "subprogram," and will refer to another flowchart, as in the above example.

A few important flowcharts which have been given "names" for reference are shown in Fig. 6.

It is common to employ different symbols in order to distinguish various kinds of program steps. Thus in Ref. 2 a step with one entrance and several exits is called a *decision* node and denoted by a lozenge; one with one exit and several exits is called a *collecting* node and denoted by a circle; and one with one entrance and one exit is called a *process* node and denoted by a rectangle.

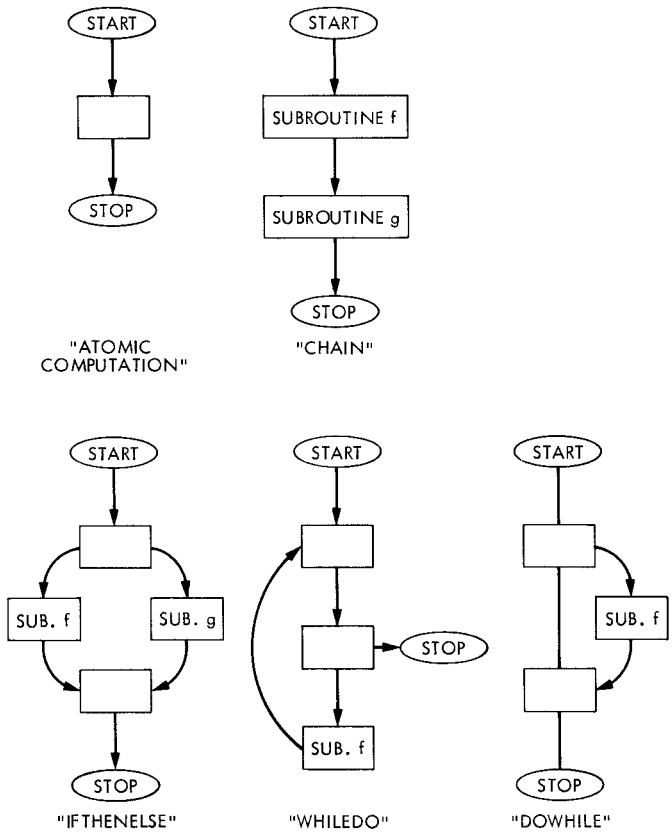


Fig. 6. Some important flowcharts

We come now to the definition of a *structured* program flowchart. If $\mathcal{B} = \{F_1, F_2, \cdots\}$ is a set of flowcharts—called $\mathcal{B}$-flowcharts—another flowchart is said to be *$\mathcal{B}$-structured* if it is either an *atomic computation*, or if it is a $\mathcal{B}$-flowchart, all of whose subprograms are $\mathcal{B}$-structured.

It is known (Ref. 2, ch. 5) that any program[2] can be rewritten in such a way that its flowchart is $\mathcal{B}$-structured if $\mathcal{B} = \{\text{CHAIN,IFTHENELSE,DOWHILE}\}$.

Since structured program flowcharts with many levels of subprograms are hard to draw, it is convenient to have a compact pictorial description of them. One such description is the *program tree*, in which each $\mathcal{B}$-flowchart is represented by a program tree *module*, consisting of a box labeled with the flowchart name, and descending arrows corresponding to the subprograms. For example, the modules for CHAIN, IFTHENELSE, DOWHILE, and WHILEDO look like this (Fig. 7):

[2]Actually this result holds only for *proper* programs; for definitions, see Ref. 2.

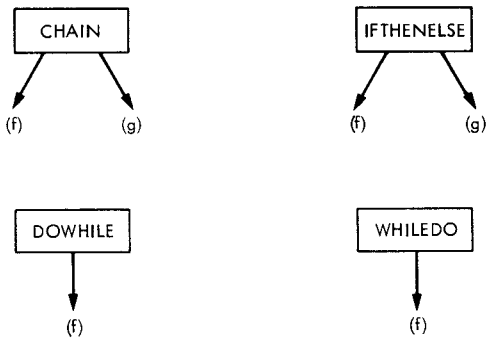


Fig. 7. Some program tree modules

The module for the special flowchart "atomic computation" is the symbol "$\equiv$".

Given a $\mathcal{B}$-structured flowchart, its program tree is defined as "$\equiv$", if it is an atomic computation, or if it is a $\mathcal{B}$-flowchart with subprograms, as the program tree module of the $\mathcal{B}$-flowchart, with the program trees of its subprograms attached to the appropriate arrows of the module We define the *level* of the first (top) $\mathcal{B}$-flowchart as 1, and for the other flowcharts as one more than its "parent" flowchart in the program tree. For example, here is the program tree of a {CHAIN,IFTHENELSE,DOWHILE}-structured program, with the level numbers written in parentheses to the right of the flowchart modules (Fig. 8):

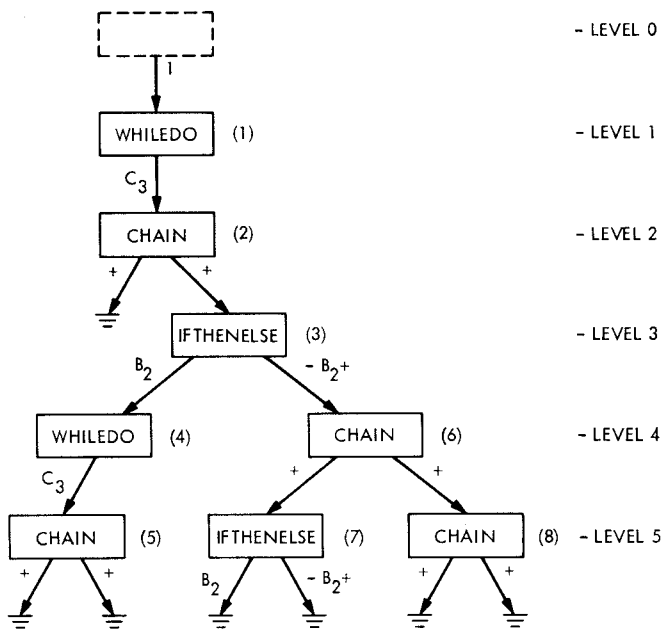


Fig. 8. A program tree for a structured flowchart

(The dummy module at level 0 and the labels on the arrows will be explained in Section III; for now they should be ignored.) If expanded into one flowchart, it would look like this (Fig. 9):

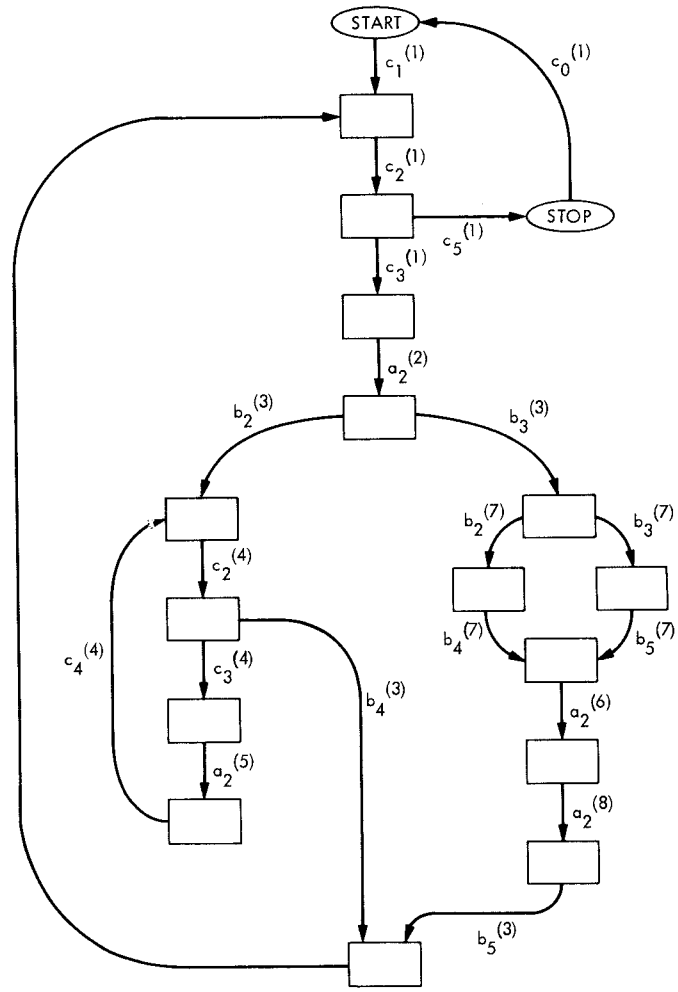


Fig. 9. Flowchart represented by the tree of Fig. 8

(This is the same as the flowchart in Figure 5.1.2 in Ref. 2; the branch labels will be explained in Section III.)

III. Solving Kirchhoff's Equations for Structured Flowcharts

In this section we will show that for a $\mathcal{B}$-structured program flowchart, Kirchhoff's equations can be solved by a simple *inspection* of the program tree. Throughout, we will illustrate the ideas with the $\mathcal{B}$-structured program of Figs. 8 and 9, with $\mathcal{B}$ = {CHAIN, IFTHENELSE, WHILEDO}.

The first step is to solve Kirchhoff's equations in each of the $\mathcal{B}$-flowcharts. This can be done by the method described in Section I, and it may take a lot of work if $\mathcal{B}$ contains many complicated flowcharts. The point is, however, that once this is done we will be prepared to solve Kirchhoff's equations for *any* $\mathcal{B}$-structured program with essentially no additional work.

For example, if $\mathcal{B} = \{$CHAIN, IFTHENELSE, WHILEDO$\}$, a set of solutions to Kirchhoff's equations is given in Fig. 10 (cf. Fig. 6; the edges are labeled with lower-case a's, b's, and c's, and the upper-case letters denote the corresponding branch variables.)

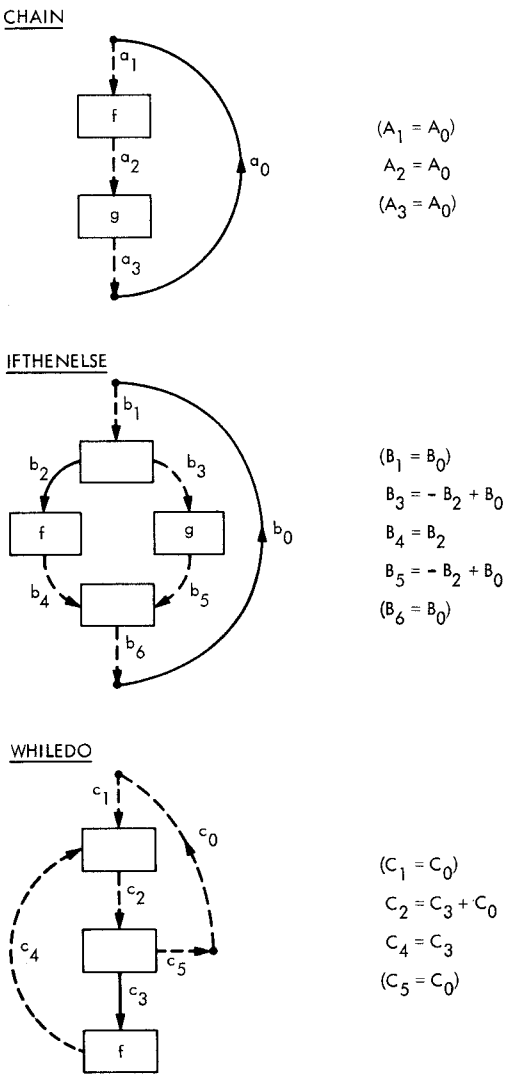


Fig. 10. Structured program flowchart

(Appendix A contains the solutions to Kirchhoff's equations for four more simple flowcharts which may be encountered as $\mathcal{B}$-flowcharts.)

Having done this preliminary work, we are prepared to solve Kirchhoff's equations for any $\mathcal{B}$-structured flowchart. Our first step is to assign *labels* to the branches in the component $\mathcal{B}$-flowcharts. To do this, we assign an identifying number to each flowchart and use that number as a superscript on the branch variables, so that identical $\mathcal{B}$-flowcharts which appear in different places in the program tree can be distinguished. For example in the flowchart of Fig. 8, the branch variables of WHILEDO Flowchart 1 will be denoted by $c_i^{(1)}$, and those of Flowchart 4 by $c_i^{(4)}$.

It is important to notice that except for the top (level 1) flowchart, not all of the flowchart branches need be considered. For example, in Fig. 8, in CHAIN Flowchart 2, branch $a_1^{(2)}$ is the entrance from WHILEDO Flowchart 1 and so it is identical to $c_3^{(1)}$. Similarly $a_3^{(2)}$ is identical to $c_4^{(1)}$. Because of this, in each $\mathcal{B}$-flowchart except the top one, the exit from START, the entrance to STOP, and the fictitious branch from STOP to START will all be missing. In the example of Fig. 8 there will be 20 labeled branches; these have been indicated in Fig. 9, although we emphasize that it will not be necessary to draw the complete flowchart in order to solve Kirchhoff's equations.

The next step is to observe that within each component $\mathcal{B}$-flowchart, *Kirchhoff's equations remain valid*, except that a special interpretation must be given the fictitious STOP $\rightarrow$ START branches. For example, in the WHILEDO Flowchart 4 in Fig. 8, the branch $c_0^{(4)}$ represents the "boundary" between this flowchart and the rest of the program. Thus $C_0^{(4)}$ is the number of times the WHILEDO Flowchart 4 is entered, and so the "boundary condition" is $C_0^{(4)} = B_2^{(3)}$, since every time the branch $b_2^{(3)}$ in IFTHENELSE Flowchart 3 is traversed, the flowchart WHILEDO Flowchart 4 is entered.

In this way we can express every branch variable in the $\mathcal{B}$-structured flowchart in terms of the *fundamental* branch variables of the component $\mathcal{B}$-flowcharts, as follows. Given a branch in one of the component $\mathcal{B}$-flowcharts, use Kirchhoff's equations within the flowchart to express the corresponding branch variable in terms of the fundamental branch variables in the *same* $\mathcal{B}$-flowchart, together with a branch variable in a $\mathcal{B}$-flowchart at a higher level. This new branch variable, if not itself fundamental, is then subject to the same procedure, until eventually the top flowchart is reached.

For example the edge $b_3^{(7)}$ in Figs. 8 and 9 yields the following series of equations:

$$B_3^{(7)} = -B_2^{(7)} + B_0^{(7)}$$
$$B_0^{(7)} = A_1^{(6)}$$
$$A_1^{(6)} = B_3^{(3)}$$
$$B_3^{(3)} = -B_2^{(3)} + B_0^{(3)}$$
$$B_0^{(3)} = A_2^{(2)}$$
$$A_2^{(2)} = A_0^{(2)}$$
$$A_0^{(2)} = C_3^{(1)}$$

Thus finally we get the equation $B_3^{(7)} = -B_2^{(7)} - B_2^{(3)} + C_3^{(1)}$, which expresses $B_3^{(7)}$ in terms of the fundamental branch variables $B_2^{(7)}$, $B_2^{(3)}$, $C_3^{(1)}$.

We can avoid this "unraveling" procedure by observing that in the above procedure we "bubble-up" through the program tree, picking up fundamental branch variables as we go. This suggests that we should label the arrows in the program tree with information that will tell us which branches to "pick up" in our rise to the top. The simplest such scheme is to label each program tree arrow with the formula which expresses the corresponding $\mathcal{B}$-flowchart branch variable in terms of the fundamental branch variables, perhaps omitting the fictitious branch variables, but retaining their signs[3]. Thus the modules in Fig. 7 would appear as this (Fig. 11):

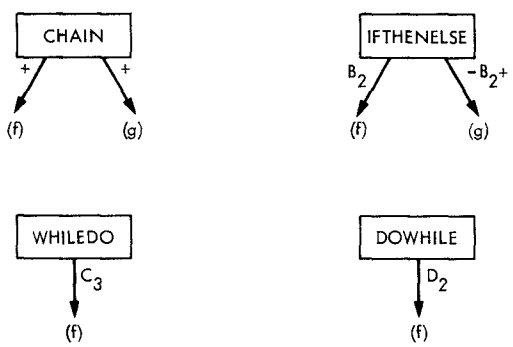


Fig. 11. Modules of Fig. 7

[3]It is possible that some branch variables will occur on the fundamental cycle which includes the fictitious branch with a minus sign. However, for any flowchart which corresponds to a realizable computer program, there will always be a properly oriented path from START to STOP, which we can take as part of this fundamental cycle. This means that we can always assume that the fictitious branch variable occurs with a "+" sign in Kirchhoff's equations.

Assuming that the program tree has been built from these arrow-labeled modules, the expression of a given branch variable (b.v.) in terms of the fundamental branch variables (f.b.v.'s) can be obtained as follows. First, express the given b.v. in terms of f.b.v.'s in its own (local) $\mathcal{B}$-flowchart. If the fictitious b.v. does not occur in this expression, we are through, since these local f.b.v.'s are global f.b.v.'s as well. If, however, the fictitious b.v. does appear, it must be expressed in terms of fundamental b.v.'s at higher levels. This is done by "climbing" the program tree, picking up arrow labels along the way, and stopping as soon as an arrow without a "+" is reached. (To be completely systematic, we should add a dummy module at level 0, whose arrow is labeled "1", so that the fictitious b.v. at level 1 will be assigned the value 1 by this procedure. This has been done in Fig. 8).

For example, using this technique on the program tree of Fig. 8, we express $B_3^{(7)}$ in terms of f.b.v.'s as follows:

$B_3^{(7)} = -B_2^{(7)} + B_0^{(7)}$ (Solution to Kirchhoff's equations in IFTHENELSE flowchart; see Fig. 10.)

$B_0^{(7)} = -B_2^{(3)} + C_3^{(1)}$ (From labeled program tree; see Fig. 8.)

In this way we obtain the following complete solution to Kirchhoff's equations for the flowchart of Figs. 8 and 9:

Flowchart number	Solution
1	$C_1^{(1)} = 1$
	$C_2^{(1)} = C_3^{(1)} + 1$
	$C_3^{(1)} =$ fundamental
	$C_4^{(1)} = C_3^{(1)}$
	$C_5^{(1)} = 1$
2	$A_2^{(2)} = C_3^{(1)}$
3	$B_2^{(3)} =$ fundamental
	$B_3^{(3)} = -B_2^{(3)} + C_{(1)}^3$
	$B_4^{(3)} = B_2^{(3)}$
	$B_5^{(3)} = -B_2^{(3)} + C_3^{(1)}$
4	$C_2^{(4)} = C_3^{(4)} + B_2^{(3)}$
	$C_3^{(4)} =$ fundamental
	$C_4^{(4)} = C_3^{(4)}$
5	$A_2^{(5)} = C_3^{(4)}$
6	$A_2^{(6)} = -B_2^{(3)} + C_3^{(1)}$
7	$B_2^{(7)} =$ fundamental
	$B_3^{(7)} = -B_2^{(7)} - B_2^{(3)} + C_3^{(1)}$
	$B_4^{(7)} = B_2^{(7)}$
	$B_5^{(7)} = -B_2^{(7)} - B_2^{(3)} + C_3^{(1)}$
8	$A_2^{(8)} = -B_2^{(3)} + C_3^{(1)}$.

In this way we have expressed each of the 20 branch variables in terms of the four fundamental branch variables $C_3^{(1)}$, $B_2^{(3)}$, $C_3^{(4)}$, $B_2^{(7)}$.

In summary, we have seen that in order to solve Kirchhoff's equations for a $\mathcal{B}$-structured program, it is first necessary to solve Kirchhoff's equations in the $\mathcal{B}$-flowcharts, and also to label the arrows of the program tree modules. Once this is done the solution to Kirchhoff's equations can be read directly from the program tree. Since the program tree is generally available anyway—it is part of the program documentation—once the solutions to Kirchhoff's equations for the $\mathcal{B}$-flowcharts are available, the solution of Kirchhoff's equations in an arbitrary $\mathcal{B}$-structured flowchart becomes trivial. For this reason, we have listed in the appendix seven flowcharts which are likely to appear in $\mathcal{B}$, together with their fundamental cycles, solutions to Kirchhoff's equations, and program tree modules.

References

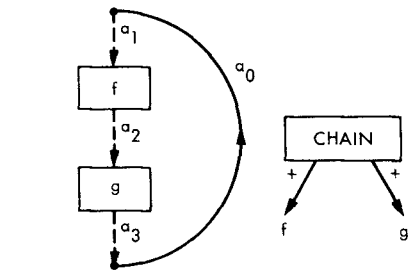
1. Knuth, Donald, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms,* Addison-Wesley, Reading, Mass., 1969.
2. Tausworthe, Robert, *Standardized Development of Computer Software,* SP 43-29, Jet Propulsion Laboratory, Pasadena, Calif. (to be published).
3. Deo, Narsingh, *Graph Theory, with Applications to Engineering and Computer Science,* Prentice-Hall, Englewood Cliffs, N. J., 1974.

Appendix A

Some Basic Flowcharts

In the flowchart diagram, *solid* branches are fundamental, *dotted* branches are not. Equations in parentheses involve branch variables whose branches are "connections" to the overall flowchart and which will not occur except at the top of the program tree.

CHAIN



FUNDAMENTAL CYCLES

$a_0 + a_1 + a_2 + a_3$

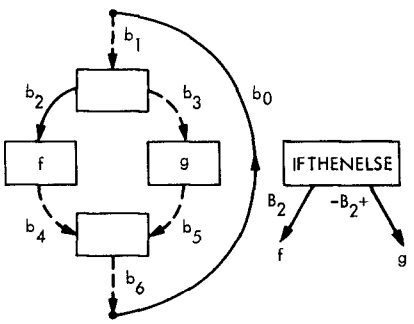
KIRCHHOFF'S EQUATIONS

$(A_1 = A_0)$

$A_2 = A_0$

$(A_3 = A_0)$

IFTHENELSE



FUNDAMENTAL CYCLES

$b_0 + b_1 + b_3 + b_5 + b_6$

$b_2 + b_4 - b_5 - b_3$

KIRCHHOFF'S EQUATIONS

$(B_1 = B_0)$

B_2 = FUNDAMENTAL

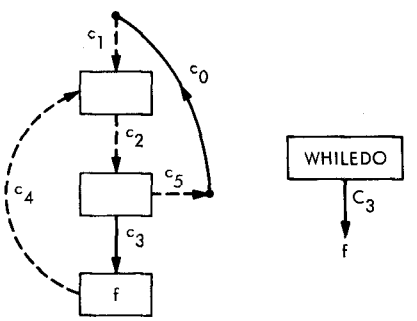
$B_3 = -B_2 + b_0$

$B_4 = B_2$

$B_5 = -B_2 + B_0$

$(B_6 = B_0)$

WHILEDO



FUNDAMENTAL CYCLES

$c_0 + c_1 + c_2 + c_5$

$c_3 + c_4 + c_2$

KIRCHHOFF'S EQUATIONS

$(C_1 = C_0)$

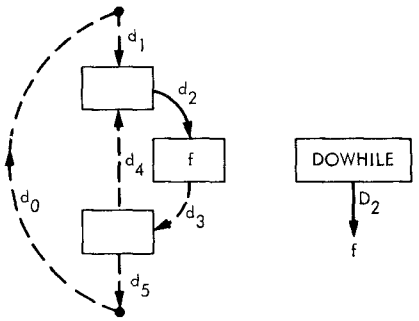
$C_2 = C_3 + C_0$

C_3 = FUNDAMENTAL

$C_4 = C_3$

$(C_5 = C_0)$

DOWHILE



FUNDAMENTAL CYCLES

$d_0 + d_1 - d_4 + d_5$

$d_2 + d_3 + d_4$

KIRCHHOFF'S EQUATIONS

$(D_1 = D_0)$

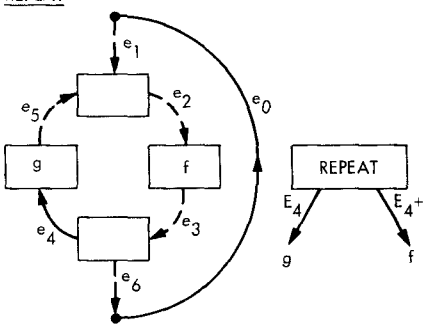
D_2 = FUNDAMENTAL

$D_3 = D_2$

$D_4 = D_2 - D_0$

$(D_5 = D_0)$

REPEAT



FUNDAMENTAL CYCLES

$e_0 + e_1 + e_2 + e_3 + e_6$

$e_4 + e_5 + e_2 + e_3$

KIRCHHOFF'S EQUATIONS

$(E_1 = E_0)$

$E_2 = E_4 + E_0$

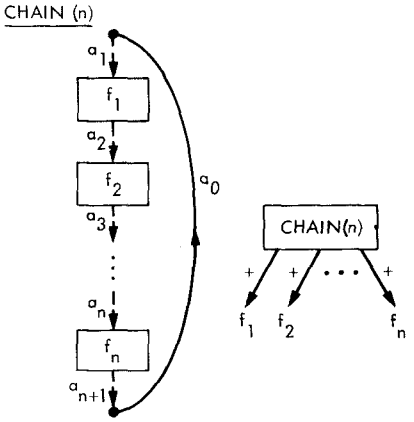
$E_3 = E_4 + E_0$

E_4 = FUNDAMENTAL

$E_5 = E_4$

$(E_6 = E_0)$

CHAIN (n)



FUNDAMENTAL CYCLE

$a_0 + a_1 + a_2 + \ldots + a_{n+1}$

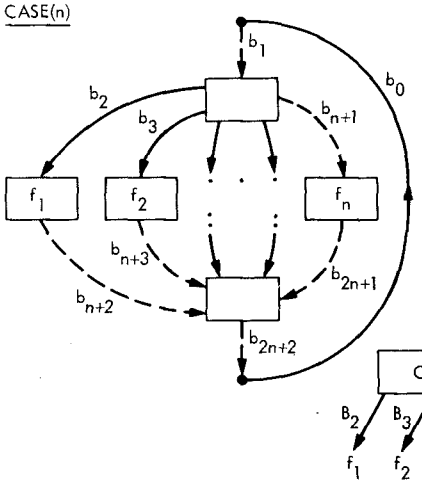
KIRCHHOFF'S EQUATIONS

$(A_1 = A_0)$

$A_2 = A_0$

$(A_{n+1} = A_0)$

CASE(n)



FUNDAMENTAL CYCLES

$b_0 + b_1 + b_{n+1} + b_{2n+1} + b_{2n+2}$

$b_i + b_{n+i} - b_{2n+1} = b_{n+1}$

$(i = 2, 3, \ldots, n)$

KIRCHHOFF'S EQUATION

$(B_1 = B_0)$

$B_{n+1} = B_{2n+1} = -B_2 - \ldots - B_n + B_0$

$B_{n+i} = B_i \quad (i = 2, \ldots, n)$

$B_{2n+2} = B_0)$