ALGEBRAIC FORMULATION OF FLOW DIAGRAMS*

By

Edward A. Voorhees

University of California
Los Alamos Scientific Laboratory
Los Alamos, New Mexico

June 20, 1957

*This work performed under the auspices of the U. S. Atomic Energy Commission

I. INTRODUCTION

Discussions involving the subject of defining problems for interpretation and coding by known automatic-coding systems generally suggest that the techniques for stating the control (or logic) of the problem are frequently difficult to understand and difficult to use. It seems that the difficulty is one of discovering a suitable language with which to define problem control. In programming a problem for hand coding, the familiar flow diagram has been successfully used (when needed) to define the control of the problem. Such flow diagrams (as we draw them) cannot be presented directly to present-day computers. It is the purpose of this paper to propose a flow diagram representation using simple algebraic language which can be directly entered into the computer.

II. DESCRIPTION

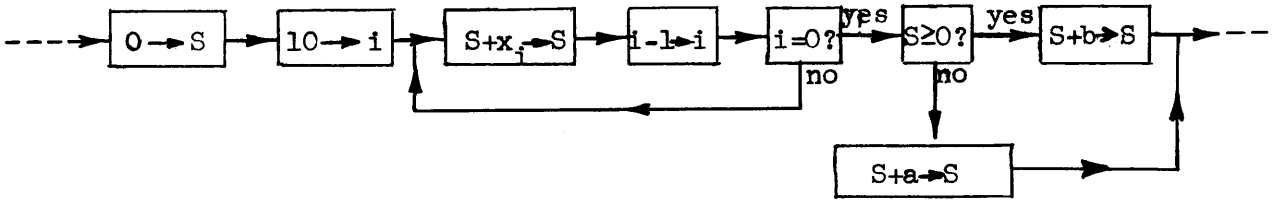
Assume we have a flow-diagram drawn such that it could be used to hand-code the problem for any stored-program computer. Suppose we remove from the boxes all equations, statements of input-output tasks and other statements not directly related to the control and logic of the problem, and list them (with identification) elsewhere as reference material. We do not include in this list statements and questions pertaining to loops, numerical conditions, switch and trigger conditions, etc. What remains, therefore, is a statement (part-picture) of the

control for the problem. We will attempt to translate this control statement into a control statement which can be easily written and entered into a computing machine.

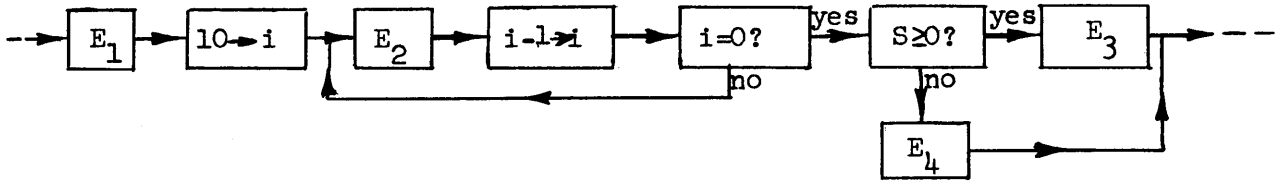
In order to illustrate the above discussion and extend the idea further, let us consider an elementary example. Suppose in a problem we wished to compute

$$S = \sum_{i=1}^{10} x_i + \left\{ \begin{matrix} a & (\text{if} \sum < 0) \\ \text{or} & \\ b & (\text{if} \sum \geq 0) \end{matrix} \right\}$$

The corresponding flow diagram might be drawn as:



The result of doing what was indicated in the previous paragraph would be:



where E_1 : $S = 0$

E_2 : $S = x_i + S$

E_3 : $S = S + b$

E_4 : $S = S + a$

Let us now make a distinction between loop ranges (such as the i-loop in the above example) and conditions for execution of equations (such

as the S test above). Suppose we define

I_1 : i = 1, 10, 1 (meaning i takes successively the values 1 through 10 in increments of 1.)

and C_1 : $S \geq 0$.

Then our example can be completely defined as follows:

Example 1

Control	Equations
$\ldots + E_1 + I_1E_2 + C_1(E_3, E_4) + \ldots$	E_1 : S = 0
	E_2 : $S = S + x_i$
I_1 : i = 1, 10, 1	E_3 : S = S + b
C_1 : $S \geq 0$	E_4 : S = S + a

where C_1 here is understood to mean "execute E_3 if C_1 is satisfied and execute E_4 if C_1 is not satisfied." An alternative form which could be used is $C_1E_3 + C_1'E_4$, provided it is known that when E_3 is executed E_4 is not also executed (as would be the case if $\sum x_i = 1$ and b = -2). C_1' is used to mean the negation of the condition C_1. Clearly, the C' convention is not essential since, C_2 : $S < 0$ could be used instead. The meaning of I_1E_2 is evident and will be defined more generally below.

Let us now define some control statement symbols. These definitions make no restrictions on the characters with regard to their use in equation writing since control statements are assumed to be handled separately from equations.

E (assumed to have a subscript) represents a single equation, or a continuous and closed set of equations (whenever one is done, all are done), or an input-output task, or a program stop, etc., or any other statement (or closed set of statements) not directly related to problem control.

C (assumed to have a subscript) represents a single condition. Several conventions have been developed using C's. They will be defined later. C's are assumed to be used for stating all conditions, except those inherent in range statements such as end-of-loop tests. C-type conditions include numerical conditions, logical conditions, console switch conditions, trigger conditions, etc. (A more-generalized C is under development.)

Any letter except C, E, S, T, or G* (assumed to have a subscript) represents a single range statement. I_1 in the example represents the range statement i = 1, 10, 1.

S (assumed to have a subscript) represents a control statement. It allows the problem control to be expressed in many control statements which are in turn connected by a single master control statement.

---

*The letters G (and G') are used in a special convention which may or may not be a desirable extension of the proposal. They do not seem to be essential, but probably provide simplifications in using the system. They will be described at the end of the paper.

$\underline{T}$ (assumed to have a subscript) represents a code which is entered by basic linkage. This concept as well as the technique for function subroutines will not be discussed further in this summary.

The plus sign, +, is used to indicate "execute the following term" as, for example, $E_1 + E_2$ means "execute E_1, then execute E_2." E_1 and E_2 are terms, of the control statement $E_1 + E_2$.

The parentheses, ( ), are used for the phrasing or grouping of terms, for indicating ranges, and for special purposes to be described. The expression $(E_1 + E_2)$ is a single term.

The comma (,) is used in special conventions.

The plus sign, + , in addition to the use described above is used to indicate "logical OR." This is a permissible logical connective only when used between C's.

The product dot, . , is used to indicate "logical AND." This is a permissible logical connective only when used between C's.

The meanings of some basic combinations of symbols used in control statements are given below. Each combination is a single term. The particular choice of subscripts has no implication.

1. I_jE_k means: Calculate E_k loopwise for the values of i indicated in the definition of I_j.

2. $I_j(....)$ means: Calculate the terms within the parentheses loopwise for the values of i indicated in the definition of I_j.

3. $\begin{cases} C_iE_j \\ C_i(....) \end{cases}$ means: Calculate E_j (or the terms within the parentheses) only if the condition C_i is satisfied.

3'. $C_i(E_j,E_k$ means: Calculate what precedes the comma if C_i is satisfied and calculate what follows the comma if C_i is not satisfied.*

4. $\begin{cases} E_kC_j \\ (....)C_j \end{cases}$ means: Calculate E_k (or the terms within the parentheses), test C_j and if C_j is not satisfied, calculate E_k again, test C_j, etc., and continue iterating until C_j is satisfied.

5. $I_k(E_i)C_j$ means: After <u>each cycle</u> of the I_kE_i loop, test C_j and if satisfied, leave the loop and calculate the term following this term. If C_j is not satisfied, do the next cycle of the I_kE_j loop, test C_j, etc. Exit from the I_kE_i loop is made as in 1 unless it is made earlies due to C_j satisfaction.

---

*$C_i(E_j,E_k)$ may be substituted for C_j in the following expressions. This usage permits sub-conditional calculations to take place before the described C control is executed.

6. $(I_k + I_\ell \ldots + I_q)E_j$

means: $I_kE_j + I_\ell E_j + \ldots + I_qE_j$. Presumably the factored notation would produce a spacewise better code.

Additional basic combinations have been defined but since they are not completely general in nature they will not be included.

Some combinations of interest which follow from the basic combinations described above are:

(a) $C_j(I_kE_i)$ means: Calculate the loop, I_kE_i, only if C_j is satisfied. This is a consequence of 3 and 1.

(b) $I_k(C_jE_i)$ means: For <u>each cycle</u> of the $I_k(\ldots)$ loop, calculate E_i only if C_j is satisfied, otherwise start the next cycle. See 2 and 3.

(c) $I_k(E_iC_j)$ means: For <u>each cycle</u> of the $I_k(\ldots)$ loop, iterate E_i until C_j is satisfied. See 2 and 4.

(d) $(I_kE_i)C_j$ means: Calculate the I_kE_i loop, test C_j and if not satisfied, calculate the I_kE_i loop again, test C_j, etc. See 4.

(e) $I_k(J_iE_\ell)$ means: Calculate the J_iE_ℓ loop for the values of i indicated in the definition of I_h. This indicates a loop within a loop. The extension to more loops is obvious. See 2.

In all of the above combinations, any E could be replaced by any control statement enclosed in parentheses. This was indicated in 3 and 4 and can be observed to be the only difference between 1 and 2.

Furthermore, in all of the combinations involving a C, the C could be replaced by a "C-statement." A "C-statement" consists of C's (only) connected by logical AND's and OR's and is totally enclosed by parentheses. As an example, let us write one of the many generalizations of 5.

5'. $I_k(E_i)(C_j + C_\ell + \ldots + C_p)$ means: After <u>each cycle</u> of the I_kE_i loop, test the "C-statement" and if any one of the C's is satisfied, exit from the loop, etc.

Let us now illustrate some of the preceding ideas with another example. Assume we wish to solve the Laplace equation for a 10 x 10 mesh with x = 1 on the boundaries, we might start with x = 0 in the interior. Then:

Example 2

CONTROL

$$I_1(J_1E_1) + I_2(J_2E_2) + (E_3 + I_2(J_2E_4))C_1 + E_5$$

I_1 : i = 1, 10, 1

J_1 : j = 1, 10, 1

I_2 : i = 2, 9, 1

J_2 : j = 2, 9, 1

C_1 : $C - 0.001 \leq 0$

EQUATIONS

E_1 : $x_{i,j} = 1$

E_2 : $x_{i,j} = 0$

E_3 : $C = 0$

E_4 : $E = (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1})/4$

$C = C + |E - x_{i,j}|$

$x_{i,j} = E$

E_5 : STOP

The first two terms of the control statement set up the mesh (using a crude technique), the third term represents the main calculation, and the fourth term stops the problem after the iteration is complete. By adding the four additional range statements,

$$I_3 : \ i = 1$$
$$I_4 : \ i = 10$$
$$J_3 : \ j = 1$$
$$J_4 : \ j = 10,$$

the mesh setup can be improved (from a time standpoint) by replacing the first two terms of the control statement by the three terms,

$$(J_3 + J_4)(I_1E_1) + (I_3 + I_4)(J_1E_1) + I_2(J_2E_2).$$

III. AN OPTIONAL CONVENTION

An additional set of symbols which possibly offer some advantages but which do not seem to be essential or necessary and which probably introduce some undesirable flexibilities have been introduced.

This convention is designed to facilitate transfer of control. It has been an interesting observation while testing the system how rarely this convention has been a convenience and still more rarely a necessity. The symbol $\underline{G}$ (with a subscript) denotes a transfer to $\underline{G'}$ (with the same subscript). There may be a many-to-one correspondence between G and G'. The two following control statements define the same calculation:

(a) $C_1E_1 + E_2 + (E_1 + E_2)C_2 + E_3$

(b) $C_1(G_1'E_1) + E_2 + C_2'G_1 + E_3$

IV. CONCLUSION

It is not the intention of the author to imply that the system just described is complete and fully developed for immediate use in an automatic coding system. We have, however, tested the system on several large problems and have yet to find any programming sequence which cannot be handled. For some situations we have created additional conventions (not defined in this summary) for brevity and elegance. We plan to examine many more problems.

The author would like to thank Mr. Bengt Carlson for his comments and encouragement and Mr. Carl R. Blancett for his generous work in testing the system on problems and for his helpful suggestions.