SNAP
A COMPUTER PROGRAM FOR GENERATING
SYMBOLIC NETWORK FUNCTIONS

P. M. Lin
G. E. Alderson

School of Electrical Engineering
Purdue University
Lafayette, Indiana

Supported by
National Aeronautics and Space Administration
Grant NGL 15-005-021
# Table of Contents

## I. Introduction ................................................................. 1

## II. A General Discussion of the Basic Algorithms ......................... 6

### II-1. Formulating the Signal-Flow Graph (SFG) .......................... 6
   a. Data Required .............................................. 6
   b. Finding a Tree ........................................... 7
   c. Rules for Formulating the Compact SFG .................... 8
   d. The Gain Formula for "Closed" SFG ......................... 11

### II-2. Manipulating SFG Branch Weights ................................. 12

### II-3. Generating First Order Loops ..................................... 16
   a. General Description ....................................... 16
   b. A detailed Description of the Path-Finding Algorithm .... 18

### II-4. Generating Nontouching Loops of Order Two or More .............. 21

## III. User's Guide ............................................................... 28

### III-1. Information Needed by User ...................................... 28

   Appendix A (A Sorting Technique for Handling Multi-
   Input, Multi-Output Networks) .................................. 36

   Appendix B (A Brief List of Limitations on the
   Size and Type of Network Allowed) ............................ 39

   Appendix C (Selecting a "Good" Tree) ........................... 40

### III-2. Modifying the Dimension of Arrays ............................. 42

## IV. Programmer's Guide ....................................................... 47

### IV-1. Definitions ........................................................ 47

### IV-2. Flow Charts ....................................................... 50

### IV-3. Program Listing .................................................... 100

## References ................................................................. 124
I. INTRODUCTION

The majority of computer aided circuit analysis programs belong to the class of "numerical programs", that is, the output is some numerical value. At the time of our research a few programs, most notably ANPL(8)*, NASAP(9), and CORNAP(10) could generate network functions as rational functions of s but did not allow the value of any element to be left in symbol form. The research project reported here represents, we believe, the first effort to generate symbolic network functions. By a symbolic network function we mean

\[
\frac{V_{out}}{V_{in}}, \frac{I_{out}}{I_{in}}, \text{or} \frac{I_{out}}{I_{in}} \text{ as a ratio of two polynomials of one of the following types:}
\]

(1) all network element values are represented by symbols (the symbols need not all be different)

Examples:

\[
\frac{V_{out}}{V_{in}} = \frac{s^2 LRC}{s^2 2LRC + s(L+R^2 C) + R}
\]

\[
\frac{V_{out}}{V_{in}} = \frac{Z Y R}{2 Z Y R + Z + R^2 Y + R}
\]

(2) some element values are specified numerically, some symbolically,

Example:

\[
\frac{V_{out}}{V_{in}} = \frac{s^2 R}{s^2 2R + s(.5x10^6 + 150R^2) + .75x10^8 R}
\]

or

(3) all element values are given numerically,

Example:

\[
\frac{V_{out}}{V_{in}} = \frac{s^2}{2s^2 + 2x10^4 s + .75x10^8}
\]

There are many reasons why one may be interested in totally or partially symbolic network functions. The following presents a few of the more important ones.

(1) Insight. To illustrate the added "insight" symbolic programs can provide
in comparison to numerical type programs, suppose we have been asked to verify that the network in Fig. 1 is a negative impedance converter for large $\beta$, i.e.,

$$Z_{in}(s) \rightarrow -Z_L(s) \quad \text{as} \quad \beta \rightarrow \infty$$  \hspace{1cm} (1)

To verify (1) with some degree of certainty using a numerical program would require evaluating $Z_{in}(s)$ (and $Z_L(s)$) for many different values of $\beta$ and frequency $\omega$, a time consuming process at best since most programs must completely re-evaluate the network response for every relatively large change in parameter values. Furthermore, the resulting verification would only be valid for the particular structure and component values chosen for $Z_L(s)$. With a symbolic program, one computer run gives the symbolic transfer function

$$Z_{in}(s) = Z_L(s) \left( \frac{2 + \beta}{2 - \beta} \right)$$

from which (1) follows immediately.

(2) Error Control. To demonstrate how a symbolic program can be used to effectively control round-off error, consider the differential amplifier \textsuperscript{(1)}
shown in Figures 1 and 2 of Section II-1. If network branches 1, 2, 5, and 11 are chosen as the tree for deriving the signal-flow graph (SFG), then the set of nontouching loops (all orders with sign attached) which belong to the numerator of the low frequency transfer function

\[
\left. \frac{I_{\text{out}}}{V_{\text{in}}} \right|_{s=0} = \frac{N}{\Delta}
\]

is given by

\[
\begin{pmatrix}
R_1 R_3 A_2 & -R_1 R_3 A_1 & R_1 R_3 A_1 & -R_1 A_1 & -R_3 R_1 A_2 & R_1 A_2 \\
(RE)^2 R_2 & (RE)^2 R_2 & (RE)^2 R_2 & RER_2 & (RE)^2 R_2 & RER_2
\end{pmatrix}
\]

and the corresponding set for the denominator is given by

\[
\begin{pmatrix}
R_1 & R_1 & R_3 & R_3 & R_1 R_3 & R_1 R_3 & R_1 R_3 & R_1 R_3 \\
R_2 & R_2 & RE & RE & R_2 RE & R_2 RE & R_2 RE & R_2 RE
\end{pmatrix}
\]

Letting \( A_2 = A_1, R_1 = 5K, R_2 = 15K, R_3 = 10K, \) and \( RE = 25 \), evaluate \( N \) and \( \Delta \) by summing the terms in the order given in the above sets keeping each number generated to 8 significant digits. Then

\[
N = A_1 [5.3333333 - 5.3333333 + 5.3333333 - 0.13333333 - 5.3333333 + 0.13333333]
\]

\[
= 3.3 \times 10^{-8} A_1
\]

and \( \Delta = 1335 \)

Thus

\[
\left. \frac{I_{\text{out}}}{V_{\text{in}}} \right|_{s=0} = \frac{3.3 \times 10^{-8} A_1}{1335}
\]

which is incorrect since \( N = 0 \). Although the above transfer function was derived using SFG theory, round-off errors which cause erroneous results can occur in any computer program restricted to numerical evaluation, and are generally very difficult to predict or control. Because round-off error enhancement in the evaluation of network functions often occurs as a result of widely separated values of some of the network elements, one method of error control would be to leave such element values in
symbolic form. This technique can be applied to the above example by noting that RE should be kept as a symbol since its value is considerably less than the other resistance values. Thus, keeping RE as a symbol and re-evaluating N gives

\[ N = A_1 \left[ \frac{3333.3333}{(RE)^2} - \frac{3333.3333}{(RE)^2} + \frac{3333.3333}{(RE)^2} - \frac{.33333333}{RE} \right. \\
\left. - \frac{3333.3333}{(RE)^2} + \frac{.33333333}{RE} \right] = 0 \]

That is,

\[ \frac{I_{out}}{V_{in}} \bigg|_{s=0} = 0 \]

(3) Sensitivity Analysis. Sensitivity of the network function to changes in a particular network parameter can be found using a symbolic program by keeping this parameter as a symbol and then performing the required differentiation. Although there exist powerful numerical techniques for sensitivity analysis, the above procedure using a symbolic program has the particularly desirable feature of being less susceptible to round-off errors.

(4) Parameter Variation. Suppose we wanted to evaluate the network function for many different values of one or more network parameters. Using a symbolic program, we could leave these parameters in symbol form and then efficiently and accurately perform the large number of required evaluations on the resulting symbolic network function. On the other hand numerical programs now available must re-derive the transfer function for every relatively large parameter change.

(5) Iterative Piecewise Linear Analysis of Resistive Nonlinear Networks. Part of this powerful analysis technique requires the solution of a
resistive linear network where some resistances and some d-c sources are kept in symbol form.

The primary objective* of this project has been the development of some new or improved concepts needed to make a symbolic network analysis program efficient with respect to program storage and execution time. The project culminated in the program SNAP (Symbolic Network Analysis Program) which finds symbolic network functions for networks containing R, L, and C type elements and all four types of controlled sources. SNAP contains the following unique features:

1. The extensive use of a path-finding algorithm in place of matrix operations,
2. Efficient techniques for finding all loops of the SFG and for enumerating all higher order loops,
3. The use of the "compact signal-flow graph" instead of the "primitive signal-flow graph", and
4. A simple coding technique which is used
   (a) manipulate symbols thereby allowing the complete program to be written in Fortran (another important aspect of the coding scheme is that it permits repeated symbols to be treated as one symbol), and
   (b) determine whether or not loop sets touch in the algorithm for enumerating higher order loops.

New techniques for handling multi-inputs and multi-outputs are also presented in this report although they have not yet been incorporated into the program SNAP.

*At about the same time the results of this project were disseminated, another symbolic program (1) (by coincidence also called SNAP) whose primary concern is the on-line use for design purposes made its appearance.
II. A GENERAL DISCUSSION OF THE BASIC ALGORITHMS

II-1. Formulating the Signal-Flow Graph (SFG)

a. Data Required

A SFG is generated by SNAP (Symbolic Network Analysis Program) from data specifying the topological structure of the network, the input-output variables, and the characteristics of each network branch. The network topology is described by

(a) A unique number for each branch, and
(b) the initial and terminal nodes of each branch as determined by the assigned current direction.

The input to the network must be a single independent voltage or current source and the output requested must be the voltage or current associated with a network branch or the voltage between any two nodes of the network.* Finally, each branch is characterized by

(a) a symbol which specifies its type, i.e.,
   passive branches: R, G, L, C, Y, Z
   control sources: VV, VC, CV, CC
   independent sources: E, I

(b) a symbol representing the branch name together with the branch value if specified, and

(c) the branch number of the control (for dependent sources only)

As an example, consider the network given below (from a paper by A. DeMari(1)).

---

* Refer to Appendix A of section III-1 for a discussion on how to handle multi-inputs and multi-outputs.
Table 1 (Network Data)

<table>
<thead>
<tr>
<th>Branch Type</th>
<th>Branch Number</th>
<th>Initial Node</th>
<th>Terminal Node</th>
<th>Symbol</th>
<th>Value</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>R1</td>
<td>$5 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>R1</td>
<td>$5 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>R2</td>
<td>$15 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>A1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>A2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>RE</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>RE</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>R3</td>
<td>$10 \times 10^3$</td>
<td></td>
</tr>
</tbody>
</table>

b. Finding a Tree

The formulation of a SFG starts with the choice of a network tree. The selection of network branches to be used in the tree is made as follows:

Independent voltage sources and controlled voltage sources are the first ones to be used. Then come the passive RLC elements in any order. In choosing the (J+1)th branch, the undirected graph formed by the J branches already selected is tested to determine whether a path exists between the two terminal nodes of the (J+1)th branch. If so, the branch under consideration is disqualified. If not, the (J+1)th branch is added to the tree. Let $n$ be the number of nodes
of the network graph. When \( n-1 \) branches are successfully chosen by the above process, we have obtained a tree.

As an example, consider the network of Fig. 2. If, following the selection of the voltage source, the passive branches are examined in the order by which they are listed, the tree shown in Fig. 3 results

\[
\text{Tree branches:} \\
\begin{align*}
\text{b}_1 &= 1, \quad \text{b}_2 = 2, \quad \text{b}_3 = 5, \quad \text{b}_4 = 9 \\
\text{Links:} \\
\ell_1 &= 3, \quad \ell_2 = 4, \quad \ell_3 = 6, \quad \ell_4 = 7 \\
\ell_5 &= 8, \quad \ell_6 = 10, \quad \ell_7 = 11
\end{align*}
\]

Figure 3

It is important to note that the complexity of the SPG and consequently the time required to evaluate the transfer function depends on the tree selected. A brief summary of the rules for choosing a "good" tree is given in Appendix C at the end of Section III-I.

c. Rules for Formulating the Compact SFG

A "compact SFG" is a signal-flow graph whose node variables consist only of tree branch voltages and link currents except when additional nodes are needed for control sources or for the output variable. This type of SFG can be more efficiently evaluated than the so-called primitive SFG which contains one node for the branch voltage and another for the branch current.

The compact SFG is constructed according to the following rules: (An example as derived from Fig. 2, Fig. 3, and Table 1 is given in Fig. 4).

Rule (1): For each link \( \ell_k \), the unique fundamental circuit \( C_k \) containing
Figure 4: SFG

Table 2 (SFG DATA)

<table>
<thead>
<tr>
<th>Initial Node</th>
<th>Terminal Node</th>
<th>Exponent of s</th>
<th>Branch Value</th>
<th>Branch Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>FB</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>-5x10^3</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>-5x10^3</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>-5x10^3</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>-1/15x10^3</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>5x10^3</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1/15x10^3</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>-5x10^3</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>A1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>5x10^3</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>0</td>
<td>0.04</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>A2</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>0</td>
<td>5x10^3</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>0</td>
<td>0.04</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>0</td>
<td>-25</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>0</td>
<td>-10^-4</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>0</td>
<td>25</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>0</td>
<td>10^-4</td>
<td>-</td>
</tr>
</tbody>
</table>
branches \( b_1, i=1,2,\ldots,m \) is found. Two sets of SFG branches can then be created.

Set (a): For each passive branch in the tree branch set \( b_i, i=1,2,\ldots,m \), a directed branch in the SFG is formed from node \( I_k \) to node \( V_{b_i} \) with weight equal to the impedance of branch \( b_i \), prefixed with the proper sign (positive, if the directions of \( I_k \) and \( b_i \) concur in \( C_k \), and negative otherwise).

Set (b): If the link \( l_k \) is a passive branch, a directed branch in the SFG is formed from each node \( V_{b_i}, i=1,2,\ldots,m \), to node \( I_k \), having weight equal to the admittance of link \( l_k \), prefixed with the proper sign (negative, if the directions of \( l_k \) and \( b_i \) concur in \( C_k \), positive otherwise).

Rule (2): If any of the four types of controlled sources are present, a directed branch is created in the SFG from the controlling variable to the controlled source, having weight equal to the constant of proportionality (\( g_m \), beta, etc.). If the controlling variable is a link voltage or a tree branch current, one more node is added to the SFG to represent this controlling variable \( X \) (node \( X_{12} \) in Fig. 4 is a node of this type). \( X \) is then expressed in terms of the tree branch voltage or link current through a simple immittance relationship.

Rule (3): If the desired output \( Y \) is neither a tree branch voltage nor a link current, then one node is added to the SFG to represent \( Y \). \( Y \) is then expressed in terms of tree branch voltages or in terms of a link current through a simple immittance relationship.

Rule (4): Finally, the SFG is "closed" by adding a branch with a symbolic weight (\( FB \), directed from the output to the input node.
d. The Gain Formula for "Closed" SFG

The purpose for introducing the closed SFG is because only all orders of non-touching loops need be found as opposed to the evaluation of Mason's formula which required enumerating certain paths as well as loops.

To derive the gain expression for the closed SFG consider first Mason's equation for the transfer function.

\[ T = \frac{X_o}{X_i} = \sum_{i=1}^{m} \frac{P_i \Delta_i}{\Delta} \]

where

\[ \Delta = 1 + \sum_{j=1}^{m} \prod_{k,j} L_{k,j} \]

is the determinant of the SFG

\[ L_{k,j} \]

is the product of the transmittances of the \( k \)th set of non-intersecting loops of order \( j \).

\[ P_i \]

is the transmittance product of the \( i \)th path between \( X_i \) and \( X_o \)

\[ \Delta_i \]

is the partial determinant obtained from \( \Delta \) after removal of all loops intersecting the \( i \)th path between \( X_i \) and \( X_o \).

Let \( \Delta_c \) be the determinant of the closed SFG. It is then noted that since \( \{P_i\}_{i=1}^{m} \) is the set of all paths from \( X_i \) to \( X_o \), the loops present in the closed SFG not present in the original SFG will be precisely \( \{(FB)P_i\}_{i=1}^{m} \)

where \( FB \) is the symbol assigned to the added branch. Further, since the path \( FB \) contains only nodes \( X_i \) and \( X_o \) which, in turn, are present in every path \( P_i \), \( i=1,2,\ldots,m \), it follows that the non-intersecting loop combinations that do not touch the loops \( (FB)P_i \), \( i=1,2,\ldots,m \) will be precisely those combinations which do not touch the path \( P_i \), \( i=1,2,\ldots,m \). It follows that

\[ \Delta_c = (FB) \sum_{i=1}^{m} P_i \Delta_i + \Delta \]

Thus, the transfer function can be found by simply sorting the terms of the determinant of the closed SFG.
II-2. **Manipulating SFG Branch Weights**

Each branch weight in the SFG is of the form

\[ \text{Constant} \cdot \text{Symbol} \cdot s^n \]

If an arbitrary branch has an initial node \( X_i \) and a final node \( X_f \), then the three parameters

\[ C(X_i, X_f) = \text{constant} \]
\[ S(X_i, X_f) = \text{symbol} \]
\[ E(X_i, X_f) = \text{exponent of} \ s \]

completely define the weight of the branch. After a loop or a set of nontouching loops has been found in the SFG, say by some path-finding technique, it is desirable to combine the weight parameters of each branch in the loop set to form a composite loop set weight. The loop set constant may be easily formed by taking the product of the constants associated with each branch. Similarly, the loop set exponent parameter is readily found by summing the exponents assigned to each branch. However, because computers are not particularly adept at symbol manipulation, it is inefficient with respect to both time and storage to form directly a composite loop set symbol. A much better technique is to convert each branch symbol into a numeric code. These codes are assigned to the SFG branches as follows: Each distinct symbol in the SFG is stored in the array \( S(j) \) and assigned a code \( B^j \) where \( B \) is some base \( B \in \{2, 4, \ldots, 2^m\} \). Now for an arbitrary SFG branch having initial node \( X_i \) and final node \( X_f \) which contains the symbol \( S(n) \), the code

\[ K(X_i, X_f) = B^n \]

is assigned.
The real value of this coding technique stems from the fact that the composite loop set code formed by summing the codes representing the individual branch symbols can be uniquely decoded provided the number of identical symbols combined into any code is less than B.

As an example of the above concepts for manipulating the SFG branch weights, refer to the SFG shown in Fig. 4. Consider, in particular, the loop defined by the node sequences

\[ V_2 - I_3 - V_2 \quad \text{and} \quad V_4 - I_5 - V_4 \]

Then

\[ \text{composite loop set constant} = (-5 \times 10^3)(1)(-5 \times 10^3)(1) \]

\[ = 25 \times 10^6 \]

and

\[ \text{composite loop set power} = 0 + 1 + 0 + 1 = 2 \]

To find the loop set code, an array of distinct symbols of the SFG and their corresponding codes must be set up.

no symbol \( \longleftrightarrow 0 \)

\[ S(1) = FB \longleftrightarrow 4^0 \]

\[ S(2) = C \longleftrightarrow 4^1 \]

\[ S(3) = A1 \longleftrightarrow 4^2 \]

\[ S(4) = A2 \longleftrightarrow 4^3 \]

Note that because there will be at most two identical symbols in any code, the base 4 was chosen. Using the above codes gives

\[ \text{composite set code} = K(V_2, I_3) + K(I_3, V_2) \]

\[ + K(V_4, I_5) + K(I_5, V_4) \]

\[ = 4^2 + 0 + 0 + 4^1 \]

\[ = 8 \]

Now to decode this number, say in the output, we would write

\[ 8 = 2(4^1) \]

\[ = (S(2))^2 \]

\[ = C^2 \]
which is indeed the symbol associated with the loop impedance product. The above coding scheme for manipulating symbols is easily adapted to the computer by incorporating the masking operation \( \text{AND} \). To determine the number of \( S(1) \) type symbols contained in a given code, the \( \text{AND} \) operation is applied to the code and \( B-1 \). In general, the number of \( S(J) \) symbols is found by dividing (using integer division so as to truncate the remainder) the code used to determine the number of \( S(J-1) \) symbols by \( B \) and then applying the \( \text{AND} \) operation. For example, consider the loop set previously discussed.

\[
\begin{align*}
\text{loop set code} &= 8 = (000000000100)_2 \\
B - 1 &= 3 = (000000000011)_2 \\
(\text{loop set code}) \text{.AND.(}B-1\text{)} &= 8 \text{.AND.} 3 \\
&= 0
\end{align*}
\]

Thus, the symbol \( S(1) = FB \) is not present. Now divide the loop code by \( B \) and repeat the above procedure

\[
\begin{align*}
\text{new code} &= \frac{8}{4} = 2 \\
(\text{new code}) \text{.AND.(}B-1\text{)} &= 2 \text{.AND.} 3 \\
&= (000000000010)_2 \\
&= 2
\end{align*}
\]

This implies \( C \) is contained in the code 8 and that its exponent is 2, i.e. \( C^2 \). The process stops when the code is reduced to zero.

Each loop set (of any order) contributes to a term in the network function. As each loop set (of any order) is generated and coded, it is compared with existing terms. If a term with the same symbol code and power of \( s \) exists, then the constant of the term is updated by adding to it the constant of the new loop set. Otherwise, a new term is created. Note that the above process is an important step towards reducing the storage requirements.
After all loop sets have been found, the transfer function is complete, and it remains only to transform the symbol code of each term into its corresponding symbol set by the .AND. operation previously described.
II-3. Generating First Order Loops

a. General Description

Let the nodes of the SFG be labelled 1,2,...,N. All first order loops which contain node J (J=1 initially) can be found by conceptually splitting node J into two nodes, one node containing all incoming branch and the other containing all outgoing branches, and then enumerating all paths between these two nodes. All branches going into node J are then removed and the process repeated for node J+1. Clearly, this procedure will produce all circuits with no duplications.

The problem of efficiently finding all circuits now becomes one of finding paths. The path-finding algorithm utilized by SNAP is based on a routing technique which conceptually resembles that proposed by Kroft\(^{(5)}\). However, because our ultimate objective is a flexible user-oriented program, we have chosen to use FORTRAN instead of SNOBOL as Kroft did. A general description of the concepts contained in the algorithm will be given here in addition to a rigorous step-by-step description presented at the end of this section.

Consider the SFG given in Fig. 4. The topological structure of the SFG can be completely described by the following routing table where the entries in the J\(^{th}\) row are the set of all nodes of distance one from node J. Note that the entries of each row are made to decrease as the column subscript M increases. This facilitates modifying the table after all paths through a particular node, say node J, have been found because only the right most non-zero entry of each row must be tested, i.e. if that entry equals J, it is a set to zero. As an example in using the routing table, the following two circuits can easily be shown to compose the complete set of circuits containing node 1.

1 - 11 - 9 - 12 - 8 - 5 - 6 - 1
1 - 11 - 9 - 10 - 7 - 2 - 6 - 1
Routing Table

A particularly important feature of the path-finding algorithm is the method by which each new node generated from the routing table must be tested to prevent loops from being formed. Rather than comparing the prospective node to each node already in the path, it is much more efficient to define the function

\[ F(I) = \begin{cases} 1 & \text{if } I \text{ is contained in the path node sequence} \\ 0 & \text{if } I \text{ is not contained in the path node sequence} \end{cases} \]

on which only one logic test need be made.

Additional insight may be obtained by viewing the path-finding technique graphically. That is, the process by which paths are generated can be observed by applying the following two rules directly to the SFG.

1. Let node J be the last node added to the path node sequence (initially J = input node). To select the next node, traverse that branch connected to node J that goes to the highest numbered node satisfying both the following requirements:
a. we did not just back up from this node while applying rule 2, and

b. this node is not included in the path node sequence.

Repeat this process until the output node is reached (then store the node sequence and go to rule 2) or until no new node can be found that satisfies (a) and (b) (then go to rule 2).

(2) Back up along the path just found (this is always possible unless we are at the input node in which case all paths have been found) until a new route can be taken according to rule 1.

The heavy lines of Fig. 5 show the path which results from applying rule 1 when node 1 is considered both the initial and terminal node. Generating a second path requires backtracking to node 9, then continuing the sequence 10-7-2-6-1. Note that the above graphical technique for listing all paths can be helpful when solving problems by hand.

b. A Detailed Description of the Path-Finding Algorithm

Algorithm PF* (Path-finding): This algorithm finds all paths between two nodes of a directed graph (without parallel edges) whose nodes are labelled 1,2,...,N. The only modification necessary to adapt the algorithm to finding

*The format used to describe the path-finding algorithm follows the style of Knuth(6).
all circuits thru node L is to set I ← L where L and I are defined below.

Notations:

I: Initial path node
L: Last path node
N: Number of nodes in graph
E_J: Number of branches leaving node J
R(J,M): Routing table
C_J: Column counter for the J^{th} row of the table R
P(V,W): The V^{th} node in the node sequence of path W
U_W: Number of nodes in path W
F(K): A function used to test whether node K is repeated, and whether the last node is reached.

PF1. (Preliminary)

Set R(J,1), R(J,2), ..., R(J,E_J) to the group of E_J nodes of distance one from node J. When using the algorithm to find circuit, make the entries of each row decrease as M increases.

Set R(J,M) ← \{-1 for M = E_J+1 and J=I
0 for M = E_J+1 and J≠I
\}

Set F(K) ← \{1 for K=I
0 for K=J and J≠I,L
-1 for K=L
\}

Set C_J ← 1 for J=1,2,...,N
Set W ← 1, V ← 2, J ← I, P(I,1) ← I

PF2. (Find the next node)

Set P(V,W) ← R(J,C_J)

PF3. (Test R)

\{< 0 stop; all paths have been found
IF R(J,C_J) = 0 set F(J) ← 0, go to step PF6
> 0 go to step PF4
PF4. (Test F)

\[
\begin{cases}
< 0 \text{ path completed; go to step PF7} \\
\text{IF } F[R(J,C_J)] \geq 0 \text{ go to step PF5} \\
\geq 0 \text{ set } C_J \leftarrow C_J + 1; \text{ go to step PF2}
\end{cases}
\]

PF5. (Prepare for next node)

Set \( J \leftarrow P(V,W) \), \( F(J) \leftarrow 1 \), \( V \leftarrow V+1 \), go to step PF2

PF6. (Back step)

Set \( C_J \leftarrow 1 \), \( J \leftarrow P(V-2,W) \), \( C_J \leftarrow C_J + 1 \), \( V \leftarrow V-1 \), go to step PF2

PF7. (Finish path)

Set \( C_J \leftarrow C_J + 1 \), \( P(K,W+1) \leftarrow P(K,W) \), \( K=1,2,\ldots,U_W-1 \), \( W \leftarrow W+1 \), go to step PF2.
II-4. Generating Nontouching Loops of Order Two or More

Preliminary results from SNAP indicate that of the following subprograms, (1) finding a SFG, (2) coding and de-coding, (3) enumerating first order loops, and (4) finding all higher order nontouching loops, the last will generally require the most time unless the network contains many distinct symbols in which case subprogram (2) may dominate. It is therefore necessary to exercise considerable care in developing an algorithm for finding all orders of nontouching loops.

In general, to find loop sets of all orders, some comparison between the node sequences of the different loops must be made. A brute force technique is simply to store all the node sequences of the first order loops and to find nontouching loops by direct comparison of the nodes contained in the loops. Of course, storage is also needed to indicate the loops contained in some of the higher order combinations, but this storage is necessary even in the more efficient techniques which follow.

The above method is improved considerably if instead of directly comparing the nodes of loop A and loop B to determine if they touch, a function $F(I)$ is defined as

$$F(I) = \begin{cases} 1 & I \in \{\text{nodes in loop A}\} \\ 0 & \text{otherwise} \end{cases}$$

and then tested as follows:

If $F(J) = 0$ all $J \in \{\text{nodes in loop B}\} \Rightarrow$ loops do not touch

$$F(J) = 1 \text{ any } J \in \{\text{nodes in loop B}\} \Rightarrow \text{loops touch}$$

For those computers which can accomodate the .AND. operation (or equivalent), the following coding technique reduces the number of logic

*Although the program was correct, the algorithm was incorrectly described in reference (3).
comparisons needed to determine if two loops touch to one and, perhaps
what is even more important, requires only a single code be stored for
each first order loop instead of the complete node sequence. As each first
order loop is generated, it is assigned an integer code whose binary repre-
sentation shows the set of nodes in the loop. For example, if loop A
contains the nodes \(\{11, 9, 12, 8, 5, 6, 1\}\) and loop B contains the nodes \(\{2, 6\}\),
then the codes are

\[
A = (110110110001)_2 = 3505 \\
B = (000000100010)_2 = 34
\]

To determine whether the two loops touch or not, the masking operation \(\text{AND}\) is used. Thus,

\[
(A) \cdot \text{AND.} (B) = (000000100000)_2 \neq 0
\]

The result is not zero, indicating that loops A and B touch.

Using the coding scheme the complete set of nontouching pairs of loops is
found and stored in the one dimension array \(N\). Let \(n\) = number of first
order of loops. Then

\[
[N(1), N(2), \ldots, N[P(1)], N[P(1)+1], \ldots, N[P(2)], N[P(2)+1], \ldots, N[P(3)], \\
N[P(3)+1], \ldots, N[P(n-1)], N[P(n-1)+1], \ldots, N[P(n)]]
\]

is the complete set of nontouching pairs of loops, where the

\[
[N(1), N(2), \ldots, N[P(1)] = \text{set of loops numbered higher than 1 which do not} \\
\text{touch loop 1.}
\]

\[
[N[P(1)+1], N[P(1)+2], \ldots, N[P(2)] = \text{set of loops numbered higher than 2} \\
\text{which do not touch loop 2.}
\]

\[
[N[P(i-1)+1], N[P(i-1)+2], \ldots, N[P(i)] = \text{set of loops numbered higher than } i \\
\text{which do not touch loop } i
\]

\[
[N[P(n-1)+1], N[P(n-1)+2], \ldots, N[P(n)] = \text{empty set because there are no} \\
\text{loops numbered higher than } n.
\]

Note that the array \(P\) is simply used to partition the array \(N\) such that the set

\[
[N[P(i-1)+1], N[P(i-1)+2], \ldots, N[P(i)]]
\]

does not touch loop \(i\).
Example: (Consider the SFG of Fig. 4)

The first order loops are

<table>
<thead>
<tr>
<th>loop</th>
<th>node sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-11-9-12-8-5-6-1</td>
</tr>
<tr>
<td>2</td>
<td>1-11-9-10-7-2-6-1</td>
</tr>
<tr>
<td>3</td>
<td>2-6-2</td>
</tr>
<tr>
<td>4</td>
<td>2-3-2</td>
</tr>
<tr>
<td>5</td>
<td>4-5-4</td>
</tr>
<tr>
<td>6</td>
<td>5-6-5</td>
</tr>
<tr>
<td>7</td>
<td>9-11-9</td>
</tr>
<tr>
<td>8</td>
<td>9-10-9</td>
</tr>
</tbody>
</table>

To find the array of nontouching pairs \( P \), SNAP codes the above loops and proceeds to use the \(.AND.\) operator. The results are

\[
N = \{4,5,7,8,5,6,7,8,7,8\}
\]

and \( P(1) = 1, P(2) = 2, P(3) = 5, P(4) = 9, P(5) = 11, P(6) = 13, P(7) = 13, P(8) = 13. \)

To systematically continue the process, an array \( S \) is created from which all higher order loop sets (2 or more) not touching loop \( L \) can be found. By incrementing \( L \) from 1 to \( n \), all higher order loops will then be enumerated.

Let

\[
S(1,1)S(1,2)\ldots S[1,U(1)] \ldots S[1,J(1)],0,\ldots
\]

\[
S(2,1)S(2,2)\ldots S[2,U(2)+1] \ldots S[2,J(2)],0,\ldots
\]

\[
S(K,1)S(K,2)\ldots S[K,U(K)] \ldots S[K,J(K)],0,\ldots
\]

where \( S(1,1) = N[P(L-1)+1] \)

\( S(1,2) = N[P(L-1)+2] \)

\[
\ldots
\]

\( S[1,J(1)] = N[P(L)] \)
and where the entire (loop numbers) of row \( M (M \geq 2) \) are those loops in the set 
\[ \{ S[M-1, U(M-1)+1], S[M-1, U(M-1)+2], \ldots, S[M-1, J(M-1)] \} \]
which do not touch the loop \( S[M-1, U(M-1)] \).

The arrows shown in the array \( S \) given above are referred to as "pointers". Note that \( U(J) \) indicates the position of "pointer" of the \( J^{th} \) row. Example: \( U(3) = 5 \) means the pointer of row 3 is currently located at the 5\(^{th} \) column.

The procedure for finding all higher order loop combinations is given in the following flow chart:

```
<table>
<thead>
<tr>
<th>Preliminary</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n ): number of first order loops</td>
</tr>
<tr>
<td>( \ell ): first order loop under consideration</td>
</tr>
<tr>
<td>( U(i) ): pointer position for row ( i ) in ( S )</td>
</tr>
<tr>
<td>( J(i) ): number of loops in row ( i ) of ( S )</td>
</tr>
<tr>
<td>( K ): the row counter indicating that row ( K ) of ( S ) is being scanned to generate a set of ( K+2 ) order loops</td>
</tr>
</tbody>
</table>

Set \( \ell = 0 \)
\( U(1) = 1 \) \( i = 1,2,\ldots,n \)
\( J(1) = 0 \) \( i = 1,2,\ldots,n \)

Set \( \ell = \ell + 1 \)

Have all first order loops been used to generate higher-order combinations? i.e. Is \( \ell = n? \)

Yes \( \rightarrow \) Stop (all higher order loops have been found)

No

Insert into row 1 of \( S \) the numbers corresponding to the loops numbered higher than \( \ell \) which do not touch loop \( \ell \).
Set \( K = 1 \)
\( J(1) = \) number of these loops

\( \leq 1 \) Test \( J(1) \)

\( > 1 \)
```
Generate row $K+1$ of $S$ as follows: Insert those loops of the set $\{S[K, U(K)+1], \ldots, S[K, J(K)]\}$ that do not touch loop $S[K, U(K)]$ into row $K+1$ of $S$. Set $J(K+1) = \text{number of these loops}$. When a new element, say $S(K+1,X)$, is generated, the weight parameters corresponding to the symbol code, constant term, and power of $s$ are stored (or when possible combined with other similar type terms) for the loop set loop $\ell \cdot \text{loop } S[1, U(1)] \ldots \text{loop } S(K+1, X)$.

- **Diagram:**
  - **Test:** $J(K+1) > 1$ → $K = K + 1$
  - **Test:** $J(K+1) \leq 1$
    - **Yes:** Can we generate row $K+1$ by incrementing the pointer of row $K$? i.e. Is $U(K) < J(K) - 1$?
      - **No:** Can we back up one row? i.e. Is $K > 1$?
        - **Yes:**
          - Set $U(K) = 1$
          - $J(K) = 0$
          - $K = K - 1$
        - **No:**
Example:

From the preceding example,

\[ N = \{4, 5, 5, 7, 8, 5, 6, 7, 8, 7, 8, 7, 8\} \]

and

\[ P(1) = 1, P(2) = 2, P(3) = 5, P(4) = 9, P(5) = 11 \]

\[ P(6) = 13, P(7) = 13, P(8) = 13 \]

Arrays \( N \) & \( P \) are more easily interpreted by setting up the following table:

<table>
<thead>
<tr>
<th>Loop J</th>
<th>Loops numbered higher than J that do not touch loop J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5, 7, 8</td>
</tr>
<tr>
<td>4</td>
<td>5, 6, 7, 8</td>
</tr>
<tr>
<td>5</td>
<td>7, 8</td>
</tr>
<tr>
<td>6</td>
<td>7, 8</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
</tr>
</tbody>
</table>

The sequence for producing the higher order loops is as follows:

Loop 1

Loops not touching loop 1 are inserted into first row of \( S \) (see Table 3)

<table>
<thead>
<tr>
<th>( S ) array</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} 4 \ 0 \ \vdots \end{array} ]</td>
<td>no 3rd order loops</td>
</tr>
</tbody>
</table>

Loop 2

<table>
<thead>
<tr>
<th>( S ) array</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} 5 \ 0 \ \vdots \end{array} ]</td>
<td>no 3rd order loops</td>
</tr>
</tbody>
</table>
**loop 3**

S array

```
5 7 8 0
0
...
```

Output

```
|
loop 3 does not touch loop 7 or loop 8 (this is determined by comparing loop codes—see section II-4)
```

```
5 7 8 0

7 8 0 0
0 0 0 0
0 0 0 0
```

|
loop 3·loop 5·loop 7
loop 3·loop 5·loop 8

|
loop 7 touches loop 8; thus, there is no 3rd row. Further if the pointer of row 1 is incremented by 1, no new 2nd row can be created. Thus, we are done with loop 3.

**loop 4**

S array

```
5 6 7 8
0 0 0 0
...
```

Output

```
|
loop 5 does not touch loop 7 or loop 8
```

```
5 6 7 8

7 8 0 0
0 0 0 0
0 0 0 0
```

|
loop 4·loop 5·loop 7
loop 4·loop 5·loop 8

|
loop 7 touches loop 8; thus, there is no 3rd row. Increment pointer of row 1.
\[ \begin{bmatrix} 5 & 6 & 7 & 8 \\ 0 & 0 & 0 & 0 \end{bmatrix} \]

loop 6 does not touch loop 7 or loop 8

\[ \begin{bmatrix} 5 & 6 & 7 & 8 \\ 7 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \]

loop 4 \cdot loop 6 \cdot loop 7

loop 4 \cdot loop 6 \cdot loop 8

incrementing pointers give no additional third order loops

loop 5

S array

\[ \begin{bmatrix} 7 & 8 & 0 & \ldots \\ 0 & \ldots \end{bmatrix} \]

Output

no 3\textsuperscript{rd} order loops

loop 7 touches loop 8

loop 6

S array

\[ \begin{bmatrix} 7 & 8 & 0 & 0 \\ 0 & \ldots \end{bmatrix} \]

Output

no 3\textsuperscript{rd} order loops

loop 7 touches loop 8

loop 7

S array

\[ [0] \]

Output

no 3\textsuperscript{rd} order loops

loop 8

S array

\[ [0] \]

Output

no 3\textsuperscript{rd} order loops
III. USER'S GUIDE

III-1. Information Needed by User

Program: SNAP (Symbolic Network Analysis Program)

Purpose: To obtain the network functions \( \frac{V_{\text{out}}}{V_{\text{in}}}, \frac{V_{\text{out}}}{I_{\text{in}}}, \frac{I_{\text{out}}}{V_{\text{in}}}, \text{or} \frac{I_{\text{out}}}{I_{\text{in}}} \) as a ratio of two polynomials of the following type:

1. all network element values are represented by symbols (the symbols need not all be different),

   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{s^2 LRC}{s^2 2LRC + s(L+R^2C) + R}
   \]

   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{Z Y R}{2 Z Y R + Z + R^2 Y + R}
   \]

2. some element values are specified numerically, some symbolically,

   Example:
   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{s^2 R}{s^2 2R + s(0.5 \times 10^6 + 150R^2) + 0.75 \times 10^8 R}
   \]

3. all element values are given numerically,

   Example:
   \[
   \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{s^2}{2 s^2 + 2 \times 10^4 s + 0.75 \times 10^8}
   \]

Description: Program SNAP is designed to handle lumped, linear, time invariant networks** containing the following type components:

1. two-terminal circuit elements -- resistance, inductance, and capacitance.

2. two-terminal networks described by an admittance or impedance parameter.

* Refer to Appendix A at end of this section for a technique of handling multi-output functions.

** See Appendix B for a brief list of additional limitations on the size and type of network allowed.
(3) all four types of controlled sources (Note: Mutual inductance, ideal transformers, gyrators, etc., can be modeled with elements in (1) and (3))

(4) one independent source; see Appendix A for a technique of handling multi-input networks.

Network Data Required: After the network components have been modeled by the type elements allowed, the branches and nodes are to be numbered consecutively starting with 1 and reference directions for each branch current are to be chosen. The following gives the sequence of data cards needed to describe the network.

<table>
<thead>
<tr>
<th>CARD 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
</tr>
<tr>
<td>1-72</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CARD 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
</tr>
<tr>
<td>1-5 (right adjusted)</td>
</tr>
<tr>
<td>6-10 (right adjusted)</td>
</tr>
</tbody>
</table>

The following three entries are optional.

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>11-15 (right adjusted)</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
</tbody>
</table>
CARD 3

<table>
<thead>
<tr>
<th>Columns</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>Network branch number of source</td>
</tr>
<tr>
<td>6-10</td>
<td>Network branch number associated with output (leave blank if output is a voltage across more than one branch)</td>
</tr>
<tr>
<td>11-15</td>
<td>Node number corresponding to the positive output voltage terminal (these columns can be left blank if columns 6-10 are not blank)</td>
</tr>
<tr>
<td>16-20</td>
<td>Node number corresponding to the negative output voltage terminal (these columns can be left blank if columns 6-10 are not blank)</td>
</tr>
</tbody>
</table>

CARDS 4 thru (b+3)  
(b = number of network branches)

Note 1: Each card describes one network branch (element).

Note 2: If output is a voltage (current) associated with a particular branch, then the data card describing this branch should be entered first (last) among the branch data cards (cards 4 thru (b+3)) to insure that this branch will be chosen as part of the tree (cotree).

Note 3: When a large number of branches share one common terminal, it is better to place these branches first starting with card 4 (card 5 if note 2 applies). The reason is given in Appendix C at the end of this section.

<table>
<thead>
<tr>
<th>Columns</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>Element type; E: voltage source</td>
</tr>
<tr>
<td>1</td>
<td>I: current source</td>
</tr>
<tr>
<td></td>
<td>G: conductance</td>
</tr>
<tr>
<td></td>
<td>R: resistance</td>
</tr>
<tr>
<td></td>
<td>L: inductance</td>
</tr>
<tr>
<td></td>
<td>C: capacitance</td>
</tr>
<tr>
<td></td>
<td>Z: impedance</td>
</tr>
<tr>
<td></td>
<td>Y: admittance</td>
</tr>
<tr>
<td></td>
<td>CC: current controlled current source</td>
</tr>
<tr>
<td></td>
<td>CV: current controlled voltage source</td>
</tr>
<tr>
<td></td>
<td>VC: voltage controlled current source</td>
</tr>
<tr>
<td></td>
<td>VV: voltage controlled voltage source</td>
</tr>
</tbody>
</table>
### Contents

<table>
<thead>
<tr>
<th>Column Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-5</td>
<td>Element number--all elements of the network must be assigned a distinct number (positive integer). For greatest efficiency, the numbering should be consecutive.</td>
</tr>
<tr>
<td>6-10</td>
<td>Initial node--this is relative to the arbitrarily chosen current direction.</td>
</tr>
<tr>
<td>11-15</td>
<td>Terminal node--this is relative to the arbitrarily chosen current direction.</td>
</tr>
<tr>
<td>17-19</td>
<td>Element symbol--the element's value, if not specified, is represented by this symbol.</td>
</tr>
<tr>
<td>20</td>
<td>Equal sign (=) if element is to be assigned a value. Leave blank if element value is to be represented in symbolic form.</td>
</tr>
<tr>
<td>21-32</td>
<td>Element value (if known)--Format is E12.5. Units should be compatible with element type as specified in columns 1-2; for example, R is expressed in ohms, G in mhos.</td>
</tr>
<tr>
<td>33-35</td>
<td>If element is a dependent source, enter the element number of its control.</td>
</tr>
</tbody>
</table>
An Example: We wish to find $\frac{I_{\text{out}}}{V_{\text{in}}}$, keeping $A_1$, $A_2$, and $C$ as symbols.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{Original Network.}
\end{figure}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{Modeled Network.}
\end{figure}

**DATA DECK**

<table>
<thead>
<tr>
<th>Type</th>
<th>Element Number</th>
<th>Initial Node</th>
<th>Terminal Node</th>
<th>Symbol</th>
<th>Value of Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-0.</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>R1= 5.00000E+03</td>
<td>-0</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>C= 0.</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>R1= 5.00000F+03</td>
<td>-0</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>C= 0.</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>R2= 1.50000F+04</td>
<td>-0</td>
</tr>
<tr>
<td>CC</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>A1= -0.</td>
<td>10</td>
</tr>
<tr>
<td>CC</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>A2= -0.</td>
<td>9</td>
</tr>
<tr>
<td>R</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>R= 2.50000F+01</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>R= 2.50000F+01</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>R3= 1.00000F+04</td>
<td>-0</td>
</tr>
</tbody>
</table>

**TABLE 1. Input Data as Reproduced in Program Output.**

---

**DIFFERENTIAL AMPLIFIER**

---

**DIFFERENTIAL AMPLIFIER**

**NUMBER OF NODES = 5**
**NUMBER OF BRANCHES = 11**
**ELEMENT NUMBER OF SOURCE = 1**
**ELEMENT NUMBER ASSOCIATED WITH OUTPUT = 6**
**BASE FOR SYMBOL CODES = 8**

**NETWORK**

**ELEMENT ELEMENT INITIAL TERMINAL ELEMENT ELEMENT ELEMENT NO.**

<table>
<thead>
<tr>
<th>Type</th>
<th>Element Number</th>
<th>Initial Node</th>
<th>Terminal Node</th>
<th>Symbol</th>
<th>Value of Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-0.</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>R1= 5.00000E+03</td>
<td>-0</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>C= 0.</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>R1= 5.00000F+03</td>
<td>-0</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>C= 0.</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>R2= 1.50000F+04</td>
<td>-0</td>
</tr>
<tr>
<td>CC</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>A1= -0.</td>
<td>10</td>
</tr>
<tr>
<td>CC</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>A2= -0.</td>
<td>9</td>
</tr>
<tr>
<td>R</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>R= 2.50000F+01</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>R= 2.50000F+01</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>R3= 1.00000F+04</td>
<td>-0</td>
</tr>
</tbody>
</table>

**TREE SELECTED**

<table>
<thead>
<tr>
<th>Type</th>
<th>Element Number</th>
<th>Initial Node</th>
<th>Terminal Node</th>
<th>Symbol</th>
<th>Value of Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-0.</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>R1= 5.00000F+03</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>R1= 5.00000F+03</td>
<td>-0</td>
</tr>
<tr>
<td>R</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>R= 2.50000F+01</td>
<td>-0</td>
</tr>
</tbody>
</table>
TABLE 2. Program Output Information Showing Signal-Flow Graph, Circuits, and Execution Times.

<table>
<thead>
<tr>
<th>SFG</th>
<th>INITIAL TERMINAL EXPONENT NODE</th>
<th>TERMINAL EXPONENT NODE OF S</th>
<th>BRANCH VALUE</th>
<th>BRANCH SYMBOL</th>
<th>1 IF SYMBOL IS INVERTED</th>
<th>1 IF SYMBOL IS USED</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>-1.000000E+00</td>
<td>FR</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1.000000E+00</td>
<td>C</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>-5.000000E+03</td>
<td>R1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1.000000E+00</td>
<td>C</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>-5.000000E+03</td>
<td>R1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>-6.66667E-05</td>
<td>R2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>5.000000E+03</td>
<td>R1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
<td>6.66667E-05</td>
<td>R2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>-5.000000E+03</td>
<td>R1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>0</td>
<td>1.000000E+00</td>
<td>A1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>5.000000E+03</td>
<td>R1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>0</td>
<td>4.000000E-02</td>
<td>RE</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>0</td>
<td>1.000000E+00</td>
<td>A2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>0</td>
<td>5.000000E+03</td>
<td>R1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>0</td>
<td>4.000000E-02</td>
<td>RE</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>0</td>
<td>-2.500000E+01</td>
<td>RE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>0</td>
<td>-1.000000E-04</td>
<td>R3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>0</td>
<td>2.500000E+01</td>
<td>RE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>0</td>
<td>1.000000E+04</td>
<td>R3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

TIME FOR FORMULATING SIGNAL FLOW GRAPH IN SECONDS: 252

CIRCUITS

<table>
<thead>
<tr>
<th>NO.</th>
<th>NODE LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 11 9 12 8 5 6 1</td>
</tr>
<tr>
<td>2</td>
<td>1 11 9 10 7 2 6 1</td>
</tr>
<tr>
<td>3</td>
<td>2 6 2</td>
</tr>
<tr>
<td>4</td>
<td>2 3 2</td>
</tr>
<tr>
<td>5</td>
<td>4 5 4</td>
</tr>
<tr>
<td>6</td>
<td>5 6 5</td>
</tr>
<tr>
<td>7</td>
<td>9 11 9</td>
</tr>
<tr>
<td>8</td>
<td>9 10 9</td>
</tr>
</tbody>
</table>

TIME FOR FINDING 8 FIRST ORDER LOOPS IN SECONDS: 0.46

TIME FOR FINDING 19 SETS OF NONTOUCHING LOOPS, IN SECONDS: 0.027

TIME FOR DECODING SYMBOLS IN SECONDS: 0.124
TABLE 3. Network Transfer Function and Total Execution Time.

<table>
<thead>
<tr>
<th>Column</th>
<th>Symbol for Given Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$A_2$ / 1</td>
</tr>
<tr>
<td>2</td>
<td>$A_1$ / 1</td>
</tr>
<tr>
<td>3</td>
<td>$C$ / $A_2$ / 1</td>
</tr>
<tr>
<td>4</td>
<td>$C$ / $A_1$ / 1</td>
</tr>
</tbody>
</table>

**Numerator Polynomial**

$$= (3.3333 \times 10^{-5} + 1.6667 \times 10^{-7} C) (A_2 - A_1)$$

<table>
<thead>
<tr>
<th>Power of $S$</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.33333E-05</td>
<td>-3.33333E-05</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.</td>
<td>1.6667E-01</td>
<td>-1.6667E-01</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.</td>
<td>0.</td>
<td>0.</td>
</tr>
</tbody>
</table>

**Denominator Polynomial**

$$= 3.3375 + 2.67 \times 10^4 s C + 5.00625 \times 10^7 s^2 C^2$$

<table>
<thead>
<tr>
<th>Column</th>
<th>Symbol for Given Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 / 1</td>
</tr>
<tr>
<td>2</td>
<td>$C$ / 1</td>
</tr>
<tr>
<td>3</td>
<td>$C*2$ / 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power of $S$</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.33750E+00</td>
<td>0.</td>
<td>0.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.</td>
<td>2.67000E+04</td>
<td>0.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.</td>
<td>0.</td>
<td>5.00625E+07</td>
<td></td>
</tr>
</tbody>
</table>

Execution time in seconds, .509

August 1970 version of SNAP.
APPENDIX A

A Sorting Technique for Handling Multi-Input, Multi-Output Networks.

Multi-Inputs

Program SNAP (the August 1970 revision) permits only one independent source branch. However, networks containing more than one source can easily be handled with the following technique. Let $W_i, i = 1, 2, \ldots, n$ represent a set of $n$ independent sources, either voltage or current. Assign $W_1$ as the permitted independent source and make $W_2, W_3, \ldots, W_n$ dependent sources which are dependent on $W_1$ with proportionality factors

$$k_2 = \frac{W_2}{W_1}, \quad k_3 = \frac{W_3}{W_1}, \quad \ldots, \quad k_n = \frac{W_n}{W_1}$$

Only the numerator polynomial in the output will contain these parameters thus permitting the user to easily put the output function into the form

$$\frac{W_{\text{out}}}{W_1} = \frac{P_1 + P_2 k_2 + P_3 k_3 + \ldots + P_n k_n}{\Delta}$$

where $\Delta$ and $P_i, i = 1, 2, \ldots, n$ are polynomials. The output function can then be written

$$W_{\text{out}} = \frac{P_1 W_1 + P_2 W_2 + \ldots + P_n W_n}{\Delta} \quad (1)$$

Although at present SNAP does not give the output function in the form of Eq. (1) directly, only a few program modifications are necessary to effect such a result. For example, the program could internally create a new input node, $I_{\text{new}}$, of the SFG and then make each independent source, $W_i$, dependent on $I_{\text{new}}$ with weight $P_i$ as shown in Fig. 1 below.
Multi-Outputs

The following technique can be used to obtain more than one output function in a single computer run: Augment the original network by appending one end of a series connection of dependent voltage sources to the given network such that

(a) to each branch current, $I_j$, desired as an output, there corresponds a dependent voltage source which depends on $I_j$ and has symbolic weight $I_{oj}$,

and (b) to each voltage $V_{AB}$ desired as an output, there corresponds a set of the dependent voltage sources each dependent upon a voltage across one of the branches in the path between nodes A and B and all having symbolic weight $V_{OAB}$.

By specifying the output to be the voltage across the entire series connection of dependent voltage sources, outputs $I_j$ and $V_{AB}$ will be those output terms which contain $I_{oj}$ and $V_{OAB}$ respectively. Only a few modifications of the present version of SNAP would be necessary to have the program internally perform the network augmentation described above (at present, the user must do the augmenting).

As an example, Fig. 2 illustrates the network augmentation needed to find the voltage $V_{14}$ and current $I_5$ for the given bridge network in one computer run.
\[ V_{47} = V_{014}(V_{12} + V_{24}) + I_{05} I_5 \]

Figure 2.
Appendix B

A Brief List of Limitations on the Size and Type of Network Allowed

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of network branches</td>
<td>35</td>
<td>SNAP cannot handle all networks having 35 branches or less. Other factors such as time considerations, SFG characteristics (number of higher order loops, for example), and number of network symbols to name a few can further limit the size of the network.</td>
</tr>
<tr>
<td>Maximum number of elements that can be represented by the same symbol</td>
<td>7</td>
<td>This number can be increased to $2^n - 1$ by increasing the symbol code base used to $2^n$, $n &gt; 3$, on the input data card 2.</td>
</tr>
<tr>
<td>Number of different powers of $s$</td>
<td>15</td>
<td>Sufficient for networks containing no more than 15 reactive elements.</td>
</tr>
<tr>
<td>Estimate of the maximum number of distinct network symbols permitted</td>
<td>12</td>
<td>This restriction results from the fact that SNAP can contain no more than 150 different symbol combinations in the output.</td>
</tr>
</tbody>
</table>
APPENDIX C

Selecting a "Good" Tree

The network tree used to generate the SFG has a very significant effect on the number of loops and higher order loops present in the SFG. The loop enumeration and evaluation, in turn, often determines the time and storage needed by a computer to solve a given network. The ladder network of Figure 1 together with Table 1 illustrated the interrelationship between the tree selected, the number of loops (all orders), computer execution time, and computer storage.

![Figure 1](image)

<table>
<thead>
<tr>
<th>Tree Branches</th>
<th>Number of loops</th>
<th>Number of higher order loops</th>
<th>Time required to find $V_o/I_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star tree: 1,3,5,7,9,11,13,15,17</td>
<td>17</td>
<td>2567</td>
<td>1.55 seconds</td>
</tr>
<tr>
<td>1,2,4,7,9,11,13,15,17</td>
<td>38</td>
<td>8096</td>
<td>3.93 seconds</td>
</tr>
<tr>
<td>1,2,4,6,9,11,13,15,17</td>
<td>117</td>
<td>19719</td>
<td>9.42 seconds</td>
</tr>
<tr>
<td>1,2,4,6,8,11,13,15,17</td>
<td>476</td>
<td>(storage for first order exceeded)</td>
<td>--</td>
</tr>
</tbody>
</table>

Unfortunately, choosing the "best" tree, that is, a tree which will minimize the number of loop combinations of all orders is a very involved process. See reference 2 and 7 for a detailed discussion of this problem.
For most networks, however, a tree that will result in a reasonable amount of execution time and computer storage can be selected by applying one of the following rules (rule 2 results in a better tree than rule 1)

**Rule 1:** Select a tree in which as many branches as possible form a star, that is, the branches share a common node. Modify this tree, if necessary, to include any branch which has two or more branches in parallel with it.

**Rule 2:** Let $T_k$ be some tree (not necessarily the best) of the network graph. For each link $\ell_i$ of the graph, define $B_{\ell_i}$ as the number of tree branches which form a circuit with $\ell_i$. Then form the sum

$$S_{T_k} = \sum_{i=1}^{L} B_{\ell_i}$$

where $L =$ number of links in the graph having $T_k$ as a tree.

Select that tree, say $T_j$, which satisfies the inequality

$$S_{T_j} < S_{T_k} \quad k = 1, 2, \ldots, N \text{ where } N = \text{number of trees}$$

The example given in section III-1 uses a tree, $T_j$, having $S_{T_j} = 11$.

The tree generated internally by program SNAP includes all voltage sources together with those passive branches read in first (starting with input data card 4) which complete the tree. Thus, to have SNAP select the tree that has been chosen by the user, it is necessary that the user's tree include all voltage sources and that all its passive branches be listed first starting with input data card 4.
III-2. Modifying the Dimension of Arrays

In order to make SNAP applicable to many different type networks, a flexible yet simple procedure is needed for modifying the dimension of the arrays. For example, storage requirements for networks containing many symbols will be determined by the number of symbols, symbol codes, etc., whereas the storage needed for networks having no symbols will be determined by the number of loops, nontouching loops of all orders, and related network characteristics. Because it is not possible to determine apriori reasonable bounds for all the network characteristics, error diagnostics have been built into the program to inform the user as to which arrays have been inadequately dimensioned. As a result, the technique for adjusting the array dimension, in SNAP can be outlined as follows:

(1) Check that those network characteristics which can be determined before running the program are within the specified limits. These limits are listed following the dimension statements of the main program for convenient reference.

(2) Run the program. If an array dimension is exceeded an error message will result which specifies the network characteristic involved. For example, if the SFG of a given network has an excessive number of circuits, the message "No. of circuits exceeds limit--increase dimensions containing NPAC" will result. The definition of NPAC (number of paths and circuits) are found immediately following the array dimensions in the main program. It is important to point out that a computer run may continue to completion even if the dimension of some arrays have been exceeded (an error message is still given, however). In this situation, the results cannot be considered reliable.
(3) Once it has been ascertained by (1) and (2) that dimension modifications are in order, refer to the next few pages to determine the arrays associated with the network characteristics of interest. Increase the dimension of all the arrays indicated by say 20% (several runs may be necessary to achieve adequate program dimensions). Then update the value of the parameter (NPAC, for example) corresponding to the network characteristic involved. This parameter is used throughout the program (as limits on DO loops etc.) thereby making it unnecessary to do any additional program modifications.

\[ NBN = \text{Number of Network Branches (Presently 35)} \]

**PROGRAM MAIN**

\[
\begin{align*}
IG(NBN), & \quad KODES(NBN), & \quad KODE(NBN,NBN) \\
SHBOL(NBN), & \quad KONC(NBN), & \quad IXPO(NBN,NBN) \\
IFLOW(NBN), & \quad N(NBN,NBN), & \quad CONS(NBN,NBN) \\
LT(NBN), & \quad NP(NBN), & \\
\end{align*}
\]

**SUBROUTINE SFG**

\[
\begin{align*}
JROW(NBN), & \quad TYPB(NBN), & \quad IQUALX(NBN), & \quad JBX(NBN) \\
NP(NBN), & \quad JB(NBN), & \quad VALX(NBN), & \quad LBX(NBN) \\
IVV(NBN), & \quad LB(NBN), & \quad NUMLX(NBN), & \quad IB(NBN,NBN) \\
NUML(NBN), & \quad MSYM(NBN), & \quad INTRE(NBN), & \quad NS(NBN,NBN) \\
LCV(NBN), & \quad IQUAL(NBN), & \quad NOTREE(NBN), & \quad NF(NBN,NBN) \\
INTREE(NBN), & \quad VAL(NBN), & \quad TYPX(NBN), & \\
LINC(NBN), & \quad SYM(NBN), & \quad NUMX(NBN), & \\
\end{align*}
\]

**SUBROUTINE FTREE**

\[
\begin{align*}
TYPX(NBN), & \quad INTRE(NBN), & \quad NF(NBN,NBN) \\
JBX(NBN), & \quad NOTREE(NBN), & \\
LBX(NBN), & \quad NP(NBN) & \\
\end{align*}
\]

**SUBROUTINE TREP**

\[
\begin{align*}
JX(NBN), & \quad JMEM(NBN), & \quad NF(NBN,NBN) \\
NP(NBN), & \quad KMEM(NBN) & \\
\end{align*}
\]
NBG = Number of Branches in SFG (Presently 100)

PROGRAM MAIN

N FIRST(NBG), SYMBUL(NBG), NEST(NBG)
N LAST(NBG), MIX(NBG), TYPE(NBG)
IXPON(NBG), CVAL(NBG),
WEIGHT(NBG), KONSO(NBG),

SUBROUTINE SFG

N FIRST(NBG), MAPY(NBG), SYMBUL(NBG)
N LAST(NBG), KONSO(NBG), MIX(NBG)
IXPON(NBG), NEST(NBG), CVAL(NBG)
WEIGHT(NBG), TYPE(NBG),

NPAC = Number of Paths Plus Circuits (Presently 300)

PROGRAM MAIN

CONST(NPAC), MAPO(NPAC), JAC(NPAC)
KODET(NPAC), NOCTOT(NPAC), NPCODE(NPAC)
IXPOT(NPAC), NUP(NPAC),
NTO = Number of Terms in Output (Presently 150)

PROGRAM MAIN

NA(NTO), POLYU(NEXPS,NTO), SEMPON(NTO,NSPT/2)
NB(NTO), POLY(NEXPS,NTO), SEMPOD(NTO,NSPT/2)
KSORT(NTO), SIMBON(NTO,NSPT/2),
ITOP(NTO), SIMBOD(NTO,NSPT/2)

SUBROUTINE ARRAY

KSORT(NTO), POLY(NEXPS,NTO)

SUBROUTINE DECODE

ITOP(NTO)

NSPT = Number of Symbols per term in Output (Presently 20)

PROGRAM MAIN

KONS(NSPT), KODF(NSPT), SEMPON(NTO,NSPT/2)
KODI(NSPT), SIMBON(NTO,NSPT/2), SEMPOD(NTO,NSPT/2)
SEMBOL(NSPT), SIMBOLD(NTO,NSPT/2),

SUBROUTINE FTREE

KCOL(NSPT)

SUBROUTINE DECODE

SEMBOL(NSPT), KODF(NSPT), KODI(NSPT)
NEXPS = Number of Different Powers of s (Presently 15)

PROGRAM MAIN

MSORT(NEXPS), POLYU(NEXPS, NTO), POLY(NEXPS, NTO)

SUBROUTINE ARRAY

MSORT(NEXPS), POLY(NEXPS, NTO)

NRI = Maximum Number of Nontouching Loops (Presently 15)

PROGRAM MAIN

ISET(NRI, NCI)

NCI = Maximum Number of Loops Not Touching any Given Loop (Presently 100)

PROGRAM MAIN

ISET(NRI, NCI)

NEON = Number of Nontouching Pairs of Loops (Presently 1200)

PROGRAM MAIN

NOTCH(NEON)

NRS = Number of Repeated Symbols (Presently 9)

PROGRAM MAIN

STAR(NRS)
IV. PROGRAMMER'S GUIDE

IV-1. Definitions

CONS(J,L)=WEIGT(I) FOR BRANCH I OF THE SFG WHERE
J=FIRST(I), L=LAST(I)

CONST(I)=COMPOSITE CONSTANT ASSOCIATED WITH CIRCUIT I. IT IS FOUND BY
TAKING THE PRODUCT OF THE CONSTANT VALUES OF EVERY SFG BRANCH IN CIRCUIT I

CVAL(NUMC)=VALX(LINK) WHERE NUMC=NUMX(LINK)
(USED ONLY FOR NETWORK BRANCHES NOT IN THE TREF)

IB(LF,JF)=IB(JF,LF)=NUMC WHERE JF=JB(NUMC) AND LF=LB(NUMC)
AND NUMC IS A NETWORK TREE BRANCH NUMBER (ASSIGNED BY USER)

IFLOW(K)=A FLAG FOR THE PURPOSE OF CHECKING WHETHER NODE K IS REPEATED
AND WHETHER THE LAST NODE IS REACHED

TG(L)=SYMBOL CODE ASSIGNED TO THE SFG BRANCHES HAVING
TERMINAL NO. L

INTRF(K)=I, THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST IS
CHOSEN AS THE K-TH BRANCH OF THE NETWORK TREE

INTRF(NUMC)=1 IF THE NETWORK BRANCH NUMBERED NUMC BY THE USER IS
SELECTED FOR THE TREE, 0 OTHERWISE

IQUAL(NUMC)=IQUALX(I) WHERE NUMC=NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)

IQUALX(I)=EQUAL SIGN (=) IF I-TH NETWORK BRANCH IN THE DATA BRANCH
LIST HAS A NUMERICAL VALUE. LEFT BLANK IF I-TH BRANCH IS TO BE REPRESENTED BY A SYMBOL

ISET(J,L)=THE INTEGER ARRAY WHICH TOGETHER WITH THE ARRAY NOTCH CAN
BE USED TO FIND ALL SETS OF NONTOUCHING LOOPS OF ORDER GREATER THAN 2

ITOP(JC)=I IF THE TERMS IN COLUMN JC OF THE ARRAY POLY BELONG TO
THE NUMERATOR OF THE OUTPUT TRANSFER FUNCTION; 0 IF THEY
BELONG TO THE DENOMINATOR

IVV(M)=NETWORK BRANCH NUMBER OF THE M-TH VOLTAGE CONTROLLED
VOLTAGE SOURCE IN THE DATA BRANCH LIST

IXPO(J,L)=IXPON(I) FOR BRANCH I OF THE SFG WHERE
J=FIRST(I), L=LAST(I)

IXPON(I)=EXPONENT OF S ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I

IXPOT(I)=COMPOSITE EXPONENT OF S FOR CIRCUIT I. IT IS FOUND BY ADDING
THE S POWERS ASSOCIATED WITH EACH BRANCH IN CIRCUIT I

JAC(J)=NUMBER OF NONZERO ENTRIES IN ROW J OF ISET

JH(NUMC)=JHX(I) WHERE NUMC=NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
JRX(I) = INITIAL NODE OF THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST
JRFM(I) = THE ROW OF THE ROUTING MATRIX FROM WHICH THE I-TH NODE IN THE PATH SEQUENCE WAS TAKEN
JROW(LF) = THE NUMBER OF NON-ZERO ENTRIES IN ROW LF OF THE ARRAY NF
JX(I+1) = NP(I)
KBASIS = NUMBER BASE OF THE SYMBOL CODES, THAT IS, THE SFG CONTAINS K00 DISTINCT SYMBOLS: SYMBOL(K), K=1,2,...,K00, NOT INCLUDING THE IAP VARIABLE & WHERE SYMBOL(K) IS ASSIGNED THE CODE KBASIS*K
KCOL = COUNTER USED TO FIND THE NUMBER OF LOOPS OF ORDER 2 OR GREATER
KINEC = ROW COUNTER OF THE MATRIX POLY
KMEM(I) = THE COLUMN OF THE ROUTING MATRIX FROM WHICH THE I-TH NODE IN THE PATH SEQUENCE WAS TAKEN
KODE(J+1) = CODE REPRESENTING THE SYMBOL OF THE SFG BRANCH HAVING J AS AN INITIAL NODE AND L AS THE TERMINAL NODE
KODES(J) = 2**{(J-1)} WHERE J IS A NODE OF THE SFG
KODET(I) = COMPOSITE CODE ASSOCIATED WITH CIRCUIT I. THIS CODE REPRESENTS THE SET OF SYMBOLS CORRESPONDING TO THE SET OF SFG BRANCHES CONTAINED IN CIRCUIT I
KODF(NZ) IS THE MULTIPLICITY OF THE SYMBOL CORRESPONDING TO THE CODE KODF(NZ)
KODI(NZ) = NZ=1,2,...,IZ IS THE SET OF INDIVIDUAL SYMBOL CODES THAT MAKE UP THE COMPOSITE CODE KSORT(JZ)
KONC(J) = COLUMN COUNTER FOR ROW J OF THE ROUTING MATRIX N(J,K)
KONS(KOZ) = 1 IF THE SYMBOL HAVING CODE KOZ IS NOT AN INVERSE SYMBOL A 0 IF THE SET OF SYMBOLS CORRESPONDING TO THE COMPOSITE CODE KSORT(J) BELONGS TO THE DENOMINATOR POLYNOMIAL
KONSO(I) = 1 IF SYMBOL OF THE SFG BRANCH I = SYMBOL(I), 0 IF SYMBOL OF THE SFG BRANCH I = SYMBOL(I)
KSORT(K) = THE CODE ASSIGNED TO COLUMN K OF THE MATRIX POLY
LH(NUMC) = LHX(I) WHERE NUMC = NUMC(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
LHX(I) = TERMINAL NODE OF THE I-TH NETWORK BRANCH IN THE DATA BRANCH LIST
LIL = A COLUMN COUNTER OF THE MATRIX POLY
LNCS(NUMC) = 1 IF THE NETWORK BRANCH NUMBERED NUMC BY THE USER IS NOT IN THE TREE, 0 OTHERWISE
LIST = NUMBER OF DIRECTED BRANCHES IN THE SFG
LISTC = 1 IF ALL CIRCUITS OF THE SFG ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE
LISTG = 1 IF SFG INFORMATION (BRANCH SYMBOLS, WEIGHTS ETC.) ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE
LISTP = 1 IF ALL PATHS FROM NODE NIN TO NODE NOOF NUT ARE TO BE LISTED IN THE PRINTOUT, 0 OTHERWISE
LT(J) = NUMBER OF POSITIVE ENTRIES IN ROW J OF N(J,K)
MAPQ(NIP) = NOCTOT(NIP) - NOCTOT(NIP-1) WHICH EQUALS THE NUMBER OF LOOPS NOT TOUCHING LOOP NIP
MXS(I) = MAPPING OF THE SFG BRANCH LIST INTO A LIST SATISFYING ONE OF THE FOLLOWING CONDITIONS
NFIRST(J)GT.NFIRST(K) FOR JGT.K OR NFIRST(J) = NFIRST(K), NLAST(J)LT.NLAST(K) FOR JGT.K
MSORT(K) = THE EXPONENT OF S ASSIGNED TO ROW K OF THE MATRIX POLY
N(J,K) WHERE K=1,2,...,LT(J) IS THE TERMINAL NODE OF SFG BRANCH HAVING J AS ITS INITIAL NODE. THE VALUE OF EACH NONZERO ENTRY IN A GIVEN ROW IS MADE TO DECREASE AS K INCREASES. THE ADDITION ENTRY N(N+LT(NIN)+1) = 1 IS ALSO MADE.
NA(J) = NUMBER OF SYMBOLS (NOT COUNTING INVERSE SYMBOLS) IN THE CODE KSORT(J)
NB(J) = NUMBER OF INVERSE SYMBOLS IN THE CODE KSORT(J)
NCIR = 1 IF CIRCUITS ARE TO BE FOUND, AND 0 IF CIRCUITS ARE NOT TO BE FOUND
NEST(1) = 1 IF THE SFG BRANCH I CONTAINS A SYMBOL IN ADDITION TO THE
LAPLACE VARIABLE S; 0 IF THE SFG BRANCH I CONTAINS NO SYMBOL
EXCEPT POSSIBLY FOR THE LAPLACE VARIABLE S.
NF(LF,JRFJ) = ROUTING TABLE FOR THE NETWORK COMPOSED ONLY OF
BRANCHES BELONGING TO THE TREE.
NFIR = 1 IF PATHS ARE TO BE FOUND (NFIR SET TO 1 IF LISP = 1), AND 0 IF
PATHS ARE NOT TO BE FOUND.
NFIRST(i) = INITIAL NODE OF THE DIRECTED SFG BRANCH I.
NIN = NETWORK BRANCH NUMBER OF THE SOURCE. THIS BECOMES THE SOURCE NODE
OF THE SFG.
NLAST(i) = TERMINAL NODE OF THE DIRECTED SFG BRANCH I.
NOS = NUMBER OF BRANCHES IN NETWORK.
NODE = NUMBER OF NODES IN NETWORK.
NODA = 0 UNLESS OUTPUT IS A VOLTAGE TAKEN ACROSS MORE THAN ONE NETWORK
ELEMENT. IN THIS CASE IT DESIGNATES THE POSITIVE TERMINAL OF
THE OUTPUT VOLTAGE.
NODR = 0 UNLESS OUTPUT IS A VOLTAGE TAKEN ACROSS MORE THAN ONE NETWORK
ELEMENT. IN THIS CASE IT DESIGNATES THE NEGATIVE TERMINAL
OF THE OUTPUT VOLTAGE.
NOC = NUMBER OF CIRCUITS (LOOPS).
NOP = NUMBER OF PATHS FROM NODE NIN TO NODE NOUT IN THE SFG.
NOUT = NETWORK BRANCH NUMBER ASSOCIATED WITH THE OUTPUT (VOLTAGE ACROSS
CURRENT SOURCE). THIS BECOMES THE SFG NODE CORRESPONDING TO 1
OUTPUT VARIABLE.
NOTCH(NOCH) AND NOCTOT(K), CONSIDER THE INTEGER SET
(I) = {1, 2, ..., N2} WHERE N2 = NUMBER OF NONTOUCHING PAIRS OF LOOPS.
NOW CONSIDER THE FOLLOWING SUBSET OF (I):
S(I) = (NOCTOT(K-1) + 1) * NOCTOT(K-1) + 2 * ... * NOCTOT(K) WHEN
NOCTOT(0) = 0. THEN THE SET (NOTCH(J), J IN S(I)) IS THE SET OF
LOOPS THAT DO NOT TOUCH LOOP K.
NOTREE(I) = 1 IF THE I-TH NETWORK BRANCH IN THE DATA LIST IS CHOSEN
FOR THE TREE, 0 OTHERWISE.
NP(I) = THE NODE SEQUENCE OF A PATH BETWEEN NODE NIN AND NODE NOUT
OF THE SFG. IF NIN = NOUT THIS IS THE NODE SEQUENCE FOR A CIRCUIT.
NPCODE(K) = COMPOSITE CODE USED TO IDENTIFY CIRCUIT I, FOUND BY SUMMING
THE CODES, NOCHS(J), ALLOTED TO EACH NODE, J, IN THE CIRCUIT.
NUML(NUMC) = NUML(1) WHERE NUMC = NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES).
NS(LF,JF) = 1 IF THE NETWORK TREE BRANCH (LF,JF) HAS INITIAL NODE
LF AND TERMINAL NODE JF AND EQUALS = 1 IF THE NETWORK TREE
BRANCH HAS INITIAL NODE JF AND TERMINAL NODE LF.
NUMX(I) = IF I-TH NETWORK BRANCH IN THE DATA BRANCH LIST IS A
DEPENDENT SOURCE, THIS ARRAY EQUALS THE NETWORK BRANCH NUMBER
ASSIGNED TO ITS CONTROL.
NUMX(I) = THE NETWORK BRANCH NUMBER ASSIGNED BY THE USEF TO THE I-TH
NETWORK BRANCH IN THE DATA BRANCH LIST.
NUP(J) DESIGNATES THE LOOP ISET(J,NUP(J))*, OF ROW J WHICH IS NOT
TOUCHED BY THE LOOPS ENTERED IN ROW J+1 OF ISET.
POLY(K,L) = MATRIX OF CONSTANTS WHERE EACH ENTRY IS ASSOCIATED WITH A
TERM IN THE NUMERATOR OR DENOMINATOR OUTPUT POLYNOMIAL.
HAVING THE S POWER OF K AND THE SYMBOL CODE ASSIGNED TO COLUMN L.
POLYU(K,L) = MATRIX OF CONSTANTS WHERE EACH ENTRY IS ASSOCIATED WITH A
TERM IN THE NUMERATOR OF THE OUTPUT POLYNOMIAL. HAVING THE S POWER
OF K AND THE SYMBOL CODE ASSIGNED TO COLUMN L.
SEMBO(KO) = SYMBOL CORRESPONDING TO THE CODE BASIS** (KO = 1)
SEMPON(J1,J2), J2 = 1, 2, ..., NA(J1), AND SEMPOU(J1,J3), J3 = 1, 2, ..., NA(J1)
ARE RESPECTIVELY THE MULTIPLICITY OF THE SYMBOLS
SIMBO(J1,J2), J2 = 1, 2, ..., NA(J1), AND SIMBOD(J1,J3), J3 = 1, 2, ..., NA(J1)
SIMBO(J1,J2), J2 = 1, 2, ..., NA(J1), AND SIMBOD(J1,J3), J3 = 1, 2, ..., NA(J1)
ARE RESPECTIVELY THE SYMBOLS AND INVERSE SYMBOLS CORRESPONDING
TO THE SYMBOL CODE KSO(TJ).
SMBOL(K) = SMBOL(I) FOR THE SFG BRANCH I WHERE I = MIX(K)
STAR(I) = **I THIS ARRAY IS GENERATED FROM DATA STATEMENTS AND IS
USED IN FORMING THE ARRAYS SEMPON AND SEMPOD
SYM(NUMC) = SYMX(I) WHERE NUMC = NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
SMBUL(I) = SYMBOL ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I
SYMX(I) = SYMBOL(3 CHARACTERS AT MOST) ASSIGNED BY USER TO THE I-TH
NETWORK BRANCH IN THE DATA BRANCH LIST. THE ELEMENT'S VALUE
IF NOT SPECIFIED IS REPRESENTED BY THIS SYMBOL
TYPX(NUMC) = TYPX(I) WHERE NUMC = NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
TYPE(NUMC) = TYPX(LINK) WHERE NUMC = NUMX(LINK)
(USED ONLY FOR NETWORK BRANCHES NOT IN THE TREF)
TYPX(I) = SPECIFIES THE ELEMENT TYPE OF THE I-TH NETWORK BRANCH
IN THE DATA BRANCH LIST. (MUST BE E, I, G, R, L, C, CC, CV, VC, OR VV
AND MUST BE COMPATIBLE WITH THE UNITS OF THE ELEMENT'S VALUE)
VAL(NUMC) = VALX(I) WHERE NUMC = NUMX(I)
(USED ONLY FOR NETWORK TREE BRANCHES)
VALX(I) = ELEMENT VALUE(E1205) OF I-TH NETWORK BRANCH IN THE DATA
BRANCH LIST
WEIGT(I) = CONSTANT TERM ASSOCIATED WITH THE VALUE OF THE SFG BRANCH I
IV-2. Flow Charts

Program SNAP is divided into the following sections:

  Program MAIN (Subprograms 1 thru 12)
  Subroutine SFG (Subprograms A thru J)
  Subroutine FTREE
  Subroutine TREP
  Subroutine ARRAY
  Subroutine DECODE

As indicated above, program MAIN is further broken down into 12 subprograms and subroutine SFG is divided into 10 subprograms.
Subprogram MAIN-1

This program reads in some preliminary network data.

Read in
(a) problem name
(b) NOD, NOB, KBASIS, LISTC, LISTP, NIN, NOUT, NODA, NODB
Set KBASIS to 8 if a zero valve has been read in.
Write out the above information for reference purposes.

To Subprogram MAIN-2
Subprogram MAIN-2

This program generates the SFG routing matrix, creates a code for each symbol (excluding s), and sets up arrays for the constants and powers of s associated with the branch weights.

---

Call subroutine SFG.
Transfer the following data into subroutine SFG: NIN, NOUT, NOD, NOB, LISTG, NODA, NODB.
(see subroutine SFG for additional data read in)
Subroutine SFG returns the following information to program MAIN-2:
LIST, NFIRST(I), NLAST(I), IXPON(I), WEIGT(I), SYMBUL(I), KONSO(I), NEST(I), MIX(I), I = 1, LIST
MG = KBASIS\*MG
initially MG=1

IBO = IBO + 1
initially IBO = 0

Compare
IBO to LIST

IBO > LIST

IBO ≤ LIST

LOB = MIX(IBO)
J = NFIRST(LOB)
L = NLAST(LOB)

LT(J) = LT(J) + 1
N(J,LT(J)) = NLAST(LOB)

After the Jth row is completed set
N(J,LT(J)+1) = \begin{cases} 0 \quad J\neq NIN \\ -1 \quad J = NIN \end{cases}

The mapping LOB = MIX(IBO) reorders
the SFG branch list so that (1) the nonzero
entries of a given row of the routing matrix
N(J,I) decreases as I increases and (2) the
Jth row is completed before elements are
entered into the J+1st row.

CONST(J,L) = WEIGHT(LOB)
IXPO(J,L) = IXPON(LOB)
where J = NFIRST(LOB), L = NLAST(LOB)
SMBOL(IBO) = SYMBOL(LOB)

Test NEST(LOB)

= 0
SFG branch LOB does not contain a symbol

= 1
SFG branch LOB contains a symbol

Test IG(L) where L = NLAST(LOB)

= 0

KP = KP + 1
Initially KP = 0

Compare SMBOL(IBO) to SMBOL(KP)

not equal

equal

Compare KONSO(LOB) and KONSO(LOBX) where LOBX = MIX(KP)

not equal
the symbol associated with branch IBO is the inverse of the symbol associated with branch KP

not equal

KP to IBO - 1

952

equal

307

KODE(J, L) = 0

307

KODE(J, L) = IG(L)

307

KODE(J, L) = IG(LX)
LX = NLAST(LOBX)

952
MG = KBASIS * MG
(Initially MG = 1)
IG(L) = MG
KODE(JL) = IB(L)

KOO = KOO + 1
Initially KOO = 0

SEMBOL(KOO) = SYMBOL(IBO)
This establishes a direct correspondence
between a symbol and its code
SEMBOL(KOO) \leftrightarrow KBASIS**KOO

Test KONSO(LOB)

SFG branch LOB contains the symbol
\( \frac{1}{SEMBOL(KOO)} \)

KONS(KOO) = 1

SFG branch LOB contains the symbol
SEMBOL(KOO)

KONS(KOO) = 0

305

KONS(KOO) = 1

KONS(KOO) = 0

305
Subprogram MAIN-3

This program codes the nodes of the SFG, and prepares the counters for finding all paths and/or circuits.

Set \( \text{POLY}(J,K) = 0, \ J=1, \text{NEXPS}; \ K=1, \text{NTO} \)
\( \text{MSORT}(J) = 0, \ J=1, \text{NEXPS} \)
\( \text{KODI}(J) = 0, \ J=1, \text{NSPT} \)
\( \text{KSORT}(J) = 0, \ J=1, \text{NTO} \)
\( \text{IR}=1, \text{NFIR}=1, \text{KNO}=0 \)

Code node \( JS \) of the SFG as follows
\( \text{KODES}(JS) = 2 \times \text{KODES}(JS-1) \)
\( JS=2, \text{NNG} \)
where \( \text{KODES}(1) = 1 \)

\[\begin{align*}
\text{TEST} = 1 & \quad \text{Write out LISTP/IN & OUT} \\
\text{TEST} = 0 & \quad \text{Prepare to find circuits thru node 1} \\
\text{LISTP} & \quad \text{23 (Subprogram MAIN-4)} \\
\text{TEST} = 1 & \quad \text{Write out LISTP/IN & OUT} \\
\text{TEST} = 0 & \quad \text{24 To Subprogram MAIN-4} \\
\end{align*}\]
Subprogram MAIN-4

This program finds all paths from node NIN to node NOUT and/or all circuits of the SFG.

Preliminary (PFL-1)
IFLOW(I1)=0  I1=1,NNG
KONC(I1)=0  I1=1,NNG
NP=KLAS
KLAS=0

I = 2

Preliminary (PFL-2)
JX(1)=NIN
J=NIN
NP(1)=NIN
IFLOH(NIN)=1
IFLOW(OUT)=-1

K = KONC(J)

PF2(FIND NEXT NODE)
NP(I)=N(J,K)

PF3 (TEST ROUTING MATRIX)
\( N(J,K) \)

\(< 0\)
All loops thru a particular node found. Go to subprogram MAIN-5 to eliminate this node.

100
(Subprogram MAIN-5)

\(> 0\)
continue to flower check

All paths out of node J checked. Return to previous node

25
Prepare for next node
set IFLOW(J)=1
I=I+1

Flow formed - try a new path out of node J

Node J is acceptable

< 0
Node J is the final node

Loop Completed
Find the composite code for the circuit node list.

1
NP Code(IR) = \sum KODE[IS]
IS=1

Find composite code, exponent, and constant associated with branch weight.

1
KODET(IR) = \sum KODE[NP(KEW-1),NP(KEW)]
KEW=2

1
CONST(IR) = \prod CONS[NP(KEW-1),NP(KEW)]
KEW=2

1
IXPOT(IR) = \sum IXPOT[NP(KEW-1),NP(KEW)]
KEW=2

Set CONEW=CONST(IR), IXNEW=IXPOT(IR), KONEW=KODET(IR)
CALL ARRAY (1, CONEW, IXNEW, KONEW, POLY, L1, L, KIK)

Look for another loop
Subprogram MAIN-5

This program determines if circuits are to be found and if so modifies the SFG by eliminating node J.

Write out time to find paths

TEST NCIR

=0

Only paths between nodes NIN and NOUT to be listed

STOP

=1

The user has specified that circuits be found

Write out time to find paths. Set

N[1,LT(1)+1]=-1

NIN=1

NOUT=1

TEST NFIR

=0

Paths have been found - circuits thru node 1 are to be found

=1

The user has specified that circuits be found

Write out time to find paths. Set

N[1,LT(1)+1]=-1

NIN=1

NOUT=1

TEST NIN-JLAS

< 0

All circuits thru node J have been found. The SFG must be modified by eliminating node J from the routing table.

NIN=J+1

NOUT=J+1

N[J,LT(J)+1]=0

Test last nonzero entry in each row of N(I1,I2). If this entry equals J set N(I1,I2)=0 and LT(I1)=LT(I1)-1

N[NIN,LT(NIN)+1]=-1

(Subprogram MAIN-4)

(Subprogram MAIN-3)
Subprogram MAIN-6

This program finds and stores all 2nd order nontouching loops

Set
NOCTOT(K1)=0  K1=1,NPAC
LOOP(M1)=0  M1=1,NNG

DO 203 LIR1=NOP+1,NOL-1

DO 202 LIR2=LIR1+1,NOL

*AND* together the node code for loop LIR1 and the node code for loop LIR2
NAN=NPCODE(LIR1).AND.NPCODE(LIR2)

Loop LIR2 touches loop LIR1

TEST
NAN=0  loop LIR1
=0  Loop LIR2 does not touch loop LIR1

TCONS2=CONST(LIR1).CONST(LIR2)
KXP02=IXPOT(LIR1)+IXPOT(LIR2)
KSYM2=KODET(LIR1)+KODET(LIR2)

CALL ARRAY (2,TCONS2,KXP02,KSYM2,POLY,LIL,KIK)
The following defines arrays NOTCH and NOCTOT:
Consider the integer set
\[ \{I\} = \{1, 2, \ldots, N_2\} \]
where \(N_2\) = number of nontouching pairs of loops
Now consider the following subset of \(I\)
\[ \{I_B\} = \{\text{NOCTOT}(K-1)+1, \text{NOCTOT}(K-1)+2, \ldots, \text{NOCTOT}(K)\} \]
Then the set
\[ \{\text{NOTCH}(J) : J \in \{I_B\} \} \]
is the set of loops that do not touch loop \(K\)
Subprogram MAIN-7

This program finds all nontouching loops of order greater than 2, and stores the associated code, power of s, and constant term.

The matrix ISET is given below in its general form to aid in understanding the flow chart of subprogram MAIN-7.

\[
\begin{bmatrix}
ISET(1,1)&ISET(1,2)&\cdots&ISET[1,NUP(1)]&\cdots&ISET[1,JAC(1)] \\
ISET(2,1)&ISET(2,2)&\cdots&ISET[2,NUP(2)]&\cdots&ISET[2,JAC(2)] \\
\vdots&\vdots&\ddots&\vdots&\ddots&\vdots \\
ISET(KAP,1)&ISET(KAP,2)&\cdots&ISET[KAP,NUP(KAP)]&\cdots&ISET[KAP,JAC(KAP)] \\
ISET(KAP+1,1)&ISET(KAP+1,2)&\cdots&ISET[KAP+1,NUP(KAP+1)]&\cdots&ISET(KAP+1,JAC(KAP+1)) \\
\end{bmatrix}
\]

where \( ISET(J,I) \), \( J=1,2,\ldots,JAC(J) \) is the subset of

\( \{ISET[J-1,NUP(J-1)+1], ISET[J-1,NUP(J-1)+2], \ldots, ISET[J-1,JAC(J-1)]\} \)

which does not touch the loop

\( ISET[J-1,NUP(J-1)] \)
DO 490 NIP=NOP+1,NOL

Generate the first row of ISET as follows:
ISET(1,1)=NOTCH[NOCTOT(NIP-1)+1]
ISET(1,2)=NOTCH[NOCTOT(NIP-1)+2]
...
ISET[1,JAC(1)]=NOTCH[NOCTOT(NIP)]
The first row of ISET is seen to be the set of loops which do not touch loop NIP

Set JAC(K)=0 } K=1,NPAC
NUP(K)=0 }

KAP=2

[KAP is the row counter of ISET]

KAP=KAP-1

440

All higher order loops not touching loop NIP have been found

TEST
KAP

≤ 0

490

KAP=KAP+1

> 0

425

JAC(KAP+1)=0
NUP(KAP)=NUP(KAP)+1

429
ISAT = ISET(KAP, NUM(KAP))

DO 435 MAPI = NUM(KAP) + 1, JAC(KAP)

.AND., together the node code of loop ISAT and the node code of loop ISOT
KAN = NPCODE(ISAT).AND.NPCODE(ISOT)
where ISOT = ISET(KAP, MAPI)

TEST KAN

Find constant, code, and exponent for the nontouching loops of order KAP+1 just found.

\[ TCONSG = \text{CONST}(NIP) \times \text{CONST}(ISOT) \times \prod_{L=1}^{KAP} \text{CONST}[\text{ISET}(L, \text{NUM}(L))] \]

\[ KSYM = \text{KODET}(NIP) \times \text{KODET}(ISOT) + \sum_{L=1}^{KAP} \text{KODET}[\text{ISET}(L, \text{NUM}(L))] \]

\[ KXPOG = \text{IXPOT}(NIP) + \text{IXPOT}(ISOT) + \sum_{L=1}^{KAP} \text{IXPOT}[\text{ISET}(L, \text{NUM}(L))] \]

CALL ARRAY (KAP+2, TCONSG, KXPOG, KSYM, POLY, LIE, KIK)
Update column counter of row $KAP+1$ of $ISET$ and insert the last loop found into $ISET$

$JAC(KAP+1) = JAC(KAP+1) + 1$
$ISET[KAP+1,JAC(KAP+1)] = ISET(KAP,MAPI)$

Row $KAP+1$ of $ISET$ has now been found and it has more than 1 non-zero entries. Thus, proceed to find the entries of row $KAP+2$.}

Row $KAP+1$ of $ISET$ has now been found but it does not contain more than one non-zero entry. Since not all loops of row $KAP$ have been exhausted, increment $NUP(KAP)$ by one and re-evaluate row $KAP+1$.

Row $KAP+1$ of $ISET$ has now been found but it does not contain more than one non-zero element. Further, all loops of row $KAP$ have been exhausted. Thus, it is necessary to back up one row and re-evaluated row $KAP$.
CALL ARRAY (2,1.,0,0,POLY,LIL,KIK)

Print out number of loops found and time required in seconds.

To Subprogram MAIN-8
Subprogram MAIN-8

This program decodes composite codes representing nontouching loops and sets up tags for use in printing out the symbolic transfer function.

Set

\[
\begin{align*}
\text{POLYU(J1,J2)} &= 0 \quad J1=1, \text{HEXPS}; \quad J2=\text{NTO} \\
\text{SEMPON(J3,J4)} &= \text{STAR(1)} \\
\text{SEMPOD(J3,J4)} &= \text{STAR(1)} \\
\text{SIMBON(J3,J4)} &= \text{SB} \\
\text{SIMBOD(J3,J4)} &= \text{SB} \\
\text{NA(J5)} &= 0 \\
\text{NB(J5)} &= 0 \\
\end{align*}
\]

\(J3=1, \text{NTO}; \quad J4=\text{NSPTU}

\(J5=1, \text{NTO}\)

SB and STAR(1) are obtained from "data" statements.

DO 646 JZ=1,LIL-1

(LIL-1 different composite symbol codes have been found in Subprogram MAIN-7)

KODY=KSORT(JZ)

ITOP(JZ)=0

If ITOP(JZ) = 0

There is no symbol associated with this value of JZ

TEST

KODY

If KODY > 0

646
CALL DECODE (KOO, KBASIS, KODY, IZ, FB, JZ, SEMBOL, KDF, KODI, ITOP)

This subroutine (a) sets
1 if terms having the code KSORT(JZ)
ITOP(JZ) = { belong in numerator of output polynomial
0 if terms belong in denominator

(b) finds the set
KODI(I), I=1, IZ
where SEMBOL[KODI(I)], I=1, IZ are the corresponding
set of symbols, and

(c) finds the multiplicity of each individual symbol,
SEMBOL[KODI(I)], and records these values in the
array KDF(I), I=1, IZ

DO 645 NZ=1, IZ

=0

TEST KONS[KODI(NZ)]

=1

The symbol corresponding to KODI(NZ) is not to be
inverted in the output

NAK=NAK+1
SIMBON(JZ,NAK)=SEMBOL[KODI(NZ)]
SEMPON(JZ,NAK)=STAR[KDF(NZ)]
NA(JZ)=NA(JZ)+1

The symbol corresponding to KODI(NZ) is to appear
inverted in the output.
 i.e.
\[
\frac{1}{SEMBOL[KODI(NZ)]}
\]

NAT=NAT+1
SIMBOD(JZ,NAT)=SEMBOL[KODI(NZ)]
SEMPOD(JZ,NAT)=STAR[KDF(NZ)]
NB(JZ)=NB(JZ)+1
increment \text{NZ}

\begin{align*}
\text{NZ} & \quad \text{increment} \quad \text{NZ} \\
\text{DO loop} & \quad \text{increment} \quad \text{NZ} \\
\text{continue} & \quad \text{increment} \quad \text{NZ} \\
\text{continue} & \quad \text{increment} \quad \text{NZ} \\
\text{Subprogram MAIN-9} & \quad \text{increment} \quad \text{NZ}
\end{align*}
Subprogram MAIN-9

This program separates POLY into the arrays POLYU and POLY for use in printing out the constant terms of the transfer function.

DO 755 JA=1,KIK-1
There are KIK-1 rows in POLY (KIK-1 different powers of s)

DO 755 JC=1,LIL-1
There are LIL-1 columns in POLY (LIL-1 different composite symbol codes)

The entry POLY(JA,JC) belongs in the denominator of the transfer function

TEST
=0

ITOP(JC)

The entry POLY(JA,JC) belongs in the numerator of the transfer function

JIB=JIB+1
POLYU(JA,JIB)=POLY(JA,JC)

JD=JD+1
POLY(JA,JD)=POLY(JA,JC)

 Increment JC until JC=LIL-1 then increment JA

Print out time for decoding symbols

Subprogram MAIN-10
Subprogram MAIN-10

This program normalizes the transfer function so as to have all positive powers of $s$.

Subprogram MAIN-11

Using MAXIM make powers of $s$ either positive or zero. i.e.,

$$\text{MSORT}(K) = \text{MAXIM} + \text{MSORT}(K)$$

$K = 1, 2, \ldots, KIK-1$
Subprogram MAIN-11

Write out the matrix of constant coefficients for the numerator polynomial of the transfer function. Also write out the symbols and $s$ powers that correspond respectively to the columns and rows of the array of constants.

Subprogram MAIN-12

Write out the matrix of constant coefficients for the denominator polynomial of the transfer function. Also write out the symbols and $s$ powers that correspond respectively to the columns and rows of the array of constants.
Subroutine SFG(NFIRST,NLAST,IXPO,WEIGHT,SYMBOL,KONSO,MIX,NEST,LIST,NIN,
NOUT,NOD,NOB,LISTG,NODA,NODB)

This subroutine generates a signal flow-graph (SFG) for the given network. The program is subdivided into subprograms A thru J.
Subprogram "A"

This program uses DATA statements to define certain variables, nulls arrays, creates the SFG feedback branch, reads in network branch information and calls FTREE to find a tree.

Use DATA statement to define the following variables: Y,G,C,IQ,R,CL,Z,E,CI,CC,CV,VV, VC,FB,ONE.

Set NS(IC,IK)=0 IC=1,NNG; IK=1,NNG
   NF(IC,IK)=0
   NEST(IG)=0 IG=1,NBG
   KONSO(IG)=0
   INTREE(II)=0 II=1,NNG
   JROW(II)=0
   MO=0,LO=0,LIST=1,LINK=0

Create SFG feedback branch used to make the SFG "closed".
   NLAST(1)=NIN
   IXPON(1)=0
   WEIGT(1)=-1
   SYMBUL(1)=FB
   KONSO(1)=0
   NEST(1)=1

Note: NFIRST(1) is determined in Subprogram "I"
Read in the following network branch information:
TYPX(I), NUMX(I), JBX(I), LBX(I), SYMX(I), IQUALX(I), VALX(I), NUMLX(I)
I=1,NOB

Choose a tree of the network for use in finding the SFG
CALL FTREE(TYPX, JBX, LBX, INTRE, NOTREE, NOD, NOB)

Subprogram "B"
Subprogram "B"

This program sets up tree branch information and creates a routing matrix and sign matrix for the tree.

DO 21 NU=1, NOD-1

Put network tree branch information in terms of the branch labels specified by user.
Let IO=INTRE(NU), NUMC=NUNX(IO)
Then TYPB(NUMC)=TYPX(IO)
   JB(NUMC)=JBX(IO)
   LB(NUMC)=LBX(IO)
   SYM(NUMC)=SYMX(IO)
   IQUAL(NUMC)=IQUALX(IO)
   VAL(NUMC)=VALX(IO)
   NML(NUMC)=NMLX(IO)
   INTREE(NUMC)=1

WRITE TYPB(NUMC), NUMC, JB(NUMC), LB(NUMC),
   SYM(NUMC), IQUAL(NUMC), VAL(NUMC), NML(NUMC)

Compare TYPB(NUMC) to VV
   .EQ.  MD=MD+1
   .NE.

Compare TYPB(NUMC) to CV
   .EQ.  LO=LO+1
   .NE.
IB(JF,LF) = NUMC
IB(LF,JF) = NUMC

The following generates a routing matrix, NF, and a sign matrix, NS, from the chosen network tree for use in finding the SFG:

JF = JB(NUMC)
LF = LB(NUMC)
JROW(JF) = JROW(JF) + 1
NF[JF,JROW(JF)] = LF
NS[JF,LF] = 1
JROW(LF) = JROW(LF) + 1
NF[LF,JROW(LF)] = JF
NS[LF,JF] = -1

Increment NU

Continue

Subprogram "C"
Subprogram "C"

This program generates SFG information from branch node to link node.

```
NES=0
NOBY=NOB

Continue

TEST KLU-NOB = 0
< 0

LINK=LINK+1
Initially 0

TEST NOTREE(LINK) = 1

= 0  Network branch NUMX(LINK) is not in tree
```

Put network link information in terms of the branch labels specified by user.
Let NUMC=NUMX(LINK)
then
```
TYPE(NUMC)=TYPEX(LINK)
JK=JBX(LINK)
LK=LBX(LINK)
SYM(NUMC)=SYMX(LINK)
IQUAL(NUMC)=IQUALX(LINK)
CVAL(NUMC)=VALX(LINK)
NUMB=NUMLX(LINK)
```

```
LINK(NUMC)=1
This records which network branches are not in the tree.
```
CALL TREP(JK, LX, HF, HF, NPL)
TREP find the path in the network tree which forms a circuit with the link NUHC.

IFIN=NUHC

LOM=LOM+1

Counter LOM will increment over the number of tree branches contained in the fundamental circuit defined by the link NUHC. For each tree branch in this list, a SPG branch will be formed from NUHC to the given tree branch.
INIT=IB[FP(LON),NP(LON+1)]
SIGN=NS[NP(LON),NP(LON+1)]

KDEPS

=1

=0 [tree branch INIT] Subprogram "D"

=0 [tree branch INIT] is passive

COMPARE
IQUAL(NUMC) and IQ

.NE.

=1

.EQ.

[Network branch NUMC has no symbol]

CONST=SIGN=CUAL(NUMC)

LIST=LIST+1

TEST NES

=0

=1

NEST(LIST)=1

Specify all branch information for LIST
KONSO(LIST)=KANSO
NFIRST(LIST)=INIT
MLAST(LIST)=IFIN
SYMUL(LIST)=SYH(IFIN)
IXFON(LIST)=IXPS
Link weight is given in impedance units and thus must be inverted.

\[
\text{WEIGHT(LIST)} = \frac{1}{\text{CONST}}
\]

\[
\text{MAPY(NUMC)} = \text{LIST}
\]

Subprogram "D"
Subprogram "D"

This program generates SFG information from link node to branch node.

```
TEST TYPB(INIT)
  .EQ. E, I, VV, or CV
  .NE. E, I, VV, or CV
  LIST = LIST + 1

TEST TYPB(INIT)
  .EQ. R or Z
  IXPON(LIST) = 0
  KONSO(LIST) = 1

  .EQ. C
  IXPON(LIST) = -1
  KONSO(LIST) = 1

  .EQ. L
  IXPON(LIST) = 1
  KONSO(LIST) = 1

  .EQ. G or Y
  IXPON(LIST) = 0
  KONSO(LIST) = 1

  COMPARE IQUAL(INIT) to IQ
  .EQ.
  TEST KONSO(LIST)
    = 0
    WEIGHT(LIST) = -SIGN * VAL(INIT)
    = 1
    WEIGHT(LIST) = -SIGN / VAL(INIT)
  .NE.
  NEST(LIST) = 1
  WEIGHT(LIST) = -SIGN
```

Subprogram tiC

NFIRST(LIST)=FIN
NLAST(LIST)=INIT
SYMBUL(LIST)=SYM(INIT)

NPLA=NPL-1-LEN

149 Subprogram "C"  = 1

TEST

NPLA

≤ 0

151 Subprogram "C"
Subprogram "E"

This program sets up SFG information for VC type control sources.

165 Numb was found from

\text{\texttt{NUMB=NUNLX(LINK)}}

Thus, Numb is the branch number of the control for the VC type source.

\text{\texttt{NUMNO=NUMB}}

\text{\texttt{TEST=INTREE(NUMB)}}

161

\text{\texttt{=1}}

\text{\texttt{=0}}

The network branch whose voltage is the controlling variable for the VC source is not in the tree. Thus a new node must be created in the SFG to represent the voltage controlling variable.

Create a SFG branch from node

\texttt{NUMB} (current thru controlling network branch) to new node

\texttt{NOBY} (voltage across controlling network branch)

\texttt{LIST=LIST+1}

\texttt{NFIRST(LIST)=NUMB}

\texttt{NOBY=NOBY+1 \texttt{(Initially NOBY=NOB)}}

\texttt{NLAST(LIST)=NOBY}

\texttt{SYMBUL(LIST)=SYM(NUMB)}

\text{\texttt{NUMNO=NOBY}}
Using the newly created SFG node representing the controlling voltage, a SFG node can now be generated for the dependent current source.
Subprogram "C"

Network branch NUMC contains a symbol

NE

NEST(LIST)=1
WEIGHT(LIST)=1.
Subprogram "F"

This program sets up SFG information for CC type control sources.

The structure of subprogram "F" is completely analogous to subprogram "E".

Subprogram "G"

This program sets up SFG information for VV type control sources.

This program is cycled over all voltage controlled voltage sources in the network. A list of these sources is maintained in the array $IVV(MI), MI=1,M0$

Other than the above, the structure of subprogram "G" is completely analogous to subprogram "E".

Subprogram "H"
Subprogram "H"

This program sets up SFG information for CV type control sources.

This program is cycled over all current controlled voltage sources in the network. A list of these sources is maintained in the array ICV(MI), MI=1,LO

Other than the above, the structure of subprogram "H" is completely analogous to subprogram "E"

515 Subprogram "I"

The SFG is now complete except for the output node.
This program generates the output node of the SFG.

User has specified that output is between nodes NODA and NODB of the network. The SFG output node NOUT is made a function of the tree branch voltages existing in the path between NODA and NODB.

CALL TREP (NODA,NODB,NP,NP,NPL)

NOUT=NODA+1

DO 510 MOP=1,NPL-1

LIST=LIST+1

510 [End of DO loop]

Generate a SFG branch between node \( IB[NP(MOP),NP(MOP+1)] \) and NOUT
\( \text{NFIRST}(\text{LIST})=IB[NP(MOP),NP(MOP+1)] \)
\( \text{NLAST}(\text{LIST})=\text{NOUT} \)
\( \text{SYMBUL}(\text{LIST})=\text{ONE} \)
\( \text{IXCON}(\text{LIST})=0 \)
\( \text{KONSO}(\text{LIST})=0 \)
\( \text{NEST}(\text{LIST})=0 \)
\( \text{WEIGHT}(\text{LIST})=\text{NS}[NP(MOP),NP(MOP+1)] \)
This is the initial node of the SFG feedback branch

Write out the following SFG information:
NFIRST(J), NLAST(J), IXPON(J), WEIGHT(J), SYMBOL(J), KONSO(J), NEST(J)
J=1, LIST

Subprogram "J"
Subprogram "J"

This program orders SFG information for input to main program. That is, a mapping function MIX is found which reorders the SFG information so that the routing matrix N(J,K) as calculated in MAIN will automatically have its entries decrease as K increases.

\[
\text{SET MIX}(J) = J, J=1, \text{NBG}
\]

\[
\text{DO 80 } K_{\text{ON}} = 1, \text{LIST}-1
\]

\[
\text{IU} = K_{\text{ON}} + 1, \text{IL} = K_{\text{ON}}
\]

\[
83
\]

\[
\text{TEST } \text{NFIRST}[\text{MIX}(\text{IU})] - \text{NFIRST}[\text{MIX}(\text{IL})] > 0
\]

\[
\text{TEST } \text{NFIRST}[\text{MIX}(\text{IU})] - \text{NFIRST}[\text{MIX}(\text{IL})] < 0
\]

\[
\text{[MIX}(\text{IU}) \text{ and MIX}(\text{IL}) \text{ must be interchanged]}
\]

\[
\text{MIX}(\text{IU}) = \text{MIX}(\text{IL})
\]

\[
\text{MIX}(\text{IL}) = \text{MIX}(\text{IU})
\]

\[
\text{MIX}(\text{IU}) = \text{MIX}(\text{IL})
\]
IL = IL - 1
IU = IU - 1

= 0

TEST IL

> 0

MIX(IU) and MIX(IL) must be interchanged

MIXL = MIX(L(IL))
MIX(IL) = MIX(IU)
MIX(IU) = MIXL

NLAST[MIX(IU)] - NLAST[MIX(IL)]

≤ 0

> 0

TEST

MIX(IU) and MIX(IL) must be interchanged

IL = IL - 1
IU = IU - 1

= 0

TEST IL

> 0

NFIRST[MIX(IU)] - NFIRST[MIX(IL)]

#0

Continue
Subroutine FTREE(TYPX, JBX, LBX, INTRE, NOTREE, NOD, NOB)

This program finds a tree of the network to use in generating a SFG.

Null the arrays
NF(12,13), KCOL(14), NOTREE(15)

\[ I = I + 1 \]
Initially \( I = 0 \)

\[ \text{TEST TYPX}(I) \]

\[ \neq E, VV, \text{or CV} \]

\[ K = K + 1 \]

\[ \text{INTRE}(K) = I \]
\[ \text{NOTREE}(I) = 1 \]

The branch just selected for the tree must now be included in a routing matrix. This matrix will be used later to test passive network branches for use in the tree.

\[ \text{NF}[\text{JBX}(I), \text{KCOL}(\text{JBX}(I))] = \text{LBX}(I) \]
\[ \text{NF}[\text{LBX}(I), \text{KCOL}(\text{LBX}(I))] = \text{JBX}(I) \]

\[ \text{TEST} \]
\[ K = \text{NOD} + 1 \]
\[ \geq 0 \quad \text{[Tree filled]} \]
\[ < 1 \]

\[ \text{TEST} \]
\[ M = 0 \]

\[ > 0 \]

Passive network branches are now being considered

\[ = 0 \]

The list of network branches are still being examined for voltage sources

All voltage sources have been chosen. The list of network branches are again examined and the necessary number of passive branches selected.
A passive branch has been tentatively accepted for the tree. It is now necessary to go to the path finding subroutine TREP to see if the branch just selected will form a loop.

CALL TREP

The passive branch just selected for the tree must now be entered into the routing matrix.
Subroutine TREP(NIN,NOUT,NF,NI,NPL)

Given the routing matrix for the network tree with input and output nodes specified, this subroutine finds a node list representing the path between these two nodes.

Preliminary
Set JX(I5)=0
NP(I5)=0
JMEM(I5)=0
KMEM(I5)=0
NPL=0, JX(1)=JX(2)=NIN,
I=1, J=NIN, NP(1)=NIN

20 K=0

25 K=K+1

Path finished

TEST NF(J,K)=NOUT ≠0

TEST NF(J,K) =0

TEST J-NIN ≠0

> 0 Flower check

TEST NF(J,K)=JX(I) =0

Flower formed

=0

≠0 Backstep

=0 no path found

45

50

100

60

100

100

100

100
Store and remember vertex
I=I+1
NP(I)=NP(J,K)
JMEM(I)=J
JX(I+1)=NF(J,K)
J=NP(J,K)
KMEM(I)=K

Backstep
J=JMEM(I)
K=KMEM(I)
I=I-1

Final path vertex and path length
NP(I+1)=NOUT
NPL=I+1

Return
Subroutine $\text{ARRAY}(\text{JSIG}, \text{XCON}, \text{JXPO}, \text{JKOD}, \text{POLY}, \text{LIL}, \text{KIK})$

This subroutine takes the constant associated with each loop or non-touching combination of loops in the SFG and stores it in the matrix $\text{POLY}$. It does this by comparing the code and exponent of the given loop combination with the codes and exponents assigned to the columns and rows of $\text{POLY}$ respectively.
DO 12 NN=1,LIL-1

10 TEST LIL-1

24 ≠ 0

DO 12 NN=1,LIL-1

NNX=NNX-1

20 ≠ 0

12 [End of DO loop]

Continue

24

KSORT(LIL)=JKOD
NNX=LIL
LIL=LIL+1

20

POLY(MMX,NNX)=POLY(MMX,NNX)+XCON*(-1.)*JSIG

Return
Subroutine DECODE(KOO, KODY, IZ, FB, JZ, SEMBOL, KODF, KODI, ITOP, KBASIS)

This program decodes the composite symbol codes.

IZ=0
M=KBASIS-1

DO 3 J=1, KOO

IPOWER=M.AND.KODY

TEST
IPOWER

= 0

≠ 0

KODY contains code for SEMBOL(J)

COMPARE
SEMBOL(J) to FB

.KEQ.
The term having code KODY belongs in numerator of transfer function

.IEQ.

IZ=IZ+1
KODF(IZ)=IPOWER
KODI(IZ)=J

ITOP(JZ)=1

3 [End of DO loop]

KODY=KODY/KBASIS

increment J
IV-3. Program Listing

***SNAP***

THIS PROGRAM FINDS THE SYMBOLIC TRANSFER FUNCTION OR IMPEDANCE FUNCTION OF A LUMPED LINEAR TIME INVARIANT NETWORK

C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC N<BR>
DIMENSION LT(35),IG(35),SYM(35),CONS(35),KONC(35)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NG
DIMENSION NLST(100),NLAST(100),IXPON(100),CLIG(100)
DIMENSION SYM(100),IX(100),CV(100)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NH
DIMENSION N(35,35),CONS(35,35),KONC(35,35),IAP(35,35)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC N<BR>
DIMENSION NLST(100),NLAST(100),IXPON(100),CWEIGHT(100)
DIMENSION SYM(100),IX(100),CV(100)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NH
DIMENSION NLST(100),NLAST(100),IXPON(100),CWEIGHT(100)
DIMENSION SYM(100),IX(100),CV(100)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NH
DIMENSION NLST(100),NLAST(100),IXPON(100),CWEIGHT(100)
DIMENSION SYM(100),IX(100),CV(100)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NH
DIMENSION NLST(100),NLAST(100),IXPON(100),CWEIGHT(100)
DIMENSION SYM(100),IX(100),CV(100)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NH
DIMENSION NLST(100),NLAST(100),IXPON(100),CWEIGHT(100)
DIMENSION SYM(100),IX(100),CV(100)
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NH
DIMENSION NLST(100),NLAST(100),IXPON(100),CWEIGHT(100)
DIMENSION SYM(100),IX(100),CV(100)
THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
CHARACTERISTICS NRI, NCI, NEON, AND NRS
DIMENSION ISET(15,100), NOTCH(1200), STAR(9)

COMMON SEMP, SEMP, PULY
COMMON/C1/MSOH, KSORT
COMMON/C2/NNG, NBG
COMMON/C3/NSPT, NTO
COMMON/C4/NSPT
EQUIVALENCE (ISET(1,1), NOTCH(1), SIMBON(1,1))
EQUIVALENCE (CONS(1,1), ISET(1,1), SIMMOD(1,1))
EQUIVALENCE (KOUT(1,1), PULYU(1,1))
DATA DASH/2H/
DATA FH, SH/3M FH, 3H /
DATA STAR(1), STAH(2), STAH(3), 3H**2, 3H**3/
DATA STAR(4), STAH(5), STAH(6), 3H**4, 3H**5, 3H**6/
DATA STAR(7), STAH(8), STAH(9), 3H**7, 3H**8, 3H**9/
DATA ONE/3H /

PROGRAM MAIN--1
PRELIMINARY INPUT INFORMATION

NBR=NUMBER OF BRANCHES IN NETWORK
NNG=NUMBER OF BRANCHES OF SFG
NTO=NUMBER OF TERMS IN OUTPUT
NSPT=NUMBER OF SYMBOLS PER TERM
NEXPS=NUMBER OF DIFFERENT POWERS OF S
NPAC=NUMBER OF PATHS PLUS CIRCUITS
NRT=MAXIMUM NUMBER OF NONTOUCHING LOOPS
NCI=MAXIMUM NUMBER OF LOOPS NOT TOUCHING ANY GIVEN LOOP
NEON=NUMBER OF NONTOUCHING PAIRS OF LOOPS
NRS=NUMBER OF REPEATED SYMBOLS(NUMBER OF NETWORK ELEMENTS ASSIGNED SAME SYMBOL)
NBR=35
NNG=100
NTO=150
NSPT=20
NEXPS=15
NPAC=300
NRT=15
NCI=100
NEON=1200
NRS=9

NSPTU=NUMBER OF SYMBOLS IN NUMERATOR OF EACH TERM
NBT=NUMBER OF BRANCHES IN TREE OF SFG
NNG=NUMBER OF NODES IN SFG
NNG=NBR
NSPTU=NSPT/2
NBT=NBR
CALL SECOND (TCOMP)
WRITE(6,160A) TCOMP
1600 FORMAT (1X,6H COMPIILATION TIME IN SECONDS, F15.8)
111 CONTINUE
WRITE (6,519)
519 FORMAT (1H1)
CALL SECOND (TSTART)
THE NEXT 6 CARDS ARE FOR PROBLEM IDENTIFICATION ON THE 1ST DATA CARD
PROGRAM MAIN--2

C

C TAKE SFG BRANCH INFORMATION AS FOUND
C BY SUBROUTINE SFG AND GENERATE
C (1) ROUTING MATRIX INFORMATION
C N(J,K), AND LI(J)
C (2) SFG BRANCH VALUES IXPO(J,L), CONS(J,L),
C KID(J,L) WHERE J=FIRST(I), L=LAST(I), AND
C I=BRANCH NUMBER
C TOGETHER WITH THE SYMBOL SEMHOL(K), K=1,2,...,M1
C
C CALL SUBROUTINE TO FORMULATE THE SIGNAL FLOW GRAPH, SFG
CALL SFG (FIRST,NLAST,IXPO,WEIGT,SYMBOH,CONS,MIX,FIRST,LST,
INOUT,NOUT,NODA,NODB)
IF (NOT) 1111, 1111, 1920

1920 CONTINUE
CALL SECOND (T2)
WRITE (6, 93A)
TSGF=T2-TSTART
WRITE (6, 1602) TSGF

1602 FORMAT (IX, 12, TIME FOR FORMULATING SIGNAL/
122H FLOW GRAPH IN SECONDS, F15.3/)
IHO=0
K=0
MICH=1
K=0
MG=1
JLAS=1
NCIR=1
ININ=NIN
OUT=NOUT
KOO=0
DO 301 INK=0, NSPT
301 KONS(INK)=0
DO 300 INK=1, NNG
300 IG(INK)=0

C FIND IXPO(J,L), CONS(J,L)
GO TO 307
305 MG=KBASIS*MG
MICH=MICH+1
307 IHO=IHO+1
IF (LIST=IHO) 14, 4, 4
4 CONTINUE
LOB=MIX([HO)
J=FIRST(LOB)
L=LAST(LOB)
IXPO(J,L)=IXPO(J,L)
CONS(J,L)=WEIGT(LOB)

C FIND ROUTING MATH
8 IF (J.EQ. JLAS) GO TO 10
LT(JLAS)=K
K1=K+1
IF (JLAS=NIN) 28, 27, 28
27 N(JLAS,K1)=-1
GO TO 29
28 N(JLAS,K1)=0
29 JLAS=JLAS+1
K=0
GO TO 8
10 K=K+1
N(J,K)=L

C FIND KMOE(J, L) AND SLOB(KOO)
SMLOB(IHO)=SMLOB(LOB)
MODE=NEST(LOB)
IF (MODE) 335, 316, 335
335 IF (IG(L)) 5+960+5
5 KODE(J,L)=IG(L)
GO TO 307
960 CONTINUE
KPU=IHO-1
IF (KPU) 953+93, 315
315 DO 952 KP=1, KPU

IF (SMHOL (IQN) .NE. SMHOL (KP)) GO TO 952
LORX=MIX (KP)
IF (KONS0 (LOA) - KONS0 (LORX)) 952, 956, 952

956 LX=NAST (LORX)
KODE (J, L) =1A (LX)
GO TO 307
952 CONTINUE
IF (SMHOL (IQN) .EQ. ONE) GO TO 316
953 IG (L) = M6
K00=K0O+1
SMBOL (K00) = SMBOL (I00)
KODE (J, L) =1A (L)
IF (KONS0 (LOA)) 3, 3, 2
2 KONS0 (K00) =1
3 CONTINUE
GO TO 305
316 KODE (J, L) =0
GO TO 307
19 LT (JLAS) = K
K11 = K+1
N (JLAS, K11) =U

C PROGRAM MAIN--3
C NULL CERTAIN ARRAYS, SET COUNTERS, AND DEFINE
C A CODE FOR EACH NODE OF THE SFG

MPL = O
K1K =1
LIL =1
DO 601 KAM=1,NEXPS
DO 601 KIM=1,NTO
601 POLY(KAM,KIM)=0
DO 602 KP1=1,NEXPS
602 MSORT(KP1)=U
DO 950 K02=1,NSPT
950 K02=K02+1,NSPT
603 KSORT(KP2)=K
IR=1
NFIR=1
KNO=0
KODES (1) =1
DO 2000 JS=2,NNG
2000 KODES (JS) =29KODES (JS-1)
IF (LISTP) 175, 175, 1116
1116 WRITE (6, 170) NIN, NOUT
170 FORMAT (17H, GATS FROM NOUE 12,9H TO NOUE 12/)
WRITE (6, 1905)
1905 FORMAT (5X, 17HNO., NODE LIST)
175 CONTINUE
IF (LISTP) 113, 1113, 23
1113 K3=L T(NIN)+1
N(NIN+K3)=0
K2=L T (1)+1
N (1, K2) =-1
NIN=1
NOUT=1
KLAS=O
24 NFIR=O
PROGRAM MAIN=4
PATH-FINDING ALGORITHM
IN ADDITION, STEPS PF7 CALCULATES THE COMPOSITE CODE, CONSTANT, AND EXPONENT OF THE PATH

C PF1 (PRELIMINARY)
DO 1112 IZO=1,NNG
1112 IFLOW(IZO)=0
   DO 31 11=1,NNG
   31 KONC(I1)=1
   NOP=KLAS
   KLAS=0
23 I=2
   J=NIN
   NP(I1)=NIN
   IFLOW(NIN)=1
   IFLOW(NOUT)=1
C 25 K=KONC(J)
C PF2 (FINI) NEXT NODE
   NP(I1)=N(J,K)
C PF3 (TEST ROUTING MATRIX)
   IF(N(J,K))100,60,34
C PF4 (TEST FOR FLOWER)
   34 NJK=M(J,K)
   IF(IFLOW(NJK))50,38,26
   26 KONC(J)=KONC(J)+1
   GO TO 25
C PF5 (PREPARE FOR NEXT NODE)
   38 J=NP(I1)
   IFLOW(J)=1
   I=I+1
   GO TO 25
C PF6 (BACKSTEP)
   60 IFLOW(J)=0
   KONC(J)=1
   J=NP(I=2)
   KONC(J)=KONC(J)+1
   I=I-1
   GO TO 25
C PF7 (FINISH PATH)
   50 KONC(J)=KONC(J)+1
   KLAS=KLAS+1
C C FIND CODE FOR NODE PATH
NPCODE(IR) = 
ISU=I-1
UO 2002 IS=1, ISU
NODS=NP(IS)
2002 NPCODE(IR) = NPCODE(IR) \times \text{KODES(NOUS)}

C
C CALL ARRAY AND WRITE
IF (NFIR.EQ.1) GO TO 179
IF (LISTC) 1208, 1208, 1206
1206 CONTINUE
KRU=1
179 KNO=KNO+1
WRITE(6*110) KNU, (NP(KK),KR=1,KNU)
110 FORMAT (4X,I3,6X,35I3)
1208 CONTINUE
C
IF (NFIR.EQ.1) GO TO 320
KODET(IR)=0
CONST(IR)=1.
IXPOT(IR)=0
IEND=I
100 IF (UO 319 KEW=2, IEND)
JNODE=NP(KEW-1)
LNODE=NP(KEW)
KODET(IR) = KODET(IR) + KODE(JNODE, LNODE)
CONST(IR) = CONST(IR) + CONS(JNODE, LNODE)
IXPOT(IR) = IXPOT(IR) + IXPOT(JNODE, LNODE)
319 CONTINUE
CONEW=CONST(IR)
IXNEW=IXPOT(IR)
KNEW=KODET(IR)
CALL ARRAY(1, CONEW, IXNEW, KNEW, POLY, LIL, KIK)
320 CONTINUE
C
IR=IR+1
IF (IR=NPAC) 361, 1361, 1360
1360 WRITE(6*1362)
1362 FORMAT (4X,3YNNO, OF CIRCUITS EXCEEDS LIMIT, INCREASE ,
126HDIMENSIONS CONTAINING NPAC)
1361 CONTINUE
GO TO 25

C
PROGRAM MAIN=5
C MODIFY THE SFA BY REMOVING EVERY BRANCH CONNECTED
NOT TO THE NODE THROUGH WHICH ALL CIRCUITS HAVE JUST
BEEN FOUND
C
100 N3=0.
IF (NCIR=1) 2610, 102, 2010
102 CONTINUE
IF (NFIR=1) 104, 2010, 104
103 K4=LT(NIN)*1
N(NIN,K4)=0
K5=LT(1)+1
N(1,K5)=-1
NOUT=1
GO TO 24
104 IF(NIN=JLAS)106,200,200
105 NIN=J+1
        NOUT=J+1
        KONC(J)=1
        NY=LT(J)+1
        NJ(N,J)=0
        DO 109 JC=NTN,JLAS
        LCOL=LT(JC)
        IF(LCOL=EV(J)) GO TO 109
        IF(N(JC,LCOL)=J)109,107,109
107 N(JC,LCOL)=0
        LT(JC)=LT(JC)-1
109 CONTINUE
        NZ=LT(NIN)+1
        N(NIN,NZ)=1
        NOUT=NIN
        GO TO 23
2010 CALL SECOND(T3)
        TPATH=T3-T12
        WRITE(6,2012)NOUT,TPATH
2012 FORMAT(1X,1HTIME FOR FINDING,110,17H PATHS,IN SECONDS,F15.3/)
        IF(NCIR=1)250,103,250
200 CONTINUE
        NOL=KLAS
        CALL SECON{T4)
        IF(T3)2014,2020,2014
2014 TCIR=T4-T3
        GO TO 2016
2020 TCIR=T4-T2
2016 WRITE(6,1601)NOL,TCIR
1601 FORMAT(1X,16HTIME FOR FINDING,110,18H FIRST ORDER LOOPS,/
111H IN SECONDS,F15.3/)

C PROGRAM MAIN--6
C FIND SECON ORDER LOOPS

        NOL=KLAS
        KMOL=0
        DO 257 KOW=1,NPAC
257 NOCTOT(KOW)=0
        LOW1=NOP+1
        NOC=0
        NOL1=NOL-1
        DO 203 L1H1=LOW1,NOL1
        LOW2=L1H1+1
        DO 202 L1H2=LOW2,NOL
        NAN=NPCODE(L1H1)*AND,NPCODE(L1H2)
        IF(NAN)202,201,202
201 CONTINUE
        TCONS2=CONST(L1H1)*CONST(L1H2)
        KXP02=IXPOT(L1H1)*IXPOT(L1H2)
        KSYM2=KODET(L1H1)*KODET(L1H2)
        CALL ARRAY(3,TCONS2,KXP02,KSYM2,POLY,L1H1,K1K)
        KMOL=KMOL+1
        NOC=NOC+1
        IF(NOC=NEUN)1396,1396,1395
1395 WRITE(*,1397)
1397 FORMAT(1X,46HINCREASE NEUN-THE DIMENSION OF THE ARRAY NOTCH)
1396 CONTINUE
PROGRAM MAIN--7
C
FIND ALL LOOPS OF ORDER
GREATER THAN 2
C
GENERATE THE FIRST ROW OF ISET
NIPL=NOP+1
KAPMAX=1
INKO=1
DO 1170 ISC=NIPL,NOL
INK1=NOCTOT(ISC)
IF(ISC=1)1171,1171,1172
1172 INK2=NOCTOT(ISC-1)+1
GO TO 1173
1171 INK2=1
1173 IF(INK1=INK2=INKO)1170,1170,1175
1175 INKO=INK1-INK2
1170 CONTINUE
IF(INKO=NCI)1391,1391,1390
1390 WRITE(6,1392) INKO
1392 FORMAT(1X,52HINCREASE NCI-THE NO. OF COLUMNS IN DIMENSION OF ISET)
1391 CONTINUE
DO 490 NIP=NIPL,NOL
INKU=NOCTOT(NIP)
IF(NIP=1)210,210,211
211 INKL=NOCTOT(NIP-1)+1
GO TO 212
210 INKL=1
212 CONTINUE
IF(INKU=INK1)490,490,410
410 JIP=0
DO 480 INK=INKL,INKU
JIP=JIP+1
480 ISET(1,JIP)=NOTCH(INK)
MAPO(NIP)=INKU-INKL+1
C
INIMATE PROCESS FOR FINDING
C HIGHER ORDER LOOPS
DO 430 KAT=N PAC
JAC(KAT)=0
430 NUP(KAT)=0
JAC(1)=MAPO(NIP)
KAP=2
440 KAP=KAP+1
IF(KAP=490,490,429
425 KAP=KAP+1
IF(KAP=NCI)1350,1350,1351
1351 WRITE(6,1352)
1352 FORMAT(1X,52HINCREASE NK1-THE NO. OF ROWS IN DIMENSION OF ISET)
1350 CONTINUE
NUP(KAP)=0
429 KAP=KAP+1
JAC(KAP)=0
NUP(KAP)=NUP(KAP)+1
C

LABEL LOOP OF FIRST CKT
NAP = NUP(KAP)
IF (KAPMAX = KAP) 1347, 1348, 1348
1347 KAPMAX = KAP
1348 CONTINUE
ISAT = ISET(KAP, NAP)
C

TEST LOOP OF REMAINING CKTS
MAPU = JAC(KAP)
MAPL = NUP(KAP) + 1
DO 435 MAPL = MAPL, MAPU
ISOT = ISET(KAP, MAPL)
KAN = NPOCUE(ISAT) AND NPOCUE(ISOT)
IF (KAN) 435, 455, 435
455 CONTINUE
C

WRITE
TCONSG = CONST(NIP)
KXPOG = IXPOT(NIP)
KSYM = KODET(NIP)
DO 477 LPO = 1, KAP
ITIC = NUP(LPO)
ITUCH = ISET(LPO, ITIC)
TCONSG = TCONSG * CONST(ITUCH)
KXPOG = KXPOG, IXPOT(ITUCH)
477 KSYM = KSYM * KODET(ITUCH)
TCONSG = TCONSG * CONST(ISOT)
KXPOG = KXPOG, IXPOT(ISOT)
KSYM = KSYM * KODET(ISOT)
KAPP = KAP + 2
CALL ARHAY(KAPP, TCONSG, KXPOG, KSYM, POLY, LIL, KIK)
KHOL = KHOL + 1
C

SET COUNTERS
423 KAP1 = KAP + 1
JAC(KAP1) = JAC(KAP1) + 1
JACK = JAC(KAP1)
ISET(KAP1, JACK) = ISET(KAP, MAPL)
435 CONTINUE
JACK = JAC(KAP1)
IF (JACK = 2) 421, 425, 425
421 IF (JAC(KAP) = NUP(KAP) - 1) 440, 440, 425
490 CONTINUE
CALL ARHAY(2, 1, 0, 0, POLY, LIL, KIK)
CALL SECOND(15)
TNTL = T5 - T4
WRITE (6, 1604) KHOL, TNTL
1604 FORMAT(I11, 'TIME FOR FINDING ' || I10, ' RH SETS OF /
130H NONTOUCHING LOOPS, IN S<CONUS, F15.3/)
C

PROGRAM MAIN--R
C

DECODE COMPOSITE SYMBOL CODE
C

AND ISOLATE SYMBOLS FROM
C

INVERSE SYMBOLS

NANU = LIL = 1
DO 691 J1 = 1, NEXP
DO 691 J2 = 1, NST
691 POLYU(J1, J2) = 0
DO 693 J1 = 1, NST
693
DO 693 J2=1,NSPTU
  SEMPON(J1,J2)=STAR(1)
  SEMP0D(J1,J2)=STAR(1)
  SIMBON(J1,J2)=SB
693 SIMBOD(J1+J2)=SH
  DO 951 J4=1,NTU
  NA(J4)=0
951 NA(J4)=0

C  DECODE KSORT(J7) AND RECORD ITEMS
  CONTAINING FEEDBACK SYMBOL #F3#
    JZU=LIL=1
    DO 646 JZ=1,JZU
    KODY=KS0H(JZ)
    ITOF(JZ)=0
    IF(KOY)715,646,715
  715 CALL DECODE(KOO,KODY,JZ+SH,JZ,SEM0L,KODF,KOUI,ITOP,KHAS0L)

C  ISOLATE NUM. SYMBOLS AND INVERSE SYMBOLS
  OF KSORT(JZ)
637 NAK=0
    NAT=0
    IF(JZ)646,646,647
647 CONTINUE
    DO 645 NZ=1,NZ
    K0ZY=K0DI(NZ)
    IARG=K0UF(N7)
    IF(IARG=KHSI)1340,1340,1341
1341 WRITE(6,1342)
1342 FORMAT(1X,30HINC11CREASE THE DIMENSION OF STAR)
1340 CONTINUE
    IF(K0S(KOZY))657,657,659
657 NAK=NAK+1
    IF(NAK-NSPTU-1)1376,1375,1375
1375 WRITE(6,1376)
1376 FORMAT(1X,40HNSPT EXCEEDS LIMIT-CREASE DIMENSIONS ,
      115HCONTAINING NSPT)
1377 CONTINUE
    SIMBON(JZ+NAK)=SEM0L(KOZY)
    SEMPO0D(JZ+NAK)=STAR(IARG)
    NA(JZ)=NA(J7)+1
    GO TO 645
659 NAT=NA+1
    IF(NAT-NSPTU-1)1381,1380,1380
1380 WRITE(6,1381)
1381 CONTINUE
    SIMBOD(JZ+NA)=SEM0L(K0ZY)
    SEMPOOD(JZ+NA)=STAR(IARG)
    NR(JZ)=NR(J7)+1
645 CONTINUE
646 CONTINUE

C  PROGRAM MAIN--9
C  SEPARATE POLY INTO ARRAYS FOR THE
C  NUMERATOR AND DENOMINATOR OF THE
C  TRANSFER FUNCTION
THE CONSTANT COEFFICIENTS IN THE TRANSFER FUNCTION ARE SEPARATED INTO ARRAYS FOR THE NUMERATOR AND DENOMINATOR.

```
KI=KIK-1
DO 755 JA=1, KIK
JIR=0
JD=0
DO 755 JC=1, NAHU
IF (ITOP(JC)) 753, 753, 751
751 JIR=JIR+1
POLYU(JA, JIR)=POLY(JA, JC)
GO TO 755
753 JD=JD+1
POLY(JA, JD)=POLY(JA, JC)
755 CONTINUE
CALL SECONDIT6)
TDECOD=F6-T6
WRITE (6,1605) TDECOD
1605 FORMAT (1X*36HTIME FOR DECODING SYMBOLS IN SECONDS,F15.3/)```

```
PROGRAM MAIN--10
MAKE POWERS OF S IN OUTPUT TRANSFER FUNCTION POSITIVE

.MAXIM=0
KAR=KIK-1
DO 522 KAR=1, KAHU
IF (MSOR(KAR)) 521, 522, 522
521 MAXIM=MAXIM+MSOR(KAR)
523 MAXIM=MAXIM+MSOR(KAH)
522 CONTINUE
DO 524 KIT=1, KAHU
524 MSOR(KIT)=MAXIM+MSOR(KIT)
```

```
PROGRAM MAIN--11
PRINT OUT NUMERATOR OF THE TRANSFER FUNCTION

LUK=0
IKU=ILU-1
WRITE (6,931)
WRITE (6,936)
WRITE (6,925)
920 FORMAT (25*20HNUMERATOR POLYNOMIAL///) WRITE (6,921)
921 FORMAT (1X,4HCOLUMN,12X,23HSYMBOL FOR GIVEN COLUMN)
DO 905 IK=1, IKU
IF (ITOP(IK)) 905, 905, 901
901 ILU=NA(IK)
IF (ILU) 710, 710, 711
710 ILU=1
711 JLU=NB(IK)
IF (JLU) 712, 712, 713
712 JLU=1
713 CONTINUE
LUK=LUK+1
WRITE (6,902) LUK, (SIMBON(IK, IL), SEMPON(IK, IL),
112

11L=1,1LU),DASH,(SIMBON(IK,JL),SEMPON(IK,JL),JL=1,JLU)

903 FORMAT(1X,15x,20X,30A3)
905 CONTINUE
   WRITE (6,930)
930 FORMAT(//)
   WRITE(6,1H21)
1A21 FORMAT(1X,7H POWER)
   WRITE (6,925)
422 FORMAT (1X,8H OF S,17X,33H CONSTANT COEFS. IN THE POLYNOMIAL)
   LML=1
   LMU=7
   IF(JIH=LMU)820,818,B18
820 LMU=JIH
818 WRITE (6,806) (LO,LD=MLM,LMU)
806 FORMAT(2X,7(8X,8H)
   KROW=IKK=1
   DO 808 KROW=1,KNOWU
   WRITE (6,812) MSORT(KROW),(POLYU(KROW,LM),LM=LMU,LMU)
   812 CONTINUE
   IF(JIH=LMU)814,814,812
812 LML=LMU+7
   LMU=LMU+7
   IF(JIH=LMU)816,816,818
816 LML=LMU
   GO TO 818
A14 CONTINUE

C PROGRAM MAIN--12
C PRINT OUT DENOMINATOR OF
C THE TRANSFER FUNCTION

LUK=0
   IKU=ILU-1
   WRITE (6,933)
931 FORMAT(50H**************5H**************5H**************5H**************5H)
   WRITE (6,933)
   WRITE (6,923)
923 FORMAT (25X,22H DENOMINATOR POLYNOMIAL//)
   WRITE (6,924)
924 FORMAT (1X,8M,LCOLUMN,12X,23HSYMBL FOR GIVEN COLUMN)
   DO 705 IK=1,IKU
   IF(ITOP(IK))701,701,705
701 ILU=NA(IK)
   LUK=LUK+1
   IF(IJU)915,915,916
915 ILU=1
916 JLU=NB(IK)
   IF(JLU)917,917,918
917 JLU=1
918 CONTINUE
   WRITE(6,703) LUK,(SIMBON(IK,IL),SEMPON(IK,IL),
   1IL=1,ILU),DASH,(SIMBON(IK,JL),SEMPON(IK,JL),JL=1,JLU)
703 FORMAT(1X,15x,20X,30A3)
705 CONTINUE
   WRITE (6,933)
   WRITE (6,1H22)
1822 FORMAT(1X,7H POWER)
   WRITE (6,925)
925 FORMAT(1x,8H OF S, 17X, 33H CONSTANT COEFS. IN THE POLYNOMIAL)
    LML=1
    LML=7
    IF (JD=LMU) $20, 518, 518
520 LMU=JD
51a WRITE (6,50a) (LO, LD=LML, LMU)
506 FORMAT(2X,7(8X,6HCOLUMN,12))
    KROWU=1
    DO 508 KROW=1, KROWU
    WRITE (6,51a) MSURT(KROW) (POLY(KROW, LM) = LML=LMU)
510 FORMAT(15,3H, 7(E12.5, 4HCOLUMN, I21)
    IF (JO-LMU) 520, 518, 512
512 LML=LML-1
    LMU=LMU+7
    IF (JO-LMU) 514, 516, 518
516 LMU=JD
    GO TO 518
514 CONTINUE
    WRITE (6,930)
    CALL SECOND (TEND)
    TEEXEC=TEND-START
    WRITE (6,1101) TEEXEC
1161 FORMAT (1X,6TH EXECUTION TIME IN SECONDS, F15.3//
    128H AUGUST 1970 VERSION OF SNAP)
250 GO TO 1111
END
SUBROUTINE SGF(NFIRST, NLAST, IXPON, NET, SYMUL, KONS, MTX, NEST,
    LIST, NMOA, NOT, NNOA, NMA, NNR)
C**************************************************************
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NAH (DEFINED IN PROGRAM MAIN-1)
DIMENSION JDOW(35), NP(35), IVV(35), NUML(35), CV(35), INTPEE(35)
DIMENSION LCAX(35)
DIMENSION NP(35, 35), LB(35, 35), MSYM(35)
DIMENSION IYX(35), JM(35), LB(35), MSYM(35)
DIMENSION IMULX(35), CAL(35), SYM(35)
DIMENSION IYX(35), VM(35), NUML(35), INTPEE(35), NOTRFF(35)
DIMENSION TYPX(35), NIMX(35), JNX(35), LNX(35), SYM(35)
C**************************************************************
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTIC NGB
DIMENSION CYMX(100)
DIMENSION NFM1HS(100), NLAST(100), IXPON(100), WEIRT(100)
DIMENSION MAPX(100), KONS(100), NEST(100), TYPE(100)
DIMENSION SYMX(100), MIX(100)
C**************************************************************
COMMON NF, NS, IB
COMMON /C2/ NNG, NGB
C
SUBPROGRAM #A#
DATA YG, CF, Y, Y, Z/WHL, 242, 229, 228, 229 /
DATA EX, YA, CC, CV, VA, VC/242, 229, 228, 229 /
DATA FH/3H FB/
DATA ONE/3H 1/
DO 710 IC=1, NNG
DO 710 I=1, NNG
NS( IC, IK ) = 0
710 NF( IC, IK ) = 0
LINK=0
DO 152 IG=1,NRG
NEST(IG)=0
152 KONSO(IG)=0
WRITE (6,266)
260 FORMAT (/ /)
WRITE (6,717)
717 FORMAT (30X'THNETWORK/)
NLAST(I)=NIN
IXPON(I)=0
WEIT(I)=-1.
SYMUL(I)=F
KONSO(I)=0
NEST(I)=1
MO=0
LO=0
LIST=1
KLU=0
DO 5 II=1,NRG
INTREE(II)=0
5 JROW(II)=0
DO 528 II=1,NRL
HEAN(S+9) TYPX(II),NUMX(II),JAX(II),LAX(II),SYM(II),
11QUALX(II),V'2LX(II),NUMLX(II)
IF(TYPX(II),EQ.CC) GO TO 1300
IF(TYPX(II),EQ.CV) GO TO 1300
IF(TYPX(II),EQ.VV) GO TO 1300
IF(TYPX(II),EQ.VC) GO TO 1300
GO TO 1301
1300 IF(NUMLX(II),GT.1301) WRITE(6,1303)
1302 WRITE(6,1303)
1303 FORMAT (1X,'***ERROR***CONTROL SPECIFICATION FOR DEP. SOURCE , 1THMISSING)
NOR=0
GO TO 1305
1301 CONTINUE
528 CONTINUE
9 FORMAT (A2,13,215,1X,A3,12,5,13)
WRITE (6,262)
261 FORMAT (2X,2PELEMET ELEMENT INITIAL TERMINAL , 2RH ELEMENT ELEMENT NO.)
WRITE (6,262)
262 FORMAT (2X,3IM TYPE NUMBER NUDE NUDE SYMBOL , 12H VALUE OF CONTROL)
DO 601 M=1,NUM
601 WRITE (6*,600) TYPX(M),NUMX(M),JBX(M),LBX(M),SYMX(M),
11QUALX(M),VALX(M),NUMLX(M)
600 FORMAT (4X,A2,6X,12,6X,12,6X,12,6X,A3,12,5,2X,12)
CALL FTHREE (TYPX,JBX,LBX,INTRE,NOTREF,NUD,NOR)
C
C SUBPROGRAM #8#
WRITE (6,51a)
518 FORMAT (30X,'13THREE SELECTED)
NUMU=NUD=1
DO 21 NU=1,NUMU
10=INTRE(NU)
NUMC=NUMX(10)
TYPB(NUMC)=TYPX(10)
JRX(NUMC)=JBX(10)
LB(NUMC)=LBX(10)
SYM(NUMC)=SYMX(10)
IQUAL(NUMC)=IQUALX(IO)
VAL(NUMC)=VALX(IO)
NUML(NUMC)=NUMLX(IO)
INTREE(NUMC)=1
WRITE (6,517) TYPB(NUMC),NUMC,JH(NUMC),LB(NUMC),SYM(NUMC),
IQUAL(NUMC),VAL(NUMC),NUML(NUMC)
517 FORMAT (4X,A2,6X,12,4X,12,F12.5,2X,I2)
KLU=KLU+1
LINC(NUMC)=0
IF (TYPB(NUMC) .NE. VV) GO TO 3
IM=IO+1
IVV(IM)=NUMC
3 IF (TYPB(NUMC) .NE. CV) GO TO 4
IL=IL+1
ICV(IL)=NUMC
4 JF=JH(NUMC)
LF=LB(NUMC)
JROW(JF)=JROW(JF)+1
JCOL(JF)=JCOL(JF)+1
NF(JF,JROW(JF))=LF
NS(JF,JROW(JF))=-1
21 CONTINUE
WRITE(6,260)
WRITE(6,715)
715 FORMAT (30X,HSF6/)
00 IF (KLU=NUMC) .LT. 32,360,532
532 LINK=LINK+1
534 IF (NOTREE(LINK)) .LT. 534,534+532
NUMC=NUMX(LINK)
TYPE(NUMC)=TYPX(LINK)
JK=JHX(LINK)
LK=LHX(LINK)
SYM(NUMC)=SYMX(LINK)
IQUAL(NUMC)=IQUALX(LINK)
VAL(NUMC)=VALX(LINK)
NUML(NUMC)=NUMLX(LINK)
TYPE2(TYPE(NUMC))=TYPE(NUMC)
CVAL(CVAL(NUMC))=CVALX(LINK)
KLU=KLU+1
LINC(NUMC)=1
KDEP=0
KANS=0
IF (TYPE(NUMC) .EQ. CL) GO TO 117
IF (TYPE(NUMC) .EQ. 0) GO TO 119
SUBPROGRAM #C#
THIS PROGRAM GENERATES SIGNAL FLOW GRAPH INFO.
FROM BRANCH NODE TO LINK NODE
NBY=NOB
CONTINUE
WRITE(6,260)
WRITE(6,715)
715 FORMAT (30X,HSF6/)
00 CONTINUE
IF (TYPE(NUMC) .EQ. Y) GO TO 119
IF (TYPE(NUMC) .EQ. E) GO TO 700
IF (TYPE(NUMC) .EQ. Z) GO TO 700
IF (TYPE(NUMC) .EQ. C) GO TO 121
KDEPS = 1
IF (TYPE(NUMC) .EQ. E) GO TO 123
IF (TYPE(NUMC) .EQ. C) GO TO 123
IF (TYPE(NUMC) .EQ. V) GO TO 165
IF (TYPE(NUMC) .EQ. C) GO TO 265
117 IXPS = -1
   KANSO = 1
   GO TO 123
119 IXPS = 0
   GO TO 123
700 IXPS = 0
   KANSO = 1
   GO TO 123
121 IXPS = 1
123 CALL THEP (JL, P, N, NPL)
IFIN = NUMC
149 LON = LON + 1
   NPI = NP (LON)
   NP2 = NP (LON + 1)
107 INIT = TH (NPI, NP2)
109 SIGN = NS (NPI, NP2)
IF (EPS) = 167 + 167 + 169
167 IF (IQUAL (NUMC) .EQ. I) GO TO 111
NES = 1
   CONST = SIGN
   GO TO 125
111 CONST = SIGN .EQ. VALU
125 LIST = LIST + 1
   IF (NES) = 502 + 503 + 504
502 NFST (LIST) = 1
503 KONS0 (LIST) = KANSO
   NFST (LIST) = INIT
   NLAST (LIST) = IFIN
   SYMBOL (LIST) = SYM (IFIN)
   IXPON (LIST) = IXPS
   IF (KONS0 (LIST)) = 505 + 505 + 504
504 WEXT (LIST) = 1 .CONST
   GO TO 506
505 WEXT (LIST) = CONST
506 MAPY (NUMC) = 'LIST
127 FORMAT (315, E12.5)
129 FORMAT (A4)
C
C SUBPROGRAM #1
C THIS PROGRAM GENERATES SIGNAL FLOW GRAPH INFO.
C FROM LINK NODE TO BRANCH NODE
169 CONTINUE
IF (TPHBI (INIT) .EQ. E) GO TO 201
IF (TPHBI (INIT) .EQ. C) GO TO 201
IF (TPHBI (INIT) .EQ. V) GO TO 201
IF (TPHBI (INIT) .EQ. C) GO TO 201
LIST = LIST + 1
IF (TPHBI (INIT) .EQ. R) GO TO 133
IF (TPHBI (INIT) .EQ. Z) GO TO 133
IF (TPHBI (INIT) .EQ. G) GO TO 702
IF (TPHBI (INIT) .EQ. Y) GO TO 702
IF (TPHBI (INIT) .EQ. CL) GO TO 135
IF(TYPH(INIT),EQ.,C) GO TO 137
133 IXPON(LIST)=0
GO TO 141
702 IXPON(LIST)=0
KONSO(LIST)=1
GO TO 141
135 IXPON(LIST)=1
GO TO 141
137 IXPON(LIST)=-1
KONSO(LIST)=1
141 IF(IQUAL(INIT),EQ.,10) GO TO 139
NEST(LIST)=1
WEIGHT(LIST)=1.*SIGN
GO TO 147
139 IF(KONSO(LIST)) 608, 608, 607
607 WEIGHT(LIST)=-SIGN/VAL(INIT)
GO TO 147
608 WEIGHT(LIST)=-SIGN*VAL(INIT)
147 NFIRST(LIST)=IFIN
NLAST(LIST)=INIT
SYMBUL(LIST)=SYM(INIT)
201 NPLA=NPL-1=4UN
IF(NPLA) 151, 151, 149
C
C SUBPROGRAM E
C THIS PROGRAM SETS UP SFG INFO. FOR VC
C TYPE CONIKUL SOURCES
165 NUMO=NUMH
IF(INTHEE(NUMH)) 163, 163, 161
163 LIST=LIST+1
NFIRST(LIST)=NUMH
NOHY=NOHY+1
NLAST(LIST)=NOHY
SYMBUL(LIST)=SYM(NUMH)
NUMO=NOHY
IF(TYPE(NUMH),EQ.,Y) GO TO 912
IF(TYPE(NUMH),EQ.,G) GO TO 912
IF(TYPE(NUMH),EQ.,C) GO TO 914
IF(TYPE(NUMH),EQ.,CL) GO TO 916
NUMO=0
IXPON(LIST)=0
GO TO 914
912 IXPON(LIST)=0
KUNO=1
GO TO 918
914 IXPON(LIST)=-1
KUNO=1
GO TO 918
916 IXPON(LIST)=1
KUNO=0
918 IF(IQUAL(NUMH),EQ.,10) GO TO 920
NEST(LIST)=1
WEIGHT(LIST)=1.
GO TO 209
920 IF(KUNO) 922, 922, 924
922 WEIGHT(LIST)=CVAL(NUMH)
GO TO 209
924 WEIGHT(LIST)=1./CVAL(NUMH)
209 KONSO(LIST)=1
161 LIST=LIST+1
NFIRST(LIST)=NUMO
NLAST (LIST) = NUMC
SYMBUL (LIST) = SYM (NUMC)
IXPON (LIST) = 0
IF (TQUAL (NUMC) .EQ. 10) GO TO 171
NEST (LIST) = 1
WEIGT (LIST) = 1.
GO TO 203
171 WEIGT (LIST) = CVALU
203 CONTINUE
GO TO 123

C
C SUBPROGRAM #F#
C THIS PROGRAM SETS UP SFG INFO. FOR CC
C TYPE CONTROL SOURCES

265 NUMO = NUMH
IF (INTKEE (NUMH)) 621, 621, 620
620 LIST = LIST + 1
NFIRST (LIST) = NUMH
NOBY = NOBY + 1
NLAST (LIST) = NOBY
SYMBUL (LIST) = SYM (NUMH)
MUNO = NOBY
IF (TYPH (NUMH) .EQ. Z) GO TO 233
IF (TYPH (NUMH) .EQ. R) GO TO 233
IF (TYPH (NUMH) .EQ. CL) GO TO 235
IF (TYPH (NUMH) .EQ. C) GO TO 237
KUNO = 0
IXPON (LIST) = 0
GO TO 241
233 IXPON (LIST) = 0
KUNO = 1
GO TO 241
235 IXPON (LIST) = -1
KUNO = 1
GO TO 241
237 IXPON (LIST) = 1
KUNO = 0
241 IF (TQUAL (NUMH) .EQ. 10) GO TO 239
NEST (LIST) = 1
WEIGT (LIST) = 1.
GO TO 247
239 IF (KUNO) 900, 900, 902
900 WEIGT (LIST) = VAL (NUMH)
GO TO 247
902 WEIGT (LIST) = VAL (NUMH)
247 KUNO (LIST) = 1.
621 LIST = LIST + 1
NFIRST (LIST) = NUMO
NLAST (LIST) = NUMC
SYMBUL (LIST) = SYM (NUMC)
IXPON (LIST) = 0
IF (TQUAL (NUMC) .EQ. 10) GO TO 271
NEST (LIST) = 1
WEIGT (LIST) = 1.
GO TO 281
271 WEIGT (LIST) = CVALU
GO TO 281
281 CONTINUE
GO TO 123

C
C SUBPROGRAM #G#
NOBY=NOBY+1
NLAST(LIST)=NOBY
SYMBOL(LIST)=SYM(NUNO)
IF(TYPH(NUNO).EQ.2) GO TO 433
IF(TYPH(NUNO).EQ.1) GO TO 433
IF(TYPH(NUNO).EQ.3) GO TO 435
IF(TYPH(NUNO).EQ.4) GO TO 437
KUNO=0
IXPON(LIST)=0
GO TO 441
433 IXPON(LIST)=0
KUNO=1
GO TO 441
435 IXPON(LIST)=1
KUNO=1
GO TO 441
437 IXPON(LIST)=-1
KUNO=0
441 IF(IQUAL(NUNO).EQ.10) GO TO 439
NEST(LIST)=1
WEIGT(LIST)=1.
GO TO 448
439 IF(KUNO):908,908+910
908 WEIGT(LIST)=VAL(NUNO)
GO TO 448
910 WEIGT(LIST)=VAL(NUNO)
448 KONSO(LIST)=1
447 CONTINUE
NUNO=NOBY
461 LIST=LIST+1
NFIRST(LIST)=NUNO
NLAST(LIST)=LI
SYMBOL(LIST)=SYM(LI)
IXPON(LIST)=0
IF(IQUAL(LI).EQ.10) GO TO 471
NEST(LIST)=1
WEIGT(LIST)=1.
GO TO 403
471 WEIGT(LIST)=VAL(LI)
403 CONTINUE
405 CONTINUE
C
C SUBPROGRAM #I#
C GENERATING OUTPUT NODE OF SFG
C
515 CONTINUE
IF(NOUT)514,512,514
512 CALL THEP(NOUT,NUNO,NODB,NP,NPI)
NOUT=NOBY+1
MOPU=NPI+1
DO 510 MOP=1,MOPU
N1=NPI(MOP)
N2=NPI(MOP+1)
LIST=LIST+1
NFIRST(LIST)=IH(N1,N2)
NLAST(LIST)=NOUT
SYMBOL(LIST)=NUNO
IXPON(LIST)=0
KONSO(LIST)=1
NEST(LIST)=0
510 WEIGT(LIST)=NS(N1,N2)
C THIS PROGRAM SETS UP SFG INFO. FOR CV
C TYPE CONTROL SOURCES
360 IF (MO) 460, 464
364 DO 305 MI = 1, MO
KI = IVV(MI)
NUNO = NUML(KI)
IF (LINC(NUNO)) 361, 361, 363
363 LIST = LIST + 1
NFIRST(LIST) = NUML(KI)
NUNY = NOBY + 1
NLAST(LIST) = NOBY
SYMBUL(LIST) = SYM(NUNO)
IF (TYPE(NUNO) .EQ. Y) GO TO 333
IF (TYPE(NUNO) .EQ. G) GO TO 333
IF (TYPE(NUNO) .EQ. C) GO TO 335
IF (TYPE(NUNO) .EQ. CL) GO TO 337
KUNO = 0
IXPON(LIST) = 0
GO TO 341
333 IXPON(LIST) = 0
KUNO = 1
GO TO 341
335 IXPON(LIST) = -1
KUNO = 0
GO TO 341
337 IXPON(LIST) = 1
KUNO = 0
341 IF (EQUAL(NUNO) .EQ. 10) GO TO 339
NEST(LIST) = 1
WEIGHT(LIST) = 1.
GO TO 348
339 IF (KUNO) 904, 904, 906
904 WEIGHT(LIST) = CVAL(NUNO)
GO TO 348
906 WEIGHT(LIST) = 1. / CVAL(NUNO)
348 CONTINUE
NUNO = NOBY
361 LIST = LIST + 1
NFIRST(LIST) = NUNO
NLAST(LIST) = KI
SYMBUL(LIST) = SYM(KI)
IXPON(LIST) = 0
IF (EQUAL(KI) .EQ. 10) GO TO 371
NEST(LIST) = 1
WEIGHT(LIST) = 1.
GO TO 303
371 WEIGHT(LIST) = VAL(KI)
303 CONTINUE
305 CONTINUE
C SUBPROGRAM #H#
C THIS PROGRAM SETS UP SFG INFO. FOR CV
C TYPE CONTROL SOURCES
460 IF (LO) 151, 151, 464
464 DO 405 MI = 1, LO
LI = ICV(MI)
NUNO = NUML(LI)
IF (LINC(NUNO)) 463, 463, 461
463 LIST = LIST + 1
NFIRST(LIST) = NUML(LI)
C THE FOLLOWING AKHAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS NNH* AND NSPT (DEFINED IN PROGRAM MAIN-1)
DIMENSION TYPS(35),JBX(35),LMX(35),INTRE(35),NOTPRT(35)
DIMENSION NTY(35),NF(35),KCOL(20)

COMMON/C2/NNG,NBG
DATA E9VV,CV/HF- ,2HV,2HV(2HCV/
DATA R,CL,C,L,22HHR,2HC,2HY,2HL/
DATA G/2HG/

C SUBROUTINE FJREJ(TYPX, JHA, JBX, INTRE, NOTRE, NUD, NOH)
C THIS PROGRAM OEHDS SFG INFORMATION
FOR INPUT TO MAIN PROGRAM
DO 87 J=1, NNG
87 MIX(J)=J
KONU=LIST-1
DO 90 KON=1, KONU
IU=KON+1
IL=KON
GO TO 83
81 MXL=MIX(IL)
MIX(IL)=MIX(IU)
MIX(IU)=MXL
IL=IL-1
IU=IU-1
IF(IL)=80, 80, 83
83 MIU=MIX(IU)
MIL=MIX(IL)
IF(NFIRST(MIU)-NFIRST(MIL))=81, 85, 80
84 MXL=MIX(IL)
MIX(IL)=MIX(IU)
MIX(IU)=MXL
IL=IL-1
IU=IU-1
IF(IL)=80, 80, 82
82 MIU=MIX(IU)
MIL=MIX(IL)
IF(NFIRST(MIU)-NFIRST(MIL))=80, 80, 84
89 IF(NLAST(MIU)-NLAST(MIL))=80, 80, 84
80 CONTINUE
1305 CONTINUE
RETURN
END

C DIMENSION TYPS(35),JBX(35),LMX(35),INTRE(35),NOTPRT(35)
C COMMON/C2/NNG,NBG
DATA E,VV,CV,2HE,2HV,2HC,2CV/
DATA R,CL,C,L,22HHR,2HC,2HY,2HL/
DATA G/2HG/
THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NFIWORK

C CHARACTERISTIC NF(NDEFINED IN PROGRAM MAIN=1)
DIMENSION JX(35),NP(35),JMEM(35),KMEM(35)
DIMENSION NF(35,35)

C*****************************************************************************
COMMON/C2/NNG,NFG
DO 80 IS=1,NNG
JX(IS)=0
NP(IS)=0
JMEM(IS)=0
KMEM(IS)=0
80 CONTINUE
RETURN
END

SUBROUTINE TREP(NINX,NOUTX,NF,NP,NPL)
C***************************************************************
NPL=0
JX(1)=NIN
JX(2)=NIN
I=1
J=NIN
NP(I)=NIN
20 K=0
25 K=K+1
  IF (NF(J,K) = OUT) 30, 50, 30
30 IF (NF(J,K)) 34, 32, 34
32 IF (J=NIN) 60, 100, 60

C
C FLOWER CHECK
34 NJK=NF(J,K)
  IF (NJ=NJX(I)) 45, 25, 45

C
C STORE AND REMEMBER VERTEX
45 I=I+1
NP(I)=NF(J,K)
JMEM(I)=J
I=I+1
JX(I)=NF(J,K)
42 J=NF(J,K)
KMFM(I)=K
GO TO 20

C
C BACKSTEP
60 J=JMF(I)
K=KMEM(I)
I=I-1
GO TO 25

C
C FINAL PATH VERTEX AND PATH LENGTH
50 II=I+1
NP(II)=NOUT
60 NPL=II
100 CONTINUE
RETURN
END

SUBROUTINE AHAY(JSIG,XCON,XPO,JKON,POLY,LIL,KIK)

C----------------------------------------------------------------------
C THE FOLLOWING AHAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS NTO, AND NEXPS (DEFINED IN PROGRAM MAIN-1)
C DIMENSION MSORT(15), KSORT(150), POLY(15,150)
C----------------------------------------------------------------------

COMMON/C1/MSORT, KSORT
COMMON/C3/NEAPS, NTO
MMX=0
WX=0
IF(KIK=1) 3, 22, 3
3 MMU=KIK-1
00 2 MM=1, MMU
MMX=MMX+1
IF(JXPO=MSORT(MM)) 2, 10, 2
2 CONTINUE
22 MSORT(KIK)=JXPO
MMX=KIK
KIK=KIK+1
IF(KIK-NEXPS-1) 1386, 1385, 1385
1385 WRITE(6,1387)
1387 FORMAT(1X,42MS-POWER EXCEEDS LIMIT-INCREASE DIMENSIONS, 
116MCONTAINING NEAPS)
1386 CONTINUE
10 IF(LIL=1)11,24+11
11 NNU=LIL=1 
   DO 12 NN=1,NNU 
   NNX=NNX+1 
   IF(JKOU=KSORT(NN))12,20+12 
12 CONTINUE
24 KSORT(LIL)=JKOU 
   NNX=LIL 
   LIL=LIL+1 
1365 WRITE(6,1366)
1366 FORMAT(1X,4AHNO. OF TERMS IN OUTPUT EXCEEDS LIMIT-INCREASE, 
125HDIMENSIONS CONTAINING NT0)
1367 CONTINUE
20 POLY(MMX,NNX)=POLY(MMX,NNX)*XCON*(N-1)**JSIG 
RETURN
END
SUBROUTINE DECIDE(KON,KOUY,IZ,FB,JZ,SEMBUL,KODF,KODI,ITOP,KRASIS)
C**********************************************************
C THE FOLLOWING ARRAYS ARE ASSOCIATED WITH THE NETWORK
C CHARACTERISTICS NSPI; AND NT0 (DEFINED IN PHORAM MAIN=1)
C**********************************************************
COMMON/C4/NSPT
IZ=0
M=KRASIS=1
DO 3 J=1,KOU 
1 IPower=M*ANU*KOU 
2 IF(IPower)=3,3,2 
3 IF(SEMBUL(J)=EQ,FB)GO TO 10 4
IZ=IZ+1 
1370 WRITE(6,1371)1370,1370
1371 FORMAT(1X,4AHNO. OF SYMBOLS PER TERM EXCEEDS OUTPUT-INCREASE, 
126HDIMENSIONS CONTAINING NSPT)
1371 CONTINUE
KODF(IZ)=IPower 
   KODI(IZ)=J 
   GO TO 3 
4 ITOP(JZ)=1 
3 KOUY=KOUY/KRASIS 
RETURN
END
REFERENCES


