ANALYSIS OF STRUCTURED PROGRAMS

S. Rao Kosaraju
Electrical Engineering Department
The Johns Hopkins University
Baltimore, Maryland 21218

Abstract

We investigate various control structures to understand their computational complexity and limitations. It is generally felt that GOTO-less programs constructed from the classical primitives are very restrictive; structured programming languages like BLISS, however, incorporating Repeat-Exit constructs appear to ease this sense of restrictiveness. In this paper we analyze this construct. We answer a conjecture of Knuth and Floyd as a special case of the general theory. We also investigate a general Top-Down Programming construct, which we call the TD_n-construct.

We structurally characterize the class of GOTO-less programs. We also generalize such an analysis and solve an open problem of Böhm and Jacopini.

I. Introduction

Recently there has been a significant attempt to move the act of programming onto a scientific plane. These ideas are not entirely new; but within the context of computer software, they were first advocated by Dikjstra [7]. He advocated avoiding unnecessary complexity of the control structures of programs by decomposing the problem into a well understood interconnection of a small number of sub-problems. The correctness of each decomposition should be verified before expanding the sub-problems. This approach seems to provide the key to treating the complexity of large problems. But it should be understood that control structures are not the only significant aspects in programming.

We analyze various control structure schemas to understand their complexity and limitations.

II. Teminology

A flow chart is composed of (basic) functions and (basic) predicates. A <u>basic function</u>, a, is a 1 entry/1 exit device represented by: $\rightarrow a \rightarrow$ or $\rightarrow\boxed{a}\rightarrow$ or $\xrightarrow{a}$. A branch without any label is always allowed. It stands for id transformation or NO OP.

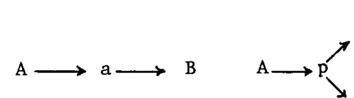
A <u>basic predicate</u>, p , is a 1 entry/2 exit device represented by:

We avoid T's and F's when there is no ambiguity or if either labelling is allowable.

Any i entry/j exit deterministic interconnection of a finite number of basic units is an (i,j) flow chart. Here deterministic refers to the constraint that any point cannot be connected to more than one other point without going through a predicate unit. In this report, unless otherwise specified a <u>flow chart</u> (or a <u>program</u>) refers to a (1,1) flow chart; i.e. we assume one entry called <u>IN</u>, and one exit called <u>OUT</u>.

We use a,b,c,... to represent basic functions
p,q,r,... to represent basic predicates,
and F,G,H,... to represent flow charts

In a flow chart there is a <u>direct connection</u> (or <u>direct path</u>) from point A to point B iff a basic unit connects A to B, i.e. A is connected to B in one of the following ways:



There is a <u>path</u> from A to B iff there are points $C_1, C_2, \ldots, C_m$, where $A = C_1$ and $B = C_m$, and there are direct connections from C_i to C_{i+1} for $i = 1, \ldots, m-1$.

There is a <u>loop</u> passing through A and B iff there are paths from A to B and from B to A. We denote such a loop as <u>loop AB</u>.

The data at the point IN is transformed by the flow chart, and the output is produced at the point OUT. Initially the control is at IN , while at termination it is at OUT. The processing of data by a basic unit is counted as one <u>step</u> of computation. A basic function transforms data and changes control from its input line to its output line. A basic predicate leaves the data unchanged but changes control from its input line to exactly one of its output lines (T or F), depending upon whether that predicate holds or not (respectively) at that instant. The sequence of steps caused by an input is called the computation sequence for that input. For a flow chart F the output resulting from an input x is represented by F(x).

Two flow charts F and G are <u>strongly equivalent</u> iff for every input, the computation sequences are identical for both flow charts. Let this be represented by $F \equiv_s G$.

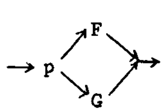
Two flow charts F and G are <u>weakly equivalent</u> iff for any interpretation of the basic elements, for every input the outputs are equal for both flow charts. Let this be represented by $F \equiv_w G$.

III. D-Flow Charts

In this section we study a class of flow charts known as D-flow charts, named after Dijkstra.

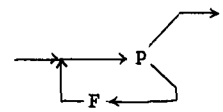
<u>D-flow charts</u> are defined recursively as follows:

1. Any basic function is a D-flow chart.
2. If F and G are D-flow charts, then $\rightarrow F \rightarrow G \rightarrow$ is a D-flow chart (BLOCK).
3. If F and G are D-flow charts, and P is a basic predicate unit, then



is a D-flow chart (IFTHENELSE).

4. If F is a D-flow chart, and p is a basic predicate unit, then

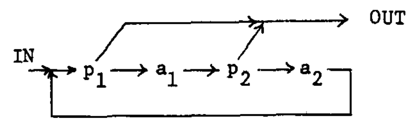


is a D-flow chart (DOWHILE).

Böhm and Jacopini [2] showed that for any flow chart there exists a weakly equivalent D-flow chart, which might use auxiliary control variables and operations on those variables. It is natural to ask whether the result can be strengthened.

<u>Theorem 1</u>: [9] The flow chart F given below cannot be strongly equivalent to any D-flow chart.

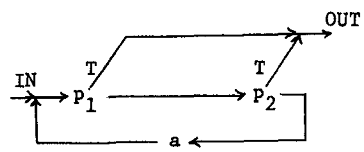
F:



The following strengthens the above theorem.

<u>Theorem 2</u>: Flow chart F given below cannot be weakly equivalent to any D-flow chart which is built up from only p_1, p_2 and a:

F:



<u>Proof</u>: If possible, let there exist such a D-flow chart G weakly equivalent to F . Then $G \equiv_w F$ for any interpretation of p_1, p_2 and a . Let

p_1 be "$x = 2^n$ for some integer n?",

p_2 be "$x = 2^n + 2^{n-1}$ for some integer n?"

a be "$x \leftarrow x+1$," and $x \in N$.

F increments any natural number x to the nearest natural number of the form 2^n or 2^n+2^{n-1}. Thus for any x where $2^n < x \leq 2^n+2^{n-1}$, the output is 2^n+2^{n-1}; and for any x where $2^n+2^{n-1} < x \leq 2^{n+1}$, the output is 2^{n+1}. If we call numbers of the form 2^n or 2^n+2^{n-1} <u>stops</u>, then for any input x, the output is the nearest higher stop.

Let there be k a-units in G. Consider a large m such that $2^{m-2} > k+1$. The output for input $2^{m-(k+1)}$ should be 2^m . Since there are only k a-units in G at least one of them must be used twice (i.e. during computation, x must have been incremented by some physical a-unit in at least 2 different instants). So x must have been processed by a loop and got out of the loop with true test on p_1. Then the input $2^m+2^{m-1}-(k+1)$ should enter and follow the same loop and exit with true test on p_1. The nearest such number is 2^{m+1} and so $G(2^m+2^{m-1}-k-1) > 2^m+2^{m-1} = F(2^m+2^{m-1}-k-1)$, a contradiction.

Before we prove it more preceisely, let us establish a few lemmas:

Lemma 2.1: If any x passes through a path composed only of the above basic units then the output will not be less than x.

Proof: x never gets decremented. QED

Lemma 2.2: Let H be any flow chart composed of the above basic units containing k a-units. For any input x, if $H(x) \geq x + k + 1$, then $H(x) \geq$ nearest higher stop of x, or undefined.

Proof: If $x + k + 1 \geq$ nearest higher stop of x, then the lemma is trivially satisfied. Otherwise, $x + k + 1 <$ nearest higher stop of x. Since x gets incremented by more than k, and since there are only k a-units, x must have passed through a loop. By the time x completes one round around the first loop, it should have been incremented by $\leq k$ since no particular a-unit operates on x more than one time. Thus, the loop should be formed out of only F exits of p_1 and p_2, and hence x could go out of that loop only a a T-condition. This coupled with lemma 2.1 establishes the lemma. QED

Lemma 2.3: Let H be any flow chart composed of the above basic units containing k a-units. Let x and y be any two inputs each of which is at a distance of at least $k+1$ to its corresponding nearest higher stop. In any t steps, if input x gets transformed to $x + \delta$, $\delta \leq k$, with control at some point $*$, then in the same t steps, y would get incremented to $y + \delta$, with control at $*$.

Proof: The complete path followed by x contains no T exit on p_1 or p_2 (since none of $x, x+1, \ldots, x+\delta$ is a stop). Hence y also follows the same path, since none of $y, y+1, \ldots, y+\delta$ is a stop (strictly speaking, we need a trivial induction along the path followed by x). Hence y gets incremented by the same amount δ, with control at $*$, and in fact follows the same path as x. QED

Lemma 2.4: Let H be any D-flow chart composed of the above basic units containing k a-units. For any m satisfying $2^{m-2} > k+1$, if $H(2^m-k-1) = 2^m$, then $H(2^m+2^{m-1}-k-1) > 2^m+2^{m-1}$ or undefined.

Proof: We prove the lemma by induction on the class of D-flow charts.

Let $x_1 = 2^m-k-1$ $\quad x_2 = 2^m + 2^{m-1}-k-1$

$s_1 = 2^m$ $\quad s_2 = 2^m + 2^{m-1}$

Basis: Lemma 2.4 is obviously true for $\rightarrow$ and $\rightarrow$ a $\rightarrow$ (hypothesis is not satisfied).

Inductive Step: Assume that the lemma holds for any D-flow charts H_1 and H_2 having k_1 and k_2 a-units respectively. Consider an H, having k a-units constructed as in each of the following cases.

Case 1: BLOCK

$H = \rightarrow H_1 \rightarrow H_2 \rightarrow$

We want to show that, if $H(x_1) = s_1$, then $H(x_2) > s_2$ or undefined.

Assume that $H(x_1) = s_1$. ....(*)

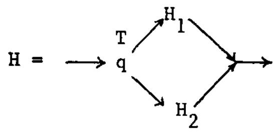
Then it cannot be the case that H_1 increments x_1 by $\leq k_1$ and H_2 increments the output of H_1 (i.e. $H_1(x_1)$) by $\leq k_2$ since $s_1 - x_1 > k$.

Subcase a: H_1 increments x_1 by more than k_1

Then by Lemma 2.2, $H_1(x_1) \geq s_1$. Hence by Lemma 2.1 and (*), $H_1(x_1) = s_1$. By hypothesis, $H_1(x_2) > s_2$ or undefined. From Lemma 2.1, it follows that $H(x_2) > s_2$ or undefined.

Subcase b: H_1 increments x_1 by $\leq k_1$ and H_2 increments the output of H_1 by more than k_2. Let $H_1(x_1) = x_1+\delta, \delta \leq k_1$. So by Lemma 2.3, $H_1(x_2) = x_2+\delta$. By (*), $H_2(x_1 + \delta) = s_1$; by hypothesis $H_2(x_2 + \delta) > s_2$ or undefined. Hence $H(x_2) = H_2(H_1(x_2)) = H_2(x_2+\delta) > s_2$ or undefined.

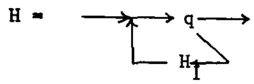
Case 2: IFTHENELSE



$q = p_1$ or p_2

Both x_1 and x_2 take F exit of q and so the result follows by applying the hypothesis to H_2.

Case 3: DOWHILE



$q = p_1$ or p_2

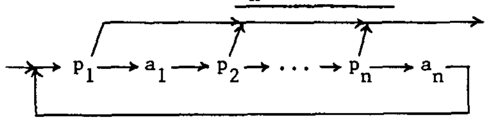
If $H(x_1) = s_1$, then q must be p_1 and the exit side must be marked 'T'. So $H(x_2)$ is 2^n for some $n \geq m + 1$, or undefined. Hence $H(x_2) > s_2$ or undefined. QED

Proof of Theorem 2 continued: Thus by Lemma 2.4, no D-flow chart using p_1, p_2 and a, containing only k a-units, can give correct outputs for both x_1 and x_2. QED

(Recently another proof was given in [3].

IV. Böhm & Jacopini Structures

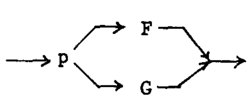
For any $n \geq 1$, the Ω_n-flow chart is:



where, for $i = 1, \ldots, n$, p_i and a_i are basic predicate and function units respectively.

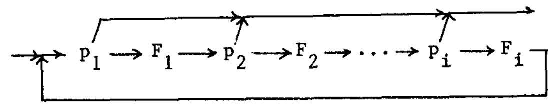
For any $n \geq 1$, BJ_n-flow charts are defined recursively as follows:

1. Any basic function unit is a BJ_n-flow chart.
2. If F and G are BJ_n-flow charts, then $\rightarrow F \rightarrow G \rightarrow$ is a BJ_n-flow chart.
3. If F and G are BJ_n-flow charts and p is a basic predicate unit, then



is a BJ_n-flow chart.

4. For $i \leq n$, if $F_1, F_2, \ldots, F_i$ are BJ_n-flow charts and $p_1, p_2, \ldots, p_i$ are basic predicate units then



is a BJ_n-flow chart.

Böhm and Jacopini [2] conjectured that: For any $n \geq 2$, the Ω_n-flow chart is not weakly equivalent to any BJ_{n-1}-flow chart composed of the same basic units. We prove a stronger result below (Theorem 3).

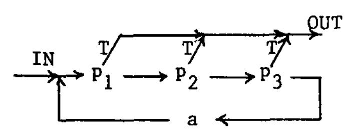
Let P_n-flow charts be defined recursively, for $n \geq 1$, as follows.

1. Any basic function unit is a P_n-flow chart.
2. If $F_1, F_2, \ldots, F_u$, for any $u \geq 0$, are P_n-flow charts and $p_1, p_2, \ldots, p_i$ for $i \leq n$, are basic predicate units, then any deterministic (1,1)-flow chart composed from the above is a P_n-flow chart (many occurrences of the same predicate are allowed).

Theorem 3: For any $n \geq 2$, the Ω_n-flow chart is not weakly equivalent to any P_{n-1}-flow chart composed from the same basic units.

Proof: For clarity, we prove it for $n = 3$ and the generalization will be obvious. Consider the Ω_3-flow chart, F, given below:

F:



We want to show that F cannot be weakly equivalent to any P_2-flow chart using only p_1, p_2, p_3 and a. If not, let there exist such a flow chart G. Consider the following interpretation:

p_1 is "$x = 2^n$ for some integer n?",

p_2 is "$x = 2^n + 2^{n-1}$ for some integer n?",

p_3 is "$x = 2^n + 2^{n-1} + 2^{n-2}$ for some integer n?",

a is "$x \leftarrow x + 1$", and $x \in N$.

As in Theorem 2, numbers of the form 2^n, $2^n + 2^{n-1}$, $2^n + 2^{n-1} + 2^{n-2}$ can be considered stops, and for any x, F(x) is the nearest higher stop of x.

Lemma 3.1: Same as Lemma 2.1

Lemma 3.2: Same as Lemma 2.2

Lemma 3.3: Same as Lemma 2.3

Lemma 3.4: Let H be any P_2-flow chart composed of the above basic units containing k a-units. For any m satisfying $2^{m-3} > k+1$, if $H(2^m-k-1) = 2^m$, and $H(2^m + 2^{m-1}-k-1) = 2^m+2^{m-1}$, then $H(2^m + 2^{m-1} + 2^{m-2}-k-1) > 2^m + 2^{m-1} + 2^{m-2}$ or undefined.

Proof: We prove the lemma by induction on the class of P_2-flow charts.

$$\begin{aligned} \text{Let}\quad x_1 &= 2^m-k-1 & s_1 &= 2^m \\ x_2 &= 2^m+2^{m-1}-k-1 & s_2 &= 2^m+2^{m-1} \\ x_3 &= 2^m+2^{m-1}+2^{m-2}-k-1 & s_3 &= 2^m+2^{m-1}+2^{m-2} \end{aligned}$$

Basis: Lemma 3.4 holds for $\to$ and $\to a \to$.

Inductive Step: Assume that the lemma holds for P_2-flow charts $H_1,H_2,\ldots,H_u$ having $k_1,k_2,\ldots,k_u$ - a-units respectively. Consider any P_2-flow chart H composed of $H_1,\ldots,H_u$ and any predicates q_1, $q_2 \in \{p_1, p_2, p_3\}$. Let H have k a-units; so $k = k_1 + k_2 + \ldots + k_u$. Any input x having an output, passes through some (possibly none) of the q_i's, then enters a H_j, and after certain transformations exits H_j and continues this process. We say that x has been processed by a global loop, if there exists a H_j such that H_j is entered at least two times.

Let H give correct outputs for both x_1 and x_2 .... (*)

For x_1, sequence of H_j's entered: $H_{i1},H_{i2},\ldots$

Case 1: Each H_{ij} increments its input by $\leq k_{ij}$. Since $s_1-x_1 > k = k_1 + k_2 + \ldots + k_u$, there should be a global loop. Let v be the minimum number such that $iv \in \{i1,i2,\ldots,iv-1\}$ (that is, we are considering the first global loop). Let the input to H_{iv} be $x_1 + \delta$. Then $\delta \leq k_{i1} + k_{i2} + \ldots + k_{iv-1} \leq k$. Hence by lemma 3.3, x_2 and x_3 will also arrive at H_{iv} with the corresponding values $x_2 + \delta$ and $x_3 + \delta$ respectively. The only way x_1 or x_2 could leave the global loop is through a T exit of q_1 or q_2. By (*) and lemma 3.1, $\{q_1,q_2\} = \{p_1,p_2\}$. Since x_3 also enters that loop, it could exit that loop only on a T condition of p_1 or p_2, i.e. T condition of q_1 or q_2, which takes it beyond s_3. Hence by lemma 3.1, $H(x_3) > s_3$ or undefined.

Case 2: Some H_{ij} increments its input by more than k_{ij}.

Let v be the minimum such j. Each of $H_{i1},H_{i2},\ldots,H_{iv-1}$ increments its input by no more than $k_{i1},k_{i2},\ldots,k_{iv-1}$ respectively. Let the input to H_{iv} be $x_1 + \delta$. Thus, as in the previous case, x_2 and x_3 will appear as $x_2 + \delta$ and $x_3 + \delta$ respectively at the input of H_{iv} (following the same path). H_{iv} increments $x_1 + \delta$ by more than k_{iv}. Hence by lemma 3.3, it increments each of $x_2 + \delta$ and $x_3 + \delta$ by more than k_{iv}. By lemma 3.2, $H_{iv}(x_2+\delta) \geq s_2$. Hence by lemma 3.1 and (*), $H_{iv}(x_2 + \delta) = s_2$. So by the hypothesis, $H_{iv}(x_3 + \delta) > s_3$ or undefined. Hence $H(x_3+\delta) > s_3$ or undefined. QED

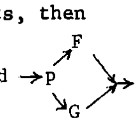
Proof of Theorem 3 continued: By lemma 3.4, no P_2-flow chart can give correct outputs for x_1,x_2, and x_3. Hence theorem 3 holds for $n = 3$.

For any n, the generalization of the proof should be obvious. The corresponding interpretation will be p_i: "$x = 2^k + \ldots + 2^{k-i+1}$ for some integer k?", for $1 \leq i \leq n$. QED

V. REPEAT-EXIT Constructs

For every $n \geq 0$, define RE_n-flow charts recursively as follows:

1. Any basic function is an RE_n-flow chart.
2. $\to$ EXIT i $\to$, $0 \leq i \leq n$, is an RE_n-flow chart.
3. If p is a predicate and F and G are RE_n-flow charts, then

   $\to F \to G \to$, and  are RE_n-flow charts.

   (BLOCK and IFTHENELSE)
4. If F is any RE_n-flow chart, then

   $\to$ RPT $\to$ F $\to$ END $\to$ is an RE_n-flow chart.

   (RPT-END loop)

Observe that RPT's and END's form matching pairs like parentheses in well parenthesized expressions.

We could easily define <u>levels</u> of RPT-END pairs, considering the innermost pair forming level 1.

Interpretation:

RPT, END and EXIT statements are NO OP's with respect to data transformation.

The control transformations associated with these statements are as follows:

1. RPT is a NO OP.
2. END returns control to the matching RPT, i.e. it has the same effect as "GO TO matching RPT".
3. EXIT i passes control to the output line of the END i levels higher. If i = 0, then it is simply a NO OP. If i levels do not exist, then the effect is the same as replacing i by the maximum possible existing level.

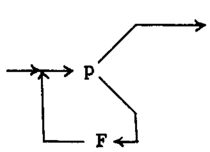
Example:

$$\underset{A_1}{\longrightarrow} \text{RPT} \underset{A_2}{\longrightarrow} \text{RPT} \underset{A_3}{\longrightarrow} \text{EXIT } i \underset{A_4}{\longrightarrow} \text{END} \underset{A_5}{\longrightarrow} \text{END} \underset{A_6}{\longrightarrow}$$

When control is at A_3: if $i = 0$, then control passes to A_4; if $i = 1$, then control passes to A_5; if $i \geq 2$, then control passes to A_6. If control is at A_4, then it passes to A_2.

For any $n \geq 0$, let us define another class of flow charts - <u>DRE_n-flow charts</u>. We allow all the constructs of <u>RE_n-flow charts</u> and in addition the DOWHILE construct. That is,

1. if F is an RE_n-flow chart, then F is a DRE_n-flow chart
2. if F is a DRE_n-flow chart and p is any predicate, then



is a DRE_n-flow chart

Remark: It can be easily seen that for any $n \geq 1$, any DRE_{n-1} flow chart is weakly equivalent to an RE_n-flow chart.

Knuth and Floyd [9] ask the following question: Does there exist a flow chart which cannot be weakly equivalent to any DRE_1-flow chart (i.e. allows DOWHILE and one level EXIT)? In the following Theorem we prove a result more general than their conjecture.

<u>Theorem 4</u>: For any $n \geq 1$, there exists an RE_{n+1}-flow chart which cannot be weakly equivalent to any RE_n-flow chart (using the same basic units and RPT, END, EXIT i, $0 \leq i \leq n$).

<u>Proof</u>: Consider the flow chart F given in Figure 1. We will establish that F is not weakly equivalent to any RE_n-flow chart, but weakly equivalent to an RE_{n+1}-flow chart. If possible, let there exist an RE_n-flow chart, G, weakly equivalent to F. The equivalence should hold for any interpretation of the predicate and the function units. We derive a contradiction for the following interpretation:

Input alphabet = $\{c,d\}$; Output alphabet = $\{[i,j,i+1,j],[i,j,i,j+1],[i,n+2,i,1],$

$[n+2,j,1,j] \ / \ i,j = 1,2,\ldots,n+1\} \cup \{c,d\}$

Thus the output alphabet consists of $(n+1)^2+2(n+1)+2$ symbols.

$p_{i,j} = p$ for $i,j = 1,2,\ldots,n+2$ where

p = "prefix of $x = c$?"

$a_{i,j,i',j'}$ = "delete the prefix symbol of x and attach symbol $[i,j,i',j']$ as a suffix symbol of x".

Function operations delete one symbol from the left of x and attach one symbol on its right. Hence the length of x remains invariant. The length of the c,d part of x cannot increase, since no function unit of F attaches a c or d to x. The non-c,d part of x can never decrease, since each operation attaches at least one non-c,d symbol, and never deletes more than one symbol.

In the following discussion let us restrict ourselves to those inputs which lead to termination and produce outputs with non-null c,d prefix, i.e. inputs of the form yz, $y \in \{c,d\}^*$, $z \in \{c,d\}^+$ which have outputs of the form zy' where y' is some string of non-c,d output symbols.

<u>Intuitive Proof:</u>

Take an inner-most RPT-END loop of G. Is that loop "effective"? That is, does there exist an input which will make use of that loop (hits that END arbitrarily many times) without going out of that loop, as it processes more and more c's and d's of x ? An in-effective loop can be easily deleted. We may assume, then, that this inner-most loop is effective. This loop cannot leave x unchanged, so it deletes some prefix c's and d's and attaches $[i_1,j_1,i_2,j_2][i_2,j_2,i_3,j_3]\ldots[i_{m-1},j_{m-1},i_m,j_m]$ $[i_m,j_m,i_1,j_1]$ to the suffix of x . That is, if we consider $[u,v,u',v']$ as a path from (u,v) to (u',v') , then the attached string forms a path starting at some (i_1,j_1) and ending at the same (i_1,j_1) on the "grid" of F. This closed path has a length of at least n+2. On this path we will be able to pick n+2 points, $(u_1,v_1),(u_2,v_2),\ldots,(u_{n+2},v_{n+2})$, one on each of (a) n+2 columns or (b) n+2 rows, due to the particular "grid structure" of F. In case (a), we will consider inputs like x, but with $(d^{n+2})^k$ (for a suitable k) inserted just after the input symbol which triggered attaching the symbol $[.,.,u_i,v_i]$ for $i = 1,2,\ldots,n+2$. (In $[.,.,u_i,v_i]$ we do not care for the first two symbols). In case (b) we insert $(c^{n+2})^k$ at the proper points. In either case we will show that G cannot properly handle all of the n+2 inputs. Admittedly this is a very vague description, but it gives all the main ingredients of the following proof.

Lemma 4.1: If G $(\equiv_w F)$ transforms any $x = yz$ into zy', then y' should be of the form $[i_1,j_1,i_2,j_2][i_2,j_2,i_3,j_3]\ldots$, where $(i_{k+1},j_{k+1}) = (i_k+1,j_k)$ or (i_k,j_k+1) where all sums are "end-around" at n+2 , i.e. $n+3 = 1$ (similar to mod n+2).

Proof: Follows from observing the flow chart F . QED

Lemma 4.2: For any sequence of the form $[i_1,j_1,i_2,j_2]\,[i_2,j_2,i_3,j_3]\ldots[i_m,j_m,i_{m+1},j_{m+1}]$, where $(i_{m+1},j_{m+1}) = (i_1,j_1)$ and $(i_{k+1},j_{k+1}) = (i_k+1,j_k)$ or (i_k,j_k+1), sums end-around at n+2, there are n+2 pairs (u_1,v_1), (u_2,v_2), $\ldots$, $(u_{n+2},v_{n+2}) \in \{(i_1,j_1),(i_2,j_2),\ldots,(i_m,j_m)\}$ such that

a) $u_1,u_2,\ldots,u_{n+2}$ are all distinct or

b) $v_1,v_2,\ldots,v_{n+2}$ are all distinct

(This corresponds to picking n+2 points in the closed loop).

Proof:
If $(i_2,j_2) = (i_1+1,j_1)$, then it can be easily verified that $(1,v_1),(2,v_2),\ldots,(n+2,v_{n+2}) \in \{(i_1,j_1),\ldots,(i_m,j_m)\}$ for some $v_1,v_2,\ldots v_{n+2}$; otherwise $(i_2,j_2) = (i_1,j_1+1)$, and it can be easily seen that $(u_1,1),(u_2,2),\ldots,(u_{n+2},n+2) \in \{(i_1,j_1),\ldots,(i_m,j_m)\}$ for some $u_1,u_2,\ldots,u_{n+2}$. QED

Lemma 4.3: In any RE_n-flow chart H, if any sub-flow chart $\to RPT \to H' \to END \to$ is replaced by $\to RPT \to H' \to H' \to END \to$ then the new flow chart is weakly equivalent to H.

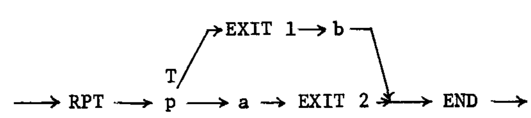
Proof: Follows from the interpretations of RPT and END. QED

Notation: Let the RPT and END of a block named ω be RPT_ω and END_ω respectively.

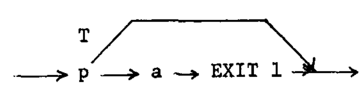
Lemma 4.4: There exists an RE_n-flow chart G_1 weakly equivalent to G in which for each inner-most RPT-END block, ω , there exists an input which results in the execution of END_ω.

Proof: This might be violated for an inner-most block ω because control never enters ω (i.e. RPT_ω never gets executed) or after entering ω it always exits and never executes END_ω. In either case weak equivalence is preserved under the following transformation:
Replace the block $\to RPT \to H \to END \to$ by $\to H' \to$, where H' is H modified as follows:

Replace any statement EXIT i, $i \geq 2$, by EXIT i-1. Delete any statement EXIT 1 and connect its input to the OUT of H. For example,



gets transformed to



The new flow chart has one less RPT-END block.

The lemma must hold after a finite number of applications of this procedure. since there are a finite number of RPT-END pairs to begin with. QED

Lemma 4.5: There exists an RE_n-flow chart G_2, weakly equivalent to G_1, in which for each innermost RPT-END block, ω, there exists an input which results in the execution of END_ω at least two times and in that computation sequence there exists 2 consecutive such executions in between which control is within ω.

Proof: Let some ω fail to satisfy the lemma. Then replace $\omega = \to RPT \to H \to END \to$ by $\omega' = \to RPT \to H \to H \to END \to$, which preserves weak equivalence (lemma 4.3), and delete $RPT_{\omega'}$, and $END_{\omega'}$, as in lemma 4.4. The lemma must hold eventually. QED

Consider an inner-most loop ω of G_2. By lemma 4.5, there exists an x which results in the execution of RPT_ω at least 3 times (one more than END_ω) and in between these three consecutive times control will be within ω. Any symbol of input x will be either c or d. Hence the prefix symbol of x must be the same for two of those executions of RPT_ω. Thus for input $x = y_1y_2z$, when it executes RPT_ω, x will be y_2zy_1' and $zy_1'y_2'$, where prefix symbol of y_2zy_1' is the same as that of $zy_1'y_2'$ (and during the processing of y_2 part of x, control will be within ω).

Lemma 4.6: In the above, a) $y_2 \neq \Lambda$, b) y_2' must be of the form

$[i_1,j_1,i_2,j_2][i_2,j_2,i_3,j_3]\ldots[i_m,j_m,i_{m+1},j_{m+1}]$

where b1) $(i_{k+1},j_{k+1}) = (i_k+1,j_k)$ or (i_k,j_k+1) where $n+3$ is taken as 1 and b2) $(i_{m+1},j_{m+1}) = (i_1,j_1)$.

Proof: If a) is not satisfied, then the processing of x never terminates. b1) holds from lemma 1.1. If b2) is not true then the output for input $y_1y_2y_2z$ would be incorrect. QED

Now let us complete the proof of theorem 4: By lemmas 4.2 and 4.6, there exist inputs $x_1z_1, x_2z_2,\ldots x_{n+2}z_{n+2}$ which while being processed by G_2 will have intermediate values $z_1w_1\,[.,.,u_1,v_1],\ z_2w_2[.,.,u_2,v_2],\ \ldots,$ $z_{n+2}w_{n+2}\,[.,.,u_{n+2}v_{n+2}]$ respectively, where $u_j = j$ for $j = 1,2,\ldots,n+2$, or $v_j = j$ for $j = 1,2,\ldots,n+2$. In addition, in each case control is within ω even before testing the corresponding prefix symbol of the z_i's.

Let us consider the first alternative; that is $u_j = j$ for $j = 1$ to $n+2$. What will happen for inputs $x_i(d^{n+2})^kz_i$, $i = 1$ to $n+2$, where k is chosen such that $(n+2)k >$ number of basic units in ω ?

From the equivalence of F and G_2, G_2 should transform $x_{n+2}(d^{n+2})^kz_{n+2}$ to $(d^{n+2})^kz_{n+2}w_{n+2}\,[.,.,u_{n+2},v_{n+2}]$ and terminate the processing (which requires an exit out of ω). For $i = 1,2,\ldots,n+1$, G_2 should transform $x_i(d^{n+2})^kz_i$ to the intermediate value $z_iw_i\,[.,.,u_i,v_i]([u_i,v_i,u_i,v_i+1]\ldots$ $\ldots[u_i,v_i+n+1,u_i,v_i])^k$. Note that the suffix symbols attached to the i^{th} input (corresponding to $(d^{n+2})^k$ part) are all distinct from the suffix symbols attached to another input corresponding to its $(d^{n+2})^k$ part. This can be seen by noting that no two columns of F have a common function operation. Just after processing each x_i part control is within ω. While processing the $(d^{n+2})^k$ part it cannot, for any given j, execute EXIT j for any two distinct sequences. Otherwise control switches to the same exit point and processes the same prefix symbol (i.e. symbol d) and hence performs the same function operation for two distinct sequences, which contradicts the previous observation. Hence there can be exits out of ω for at most $n-1$ of the first $n+1$ sequences. So, for at least two of the first $n+1$ sequences, the processing of $(d^{n+2})^k$ part should be done within loop ω. For such sequences END_ω is processed at least once, since $(n+2)k >$ number of basic units in ω and ω is an innermost loop. After its execution both sequences have the same input prefix symbol with control at the same point, i.e. at RPT_ω. Hence in both cases the same basic function is applied, a contradiction.

We argue similarly the other alternative; i.e. $v_j = j$ for $j = 1$ to $n+2$. Hence F is not weakly equivalent to any RE_n-flow chart.

In the following we show that F is weakly equivalent to an RE_{n+1}-flow chart. Let us call the off-diagonal predicates on the grid of F $q_1, q_2, \ldots, q_{n+1}$; where $q_i = p_{n+2-i,i}$ for $1 \le i \le n+1$. There is no loop entirely above or below the off-diagonal and any path from above the off-diagonal to below the same must pass through one of $q_1, \ldots, q_{n+1}$. Hence any loop on the grid of F must pass through at least one of $q_1, \ldots, q_{n+1}$. (Such a property does not hold for the main diagonal: $p_{1,1}, p_{2,2}, \ldots, p_{n+1,n+1}$).
Hence if we "expand" F into an acyclic graph starting at each q_i and terminating whenever we encounter one of $q_1, \ldots, q_{n+1}$, the expansion gives a finite flow chart in each case.

Now we define functions $H_1, \ldots, H_{n+1}$ each mapping permutations of $(1,2,\ldots,n+1)$ into (1,1) flow charts. Let $H_k(i_1, i_2, \ldots, i_k, \ldots, i_{n+1})$ be a flow chart H. H will process data appearing at the input to q_{i_1} in F as follows:

1) if the data path followed in F leads to OUT without encountering any $q_{i_{k+\delta}}$, $1 \le \delta \le n+1-k$, then H transforms the input exactly as F does, and then transfers control, via an EXIT, to $n+1-k$ levels above the level of H.
2) if the data path followed in F encounters some $q_{i_{k+\delta}}$, $1 \le \delta \le n+1-k$, then H transforms the input exactly as F does up to the first such encounter, and then transfers control, via an EXIT, to $\delta-1$ levels above the level of H.

Verify that the following inductive construction is correct.

For $k = 1, \ldots, n$, let $H_{k+1}((i_1, \ldots\ldots, i_{n+1})) =$
$\rightarrow \mathrm{RPT} \rightarrow H_k((i_1, \ldots, i_{n+1})) \rightarrow$

$\rightarrow H_k((i_{k+1}, i_2, \ldots, i_k, i_1, i_{k+2}, \ldots, i_{n+1})) \rightarrow \mathrm{END} \rightarrow$.
Hence if H_1 is properly defined, then $H_{n+1}((1, \ldots, n+1))$ will be weakly equivalent to F by property 1).

"Expand" the flow chart, calling the input of the expansion IN_1, by following the connections of F, starting at q_{i_1} and terminating whenever one of $q_{i_1}, \ldots q_{i_{n+1}}$ is encountered. There are two places at which we must insert EXIT's in order to preserve properties 1) and 2) above.

i. Connect the False exit of each $p_{n+2,i}$ to an EXIT n+1, and then connect this to the input of $p_{1,i}$ (this is purely arbitrary since the execution of this EXIT will remove control from H). The true exit of each $p_{i,n+2}$ is connected to the input of $p_{i,1}$.
ii. Replace each terminating q_{i_k} by an EXIT k-1, and connect all these EXIT's to a single node, OUT_1.

Then let $H_1(q_{i_1}, \ldots, q_{i_{n+1}}) =$

$$\longrightarrow \mathrm{RPT} \xrightarrow{IN_1} \begin{array}{c}\text{Expanded}\\\text{Flow Chart}\end{array} \xrightarrow{OUT_1} \mathrm{END} \longrightarrow$$

This technique is illustrated by an example in Figure 2. Note that in case i) each exit will transfer control to exactly n levels above the enclosing RPT-END pair; and in case ii) control will be transferred exactly k-2 levels above the enclosing RPT-END pair when the data path for an input at q_{i_1} in F would encounter q_{i_k} (in $q_{i_1}, \ldots, q_{i_{n+1}}$). Thus conditions 1) and 2) are preserved, and H_1 is properly defined.

QED

<u>Theorem 5</u>: For every flow chart having n basic predicate units, there exists a weakly equivalent RE_n-flow chart (using the same basic elements and RPT, END, and EXIT i, $i = 1, \ldots, n$).

<u>Proof</u>: The proof is closely related to Böhm and Jacopini construction [2]. One should not have any problem in replacing the control variables used in Böhm and Jacopini's construction by RPT, END, EXIT constructs applicable to this class of flow charts. We leave the details to the reader. QED

VI. Generalized REPEAT-EXIT Model

In the above model, we allowed exits out of a RPT-END block to the n immediately enclosing levels. But in programming applications cases occur which need further generalization. For example, when a trap occurs, irrespective of where it occurs, control passes to the trap location. Hence we use the following generalization:

GRE_n-flow charts:

Instead of just allowing EXIT i, $i \leq n$, we allow EXIT i for any i. But the restriction is that in any level of RPT-END there can be at most n distinct i's occurring as exit numbers.

Theorem 6: For any n, there exists a GRE_{n+1}-flow chart which cannot be weakly equivalent to any GRE_n-flow chart.

Proof: The proof of Theorem 4 remains valid for this generalization.

VII. Top-Down Programming Models

Program development can be generally classed as either top-down or bottom-up. In top-down programming, the original problem is decomposed into a number of inter-connected sub-problems. We iterate the process on each one of these sub-problems, until we come to the basic instruction level of the programming language. In bottom-up programming we do just the reverse; we take some basic instructions and assemble them into a module which is used as a sub-module for a bigger module. We continue this process until the problem is realized.

Thesis: For "Reliability in Programming" - at each step in the development of a program:

I. A module should be decomposed into as few sub-modules as possible in the top-down case; or composed from only a few sub-modules in the bottom-up case. This is particularly important for the multi-exit sub-modules.

II. The number of entries (input lines) and exits (output lines) of each sub-module should be as small as possible.

Thus the following class of TD_n-flow charts is a logical consequence of the general philosophy of top-down programming. We recursively define TD_n and $(1,2)_n$-flow charts as follows:

1. A basic function is a TD_n-flow chart
2. A basic predicate p is a $(1,2)_n$-flow chart
3. A deterministic interconnection of at most n $(1,2)_n$-flow charts and any number of TD_n-flow charts is a $(1,2)_n$-flow chart if it is a (1,2) flow chart, and is a TD_n-flow chart if it is a (1,1) flow chart.

Theorem 7: For any $n \geq 1$, any TD_n-flow chart is weakly equivalent to an RE_n-flow chart.

Proof: In converting from $(1,2)_n$ to TD_n-flow charts, we will introduce new function instructions STOP 1 and STOP 2 which, intuitively, will act as position markers. These instructions are used only in converting $(1,2)_n$-flow charts, and will not be present in converted TD_n-flow charts.

We convert TD_n-flow charts and $(1,2)_n$-flow charts preserving the following properties:

1. Any TD_n-flow chart is converted into a weakly equivalent RE_n-flow chart.
2. Any $(1,2)_n$-flow chart F, is converted into an RE_n-flow chart, G (STOP 1 and STOP 2 are also allowed). Any computation sequence in G will not contain any function transformation (including STOP 1) after a STOP 1 or STOP 2. Label the two output lines of F OUT 1 and OUT 2. Then there is a 1-1 correspondence between computation sequences in F ending at OUT i, and computation sequences in G containing STOP i. This correspondence is such that data undergoes the same function transformations in both the sequences, ignoring STOP i.

This conversion is achieved inductively as follows:

Basis:

1) Convert any basic function → a → into → a →.

2) Convert any basic predicate → p ↗↘ into

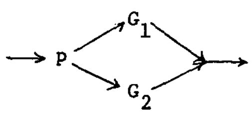
→ p ↗ STOP 1 ↘ →
    ↘ STOP 2 ↗

1) General TD_n Construct:

For any TD_n-flow chart, composed of i

$(1,2)_n$ sub-flow charts and any number of TD_n sub-flow charts, consider the i $(1,2)_n$ sub-flow charts to be distinct predicate units and the TD_n sub-flow charts to be distinct function units. Use Theorem 5 to convert the resulting flow chart into a weakly equivalent RE_n-flow chart R (note ($i \leq n$ by definition of TD_n). By induction, each of the original TD_n sub-flow charts corresponds to an RE_n-flow chart; hence replace the function units in R by the corresponding RE_n-flow charts.

Convert each of the IFTHENELSE constructs in R , starting at the innermost, as follows:



Note: G_1 and G_2 are already RE_n-flow charts since we are starting at the innermost const

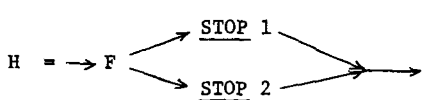
Let H be the $(1,2)_n$ sub-flow chart corresponding to p ; by induction we can assume that H is already converted to H', an RE_n-flow chart which satisfies property 2. Replace the above IFTHENELSE construct by

$$\rightarrow H'' \rightarrow$$

where H" is obtained from H' by substituting G_1 and G_2 for STOP 1 and STOP 2 respectively. Note that all STOP statements have been eliminated from H" . Thus, after all of these IFTHENELSE forms have been converted, the final RE_n-flow chart must be composed only of RPT, END, EXIT i statements and units from the original TD_n-flow chart.

2) General $(1,2)_n$ Construct:

Convert any $(1,2)_n$-flow chart F into a TD_n-flow chart



We will assume by induction that each of the $(1,2)_n$ and TD_n sub-flow charts in F may be converted to RE_n-flow charts. (STOP 1 and STOP 2 are new basic functions.) Then convert H to an equivalent RE_n-flow chart, F' , as in step 1. Note that weak equivalence of H and F' allows us to assume that no function transformations can occur after any STOP i statement in a computation sequence for F'. Now replace STOP i by STOP i. QED

As a consequence of theorems 4 and 7, we have the following theorem.

Theorem 8: For any n, there exists a flow chart which is not weakly equivalent to any TD_n-flow chart (using the same basic units).

VIII. Characterization

What makes a flow chart non-reducible to an equivalent D-flow chart? We are interested in the structural property that characterizes reducibility, i.e. we want to eliminate considerations based on specific data structures and relationships between various basic operations. Hence we assume that the flow chart has the following property:

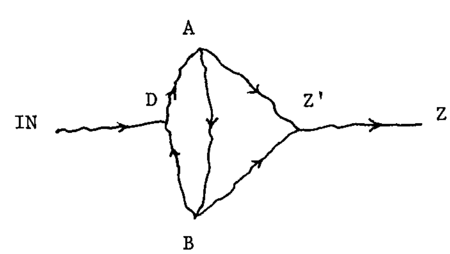
1. No two basic units have the same name
2. Every branch has a function label.

We name such flow charts S-flow charts (S for structure). Theorem 9 characterizes the sub-class of S-flow charts, each member of which is reducible to a weakly equivalent D-flow chart, using the same basic components.

Def: A loop AB is reachable iff there is a path from the entry point IN, to some point (hence all points) on that loop.

Def: A path is an exit of a loop iff it is of non-zero length; it starts at some point on the loop; no other point in the path is on the loop, and the path ends at the OUT of the flow chart. The common point of the loop and the path is an exit-point of the loop.

Theorem 9: An S-flow chart is non-reducible to an equivalent D-flow chart iff there exists a reachable loop with at least 2 distinct exit points, i.e. there exists a sub-graph of the form:



where each wiggly line is a path; loop AB is reachable from IN; and there exist paths AZ'Z, BZ'Z which are exits of loop AB.

IX. Conclusions

We have made an effort to study the characteristics of certain classes of flow charts. In our study we used one common definition for weak equivalence. There are other interesting notions of weak equivalence. For example, Ashcroft and Manna [1] allow predicates of the form $p(\underline{a})$ where p is a basic predicate and $\underline{a}$ is a composition of basic functions. Such predicates ask "is $p(\underline{a}(x))$ true?" but do not change x itself. They also allow boolean combinations of such predicates to be a prediate. Most of our results hold even for this variation.

Some of the problems we have not considered, but which deserve further investigation are: (Structurally) Characterize the class of flow charts reducible to BJ_n, P_n, TD_n, RE_n and GRE_n flow charts etc.

Acknowledgements

The author expreses deep appreciation to Dr. Harlan Mills who introduced him to the general area of Structured Programming. He is also grateful to Mr. Lee Hoevel for his careful reading of the manuscript and his many constructive comments. Thanks are due to Mrs. Gloria Baker for her excellent typing. This research was supported by National Bureau of Standards.

References

1. Ashcroft, E. and Manna, Z., The translation of "go to" Programs, Booklet TA-2, Preprints of the IFIP Congress 1971, Ljubljana, Yugoslavia, August 1971.

2. Böhm, C. and Jacopini, G., Flow diagrams, Turing Machines and Languages with only two formation rules, CACM, May 1966, pp. 366-371.

3. Bruno, J. and Steiglitz, K., The expression of algorithms by charts, JACM, July 1972, pp. 517-525.

4. Cooper, D. C., Böhm and Jacopini's reduction of flow charts, CACM, August 1967, p. 463.

5. Cooper, D.C., Some transformations and standard forms of graphs, with applications to computer programs. Machine Intelligence 2, 1967, pp. 21-32.

6. Dijkstra, E. W., GoTo statement considered harmful, CACM, March 1968, pp. 147-148.

7. Dijkstra, E. W., Notes on Structured Programming, T.H. Report 70-WSK-03 (Technological University Eindhoven, Netherlands, 1970)

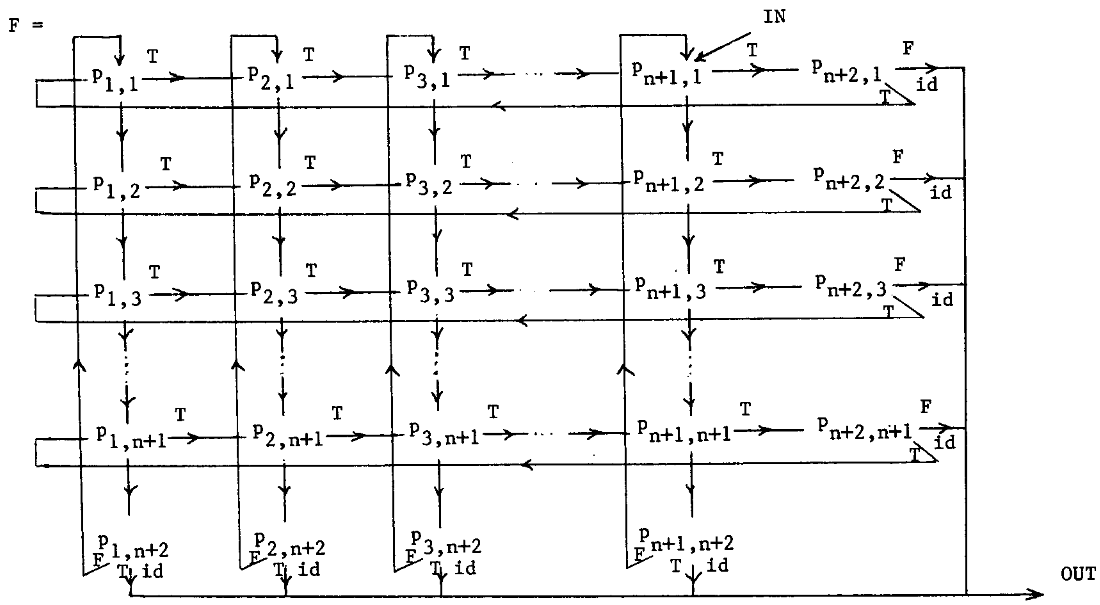
8. Hecht, M. S. and Ullman, J. D., Flow Graph reducibility, Proc. of 4th Annual ACM Symp. on Theory of Computing 1972, pp. 238-250.

9. Knuth, D. E. and Floyd, R. W., Notes on Avoiding "Go To" statements, Report No. CS-148, C.S. Dept., Stanford University, 1970.

10. Kosaraju, S. Rao, Analysis of Structured Programs, Tech. Report, Electrical Engineering Department, The Johns Hopkins University, November 1972.

11. Mills, H. D., Mathematical Foundations for Structured Programming, FSC 72-6012 (Federal Systems Division, IBM Corp., Gaithersburg, Md. 1972).

12. Wulf, W. A., Programming without the Go To, Booklet TA-3, Preprints of the IFIP Congress 71, Ljubljana, Yugoslavia, August 1971



For the connections shown above: Function connecting $p_{i,j}$ and $p_{i',j'} = a_{i,j,i',j'}$

Figure 1

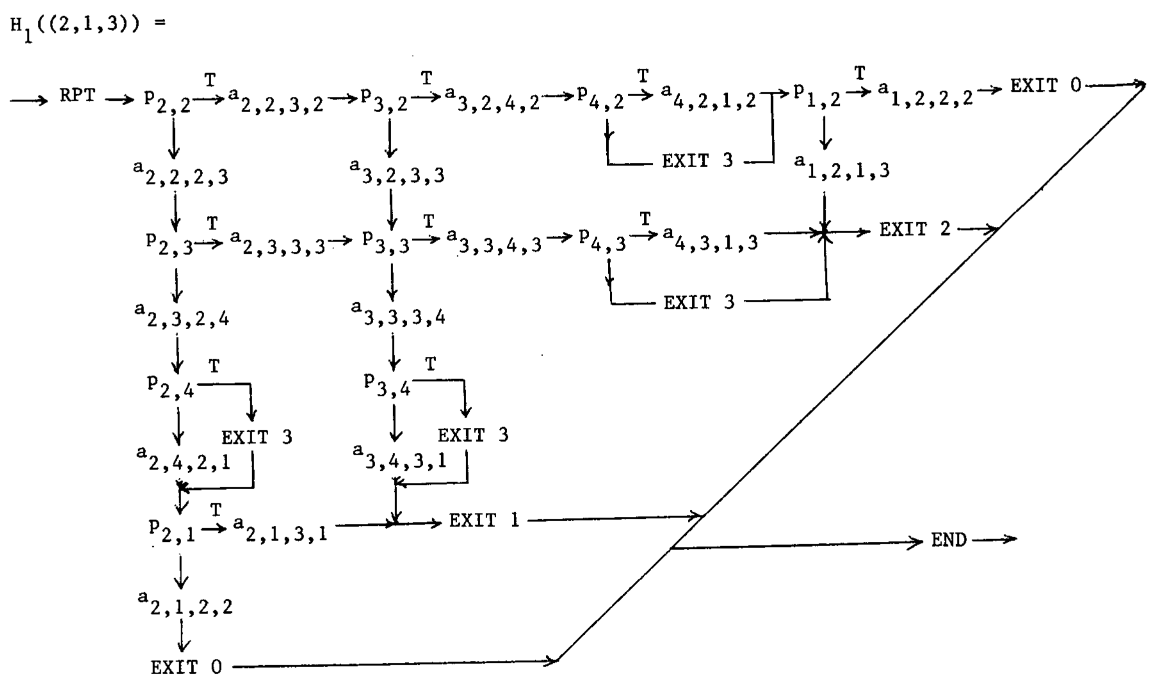


Figure 2