



A
TOPOLOGICAL APPROACH
TO
LINEAR NETWORK ANALYSIS

Thesis Submitted in Fulfilment of the Requirements
for the Degree of
Doctor of Philosophy

by

R. A. Brownell, B.Sc., B.E.
(Robert Alan)

Department of Electrical Engineering
University of Tasmania
Hobart

July, 1973
Conferred 1974

Statement of Originality

This thesis contains no material which has been submitted for a degree or diploma at any university.

To the best of the candidate's knowledge and belief, this thesis contains no copy or paraphrase of material previously published or written by another person.

R.A. Brownell

.....

(R.A. Brownell)

July, 1973.

Acknowledgements

The author is indebted for advice and encouragement to the staff of the Electrical Engineering Department and, in particular, to his supervisor, Mr. John H. Brodie.

The author is also appreciative of the interest taken in his work by his fellow students, including those from other disciplines, and of the criticism of the computer programs and analysis techniques volunteered by the numerous users within and without the University; all have influenced the shape of this work.

Grateful acknowledgement is made to General Motors-Holden Limited for their material support with a Post-Graduate Research Fellowship for the period from March, 1965 to March, 1968. The author was also supported with a University Research Scholarship from April to September, 1968.

His present employer, Amalgamated Wireless (Australasia) Limited, assisted with the reproduction of the thesis.

CONTENTS

Statement of Originality	
Acknowledgements	
Contents	
Figures	
Tables	
Prologue - General Introduction and Summary	

I THEORY

Page

1	TOPOLOGICAL ANALYSIS OF STRUCTURES OF MULTI-PORT NETWORKS	
1.1	Introduction	1.1
1.2	Network Characterisation	1.2
	Network Polynomials	1.2
	Structures of Multiport Networks	1.6
1.3	The Analysis Process	1.8
	Calculation of Natural Polynomials	1.8
	Calculation of High-Order Polynomials	1.12
	The Analysis Algorithm	1.13
1.4	Application of the Analysis Process	1.14
	General Structures	1.14
	Structures of 2-port Networks	1.15
1.5	Conclusion	1.16
2	POLYNOMIALS AND NETWORK BEHAVIOUR	
2.1	Introduction	2.1
2.2	Identical Networks in Parallel	2.1
2.3	Polynomials of Equivalent Networks	2.7
2.4	Polynomials and the Short-Circuit Admittance Matrix	2.12
2.5	The Unit Gyrator	2.13
2.6	Topological Formulae	2.15
2.7	Polynomial Identities	2.17
2.8	Conclusion	2.18
3	ALTERNATIVE ANALYSIS METHODS	
3.1	Introduction	3.1
3.2	Inversion of Network Matrices	3.2
3.3	Addition of Network Matrices	3.4
3.4	Conclusion	3.7

CONTENTSII PRACTICE

Page

4	TOPOLOGICAL ANALYSIS OF 2-PORT NETWORKS	
4.1	Introduction	4.1
4.2	Analysis Methods	4.2
4.3	Calculation of Network Functions	4.3
4.4	Network Tearing	4.6
4.5	Structure Graphs	4.8
4.6	Algebraic Reduction	4.10
4.7	Polynomials of 2-port Networks	4.12
4.8	Analysis of 2-port Network Structures	4.13
4.9	Implementing the Analysis Method	4.16
4.10	Numerical Accuracy	4.19
4.11	Polynomial Representation	4.20
4.12	Conclusion	4.21
5	ALGORITHMS FOR COMPUTER PROGRAMS	
5.1	Introduction	5.1
5.2	Programming Language	5.2
5.3	Data Structure	5.5
5.4	Utility Routines	5.7
5.5	Algebraic Reduction	5.9
	Syntax of Network Expressions	5.10
	Semantics of Network Expressions	5.13
	Evaluation of Network Expressions	5.14
5.6	Topological Analysis	5.17
	Interface with the Algorithm	5.17
	Action of the Algorithm	5.19
	Application of the Algorithm	5.25
5.7	Conclusion	5.28
6	DEMONSTRATION OF COMPUTER PROGRAMS	
6.1	Introduction	6.1
6.2	Large Passive Filter	6.3
6.3	Three-Stage IC Amplifier	6.6
6.4	General Converter	6.9
6.5	Position Control System	6.11
6.6	Conclusion	6.13

References

Epilogue - General Conclusions

Appendix - Reprint of paper

"Growing the Trees of a Graph"

FIGURES

	Page
1.1 Polynomials of a network A, and their representation	1.4
1.2 Equivalent interconnection of ports	1.7
1.3 A structure of three constituent networks	1.9
2.1 Equivalent star and mesh networks A and B	2.7
2.2 Cascade connection of network A with a unit gyrator at port C	2.14
4.1 Circuit diagram of two-stage transistor amplifier	4.8
4.2 Network diagram highlighting small-signal behaviour	4.8
4.3 Complete network as a structure of 2-port networks	4.8
4.4 Sign convention of voltages and currents of a 2-port network	4.8
4.5 Structure graph	4.9
4.6 Complete network as an algebraically reduced structure	4.11
4.7 Algebraically reduced structure graph	4.11
4.8 Nine pointer settings for a paralleled pair of 2-port networks A and B.	4.15
5.1 Typical arrangement of polynomials in the equivalent two- dimensional array. Arithmetic expressions define the subscripts of corresponding locations in the actual array	5.6
5.2 Algorithm for evaluation of network expressions	5.14
5.3 Algorithm for topological analysis	5.18
5.4 An equivalent 2-port network representing the general branch of a network graph	5.25
5.5 Signal-flow graph	5.27
5.6 Electrical analogue of signal-flow graph	5.27
5.7 Structure graph corresponding to signal-flow graph	5.27
6.1 Large passive filter and its subnetworks	6.3
6.2 Three-stage IC amplifier: (a) circuit diagram; (b) transistor models; (c) structure of 2-port networks; (d) structure graph	6.6
6.3 General converter : (a) prototype circuit; (b) structure of 2-port networks; (d) structure graph	6.9
6.4 (a) Equivalent circuit for transistor; (b) Transformation from common base to common emitter; (c) Transformation from common base to common collector	6.9
6.5 Lineprinter output for analysis of general converter	6.11
6.6 Position control system	6.11
6.7 Position control system: (a) system equations; (b) signal- flow graph	6.11
6.8 Structure graph corresponding to signal-flow graph	6.12
6.9 (a) Equivalent network of position control system (b) Structure graph of equivalent network	6.13

TABLES

	Page
1.1 The number of mth-order linkage polynomials of a n-port network	1.6
4.1 Basic 2-port networks and their polynomials	4.10
4.2 Calculated frequency response of a filter complex containing 50 reactive components, illustrating the effect of a frequency transformation	4.19
6.1 Results of two analyses of the large passive filter	6.5
6.2 The sensitivities to parameters L and C of the natural frequencies of the three-stage IC amplifier with its input short- circuited.	6.8

P R O L O G U E

General Introduction and Summary

The work reported in this thesis was motivated by a desire to develop better practical methods for linear network analysis. The practical aspects of existing methods, together with the new methods arising from this work, are discussed in part II of the thesis. Part I is devoted to a theoretical foundation for the new methods.

The analysis method centres on network polynomials — their relationship with network behaviour and with each other. Until recently there has been no satisfactory formal treatment of network polynomials; they tend to be regarded as numerical conveniences arising in various analysis methods. For example, ratios of polynomials may express network transfer functions; they characterise linear dynamic systems; and their roots determine the natural frequencies of networks. In particular, when we analyse a network by inverting the nodal admittance matrix whose elements have been expressed as ratios of polynomials, the polynomials proliferate. It is from this background that most of the theory described here was developed.

In 1968, Dr. D.B. Pike, who had been working independently, submitted his Ph.D. thesis on "Linkage Polynomials" to the University of Sydney. That work, which this writer considers to be definitive in its treatment of many aspects of the subject, was motivated by problems in the realisation of multiport networks, and defines the polynomials by their occurrence as minor determinants of hybrid matrices of multiport networks. This definition relates them directly to network behaviour, and their relationships with each other are obtained from Laplace expansions of minor determinants. The most important contribution of Pike's thesis is concerned with the interconnection of two multiport networks; it enunciates the relationships between the polynomials of the complete network and the polynomials of its two constituent networks. In that work the relationships are obtained with Laplace expansions of the minor determinants of the sum of the two appropriate hybrid matrices of the constituent networks.

It is a different enunciation of these same relationships which is considered to be the most significant contribution of part I of this thesis. But in this work the subject of network polynomials is approached from an altogether different point of view. Both the point of view and the alternative statement of the main results have an important bearing on the practical implementation of the analysis methods, and it is the intended application of the theory which dictates the form of its presentation in part I.

The evolution of this approach may be traced from the analysis of networks by the solution of simultaneous linear equations. The conventional elimination techniques are satisfactorily proficient in solving equations with numerical coefficients but are quite clumsy when handling coefficients represented symbolically. In the latter case, however, application of Cramer's rule leads to a suitable expression of the solution in the form of ratios of determinants, and it is left to the numerical analyst to find suitable means for expanding the appropriate determinants.

For large determinants containing symbolic entries this task is cumbersome, and, for determinants derived from physical structures such as electrical networks, concludes with the cancellation of large numbers of terms. It is to this task that the network topologist, with a different point of view of the analysis problem, makes a significant contribution. Each term in the expansion is related to a unique set of branches of the network graph and its value is the product of the admittances of those branches. The sets of branches associated with a particular determinant constitute k -trees* of the network graph, and the analysis task is therefore one of generating, without duplication, all the k -trees of a graph. Unfortunately, this approach, even with the aid of a digital computer, is impractical for moderately-sized networks because of the prohibitively large numbers of trees associated with them.

*A k -tree of a graph is a tree of a subgraph which, although it includes all the nodes of the graph, is in k separate parts.

If large networks are to be analysed with topological methods, some form of network partitioning — otherwise known as network tearing, or diakoptics — must be employed. For reasons discussed in chapter 1 the approach taken throughout this work is to tear networks apart only at internal ports. It is then convenient to view polynomials as topological quantities (sums of branch-admittance products) of multiport networks, and it is the aim of chapter 1 to present an algorithm for combining the polynomials of constituent multiport networks to form the polynomials of any structure of those multiport networks.

This algorithm permits the analysis of networks in terms of topological quantities. It is the purpose of chapter 2 to relate the topological quantities, the polynomials, to network behaviour.

Consideration is first given to two identical networks in parallel, and comparison of the complete network polynomials obtained by the analysis process with those deduced from fundamental principles embodied in lemma 2.1 proves theorem 2.1. This major theorem relates all the linkage polynomials of an n -port network to an $n \times n$ matrix of rational polynomials, which is proved later, as theorem 2.3, to be the short-circuit admittance matrix of the network.

The second major theorem, theorem 2.2, establishes the relevance of the topological quantities by asserting their uniqueness in characterising networks. Proof of this theorem is centered on a study of a star network and its equivalent mesh network which has no internal nodes. The equations for the latter network, when generalised and reinforced by the two major theorems, also prove theorem 2.3 and lead directly to the classical topological formulae which express the various driving point and transfer functions of a network in terms of topological quantities.

Chapter 3 links the topological analysis process with methods of analysis based on matrix manipulation. It is shown that the application of only two generalised polynomial identities is sufficient to calculate the elements of all hybrid matrices from the given elements of any one hybrid matrix. They therefore provide a means of inverting any hybrid network matrix, a task which is central to many analysis methods, and

confirm the method adopted by Downs [19] for directly inverting a matrix of rational polynomials. Finally, the analysis of any multiport network structure by the process of chapter 1 is interpreted as the Laplace expansion of minor determinants of a matrix formed from the sum of appropriate hybrid matrices representing the individual constituent networks. The main theoretical development of the thesis thus concludes with an indication that the new analysis process could be derived solely from a matrix point of view, instead of from the topological point of view.

Chapter 4 opens part II of the thesis with a survey of existing methods of linear network analysis and an introduction to a new practical approach which is confined to structures composed only of 2-port networks. An example illustrates the method of representing electronic circuits and serves to introduce two important concepts to be developed later in the thesis: structure graphs and algebraic reduction. A simpler notation for the polynomials of 2-port networks is introduced, and the general topological analysis algorithm of chapter 1 is recast in a form better suited to the analysis of structure graphs. Practical aspects of the analysis methods are discussed, and particular attention is given to the problem of numerical accuracy and to schemes for representing polynomials.

The course of the practical work has been largely determined by progress with computer programs design to prove the logic of the analysis algorithms and to demonstrate their overall effectiveness as analytical tools.

Chapter 5 presents the results of this work in the form of two major algorithms: one for algebraic reduction, and the other for topological analysis of structure graphs; they are expressed in a high-level computer language and cover the essential aspects of all programs that implement the analysis method. The chapter also includes a rigorous definition of a language for describing networks and controlling the analysis process. It elaborates the concept of a network algebra and is designed to accept circuit models and network parameters in a variety of alternative forms.

Chapter 6 assesses the analysis method in a variety of situations. Program data and execution times are given for the frequency-response tabulation of a large passive filter, for a sensitivity investigation of a

multistage amplifier, and for a symbolic analysis of an impedance-converter circuit. A comparison is also made of two approaches to the analysis of a control system: it is represented for analysis both as a signal-flow graph and, more naturally, as a structure of 2-port networks.

* * *

Topological analysis of large networks is repressed by the curse of large numbers - not only large numbers of trees in a particular set, but large numbers of sets of trees. Responsible for this state of affairs are those aspects of the analysis method which make it attractive: its thoroughness, its flexibility, and its generality. Throughout the thesis, in its progress toward a tractable analysis method, disciplines and restrictions have been imposed. It is, perhaps, a signal achievement that the thesis is able to conclude with reports of practical computer programs possessing unique and powerful analytical facilities.

TOPOLOGICAL ANALYSIS OF STRUCTURES OF MULTI-PORT NETWORKS

1.1 INTRODUCTION

Broadly speaking, network analysis is a process whereby the behaviour of a network as a whole is ascertained from the known behavioural characteristics of its parts. For a network composed of linear, time-invariant, 2-terminal devices, powerful methods of analysis can be derived from a study of the topology of the network; trees and k-trees of the network graph are enumerated, products of the branch admittances are formed for each tree, and the products are summed over all trees in particular sets. If the branch admittances are represented by their Laplace transforms the resulting topological quantities have the form of polynomials in the Laplace operator(s), and it can be shown that ratios of the polynomials determine the various network functions such as transfer functions and driving-point immittances. Thus the behaviour of the network as a whole is directly related to the admittances of the individual network elements.

As powerful as these analysis methods are, they leave much to be desired. Of prime concern is the large number of trees, associated with only moderately-sized networks [25, 32], which are costly to enumerate and evaluate. Some attempts have been made to alleviate this problem with various forms of network partitioning, and thereby directly evaluating partial sums of admittance products without generating individual k-trees [21, 37]. But as yet there is no report of these methods being extended to cover active networks, or of their application in computer programs. Of secondary concern are the difficulties in handling mutual inductances, active devices such as controlled sources, degenerate devices such as ideal transformers and operational amplifiers, and other 2-port devices. Procedures have been developed to handle most of these devices [13, 14, 34, 36, 48] but at the expense of increased complexity and effort in the analysis process. The modelling of transformers in a manner which preserves the isolation between their ports is particularly cumbersome [6].

Any procedure for tearing networks would afford an opportunity to avoid the generation of large numbers of trees. But by imposing the following discipline on the manner in which networks may be torn, the difficulty in modelling the isolating character of transformers is also avoided. A network port is defined in the conventional way; that is, a pair of terminals with which is associated one voltage, measured between the terminals, and one current, which leaves the network at one terminal and re-enters through the other terminal. It is then stipulated that networks may only be interconnected at their ports. Because an appropriate method for interconnecting networks must assume that the currents in the terminals are equal and opposite, the tearing of a network is therefore valid only if the behaviour of the network is not altered by the introduction of isolating transformers at the interconnections between the subnetworks. The isolating character of transformers and mutually coupled coils is thereby taken into account automatically by the assumed nature of the interconnections.

With the above stipulation, the tearing procedure requires that a network be represented as a structure of multiport networks, and it is for this reason that attention is focused on the general multiport network and the topological quantities which characterise it.

1.2 NETWORK CHARACTERISATION

1.2.1 Network Polynomials

Network polynomials are here defined as topological quantities: each polynomial is associated with a set of trees (or k-trees) and is equal to the sum, over all trees in the set, of the products, over all branches in a tree, of the branch admittances. This quantity is referred to as a branch-admittance-product-sum (BAPS). Although every polynomial is the BAPS of some set of trees, not every BAPS which occurs in this study is necessarily a proper network polynomial.

To define the sets of trees it is assumed that every network can be represented by some equivalent network containing only unistors, resistors, and gyrators, following the method of Mason [34]. However, it will not be necessary to construct such equivalent networks or to be concerned with any practical difficulties, such as the need for limiting processes on

the values of some branch admittances, that might be entailed by this process.

Because ideal isolating transformers may be inserted at the ports of a network without affecting its behaviour, it is further assumed, in order to simplify the definition of the sets of trees, that one terminal of every port is connected to some common ground terminal. The ungrounded terminal of a port is referred to simply as the port terminal of that port.

The many polynomials of a network are related to various sets of k-trees of the graph of the network. A k-tree is generally understood to be a tree of a subgraph which is derived from the original graph by removing branches in such a way that the subgraph has k separate parts. An alternative view of the necessary modification to the original network employs the concept of a collapsed port: it is regarded as being short-circuited, with both terminals tied together to form a single terminal. A k-tree is then a tree of the network with k collapsed ports.

It is important for the development of this thesis, however, that a collapsed port be interpreted in a slightly different way. A collapsed port is here regarded as a port for which the path from the port terminal to ground lies not in the network itself but in some external network. This external path must be included in the trees of the network, but, when calculating a BAPS, its branch admittances are ignored.

Definition 1.1 ("natural polynomials")

The set of natural polynomials of the general network N is now introduced. The general member of the set is denoted by

$$N_{pqr..}^{abc..}$$

It is defined as the BAPS of the set of trees of N with the ports p, q, r,.. collapsed and the remaining ports a, b, c,.. unaltered (open-circuit).

Definition 1.2 ("transfer polynomials")

Natural polynomials are only particular (zero-order) instances of the set of multiple-order transfer polynomials, of which the typical member (mth-order) is denoted by

$$N_{x_1 y_1 \dots x_m y_m}^{y_1 \dots y_m abc..}$$

This polynomial is defined as the BAPS of that set of trees of N whose ports $x_1, \dots, x_m, p, q, r, \dots$ are collapsed, which each contain m branch paths from the port terminals of ports y_1, \dots, y_m to the respective port terminals x_1, \dots, x_m . This set of trees is a subset of the set associated with the natural polynomial

$$N_{x_1 \dots x_m p q r \dots}^{y_1 \dots y_m a b c \dots}$$

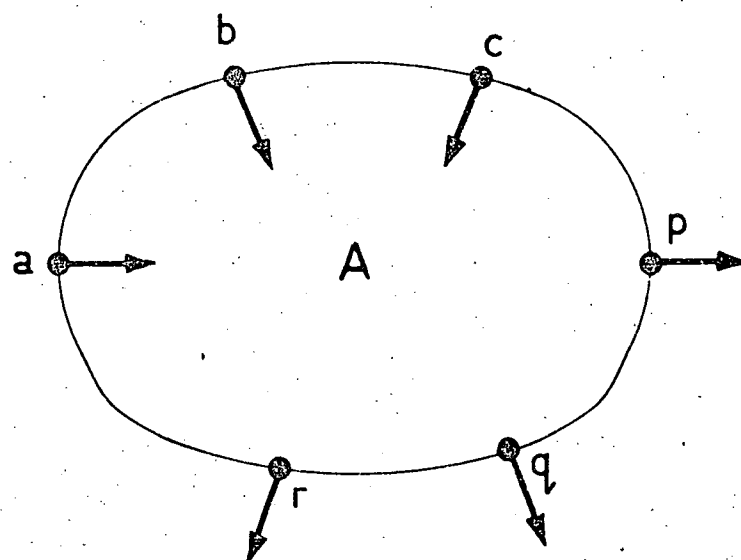
Because the ports x_1, \dots, x_m are collapsed, the m branch paths are necessarily separate.

A pictorial representation of these polynomials highlights their distinguishing features. With each port of a network is associated a pointer. If a port is collapsed, its pointer is directed out of the network; otherwise, it is directed into the network. In either case the pointer indicates the initial direction of the branch paths from the port terminal to ground, and a setting of all the pointers of a network, or pointer setting, thus determines the set of trees associated with a particular natural polynomial. Its subscripts identify the ports whose pointers are directed out of the network, and its superscripts identify the ports whose pointers are directed into the network (for example, see figure 1.1a).

A subset of trees which each contain m branch paths between pairs of port terminals is represented by a set of m lines called pointer paths drawn across the network between the respective pairs of ports, with directions determined by the pointers (for example, see figure 1.1b). Hence, in the representation of a multiple-order transfer polynomial the pointer setting determines the subscripts and superscripts, while the pointer paths determine the pairs of port indices beneath the network's base symbol.

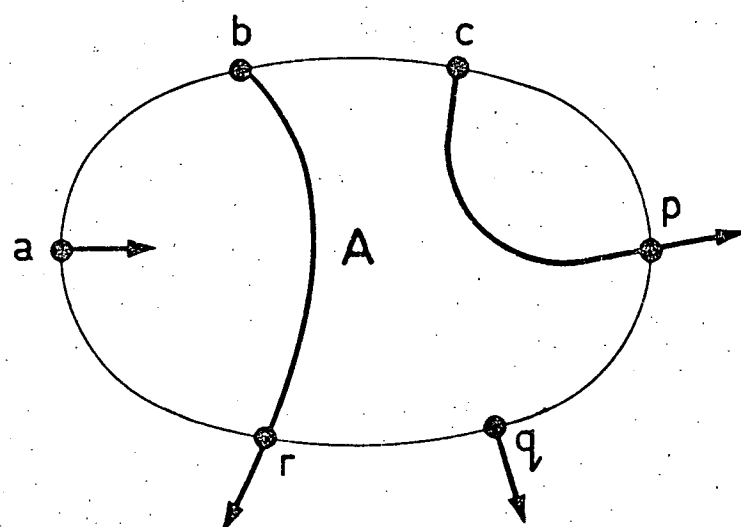
Definition 1.3 ("linkage polynomials")

It is left until chapter 2 to relate the network polynomials to network behaviour and to explore the nature of the characterisation which they provide. It will be seen that in the analysis of networks certain groups of m th-order transfer polynomials always occur in combination. Such combinations, which have been called linkage polynomials by Pike [41], are here defined in terms



$$A_{pqr}^{abc}$$

(a)



$$A_{pqr}^{abc}$$

$$\begin{matrix} rb \\ pc \end{matrix}$$

(b)

Figure 1.1 Polynomials of a network A, and their representation.

of the transfer polynomials by

$$\begin{matrix} y_1 \dots y_m \\ x_1 \dots x_m \end{matrix} N_{pqr\dots}^{abc\dots} \equiv (-1)^m \sum \delta_{j_1 \dots j_m}^{j_1 \dots j_m} \cdot \begin{matrix} y_{j_1} \dots y_{j_m} \\ x_1 y_{j_1} \\ \vdots \\ x_m y_{j_m} \end{matrix} N_{pqr\dots}^{abc\dots} \quad (1.1)$$

where the summation is over all the $m!$ permutations j_1, \dots, j_m of $1, \dots, m$, and δ is the generalised Kronecker delta which is +1 or -1 depending on whether the permutation is even or odd.

The notation for linkage and transfer polynomials is made more compact by replacing the various sets of port indices with Greek symbols. Thus the general linkage polynomial of equation 1.1 becomes ${}_{\alpha}^{\beta} N^{\gamma}$ where

$$\alpha = \{x_1 \dots x_m\}, \quad \beta = \{y_1 \dots y_m\} \quad \gamma = \{abc\dots\}.$$

By convention, all the remaining port indices p, q, r, \dots which are not included in the sets α, β, γ are assumed to be in the suffixed subscript position.

The cardinal number of a set is denoted by square brackets. Thus, in this example, $[\alpha] = [\beta] = m$.

To determine any one of the m th-order linkage polynomials of an n -port network, $2m$ of the n ports are chosen to either originate or terminate m pointer paths, m of these $2m$ ports are chosen to terminate pointer paths, and the pointers of the remaining $n - 2m$ ports may be directed either into or out of the network. Thus the total number of m th-order linkage polynomials is

$$\begin{aligned} L(n, m) &= \binom{n}{2m} \cdot \binom{2m}{m} \cdot 2^{n-2m} \\ &= \frac{n!}{(n-2m)! (m!)^2} \cdot 2^{n-2m}. \end{aligned}$$

This function is tabulated in table 1.1.

<div>m \ n</div>	0	1	2	3	TOTAL
2	4	2			6
3	8	12			20
4	16	48	6		70
5	32	160	60		252
6	64	480	360	20	924

Table 1.1 The number of mth-order linkage polynomials of an n-port network.

When counting the total number of linkage polynomials of all orders (including the natural polynomials) we note that each linkage polynomial can be associated with a unique selection of n symbols from 2n symbols. For example, with a total population consisting of n "currents" i_1, \dots, i_n and n "voltages" e_1, \dots, e_n , the general linkage polynomial ${}^\beta_\alpha N^\gamma$ may be uniquely associated* with the selection of n symbols which includes those voltages whose indices are included in the sets β and γ and those currents whose indices are not included in either of the sets α or γ . Thus the total number of linkage polynomials is

$$\sum_{m=0}^{\lfloor \frac{n}{2} \rfloor} L(n,m) = \binom{2n}{n}.$$

1.2.1 Structures of Multiport Networks

In keeping with the multiport-network characterisation developed above, all interconnections between networks may be made only at their ports. It is further stipulated that a connection between ports must be characterised by either a voltage or current which is common to all the ports, i.e. the ports are either in parallel or in series. Any connection of ports can be made to

* This association of linkage polynomials with segregations of port currents and voltages, as occurs in the selection of a set of independent variables with which to describe the behaviour of an n-port network, is actually substantiated by the theorems of chapter 2.

conform to this rule by introducing simple 2-port networks as, for example, in figure 1.2. A set of multiport networks connected together in this way is here called a structure of networks. The individual networks in a structure are called constituent networks, and the network formed by the structure is called the complete network.

When connected together, a set of ports of different constituent networks is regarded, for identification purposes, as a single port of the structure. If a port of the structure has connections only to constituent networks and not to some external network, i.e. it does not correspond to a port of the complete network, it is called an internal port. Otherwise it is called an external port.

Polynomials and trees of the complete network are called complete polynomials and complete trees respectively; polynomials and trees of the constituent networks are called constituent polynomials and constituent trees.

The task of analysing a structure can now be simply stated as that of calculating the complete polynomials from the given constituent polynomials. But before we begin this task, a further simplification is made, without loss of generality, by considering only those structures in which all ports are of the parallel type. Structures with ports connected in series may be converted to equivalent structures containing only parallel ports by inserting a unit gyrator in every port which is connected in series. The effect of a unit gyrator connected to a port is to interchange the voltage and current values, so that whereas a series connection of ports constrains the currents to be equal, the ports of the equivalent structure must be connected in parallel to constrain the voltages to be equal. It is seen in chapter 2 that cascading a network with unit gyrators only interchanges some polynomials—because ports that were originally open-circuit become collapsed, and vice versa—and changes the sign of others, due to the antireciprocal nature of a gyrator.

With all the ports of a structure now of the parallel type and, if necessary, isolated from the constituent networks by ideal transformers, one terminal of every port is connected to a common ground. With each port is associated a pointer which may be directed to any one of the constituent networks attached to that port. Because it is possible to interpret the

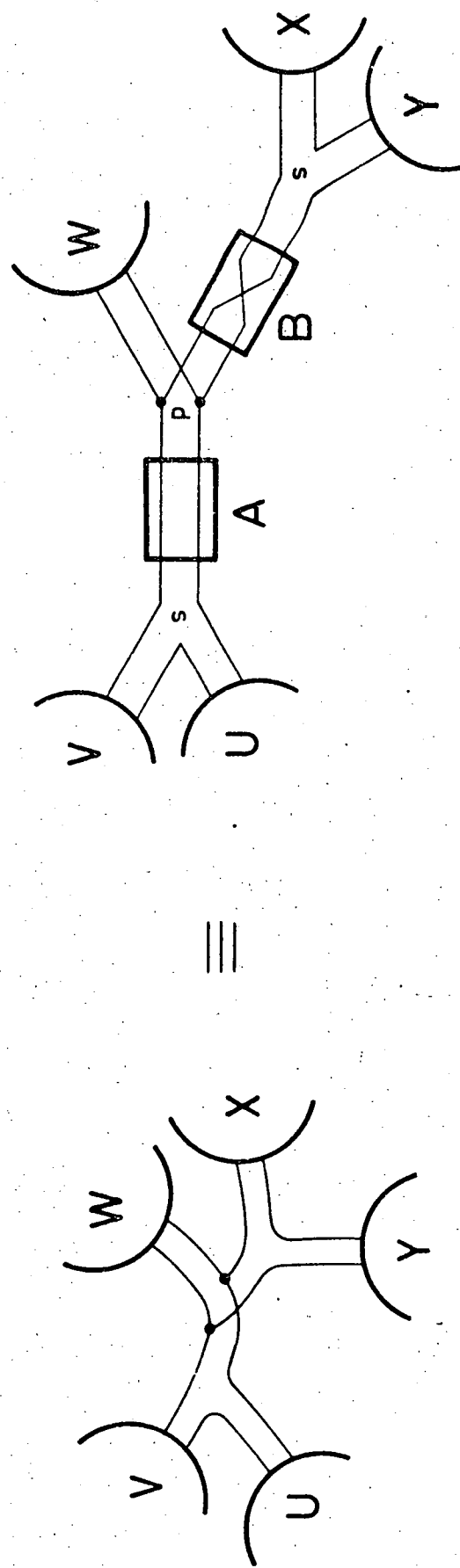


Figure 1.2 Equivalent interconnections of ports.

direction of a pointer as indicating the initial direction of paths from the ungrounded port terminal to ground, a setting of all the pointers of a structure may be interpreted as specifying particular polynomials of the constituent networks, in the same way that they do for an isolated network. Pointers of internal and external ports are called internal pointers and external pointers respectively.

1.3 THE ANALYSIS PROCESS

1.3.1 Calculation of Natural Polynomials

The process for analysing a complete network structure which, together with its constituent parts, is characterised by polynomials, is based on an analysis of the various trees of the complete network.

It is first noted that a complete polynomial is the BAPS (branch-admittance-product-sum) of complete trees, and the polynomial or its associated set of complete trees is represented by a setting of internal pointers.

Attention is focused on the ungrounded terminals of both the internal and external ports. Because each complete tree, by definition, contains a unique path to ground from every node in the complete network, the complete trees are classified uniquely according to the initial direction taken by the paths from these port terminals to ground. The classification concerns only the first constituent networks through which these paths pass, and each class is therefore represented by a setting of all the pointers.

A pointer setting thus determines a set of constituent polynomials and also a class or subset of complete trees associated with a complete polynomial. Furthermore, every complete tree in the class is a union of constituent trees associated with the constituent polynomials. However, not every union of constituent trees determined by the pointer setting is necessarily a complete tree. The path from a node to ground either lies wholly in one constituent network or passes through a port into an adjacent constituent network. The path in the adjacent network may also pass through another port to connect with a path in yet another constituent network, and so on, but unless the path leads to a port already passed by itself—and thus forms a loop of branches—it will eventually terminate at the ground node. Hence a union of constituent trees is either a complete tree or forms one or more loops of branches.

The branch loops that are formed by some unions of trees are themselves classified according to the constituent networks traversed by the loops, and the classes are represented by pointer loops drawn across the constituent networks and passing through the ports in the direction of the pointers (for example, see figure 1.3b). The pointer loops not only represent subsets of the unions of constituent trees, but their segments, which traverse individual constituent networks, also determine subsets of the constituent trees to which the trees in the union must belong.

To fully analyse a structure all the possible pointer settings must be considered. Suppose that pointer settings are generated in some regular manner, and consider a pointer setting which defines the i -th class of complete trees associated with a particular complete natural polynomial. Assume that in this general case the pointer setting will allow many pointer loops p, q, r, \dots to be drawn. The sets and subsets of trees and unions of trees are identified with the following symbols:

T_{ik} , the set of trees of the k -th constituent network determined by the i -th setting of pointers;

$T_{ik}^{p,q,r,\dots}$, the set of trees of the k -th constituent network determined by the i -th setting of pointers and the pointer loops p, q, r, \dots (note that

$$T_{ik}^{p,q,r,\dots} \subseteq T_{ik},$$

and the equality holds if and only if none of the pointer loops traverses the k -th constituent network);

$I_i = \left\{ \bigcup_k T_{ik} \mid T_{ik} \in T_{ik} \right\}$,
the universal set of unions;

$L_i^j = \left\{ \bigcup_k T_{ik}^j \mid T_{ik}^j \in T_{ik}^j \right\}$,
the j -loop set of unions;

$T_i = I_i - \bigcup_j L_i^j$,
the tree set, i.e. the set of all unions of constituent trees which are complete trees.

$S(X_i)$ is defined as the BAPS of the trees or unions of trees in the set X_i .

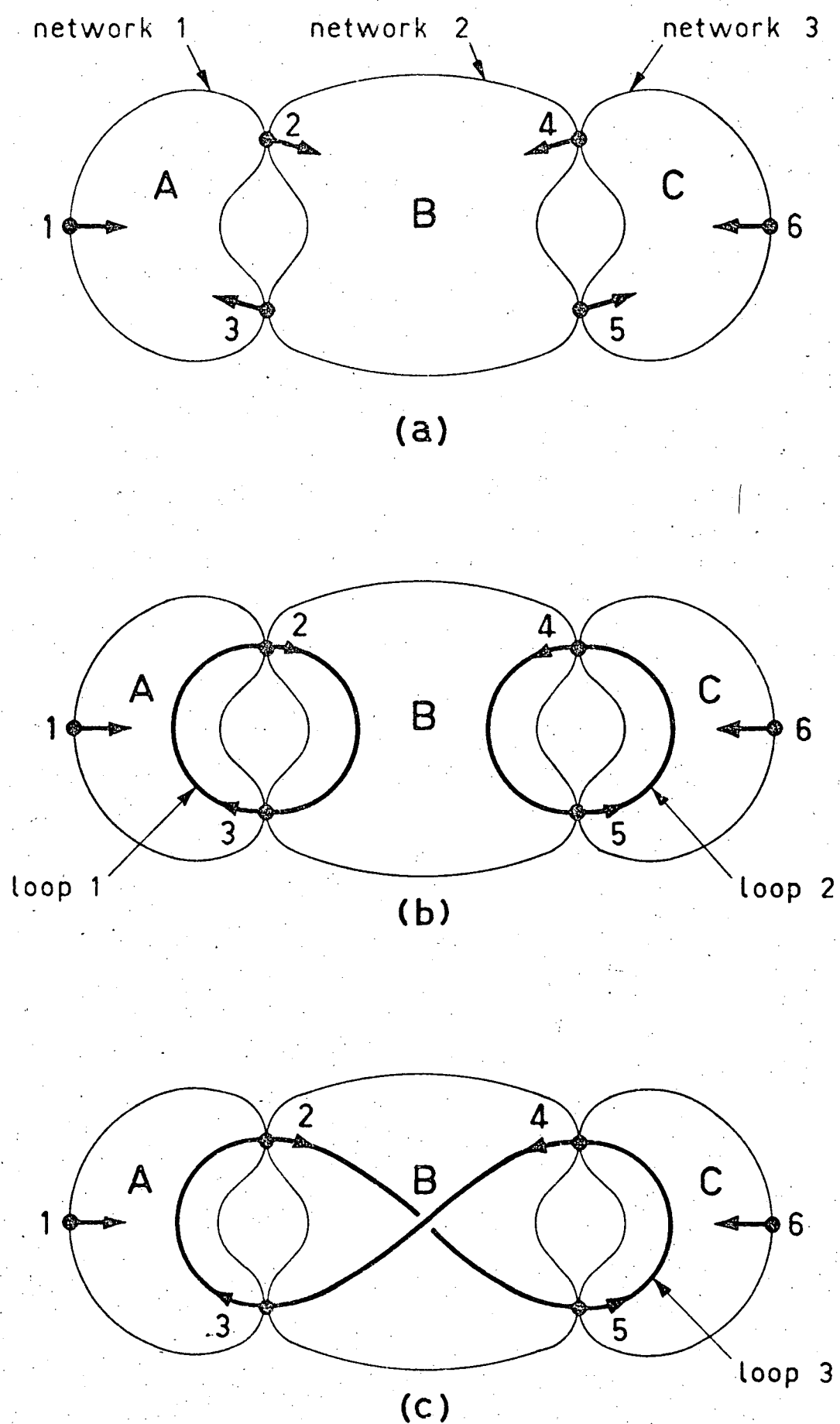


Figure 1.3 A structure of three constituent networks

The BAPS of all unions of trees, with one tree taken from the set for each constituent network, equals the product of the BAPS's of the sets of trees for each constituent network. Hence, by considering the sets of constituent trees determined by the setting of pointers and the various combinations of pointer loops,

$$\begin{aligned}
 S(I_i) &= \prod_k S(T_{ik}), \\
 S(L_i^j) &= \prod_k S(T_{ik}^j), \\
 S(L_i^m \cap L_i^n) &= \prod_k S(T_{ik}^{m,n}), \\
 S(L_i^p \cap L_i^q \cap L_i^r) &= \prod_k S(T_{ik}^{p,q,r}); \text{ etc}
 \end{aligned} \tag{1.2}$$

The factors $S(T_{ik})$, $S(T_{ik}^j)$, $S(T_{ik}^{m,n})$, ... are polynomials of the constituent networks; $S(T_{ik})$ is a natural polynomial, and $S(T_{ik}^{p,q,r,\dots})$ is a nonzero-order transfer polynomial — unless $T_{ik}^{p,q,r,\dots} = T_{ik}$.

The desired quantity is the BAPS of the complete trees which belong to the class represented by the i -th pointer setting, and is given by

$$\begin{aligned}
 S(T_i) &= S(I_i) - S(\bigcup_j L_i^j) \\
 &= S(I_i) - \sum_j S(L_i^j) + \sum S(L_i^m \cap L_i^n) \\
 &\quad - \sum_{p,q,r} S(L_i^p \cap L_i^q \cap L_i^r) + \dots \pm S(\bigcap_j L_i^j)
 \end{aligned} \tag{1.3}$$

The steps required for each setting of pointers are reviewed and illustrated with reference to the structure of networks and setting of pointers shown in figure 1.3.

- (a) Search for all possible pointer loops. In the example, three pointer loops can be drawn with this, say the i -th, setting of pointers. Representations of the sets of tree unions L_i^1 and L_i^2 , alone, are not illustrated, although figure 1.3b represents the set of tree unions $L_i^1 \cap L_i^2$ and determines the sets of constituent trees $T_{i1}^{1,2}$, $T_{i2}^{1,2}$, and $T_{i3}^{1,2}$. Figure 1.3c represents the set of tree unions L_i^3 and determines the sets of constituent trees T_{i1}^3 , T_{i2}^3 , and T_{i3}^3 .

(b) The BAPS's of tree unions are calculated using the equations 1.2. Thus:

$$\begin{aligned}
 S(I_i) &= A_2^{13} B_{35}^{24} C_4^{56}, \\
 S(L_i^1) &= A_{23}^{13} B_{32}^{24} C_4^{56}, \\
 S(L_i^2) &= A_2^{13} B_{54}^{24} C_{45}^{56}, \\
 S(L_i^3) &= A_{23}^{13} B_{52}^{24} C_{45}^{56}, \\
 \text{and } S(L_i^1 \cap L_i^2) &= A_{23}^{13} B_{32}^{24} C_{45}^{56}.
 \end{aligned}$$

The sets $L_i^1 \cap L_i^3$, $L_i^2 \cap L_i^3$ and therefore $L_i^1 \cap L_i^2 \cap L_i^3$ are empty. Note that the subscripts and superscripts, determined by the pointer setting, are the same in each product, and that only the pairs of transfer indices beneath the base symbols, determined by pointer paths, vary from one product to another.

(c) The BAPS of the complete trees is calculated with equation 1.3. Thus

$$S(T_i) = ABC - ABC - ABC - ABC + ABC.$$

$\begin{matrix} & & & & \\ & & 23 & 32 & \\ & & 54 & 45 & \\ & & 23 & 52 & 45 \\ & & & 34 & \\ & & & & 23 & 32 & 45 \\ & & & & & 54 & \end{matrix}$

(The subscripts and superscripts have been omitted for clarity.)

Because the double-transfer polynomials B_{32}^{24} and $-B_{52}^{24}$ have the cofactor AC_{2345} they are combined as the one linkage polynomial B_{35}^{24} and the BAPS is expressed entirely in terms of linkage polynomials as follows:

$$\begin{aligned}
 S(T_i) &= A_2^{13} B_{35}^{24} C_4^{56} - A_2^{13} B_{35}^{24} C_4^{56} - A_2^{13} B_{35}^{24} C_4^{56} - A_2^{13} B_{35}^{24} C_4^{56} + A_2^{13} B_{35}^{24} C_4^{56} \\
 &+ A_2^{13} B_{35}^{24} C_4^{56}.
 \end{aligned} \tag{1.4}$$

In the general case, transfer polynomials of a network N which group together into one linkage polynomial will always have the same cofactor in equation 1.3 because the pointer paths which complete pointer loops by traversing the networks external to N are independent of the pointer paths which traverse N itself. The sign change manifested by the Kronecker delta in equation 1.1 takes into account the change in the number of distinct pointer loops as the transfer indices are permuted.

The final stage in the calculation of a complete natural polynomial corresponding to a particular setting of external pointers involves the summation of BAPS contributions obtained from every possible setting of internal pointers; that is,

$$S(\cup T_i) = \sum_i S(T_i). \tag{1.5}$$

1.3.2 Calculation of High-Order Polynomials

A transfer polynomial of the complete network is the BAPS of a set of trees which all have branch paths between certain pairs of external ports. The trees are classified according to the constituent networks traversed by the paths, and the classes are represented by pointer paths drawn across the networks and through the internal ports in the direction of the pointers. As with complete natural polynomials, the trees are unions of trees of constituent networks, and, in company with the pointer settings, the segments of the pointer paths determine the sets of constituent trees to which the trees in the union must belong. The possible existence of pointer loops elsewhere in the structure must again be taken into account.

Rather than describe a different algorithm to calculate the high-order transfer polynomials of a complete network, the concept of a closing network is introduced in order that the one algorithm should generate the transfer and linkage polynomials of all orders.

The closing network is an imaginary network which connects all the external ports of the complete network. It may be regarded as the environment of the complete network or the complement of the complete network in the "universal" system; it is the network into which the external pointers are directed when they are directed away from the complete network. In this sense it has the same status as a constituent network, and the external ports thus lose their distinction from internal ports. It is called a closing network because it provides imaginary paths to close the pointer paths through the complete network and so form pointer loops.

A high-order transfer polynomial can now be defined as the BAPS of those trees which are capable of forming branch loops through the closing network, and these trees are subject to the same classification and rules of evaluation as the unions of trees which form branch loops through the normal constituent networks.

1.3.3 The Analysis Algorithm

The one algorithm which calculates the complete linkage (or transfer) polynomials of all orders is summarised in the following steps:

- (a) To the set of real constituent networks add the closing network.
- (b) With each port associate a pointer which may be directed into any attached constituent network (real or closing).
- (c) Set the polynomials of the closing network to zero. These polynomials will be employed as accumulating sums of products of polynomials of the real constituent networks.
- (d) Generate every possible setting of pointers once and only once.

For every setting take the following steps:

- (i) Search for all possible pointer loops.
- (ii) Determine all the polynomial products given by equations 1.2.

For every polynomial product take the following steps:

- (1) Give the product a sign as determined by equation 1.3:
if the product is represented by an even number of pointer loops, the sign is positive; otherwise, the sign is negative.
 - (2) Every product will include one polynomial from every real constituent network and will also determine a polynomial of the closing network. The polynomials of the real constituent networks are multiplied together and the product is added — or subtracted, depending on the sign from step (1) — to the accumulated polynomial of the closing network.
- (e) At the completion of step (d), calculate the polynomials of the complete network N from the accumulated polynomials of the closing network \bar{N} with either of the equations

$$\begin{aligned} {}^{\beta}N_{\alpha}^{\gamma} &= (-1)^{[\alpha]} \cdot {}^{\alpha}\bar{N}_{\beta}^{\delta} \\ \text{or } N_{\alpha/\beta}^{\beta\gamma} &= (-1)^{[\alpha]} \cdot \bar{N}_{\beta\alpha}^{\alpha\delta} \end{aligned}$$

1.4 APPLICATION OF THE ANALYSIS PROCESS

1.4.1 General Structures

In common with all topological methods the analysis process described above suffers severe limitations with regard to the magnitude of its task. As the number of constituent networks and ports is increased the analysis task tends to grow exponentially. Even analysis of the apparently simple structure of figure 1.3 (with ports 1 and 6 the external ports) is tedious if done by hand: it requires attention to 2^6 different pointer settings, each involving a search for pointer loops and the calculation of one or more polynomial products.

It has not been practical to implement the general algorithm as a computer program for several reasons: for instance, one potentially difficult problem concerns the storage and addressing of the large numbers of polynomials associated with each network (see table 1.1). To make the algorithm more practical, further investigation is required to find suitable routines to recognise pointer loops, and to group together polynomial products in such a way that sums of transfer polynomials may be replaced by their equivalent linkage polynomials. It seems likely, though, that a practical algorithm would somehow combine these two routines with a special routine for generating the pointer settings.

Nevertheless, these practical difficulties are alleviated by the diakoptic approach which the method permits. Because the results of analysis of one network — the linkage polynomials — can be used directly as ingredients for the analysis of some larger network, it is possible, and generally advantageous, to tear a network apart into progressively smaller substructures and analyse them separately, so that at any stage only a relatively simple structure needs to be analysed. This approach is particularly attractive when the systems to be analysed are, irrespective of size, only loosely interconnected.

Unfortunately, a diakoptic approach raises the problem of specifying and controlling the manner in which subnetworks are created and manipulated. Experience with other diakoptic methods, such as that of Kron [28], suggests that this task is better done manually than with a computer routine, and it is notable that Ishizaki et al [27] have developed a language notation with which to specify the algebraic manipulation of multiport networks. But, again because of the large numbers of linkage polynomials, this aspect of the topological method has not been investigated in the general case.

1.4.2 Structures of 2-Port Networks

Limiting all networks to two ports effects a drastic simplification in the analysis process without seriously limiting its application. Most system components and circuit devices can be modelled directly as constituent 2-port networks, and two ports allow sufficient access to a complete network to determine any transfer or driving-point immittance functions that may be sought.

In the following analysis all networks have exactly two ports. Consequently all networks are characterised by six polynomials, and the organisation of polynomial storage and manipulation is comparatively simple.

When analysing large structures of 2-port networks with the general algorithm, pointer settings are best generated by setting pointers one at a time in an order which attempts to follow the formation of pointer paths, so that pointer loops are detected automatically as they are formed. Because a pointer determines the polynomials of the 2-port networks which it traverses, even though pointers elsewhere in the structure may not be set, it is possible to factorise the sum of polynomial products associated with a pointer setting.

Consider one setting of pointers for which there are p pointer loops.

Let t_j be the product of transfer polynomials determined by the j -th pointer loop,

n_j be the corresponding product of natural polynomials determined by the j -th pointer loop,

and n_0 be the product of natural polynomials of networks which are not traversed by any pointer loop.

From equation 1.3 the contribution to the appropriate closing polynomial is

$$\begin{aligned}
 S &= n_0 n_1 \dots n_p - (n_0 t_1 n_2 \dots n_p + n_0 n_1 t_2 n_3 \dots n_p + \dots) \\
 &\quad + (n_0 t_1 t_2 n_3 \dots n_p + n_0 n_1 t_2 t_3 n_4 \dots n_p + \dots) \\
 &\quad \dots \quad (-1)^p n_0 t_1 t_2 \dots t_p \\
 &= n_0 n_1 \dots n_p \left(1 - \sum_i \frac{t_i}{n_i} + \sum_{i,j} \frac{t_i t_j}{n_i n_j} - \dots (-1)^p \frac{t_1 t_2 \dots t_p}{n_1 n_2 \dots n_p} \right) \\
 &= n_0 n_1 \dots n_p \prod_i \left(1 - \frac{t_i}{n_i} \right) \\
 &= n_0 \prod_i (n_i - t_i) \tag{1.7}
 \end{aligned}$$

By calculating the factors $(n_i - t_i)$ as the pointer loops are formed, many polynomial manipulations and associated book-keeping chores are avoided and some sources of numerical round-off error are eliminated.

The practical application of algorithms using the above expression in the analysis of structures of 2-port networks has been thoroughly investigated and is the subject of part II of the thesis.

1.5 CONCLUSION

This chapter introduced a set of topological quantities as parameters to characterise multiport networks, and developed an analytical process which relates the parameters of a complete network with the parameters of its constituent parts. The process has two important features. First, it permits a diakoptic approach to the analysis of large systems; and second, because the parameters need only be multiplied together, added, or subtracted, it permits a totally symbolic analysis. The advantages of both features are discussed further in chapter 4.

POLYNOMIALS AND NETWORK BEHAVIOUR

2.1 INTRODUCTION

The process presented in chapter 1, for analysing structures of multiport networks in terms of topological quantities, is not complete as a useful analysis theory because the parameters which are used to characterise networks have in no way been related to the observable behaviour of the networks. The aim of this chapter is to establish such a relationship.

In the strict logical development of this relationship the first important goal is to establish that the characterisation of a network by a set of linkage polynomials is, in some sense, unique. But the proof of the relevant theorem (2.2) is supported by a special case of another theorem (2.1) that relates all linkage polynomials to a particular subset of the linkage polynomials. Because the proof of the latter theorem relies largely on an application of the analysis process, it is introduced first. Proof of the general case for theorem 2.1 must, however, be reserved until theorem 2.2 is proved.

The first two theorems constitute the major part of the chapter. It is a relatively simple step to theorem 2.3 which establishes a connection between the linkage polynomials and the behaviour of a network characterised by its short-circuit admittance matrix. Theorem 2.4 follows from another simple application of the analysis process and provides an effective means of generalising any identities involving polynomials and port variables.

2.2 IDENTICAL NETWORKS IN PARALLEL

The connection of two, or more, identical multiport networks in parallel allows a simple demonstration of the analysis process of chapter 1, and leads to a theorem which establishes all the relationships between the polynomials of a network.

Lemma 2.1

For a structure of k identical n -port networks, with their corresponding ports connected in parallel, the polynomials of the complete network M are related to the polynomials of the constituent networks N by the expression

$$\frac{\beta}{\alpha} M^{\delta} = k^{[\beta]+[\delta]} \cdot (N)^{k-1} \cdot \frac{\beta}{\alpha} N^{\delta}.$$

Proof (for networks without internal nodes)

If the network contains no internal nodes then the natural polynomial N is unity because, with all ports collapsed, the trees contain no branches. The complete network M behaves as a single network N with all its branch admittances multiplied by the factor k , and the theorem is proved for this special case by noting that the trees associated with the general polynomial contain $[\beta]+[\delta]$ branches.

Definition 2.1 ("the X matrix")

Throughout this chapter a particular matrix, whose elements are ratios of polynomials, will play a major role. It is introduced at this stage simply as the X matrix. Its minor determinant, comprising columns a, b, c, \dots and rows p, q, r, \dots , is denoted by

$$X_{pqr\dots}^{abc\dots}$$

With this notation the X matrix is defined by its elements, as follows:

$$X_i^j = \frac{N_i^j}{N} \quad (i \neq j),$$

and

$$X_i^i = \frac{N_i^i}{N}.$$

Theorem 2.1

The general linkage polynomial is related to a minor determinant of the X matrix by the identity

$$\frac{\beta}{\alpha} \frac{N^{\delta}}{N} = X_{\alpha\delta}^{\beta\delta}.$$

Proof

The theorem will be proved by induction on e , where $e = [\beta] + [\delta]$.

The truth of the theorem for $e = 1$ is established by definition 2.1.

We now assume that the theorem is true for all $e < f$, and proceed to establish the theorem for $e = f$ by considering two identical n -port networks N connected in parallel.

If $[\beta] + [\gamma] = f$ then by lemma 2.1 (which at this stage is proved only for networks without internal nodes), the general linkage polynomial of the complete network M is given by

$${}_{\alpha}^{\beta}M^{\gamma} = 2^f \cdot N \cdot {}_{\alpha}^{\beta}N^{\gamma} \quad (2.1)$$

The same polynomial is now calculated using the analysis process developed in chapter 1. The pointers of the ports denoted by β and γ may be directed into either of the constituent networks, and the remaining pointers are directed into the closing network. There are, therefore, a total of 2^f pointer settings to be considered.

In the typical pointer setting of those to be considered, suppose that, of the set β , the subset of pointers β^1 are directed into the first constituent network and the remaining subset β^2 are directed into the second constituent network. Let the subsets of α which correspond to β^1 and β^2 (by virtue of the order of the members of α and β) be α^1 and α^2 respectively. Thus, if $\beta = \{\gamma_1 \dots \gamma_m\}$, $\alpha = \{x_1 \dots x_m\}$, and $\beta^1 = \{\gamma_2 \gamma_3 \gamma_7\}$ then $\alpha^1 = \{x_2 x_3 x_7\}$. Similarly, let γ^1 and γ^2 denote the two subsets of γ whose pointers are directed to the first and second networks respectively.

The transfer polynomials constituting the complete linkage polynomial are represented by pointer paths drawn across the structure, starting from the ports β^1 and β^2 , and terminating at the ports α^1 and α^2 . The major task is to find all such pointer paths and all the pointer loops, find all the transfer-polynomial products determined by equation 1.3, and group them together into products of linkage polynomials.

It is noted that pointer loops can only be drawn through ports belonging to the sets γ^1 and γ^2 , although any path from a port in β to a port in α may pass through any numbers of ports in γ^1 and γ^2 alternately; for instance, a path may originate at a port in β^1 , pass through different ports in γ^2 , γ^1 , γ^2 , γ^1 successively, and terminate at a port in α^1 .

Consider a sum of products of transfer polynomials which constitute a typical product of linkage polynomials. Suppose that in one constituent network,

Hence, the sign associated with the linkage product (2.2) is

$$\begin{aligned} & \delta_{\alpha_1' \alpha_1^2 \alpha_3^2 \alpha_2^2 \alpha_2'} \cdot (-1)^{[\gamma_2'] + [\gamma_3']} \\ &= \delta_{\alpha_1' \alpha_1^2 \alpha_3^2 \gamma_1^2 \gamma_1' \gamma_2^2 \gamma_2' \alpha_2^2 \alpha_2' \gamma_2^2} \end{aligned}$$

For a particular pointer setting every product of transfer polynomials determined by equation 1.3 is contained in the expansion of one and only one linkage polynomial product of the form 2.2, and, conversely, every term in the expansion of every possible product of this form is a term of equation 1.3. Hence the contribution from one pointer setting to the complete linkage polynomial $\beta_\alpha M^\gamma$ is

$$C = \sum \left(\delta_{\alpha_1' \alpha_1^2 \gamma_1^2 \gamma_2^2 \alpha_2^2 \alpha_2'} \cdot \beta_{\alpha_1' \alpha_1^2 \gamma_1^2} N^{\gamma_1'} \cdot \gamma_2^2 \beta_{\gamma_2^2 \alpha_2^2 \alpha_2'} N^{\gamma_2^2} \right), \quad (2.5)$$

where the summation is over all sets of α_2', α_1^2 (which replaces both α_1^2 and α_3^2 in the preceding example), γ_2' (which replaces both γ_2' and γ_3'), and γ_1^2 . To include the general term, the only relationship between the sizes of these sets is expressed by $[\alpha_2'] + [\gamma_2'] = [\alpha_1^2] + [\gamma_1^2]$.

(The preceding example considered the case in which $[\gamma_2'] > [\gamma_1^2]$). If each constituent network has less than f pointers directed into it then the assumed validity of the theorem relates the polynomials to minors of the X matrix, with the result that expression 2.5 divided by $(N)^2$ is recognised as a Laplace expansion of a minor of the X matrix. That is,

$$\begin{aligned} C &= (N)^2 \cdot \sum \left(\delta_{\alpha_1' \alpha_1^2 \gamma_1^2 \gamma_2^2 \alpha_2^2 \alpha_2'} \cdot X_{\alpha_1' \alpha_1^2 \gamma_1^2 \gamma_2^2}^{\beta_{\alpha_1' \alpha_1^2 \gamma_1^2}} \cdot X_{\gamma_2^2 \alpha_2^2 \alpha_2'}^{\gamma_2^2 \beta_{\gamma_2^2 \alpha_2^2 \alpha_2'}} \right) \\ &= (N)^2 \cdot X_{\alpha_1' \alpha_2^2 \gamma_1^2 \gamma_2^2}^{\beta_{\alpha_1' \alpha_2^2 \gamma_1^2 \gamma_2^2}} \\ &= (N)^2 \cdot X_{\alpha \gamma}^{\beta \gamma} \end{aligned} \quad (2.6)$$

In this Laplace expansion the first minor comprises columns β' and γ' corresponding to the pointers directed into the first constituent network, and the second minor comprises columns β^2 and γ^2 , corresponding to the pointers

directed into the second constituent network. In the "diagonal" term of the expansion, whose sign is

$$\delta_{\alpha' \gamma' \alpha^2 \gamma^2}^{\alpha' \gamma' \alpha^2 \gamma^2} = +1,$$

the first minor comprises rows α' and γ' , and the second minor comprises rows α^2 and γ^2 . Other terms are obtained by interchanging rows α'_2 and γ'_2 of the first minor with rows α^2_1 and γ^2_1 of the second minor.

The contribution to the complete polynomial determined by two of the pointer settings cannot be expressed in this form because one of the constituent networks has f pointers directed into it. But, for both these pointer settings, the other constituent network has no pointers directed into it, in which case the contribution is simply

$$N \cdot {}^\beta_\alpha N^\gamma. \quad (2.7)$$

The analysis process is completed by combining the contributions of all 2^f pointer settings given by the expressions 2.6 and 2.7, whence

$${}^\beta_\alpha M^\gamma = (2^f - 2) \cdot (N)^2 \cdot X_{\alpha\gamma}^{\beta\gamma} + 2 \cdot N \cdot {}^\beta_\alpha N^\gamma. \quad (2.8)$$

Elimination of the complete polynomial ${}^\beta_\alpha M^\gamma$ from equations 2.1 and 2.8 establishes the theorem for $e = f$, and hence, by induction, for all e .

Q.E.D.

This theorem is the key to all the relationships between the linkage polynomials of a multiport network, and is used in section 2.7 to establish two important polynomial identities. In particular, we note the following corollary, without proof:

Corollary 2.1

All the linkage polynomials of an n -port network may be derived from the set of $(n^2 + 1)$ polynomials comprising the $(n + 1)$ natural polynomials and the $n(n - 1)$ single-transfer polynomials that occur in the X matrix.

2.3 POLYNOMIALS OF EQUIVALENT NETWORKS

In order to relate the polynomials of a network, defined as branch-admittance-product-sums (BAPS) of trees, to the electrical behaviour of the network observed at its ports, we must at some stage investigate the relationships between the voltages and currents in a network. Considering first a two-terminal device by itself, the current through it, and the voltages between both its terminals and some common ground point, the observance of a linear relationship—such as Ohms Law—is implied by the adoption of the admittance parameter to characterise the resistors, unistors and gyrators with which we model an electrical circuit. The consequences of Kirchoff's Law, however, make their first appearance in this section.

The practical value of linkage polynomials, as a set of parameters to characterise a network, is assured by the following theorem.

Theorem 2.2

Electrically equivalent networks, i.e. networks which exhibit the same electrical behaviour when observed at their ports, are characterised by sets of linkage polynomials for which the ratios between corresponding pairs of polynomials are equal. In other words, the polynomials of equivalent networks are identical, except for some multiplicative constant which applies to all the polynomials of a network.

Proof

Networks with the same behaviour but different internal topological structures can be transformed from one to another by successively introducing or eliminating internal nodes. Hence, to prove the theorem, it is sufficient to show that a transformation which eliminates an internal node without changing the network's behaviour also preserves the ratios between network polynomials.

Suppose that a star network A with internal node r and n external nodes is replaced by an equivalent mesh network B, as in figure 2.1. In network A the connection between node r and an external node i will, in general, comprise two unistors, one directed from node i with admittance y_i^1 , and the other directed to node i with admittance y_i . The equivalent mesh contains unistors directed

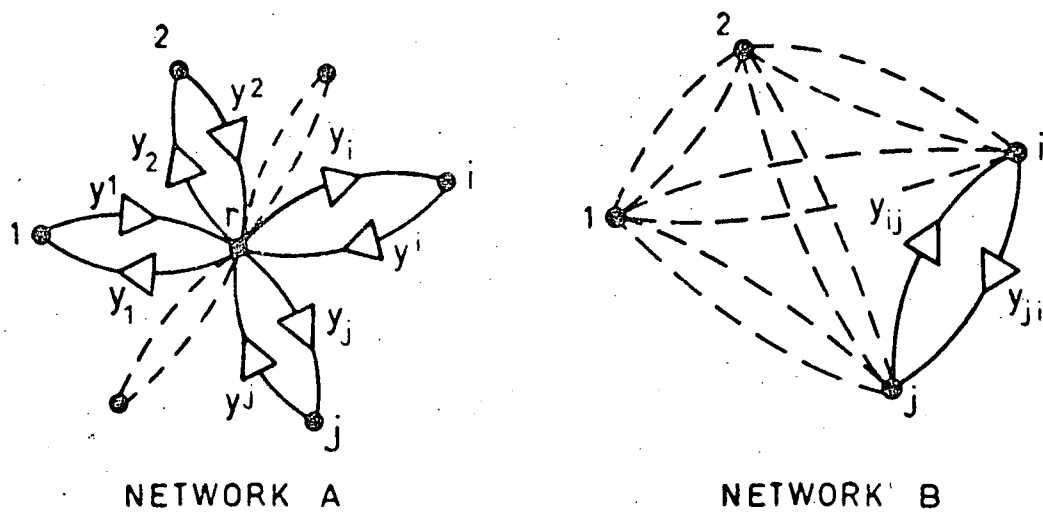


Figure 2.1 Equivalent star and mesh networks A and B.

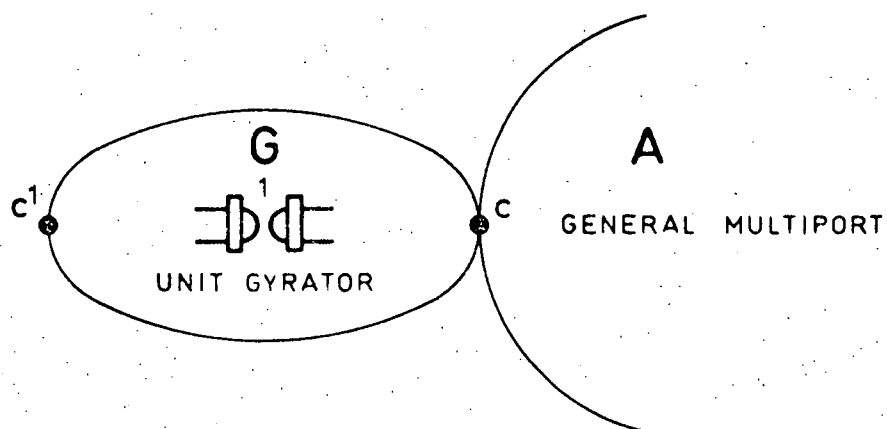


Figure 2.2 Cascade connection of network A with a unit gyrator at port c .

from node j to node i with admittance

$$y_{ij} = y_i y^j / Y$$

where
$$Y = \sum_k y_k . \quad (2.9)$$

That the two networks are in fact equivalent is demonstrated by comparing their driving-point and transfer admittances.

We define E_i to be the voltage between node i and some common ground point, and I_i to be the current entering the network at node i . Note that, by definition, the current in a unistor directed from node i with admittance y^i is $y^i \cdot E_i$.

Applying Kirchoff's current law to the internal node of network A,

$$I_r = \sum_i y_i \cdot E_r - \sum_i y^i \cdot E_i = 0,$$

i.e.
$$E_r = \sum_i y^i / Y \cdot E_i .$$

If all the external nodes except j are short-circuited then

$$E_r = y^j / Y \cdot E_j ,$$

$$I_i = - y_i \cdot E_r ,$$

and
$$I_j = y^j \cdot E_j - y_j \cdot E_r .$$

The short-circuit transfer and driving point admittances are therefore given by

$$Y_{ij} = I_i / E_j = -y_i \cdot y^j / Y$$

and
$$Y_{jj} = I_j / E_j = y^j - y_j \cdot y^j / Y$$

$$= (Y - y_j) \cdot y^j / Y \quad (2.10a)$$

The short-circuit transfer and driving-point admittances of network B are given by

$$Y_{ij} = -y_{ij}$$

and
$$Y_{jj} = \sum_i y_{ij} \quad (2.10b)$$

They prove to be identical to those of A when the relations 2.9 are invoked.

To calculate the effect of this star-to-mesh transformation on the polynomials of any network in which the star network might be embedded, the complete network is torn in order to isolate the star as a constituent n-port

network. For instance, the star's external node j , together with the common ground terminal, becomes the j -th port of the constituent network. We now compare the polynomials of networks A and B.

With all ports collapsed, the trees of network A contain a single unistor directed from node r to any external node i .

$$\therefore \mathbf{A} = \sum_i y_i = Y. \quad (2.11a)$$

If port j alone is not collapsed, a tree contains the unistor y^j with any other unistor y_i ($i \neq j$).

$$\therefore \mathbf{A}^j = \sum_{i, i \neq j} y_i \cdot y^j = (Y - y_j) \cdot y^j. \quad (2.11b)$$

Only one of these trees contains a path from port j to port i .

$$\therefore {}^j_i \mathbf{A} \equiv -\mathbf{A}^j_{ij} = -y_i \cdot y^j. \quad (2.11c)$$

Further inspection of network A reveals that the general natural polynomial is

$$\mathbf{A}^{abc\dots} = [Y - (y_a + y_b + y_c + \dots)] \cdot y^a \cdot y^b \cdot y^c \dots, \quad (2.11d)$$

and the general single order transfer polynomial is

$${}^i_j \mathbf{A}^{abc\dots} \equiv -\mathbf{A}^{abc\dots}_{ij} = -y_i \cdot y^j \cdot y^a \cdot y^b \cdot y^c \dots \quad (2.11e)$$

The transfer polynomials (and therefore the linkage polynomials) of order greater than 1 are zero.

With all the ports of network B collapsed, there are no trees. If port j alone is not collapsed, each tree consists of a single unistor y_{ij} which also provides a path from port j to port i .

$$\therefore \mathbf{B} = 1, \quad (2.12a)$$

$$\mathbf{B}^j = \sum_i y_{ij} = (Y - y_j) \cdot y^j / Y, \quad (2.12b)$$

$$\text{and } {}^j_i \mathbf{B} \equiv -\mathbf{B}^j_{ij} = -y_{ij} = -y_i \cdot y^j / Y. \quad (2.12c)$$

Because network B has no internal nodes, its remaining linkage polynomials may be determined by application of theorem 2.1. It will be proved by induction on m that

$$\mathbf{B}^{j_1 \dots j_m} = (1 - \sum_{p=1}^m y_{j_p} / Y) \prod_{p=1}^m y^{j_p}, \quad (2.12d)$$

$${}^j_q \mathbf{B}^{j_1 \dots j_{q-1} j_{q+1} \dots j_m} = -y_i / Y \prod_{p=1}^m y^{j_p}, \quad (2.12e)$$

$${}^{j_q j_m}_{ik} \mathbf{B}^{j_1 \dots j_{q-1} j_{q+1} \dots j_{m-1}} = 0. \quad (2.12f)$$

The equations 2.12b and 2.12c establish 2.12d and 2.12e for $m=1$; equation 2.12f is established for $m=2$ by considering the expansion

$$\begin{aligned}
 {}^{j_1 j_2}_{ki} B &= X_{ki}^{j_1 j_2} \\
 &= X_k^{j_1} \cdot X_i^{j_2} - X_i^{j_1} \cdot X_k^{j_2} \\
 &= {}^{j_1}_{k} B \cdot {}^{j_2}_{i} B - {}^{j_1}_{i} B \cdot {}^{j_2}_{k} B \\
 &= \gamma_k / Y \cdot \gamma^{j_1} \cdot \gamma_i / Y \cdot \gamma^{j_2} - \gamma_i / Y \cdot \gamma^{j_1} \cdot \gamma_k / Y \cdot \gamma^{j_2} \\
 &= 0.
 \end{aligned}$$

Expanding, about column j_{m+1} , the minor comprising rows and columns j_1, \dots, j_{m+1} ,

$$\begin{aligned}
 {}^{j_1 \dots j_{m+1}} B &= X_{j_1 \dots j_{m+1}}^{j_1 \dots j_{m+1}} \\
 &= X_{j_1 \dots j_m}^{j_1 \dots j_m} \cdot X_{j_{m+1}}^{j_{m+1}} - \sum_{q=1}^m X_{j_1 \dots j_q \dots j_m}^{j_1 \dots j_q \dots j_m} \cdot X_{j_q}^{j_{m+1}} \\
 &= B^{j_1 \dots j_m} \cdot B^{j_{m+1}} - \sum_{j_q} B^{j_1 \dots j_q \dots j_{m+1}} \cdot B^{j_q} \\
 &= (1 - \sum_{p=1}^m \gamma_{j_p} / Y) \prod_{p=1}^m \gamma^{j_p} \cdot (1 - \gamma_{j_{m+1}} / Y) \cdot \gamma^{j_{m+1}} \\
 &\quad - \sum_{q=1}^m [\gamma_{j_{m+1}} / Y \cdot \prod_{p=1}^m \gamma^{j_p} \cdot \gamma_{j_q} / Y \cdot \gamma^{j_{m+1}}] \\
 &= (1 - \sum_{p=1}^{m+1} \gamma_{j_p} / Y) \prod_{p=1}^{m+1} \gamma^{j_p},
 \end{aligned}$$

which proves equation 2.12d for all m . Repeating the same expansion, but with row j_{m+1} replaced by row i ,

$$\begin{aligned}
 {}^{j_{m+1} j_1 \dots j_m}_i B &= X_{j_1 \dots j_m i}^{j_1 \dots j_m j_{m+1}} \\
 &= X_{j_1 \dots j_m}^{j_1 \dots j_m} \cdot X_i^{j_{m+1}} - \sum_{q=1}^m X_{j_1 \dots j_q \dots i \dots j_m}^{j_1 \dots j_q \dots j_m} \cdot X_{j_q}^{j_{m+1}} \\
 &= B^{j_1 \dots j_m} \cdot B^{j_{m+1}} - \sum_{j_q} B^{j_1 \dots j_q \dots j_{m+1}} \cdot B^{j_q} \\
 &= -(1 - \sum_{p=1}^m \gamma_{j_p} / Y) \prod_{p=1}^m \gamma^{j_p} \cdot \gamma_i / Y \cdot \gamma^{j_{m+1}} \\
 &\quad - \sum_{q=1}^m [\gamma_i / Y \cdot \prod_{p=1}^m \gamma^{j_p} \cdot \gamma_{j_q} / Y \cdot \gamma^{j_{m+1}}] \\
 &= -\gamma_i / Y \cdot \prod_{p=1}^{m+1} \gamma^{j_p},
 \end{aligned}$$

which proves equation 2.12e for all m . Again repeating the expansion, but with rows j_m and j_{m+1} replaced by rows k and i respectively,

$$\begin{aligned}
 {}^{j_m j_{m+1} j_1 \dots j_{m-1}}_{ki} B &= X_{j_1 \dots j_{m-1} ki}^{j_1 \dots j_{m-1} j_m j_{m+1}} \\
 &= X_{j_1 \dots j_{m-1} k}^{j_1 \dots j_{m-1} j_m} \cdot X_i^{j_{m+1}} - X_{j_1 \dots j_{m-1} i}^{j_1 \dots j_{m-1} j_m} \cdot X_k^{j_{m+1}} - \sum_{q=1}^{m-1} X_{j_1 \dots j_q \dots i \dots k}^{j_1 \dots j_q \dots j_m} \cdot X_{j_q}^{j_{m+1}} \\
 &= {}^{j_m}_{k} B \cdot {}^{j_{m+1}}_i B - {}^{j_m}_{i} B \cdot {}^{j_{m+1}}_k B - \sum_{j_q} {}^{j_m}_{ik} B^{j_1 \dots j_q \dots j_{m+1}} \cdot B^{j_q} \\
 &= \gamma_k / Y \cdot \prod_{p=1}^m \gamma^{j_p} \cdot \gamma_i / Y \cdot \gamma^{j_{m+1}} - \gamma_i / Y \cdot \prod_{p=1}^m \gamma^{j_p} \cdot \gamma_k / Y \cdot \gamma^{j_{m+1}} - 0 \\
 &= 0,
 \end{aligned}$$

which proves equation 2.12f for all m .

Any linkage polynomial of order greater than 2 can be expanded in terms containing a linkage polynomial of order 2, and is, therefore, also zero.

Comparison of the linkage polynomials (2.11) of network A with the polynomials (2.12) of network B indicates that they are identical except for the multiplicative constant Y , which applies to all the polynomials.

Because, with the analysis process of chapter 1, the polynomials of any complete network are homogeneous functions of the polynomials of the constituent networks, it follows that the effect of a network transformation involving a node elimination preserves the ratios between the polynomials.

Q.E.D.

The above proof would have been shorter if corollary 2.1 could have been applied to both networks A and B; for then it would have been necessary to compare only the $n^2 + 1$ polynomials which determine the respective X matrices. But this corollary could not be applied to network A because lemma 2.1 is, at this stage, proved only for networks without internal nodes. However, with theorem 2.2 now established, it is possible to prove that the lemma, its theorem, and its corollary are valid for all networks.

Proof of Lemma 2.1

Because the behaviour of the structure M would not be altered if every internal node of one network was connected to the corresponding nodes in the other networks, the structure is equivalent to a single network P obtained from N by multiplying all its branch admittances by the factor k . If g is the number of internal nodes then the number of branches in each tree associated with the general polynomial is $g + [\beta] + [\gamma]$, and the general polynomial of the equivalent structure is

$$\beta_{\alpha} P^{\gamma} = k^{g + [\beta] + [\gamma]} \beta_{\alpha} N^{\gamma}$$

But, by theorem 2.2,

$${}^{\beta}_{\alpha} M^{\gamma} = K \cdot {}^{\beta}_{\alpha} P^{\gamma}$$

for some constant, K.

$$\therefore {}^{\beta}_{\alpha} M^{\gamma} = K \cdot k^{g+[\beta]+[\gamma]} \cdot {}^{\beta}_{\alpha} N^{\gamma}. \quad (2.13)$$

If all the ports are collapsed then equation 2.13 becomes

$$M = K \cdot k^g \cdot N. \quad (2.14)$$

The structure may also be analysed by the process of chapter 1, in which case it is seen that

$$M = (N)^k. \quad (2.15)$$

Elimination of M and $(K \cdot k^g)$ from equations 2.13 and 2.14 proves the lemma for all networks.

Q.E.D.

We note an obvious corollary to theorem 2.2:

Corollary 2.2

The X matrices of equivalent networks are equal.

2.4 POLYNOMIALS AND THE SHORT-CIRCUIT ADMITTANCE MATRIX

The importance of theorem 2.2 is recognised by its contribution (in the form of corollary 2.2) to the proof of the following theorem which is the key to the relationship between a network's behaviour and its polynomials.

Theorem 2.3

The X matrix, whose elements (by definition 2.1) are ratios between linkage polynomials of a network, is equal to the short-circuit admittance matrix of the network.

Proof

The theorem is first proved for the general n-port network B without internal nodes, as in figure 2.1. Such a network was discussed in the proof of theorem 2.2: its short-circuit admittances are given by equations 2.10b, and its polynomials are given by equations 2.12. From these equations it is deduced that its X matrix is given by

$$X_i^j \equiv {}^j B / B = Y_{ij} \equiv I_i / E_j$$

and $X_i^j \equiv B^j / B = Y_{jj} \equiv I_j / E_j$

Any network N equivalent to B will, by definition, have the same short-circuit matrix, and, by corollary 2.2, have the same X matrix. Thus the theorem is proved for any network.

Q.E.D.

2.5 THE UNIT GYRATOR

In a simple application of theorem 2.3 we determine the 6 polynomials that characterise the unit gyrator G, whose behaviour is described by the equation

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} \quad (2.16)$$

For simplicity, it is assumed that the common denominator polynomial of the X or admittance matrix is

$$G_{12} = 1. \quad (2.17a)$$

The numerator polynomials are then equated with the elements of the admittance matrix, i.e.

$$\begin{aligned} G_2^1 &= G_1^2 = 0, \\ {}_1^2 G &= -G_{12}^2 = 1, \\ \text{and } {}_2^1 G &= -G_{21}^1 = -1. \end{aligned} \quad (2.17b)$$

The sixth linkage polynomial is equated with the determinant of the admittance matrix, i.e.

$$G^{12} = 1. \quad (2.17c)$$

These polynomials are required in the proof of the following theorem which, incidentally, gives a further demonstration of the analysis process.

Theorem 2.4

If an n-port network A is cascaded at port c with a unit gyrator to form a new n-port network B, the linkage polynomials of the two networks A and B are related with the equations:

$$\begin{aligned} B^c &= A_c, \\ B_c &= A^c, \\ {}^c B &= {}^c A, \\ {}_c B &= -{}_c A. \end{aligned} \quad (2.18)$$

The positions of the indices of all ports other than c are not affected and are therefore not shown in these equations.

Proof

While developing the proof the port of the complete network B which corresponds to port c of the constituent network A is denoted by c' (see figure 2.2). The first and second ports of the unit gyrator G are therefore c' and c respectively, and from equations 2.17 its polynomials are

$$\begin{aligned} G_{c'/c} &= -G_{c'c}^c = G_{cc}^{c'} = G^{cc'} = 1, \\ \text{and} \quad G_c^{c'} &= G_{c'}^c = 0. \end{aligned} \quad (2.19)$$

In applying the analysis process it is noted that for every setting of external pointers there can be no pointer loops, and there are only two settings of the internal pointer to consider. Therefore, if c' is not among the transfer indices,

$$\begin{aligned} B^{c'} &= G_c^{c'} A^c + G^{cc'} A_c \\ \text{and} \quad B_{c'} &= G_{c'c}^c A^c + G_{c'}^c A_c. \end{aligned} \quad (2.20a)$$

If a pointer path of the complete network originates from, or terminates at, port c' then it must pass through the internal port c.

$$\begin{aligned} \therefore B_{c'}^{c'} &= G_{cc}^{c'} A_c^c \\ \text{and} \quad B_{c'}^{c'} &= G_{c'c}^c A_c^c. \end{aligned} \quad (2.20b)$$

All the transfer polynomials in the expansion of any one linkage polynomial are associated with the same gyrator polynomial. Therefore, after substitution for the gyrator polynomials (2.19), the equations 2.20 relate both the transfer and linkage polynomials, and prove the theorem.

Q.E.D.

In effect, cascading one port of a network with a unit gyrator interchanges the roles that its voltage and current play in the characterisation of the network's behaviour. This property has already been exploited (in section 1.2) in converting series connections of ports to equivalent parallel connections prior to the analysis of general structures of multiport networks. It is exploited further in generalising any identity relating the various polynomials, voltages, and currents of a network.

2.6 TOPOLOGICAL FORMULAE

It is established by theorem 2.3 that the transfer admittance from port j to port i , with all other ports α short-circuited, is given by the ij -th element of the X matrix, i.e.

$$\frac{I_i}{E_j} \bigg/_{E_{i,\alpha}=0} = \frac{N_{ij}^j}{N_{ij}^j} \quad (2.21)$$

If the set of ports α is divided into two sets β and γ , and all the ports γ are cascaded with unit gyrators, then, with the help of theorem 2.4, we obtain from equation 2.21 the more general identity for transfer admittances:

$$\frac{I_i}{E_j} \bigg/_{E_{i,\beta}=0, I_{\gamma}=0} = \frac{N_{ij}^{\gamma}}{N_{ij}^{\gamma}} \quad (2.22a)$$

Cascading ports i and j with unit gyrators, either at the same time or one at a time, yields the general identities for transfer impedances, voltage ratios, or current ratios:

$$\frac{E_i}{I_j} \bigg/_{E_{\beta}=0, I_{\gamma}=0} = - \frac{N_{ij}^{\gamma}}{N_{ij}^{\gamma}}, \quad (2.22b)$$

$$\frac{E_i}{E_j} \bigg/_{E_{\beta}=0, I_{\gamma}=0} = - \frac{N_{ij}^{\gamma}}{N_{ij}^{\gamma}}, \quad (2.22c)$$

$$\frac{I_i}{I_j} \bigg/_{E_{i,\beta}=0, I_{\gamma}=0} = \frac{N_{ij}^{\gamma}}{N_{ij}^{\gamma}} \quad (2.22d)$$

The general identity for a driving-point immittance is deduced in a similar way:

$$\frac{I_j}{E_j} \bigg/ E_\alpha = 0 = \frac{N_\alpha^j}{N_{j\alpha}} \quad (2.23)$$

$$\therefore \frac{I_j}{E_j} \bigg/ E_\beta = 0, I_\gamma = 0 = \frac{N_\beta^{j\gamma}}{N_{j\beta}^\gamma} \quad (2.24)$$

In so far as the polynomials are defined here as topological quantities — branch-admittance-product-sums of sets of trees — the identities 2.22 and 2.24 embrace the classical topological formulae for all network functions.

It is also deduced from these identities that the zeros of transmission from port j to port i , with ports β short-circuited and ports γ open-circuited, are zeros of the single-order linkage polynomial $N_\beta^{j\gamma}$. The zeros of the natural polynomial N_β^γ determine the natural frequencies of the network with ports β short-circuited and ports γ open-circuited, because this polynomial is the common denominator of the hybrid matrix with which the voltages E_γ and currents I_β are expressed as linear functions of the currents I_γ and voltages E_β .

A familiar, particular case of the identities arises when all ports are cascaded with a unit gyrator. All port currents become port voltages and vice versa, and the short-circuit admittance matrix (the X matrix) becomes the open-circuit impedance matrix. The transfer and driving-point impedances from port j follow from identities 2.21 and 2.23:

$$\frac{E_i}{I_j} \bigg/ I_{i,\alpha} = 0 = - \frac{N_\alpha^j}{N_{ij\alpha}} \quad (2.25)$$

and

$$\frac{E_j}{I_j} \bigg/ I_\alpha = 0 = \frac{N_j^\alpha}{N_{j\alpha}} \quad (2.26)$$

These identities confirm the well-known result that the common denominator of all the open-circuit impedances is associated with the branch-admittance-product

-sum of the trees of the network. With the application of theorem 2.1 the impedance of identity 2.25 is recognised as the ratio of the cofactor of the ji -th element of the admittance matrix to the determinant of the admittance matrix, which is, of course, the ij -th element of the inverse of the admittance matrix.

2.7 POLYNOMIAL IDENTITIES

To further illustrate the application of theorems 2.1 and 2.4 we derive two simple but important generalised polynomial identities.

Consider, first, the ratio of polynomials

$$\begin{aligned} \frac{N^{uv}}{N_{uv}} &= X_{uv}^{uv} \\ &= \frac{1}{(N_{uv})^2} \begin{vmatrix} N_v^u & N_u^v \\ N_v^u & N_u^v \end{vmatrix} \end{aligned}$$

Which, on expansion of the determinant, yields the identity

$$N^{uv} \cdot N_{uv} = N_v^u \cdot N_u^v - N_v^u \cdot N_u^v \quad (2.27)$$

This identity may be applied to a general multiport network, in which case the notation convention already adopted implies that all port indices not specifically included are assumed to be in the suffixed-subscript position. However, cascading any of these ports with unit gyrators would transfer their indices from the subscript to the superscript position. Therefore, in interpreting this (as well as any other) identity, any missing port index may be inserted in either the suffixed-subscript or superscript position, uniformly throughout the identity. Thus, an instance of this identity for a 4-port network is

$$N_1^{234} \cdot N_{123}^4 = N_{13}^{24} \cdot N_{12}^{34} - N_3^2 \cdot N_1^3 \cdot N_1^4$$

Consider, second, the ratio of polynomials

$$\begin{aligned} \frac{{}_u^v N^w}{N_{uvw}} &= X_{uw}^{vw} \\ &= \frac{1}{(N_{uvw})^2} \begin{vmatrix} {}_u^v N_w & {}_u^w N_v \\ {}_w^v N_u & N_{uv}^w \end{vmatrix} \end{aligned}$$

which, on expansion of the determinant, yields the identity

$${}_u^v N^w \cdot N_{uvw} = {}_u^v N_w \cdot N_{uv}^w - {}_w^v N_u \cdot {}_u^w N_v.$$

With regard to missing indices, this identity is open to the same interpretation as 2.27. Yet, cascading either or both of the ports u and v with unit gyrators results in even more identities; therefore, a more general form of the identity is

$${}_u^v N^w \cdot N_w = {}_u^v N_w \cdot N^w - {}_w^v N_u \cdot {}_u^w N. \quad (2.28)$$

These two little-known polynomial identities are sufficient to calculate all the natural and first-order polynomials from a given X matrix, and, as such, are the basis for an alternative analysis method discussed in chapter 3.

It will be appreciated that a very large number of new polynomial identities may be derived by considering any Laplace expansion of any minor of the X matrix and then cascading any of the ports with unit gyrators.

2.8 CONCLUSION

This chapter completes the exposition of a theory for the topological analysis of multiport networks. Its aim was to validate the approach to analysis taken in chapter 1 by building a logical bridge from the new analysis method to the well-established topological theory as it applies to the analysis of linear networks. The new approach was begun in chapter 1 with the definition of a set of polynomials as topological quantities, and with the development of an analysis process which dealt with these polynomials.

Largely as a consequence of the analysis process, all the linkage polynomials were related, by theorems 2.1 and 2.3, to minors of the short-circuit admittance matrix, and thus their relevance in describing the observable

behaviour of a network was established. The effect of cascading a port with a unit gyrator was investigated in theorem 2.4, and was exploited as a simple means of generalising all results obtained from the preceding theorems.

The main objective of the chapter was reached with the identities 2.22 and 2.24, which express the topological formulae for the various functions of a network. The other identities and properties of linkage polynomials derived throughout the chapter serve in presenting a more complete picture of the topological quantities, and demonstrate the facility with which theoretical results may be obtained. Although the proof of the four theorems, and the derivation of the polynomial properties which emerge from them are unique to this approach, the results are not new. A thorough examination of the properties of linkage polynomials has been presented elsewhere by Pike [41] .

ALTERNATIVE ANALYSIS METHODS

3.1 INTRODUCTION

The preceding chapters present a complete theory for the analysis of linear networks based on an analysis process which avoids the compilation and manipulation of matrices. Although this process has unique features which appear to give it an advantage in performing symbolic analysis of large networks, a realistic assessment of its merits cannot be made without establishing some link with other analysis methods. This chapter therefore investigates analysis methods based on the manipulation of network matrices and, where possible, attempts to interpret one method in terms of the other.

3.2 INVERSION OF NETWORK MATRICES

A common method of network analysis entails the compilation of the so-called "nodal-admittance matrix" and its subsequent inversion. If the network is regarded as a multi-port network, the nodal admittance matrix is recognised as the short-circuit admittance matrix or, if all elements have a common denominator, the X matrix of definition 2.1. The compilation of this matrix is generally straightforward and is here taken for granted; this section addresses itself to the task of inverting either the X matrix or any other hybrid matrix which relates one set of port variables (a voltage or current from each port) to its complement set.

The inversion of an $n \times n$ matrix may be achieved in n steps, each one involving the interchange of the voltage and current variables for one port, and thereby forming a new hybrid matrix. Because the steps are similar in principle, it is sufficient to detail only one such step. Further, because the formulae concerning any hybrid matrix may be obtained, with application of theorem 2.4, from similar formulae concerning the X matrix, this step will be demonstrated only with the X matrix.

Suppose that the voltage and current variables of the k -th port are interchanged, thus forming the hybrid matrix whose ij -th element is denoted by w_{ij} . If the transformation of the X matrix was achieved by cascading the k -th port

with a unit gyrator then, by theorem 2.4,

$$\begin{aligned}
 W_j^j &= N^{jk} / N^k, \\
 W_i^j &= {}^j_i N^k / N^k, \\
 W_i^k &= {}^k_i N / N^k, \\
 W_k^j &= -{}^j_k N / N^k, \\
 W_k^k &= N / N^k.
 \end{aligned} \tag{3.1}$$

and

All the polynomials except N^{jk} and ${}^j_i N^k$ are obtained directly from the X matrix.

The exceptions are calculated from polynomials of the X matrix using the polynomial identities 2.27 and 2.28, i.e.

$$\begin{aligned}
 N^{jk} &= (N^j \cdot N^k - {}^k_j N \cdot {}^j_k N) / N \\
 \text{and } {}^j_i N^k &= ({}^j_i N \cdot N^k - {}^k_i N \cdot {}^j_k N) / N.
 \end{aligned} \tag{3.2}$$

Combination of the identities 3.1 and 3.2 with the X matrix definition (2.1)

yields expressions relating elements of the two matrices, i.e.

$$\begin{aligned}
 W_j^j &= X_j^j - X_j^k \cdot X_k^j / X_k^k, \\
 W_i^j &= X_i^j - X_i^k \cdot X_k^j / X_k^k, \\
 W_i^k &= X_i^k / X_k^k, \\
 W_k^j &= -X_k^j / X_k^k, \\
 \text{and } W_k^k &= 1 / X_k^k.
 \end{aligned} \tag{3.3}$$

These expressions confirm the direct matrix inversion method of Shipley and Coleman [47].

The complete inversion process is demonstrated with a general 3-port network. The short-circuit admittance matrix (the X matrix) is given by

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \frac{1}{N_{123}} \begin{vmatrix} N_{23}^1 & {}^2_1 N_3 & {}^3_1 N_2 \\ {}^1_2 N_3 & N_{13}^2 & {}^3_2 N_1 \\ {}^1_3 N_2 & {}^2_3 N_1 & N_{12}^3 \end{vmatrix} \cdot \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix}$$

The hybrid matrix obtained by interchanging I_3 and E_3 is given by

$$\begin{bmatrix} I_1 \\ I_2 \\ E_3 \end{bmatrix} = \frac{1}{N_{12}^3} \begin{bmatrix} N_2^{13} & {}^2N_3 & {}^3N_2 \\ {}^1N_2^3 & N_1^{23} & {}^3N_1 \\ -{}^1N_3 & -{}^2N_1 & N_{123} \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ I_3 \end{bmatrix},$$

for which the new polynomials are determined by

$$N_2^{13} = (N_{23}^1 \cdot N_{12}^3 - {}^1N_2 \cdot {}^3N_2) / N_{123},$$

$$N_1^{23} = (N_{13}^2 \cdot N_{12}^3 - {}^2N_1 \cdot {}^3N_1) / N_{123},$$

$${}^2N_3^3 = ({}^2N_3 \cdot N_{12}^3 - {}^3N_1 \cdot {}^1N_2) / N_{123},$$

$${}^1N_2^3 = ({}^1N_3 \cdot N_{12}^3 - {}^3N_2 \cdot {}^2N_1) / N_{123}.$$

A similar step interchanges I_2 and E_2 , and requires the calculation of three more polynomials:

$$\begin{bmatrix} I_1 \\ E_2 \\ E_3 \end{bmatrix} = \frac{1}{N_1^{23}} \begin{bmatrix} N^{123} & {}^2N^3 & {}^3N^2 \\ -{}^1N^3 & N_{12}^3 & -{}^3N_1 \\ -{}^1N^2 & -{}^2N_1 & N_{13}^2 \end{bmatrix} \begin{bmatrix} E_1 \\ I_2 \\ I_3 \end{bmatrix},$$

$$N^{123} = (N_2^{13} \cdot N_1^{23} - {}^1N^3 \cdot {}^2N^3) / N_{12}^3,$$

$${}^3N^2 = ({}^3N_2 \cdot N_1^{23} - {}^3N_1 \cdot {}^2N^3) / N_{12}^3,$$

$${}^1N^2 = ({}^1N_2 \cdot N_1^{23} - {}^1N^3 \cdot {}^2N_1) / N_{12}^3.$$

The final step interchanges I_1 and E_1 , requiring calculation of another three polynomials, and yields the open-circuit impedance matrix:

$$\begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} = \frac{1}{N^{123}} \begin{bmatrix} N_1^{23} & -{}^2N^3 & -{}^3N^2 \\ -{}^1N^3 & N_2^{13} & -{}^3N_1 \\ -{}^1N^2 & -{}^2N_1 & N_3^{12} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix},$$

$$N_3^{12} = (N^{123} \cdot N_{13}^2 + {}^1N^2 \cdot {}^3N^2) / N_1^{23},$$

$${}^3N_1^1 = ({}^3N_1 \cdot N^{123} - {}^1N^2 \cdot {}^2N^3) / N_1^{23},$$

$${}^2N_3^1 = ({}^2N_1 \cdot N^{123} - {}^1N^3 \cdot {}^3N^2) / N_1^{23}.$$

Thus all 20 natural and single-order linkage polynomials of a 3-port network are calculated in three stages, using only the identities 2.27 and 2.28.

Practical methods of network analysis based on the above process have been developed by Downs [18, 19, 20]. With many networks two economies are exercised: the symmetry of matrices of reciprocal networks is exploited to save storage space and avoid repeated computation, and the use of a common denominator polynomial is not enforced.

In the initial compilation of the admittance matrix and, to a lessening extent, in the calculation of subsequent hybrid matrices, the use of a common denominator polynomial for all elements of the matrix would incur the introduction and subsequent cancellation of many polynomial factors. Without a common denominator polynomial the degrees of the numerator polynomials are generally smaller, but more polynomials must be manipulated and the computational algorithm is more complicated. This aspect of the method has been studied at length by Downs and need not be pursued here; the purpose of the chapter is served by the link between this matrix inversion method and the approach to network analysis which is the subject of the thesis.

3.3 ADDITION OF NETWORK MATRICES

With a conventional approach, the analysis of a network as a whole requires the inversion of a network matrix as discussed in the previous section; for a structure of separately analysed constituent networks as described in chapter 1, the same conventional approach to analysis requires the inversion of a matrix which is the sum of appropriate hybrid matrices representing the individual constituent networks. We shall develop an analysis method based on this conventional approach and draw a parallel between the computational aspects of the method with those of the topological analysis process of chapter 1.

It was demonstrated in chapter 1 that it is sufficient to consider only those structures in which the constituent networks are interconnected with their ports in parallel. In that case the appropriate hybrid matrix is the short-circuit admittance matrix and we assume that the matrix of each individual constituent network is augmented with rows and columns of zero elements, where

necessary, so that it has a row and column corresponding to every port in the structure.

Throughout the structure the corresponding port voltages of the constituent networks are equal, and the corresponding port currents add together to produce the port currents of the complete network; therefore the admittance matrix of the complete network is the sum of the admittance matrices of the constituent networks.

If each constituent network is fully analysed, there is known a numerator polynomial and a common denominator polynomial for every minor (including the determinant and individual elements) of its matrix. The goal of the analysis is to calculate the numerator polynomials and common denominator polynomial of every minor of the matrix of the complete network.

Let N denote the complete network, and A, B, C, \dots denote generic representatives of the constituent networks. Let NX refer to the X matrix of network N , let $NX_{pqr\dots}^j$ denote the column vector comprising the elements of rows p, q, r, \dots common to column j of NX , and let $NX_{pqr\dots}^{abc\dots}$ denote the minor determined by the columns a, b, c, \dots and rows p, q, r, \dots . Then the equation

$$NX = \sum_A AX$$

expresses the matrix of the complete network as the sum of the matrices of the constituent networks. A general minor of the complete network is expanded as follows:

$$\begin{aligned} NX_{pqr\dots}^{abc\dots} &= \begin{vmatrix} NX_{pqr\dots}^a & NX_{pqr\dots}^b & NX_{pqr\dots}^c & \dots \end{vmatrix} \\ &= \begin{vmatrix} \sum_A AX_{pqr\dots}^a & \sum_B BX_{pqr\dots}^b & \sum_C CX_{pqr\dots}^c & \dots \end{vmatrix} \\ &= \sum_A \begin{vmatrix} AX_{pqr\dots}^a & \sum_B BX_{pqr\dots}^b & \sum_C CX_{pqr\dots}^c & \dots \end{vmatrix} \\ &= \sum_A \sum_B \sum_C \dots \begin{vmatrix} AX_{pqr\dots}^a & BX_{pqr\dots}^b & CX_{pqr\dots}^c & \dots \end{vmatrix} \end{aligned} \quad (3.4)$$

The first observation from this expression is that every determinant in the sum of determinants corresponds to an appropriate setting of pointers. For instance, the generic determinant of this expression corresponds to the setting in which the pointer of port a is directed into the constituent network A , b into B , c into C ; etc.

A particular determinant is evaluated by a Laplace expansion with the columns contributed by the same constituent network grouped together. Thus each term in the Laplace expansion is a product of minors, one minor from each constituent network, and corresponds to a term in the expression 1.3 for the BAPS associated with a particular pointer setting.

All terms in the expansion of any minor of the complete network matrix possess a common denominator polynomial which is the product of the common denominator polynomials of all the constituent network matrices, i.e.

$$N = \prod_A A$$

where N and A denote the common denominator polynomials of the matrices of the complete network N and the generic constituent network A . To maintain this common denominator, even for terms corresponding to pointer settings which leave some constituent networks without pointers directed into them, it is convenient to conceive a minor with no columns or rows. Because it must have a value of unity, its numerator polynomial is equal to the common denominator polynomial of the matrix.

The addition of matrices is illustrated with the structure of figure 1.3.

The matrices of the three networks are denoted by

$$AX = \frac{1}{A_{123}} \begin{vmatrix} A_{23}^1 & A_3^2 & A_2^3 & 0 & 0 & 0 \\ A_3^1 & A_{13}^2 & A_1^3 & 0 & 0 & 0 \\ A_2^1 & A_1^2 & A_{12}^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix},$$

$$BX = \frac{1}{B_{2345}} \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & B_{345}^2 & B_{45}^3 & B_{35}^4 & B_{34}^5 & 0 \\ 0 & B_{45}^2 & B_{245}^3 & B_{25}^4 & B_{24}^5 & 0 \\ 0 & B_{35}^2 & B_{45}^3 & B_{235}^4 & B_{23}^5 & 0 \\ 0 & B_{34}^2 & B_{45}^3 & B_{23}^4 & B_{234}^5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

$$CX = \frac{1}{C_{456}} \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{56}^4 C_{64}^5 C_{45}^6 \\ 0 & 0 & 0 & C_{56}^4 C_{46}^5 C_{64}^5 \\ 0 & 0 & 0 & C_{65}^4 C_{56}^5 C_{45}^6 \end{vmatrix}$$

The pointer setting shown in the figure (1.3) corresponds to an expansion of the determinant

$$\begin{vmatrix} A_{23}^1 & 0 & A_{12}^3 & 0 & 0 & 0 \\ A_{32}^1 & B_{345}^2 & A_{21}^3 & B_{35}^4 & 0 & 0 \\ A_{23}^1 & B_{45}^2 & A_{12}^3 & B_{25}^4 & 0 & 0 \\ 0 & B_{35}^2 & 0 & B_{235}^4 & C_{46}^5 & C_{45}^6 \\ 0 & B_{34}^2 & 0 & B_{23}^4 & C_{46}^5 & C_{45}^6 \\ 0 & 0 & 0 & 0 & C_{45}^5 & C_{45}^6 \end{vmatrix}$$

by the three sets of columns $\{1,3\}$, $\{2,4\}$, and $\{5,6\}$. Due to the many zero elements most of the terms in the expansion are zero; for non-zero terms the minors determined by columns 1 and 3 must include row 1 and either row 2 or row 3, while the minors determined by columns 5 and 6 must include row 6 and either row 4 or row 5. Therefore, there is a total of four non-zero terms:

$$\begin{aligned} & \delta_{132456}^{132456} \cdot AX_{13}^{13} \cdot BX_{24}^{24} \cdot CX_{56}^{56} \\ & + \delta_{132546}^{132456} \cdot AX_{13}^{13} \cdot BX_{25}^{24} \cdot CX_{46}^{56} \\ & + \delta_{123456}^{132456} \cdot AX_{12}^{13} \cdot BX_{34}^{24} \cdot CX_{56}^{56} \\ & + \delta_{123546}^{132456} \cdot AX_{12}^{13} \cdot BX_{35}^{24} \cdot CX_{46}^{56} \\ & = \frac{A_{23}^1 \cdot B_{35}^2 \cdot C_{45}^6 - A_{25}^1 \cdot B_{34}^2 \cdot C_{45}^6 - A_{23}^1 \cdot B_{35}^2 \cdot C_{45}^6 + A_{25}^1 \cdot B_{34}^2 \cdot C_{45}^6}{A_{123} \cdot B_{2345} \cdot C_{456}} \end{aligned}$$

This expression agrees with the expression 1.4 obtained with the topological analysis method.

The analysis of this structure is far from complete. Fifteen more pointer settings must be considered before all the terms in the expansion of the determinant of the complete network matrix are obtained. If all ports are external, a total of $2^2 \times 3^4 = 324$ pointer settings must be considered before all the complete polynomials are obtained. Nevertheless, the example demonstrates the computational equivalence of the matrix-addition and topological analysis methods.

Although the methods are computationally equivalent, the respective algorithms which control the computations are entirely different. The topological analysis algorithm is admittedly quite complex but it only computes terms which are, in general, non-zero. On the other hand, an algorithm to evaluate determinants with appropriate Laplace expansions would be comparatively simple but, without suitable traps, would generate all terms, both zero and non-zero.

In the expansion illustrated above there are a total of $(6!)/(2!)^3 = 90$ terms, of which only 4 are non-zero. The relative merits of the methods therefore depend largely on the degree of interconnection between constituent networks, though it is worth remarking that the structure of figure 1.3, which in this respect is not atypical of electrical networks, appears to be better served by the topological analysis method.

3.4 CONCLUSION

The link between the topological approach and the more conventional matrix methods, which concludes the theoretical part of the thesis, places the new method in a broader perspective. It suggests an alternative development of the topological methods, starting from the matrix methods rather than from an investigation of topological quantities, and it provides another interpretation of the analysis process. Thus, if a network is strongly interconnected, the topological analysis algorithm, which might become preoccupied with the search for pointer loops and the collection of transfer

polynomials to form linkage polynomials, can be abandoned in favour of the direct expansion of determinants—without compromising either the diakoptic approach or the facility for achieving a fully symbolic analysis.

Due to the sparsity of non-zero elements the use of matrices, especially those whose elements are rational polynomials, in the analysis of most large, electrical networks is wasteful in terms of both storage allocation and computational effort. In a broad view, the topological approach is seen to directly exploit matrix sparsity and should prove superior to the conventional sparse-matrix techniques which gain their efficiency from a purely numerical analysis of the matrix rather than from a knowledge of the topological features of the real system.

4.1 INTRODUCTION

When it is required to predict the behaviour of any linear network, from simple passive filters to multi-stage frequency-selective amplifiers containing many feedback loops, the easiest, the most productive, and therefore usually the first endeavour is to lump any distributed components together, estimate the small-signal behaviour characteristics of all the components, and calculate the response of the network at many frequencies. Although only a starting point for more thorough investigations of network characteristics such as noise, non-linear behaviour, transient response, and sensitivity to parameter changes, a frequency response analysis provides a broad insight to the performance of a network, and presents data which can be readily corroborated with measurements on the physical realisation of the network.

That a digital computer is an invaluable tool for the analysis of large networks cannot be disputed; indeed, the importance of ac analysis is underscored by the large number of computer programs which have been developed to perform this task.

The first generation of programs, not unnaturally, used the simplest formulation of the analysis problem: the nodal-admittance matrix.

Two programs are typical: ECAP, which also performs dc and transient analysis, and has been implemented on most types of large computers; and ACNET, which is widely known due to its support on the Honeywell Mark I computer time-sharing service. However, these programs are inefficient in their use of both computer time and computer store-space. For time-shared computers which have a limited space available in their core-store this aspect is critical and often precludes the analysis of large networks — especially those for which a computer analysis would be most valuable.

Beside a program's computational efficiency and accuracy, another aspect which influences its popularity is the form in which data describing the network must be presented to it. Most programs will accept, in a uniform and simple manner, networks comprising only R, L and C components, but few of the general-purpose programs will recognise a more complete set of network components. The onus is then on the program user to model devices such as transistors and transformers with, for example, only R, L, C and voltage-dependent current-source elements.

It is often practical to include extra routines in a program to either perform this modelling directly, or otherwise handle an enlarged set of basic elements which might include all types of dependent sources and other two-port devices. However, for some analysis methods there still remains a fundamental difficulty in handling degenerate devices such as ideal isolating transformers and operational amplifiers.

This chapter first surveys the known methods for ac analysis of linear networks, examines briefly the methods for calculating network functions, and discusses the desirability of network tearing.

The approach to network tearing introduced in chapter 1 is illustrated by its application to a simple 2-port amplifier circuit, and from this exercise there emerges the concept of a structure graph, used to describe any 2-port network. It is in attempting to analyse structure graphs in the most efficient manner that the concept of algebraic reduction arises, and this, too, is illustrated with reference to the amplifier circuit.

A simpler notation for the polynomials of 2-port networks is introduced, and the general topological analysis algorithm of chapter 1 is recast in a form better suited to the analysis of structure graphs.

The remainder of the chapter discusses implementation of the analysis method. One major problem is the loss of numerical accuracy due to truncation errors in the polynomial coefficients, and this is tackled with the introduction of a novel, computationally-simple frequency transformation.

The final section discusses various forms of polynomial representation and their roles in the symbolic analysis of lumped parameter networks, in the frequency-by-frequency analysis of distributed-parameter networks, and in parameter-sensitivity analyses.

4.2 ANALYSIS METHODS

If the response is required at a large number of frequencies it is desirable to use Laplace transform techniques and first calculate the transfer functions as ratios of polynomials in the complex-frequency variable s . From these the response can be calculated quite simply and hence more rapidly than the point-by-point methods which repeat the whole analysis at each frequency. However, in the past, the use of network polynomials has lost favour [5] and point-by-point methods have been improved to the extent that, after the first analysis, much of the effort required to invert a matrix is avoided.

Notable among the point-by-point methods is that of Pinel and Blostein[42] embodied in a program called KRON, which first compiles the nodal admittance matrix of a tree of the network and calculates the Laplace transform of its inverse. At each frequency it is only necessary to evaluate this inverse and adjust it, with a routine developed by Branin[4] as each link and controlled source is added to the tree to complete the network. Branin's method, based on the work of Kron[29] by which the solution matrix is simply updated rather than calculated anew when a link is added, is also suited to the calculation of sensitivities with respect to component changes.

Rational polynomials have not been favoured in the analysis of large networks for two reasons: their coefficients are difficult to calculate, and the response at some frequencies can be intolerably sensitive to errors in the coefficients. However, these two difficulties are largely overcome with the methods demonstrated in this thesis (chapter 6), and the many advantages of the classical approach to analysis are more easily realised.

Besides allowing rapid calculation of frequency response, network polynomials yield other information on the behaviour of a network. Provided that suitable polynomial root-finding routines are available, the zeros and poles can be calculated to determine a network's stability and natural frequencies. Transient response can be calculated, either by finding the poles and their residues and inverting the Laplace transform in the conventional manner, or by working directly from the polynomial coefficients and thus avoiding the difficulties associated with multiple roots [15,31]. Further, by analytically differentiating the network-function polynomials, the group delay can be calculated more accurately than by numerically differentiating the phase response with respect to frequency.

4.3 CALCULATIONS OF NETWORK FUNCTIONS

Methods for calculating network functions fall into three essentially different classes.

The first class contains those methods which manipulate polynomials. Starting with the simpler polynomials representing individual components, the polynomials are combined as the components are interconnected until the polynomials representing the complete network are obtained. An early method of this type was that of Bashkow[2], although it can only analyse ladder structures with passive components it achieved widespread use. A more versatile method described by

Riordan [44] is intended to handle active components, but while its forte is cascaded 2-port networks, all but the simplest feedback paths must be approximated. It is a general characteristic of methods of this type, however, that they provide the most rapid analysis of those networks restricted to a certain structural type.

The greatest danger which must be avoided with methods in this class is the generation of spurious common factors in the numerators and denominators of network functions. Apart from occupying valuable store space in the computer, the factors are difficult to recognise and cancel, because in practice a slight variation in polynomial coefficients due to round-off error can significantly alter their values. This is the reason why large matrices with rational-polynomial elements cannot be inverted with the conventional methods such as Gaussian elimination.

Probably the best technique for inverting an admittance matrix of rational polynomials has been developed into a practical method by Downs [18]. The inversion of an $n \times n$ matrix is accomplished in n similar stages, each stage resulting in a different hybrid matrix of the network. The degrees of polynomials are kept within manageable bounds by dividing out the predicted common factors at every stage. Although the process is efficient for matrices of moderate size it is reported by Neill [40] that for larger matrices a severe loss of significant figures occurs during the division of polynomials and the increase in computing time makes the process uneconomic.

The second class contains those methods based on topological formulae for network functions [13,14,34,48]. Typical of this approach is a program written by Calahan [11].

As a class these methods are notable for their ability to handle component values either numerically or symbolically and to thus establish the functional dependence of the network functions of any set of network parameters. The modelling of ideal isolating transformers, mutual inductances, gyrators, and active components is difficult, and the methods vary from one another mainly in the techniques adopted to overcome these difficulties. Common to all these methods is the severe limitation imposed on network size by the need to generate all the trees of a network graph. The number of trees tends to grow exponentially with the number of nodes and branches, so that although some improvement may be made in the algorithms to generate trees and calculate their branch-admittance

products, the increase in the size of networks that can be analysed is not likely to be significant.

Also in this class are the methods based on the signal-flow graph techniques first developed by Mason [33]. Their applications have become widely known through documentation of the several versions of the program NASAP [52]. Signal-flow graphs have an advantage in that they represent the mathematical relationship between the system variables rather than the physical interconnections which are represented by the network graph. Consequently the handling of complex or degenerate devices is relatively straightforward, but this facility is obtained at the expense of approximately doubling the size of the graph to be analysed. The transfer functions are found with the application of topological formulae requiring enumerative schemes for loops in the graph, and these methods suffer the same limitations on network size as the other topological methods.

The third class includes those methods which are based on the concept of state-variables and which characterise a network by its A matrix [1]. The chief difficulty concerns the selection of a suitable vector of state variables on which to base the A matrix of the general network, and it is in this respect that many methods differ [10,30]. The usual analysis procedure is to calculate the eigenvalues and eigenvectors of A by an iterative method; the eigenvalues are the natural frequencies of the network and hence the poles of the transfer functions, and the eigenvectors determine the residues of the poles in all the transfer functions. An alternative approach finds the transfer-function zeros as the eigenvalues of related network matrices [46].

These methods, especially those which, like the program CORNAP [43], employ the Q-R transformation of Francis [24,51], have been preferred for calculating the natural frequencies of a network because, in working with the A matrix, the iterated loop links the frequencies more directly to the network parameters than do the methods whose iterations work with the coefficients of the characteristic polynomial. It is well known that round-off errors in the coefficients of a polynomial may strongly influence the accuracy of its roots.

Related to the third class, with regard to their calculation of network poles and zeros, are those methods which apply the Müller [39] routine directly to the determinant and minor of the nodal admittance matrix. Matrix inversion is not required, but the determinant must be evaluated at one complex frequency for each of many iterative steps toward successive natural frequencies. When used with conventional Gaussian elimination, as in the program LISA [17],

computational efficiency is sacrificed for the inherent superior accuracy; but when combined with a determinant evaluation procedure which uses the efficient row- and column-ordering scheme of a sparse matrix technique [3] as in program FRANK and its successor SLIC [26] it results in one of the most efficient and accurate tools for the calculation of natural frequencies that is currently available.

However, the use of an iterative routine, with its attendant problems of control, difficulty with pathological cases, and questionable accuracy, is not attractive for the calculation of steady-state frequency response.

Beside the iterative methods, there are direct methods for calculating the coefficients of the characteristic polynomial of A , and, at the same time, calculating the adjoint matrix of $sI - A$ from which the numerator polynomials of all the transfer functions may be obtained [38]. A reliable method of this type is a modification by Faddeev [22] of Leverrier's method. But the storage requirements and tremendous computational effort involved with these methods soon become prohibitive. If A is an $n \times n$ matrix, the number of coefficients in the polynomials of the adjoint matrix is proportional to n^3 , and the number of arithmetic operations needed to calculate them is proportional to n^4 .

4.4 NETWORK TEARING

If both computer store-space and computer time are to be used efficiently to permit the analysis of very large networks, it must be possible to tear a network apart into sub-networks, analyse them separately, and somehow combine the results of their separate analyses to achieve an analysis of the whole network.

The practicability of this procedure depends entirely on the existence of a suitable method for combining the analyses, or solutions, of separate networks. The powerful techniques developed by Kron [29] for interconnecting the solutions of any physical systems, employ matrix methods and require the inversion of matrices; it is therefore not considered feasible to adapt these methods directly to the rational polynomials of the Laplace transform technique. As alternatives to the tearing procedure, which gains its computational effectiveness by exploiting the sparsity of non-zero elements in network matrices, there are techniques involving schemes for matrix decomposition [49] but they too, for the same reason, are not amenable to Laplace transform techniques.

Before demonstrating the tearing procedure introduced in chapter 1, some of its expected advantages are reviewed. An advantage of any tearing procedure is that if a topological method is used to analyse the subnetworks then the total number of trees that must be found in all the subnetworks is considerably less than the number of trees of the complete network.

When the effect of a network modification is required, only the analysis of those subnetworks containing the modified components need be repeated. If the modification involves a change in the admittance of a component, it can be effected by the addition of a similar component, with either a positive or negative admittance, in parallel with the original network, and the additional analysis effort is concerned only with this connection. This feature is particularly useful in the analysis facility of a computer program for on-line network design.

A tearing procedure can take advantage of a situation in which a network has some identical subnetworks, for a standard subnetwork can be analysed once, and its solution stored and used later in interconnections with any other networks. A network can also be torn apart to separate those subnetworks which can best be analysed by different methods; for example, devices with three or more terminals need not be modelled with networks of two-terminal devices, as they must be for the topological methods, but may be represented directly as a subnetwork.

Probably the most significant attribute of a tearing procedure is that by applying it successively to subnetworks, and their subnetworks in turn, until each separate network contains a single component, it becomes a method of analysis in itself. Characterisation of single components in a manner suited to the interconnection of solutions is elementary, hence all the effort of analysis is associated with the interconnection of solutions.

To eliminate the difficulty in representing isolating transformers and mutually coupled coils, the tearing procedure of chapter 1 requires that a network be represented as a structure of multiport networks and that the ports may only be connected in series or in parallel. When connected together, a set of ports of different networks is considered to be one port of the structure, and will be identified here by a number followed by the symbol *s* or *p* to indicate the type of connection.

The generality of this approach to network tearing is assured by the fact that any multiport network composed of 2-terminal, 3-terminal and 2-port devices can be represented as a valid structure of 2-port networks. The representation for strongly interconnected networks, although difficult to conceive, can be generated systematically, while the representation for many practical networks is readily apparent, as will be appreciated with the following example.

4.5 STRUCTURE GRAPHS

The desired approach to network analysis, outlined above, is demonstrated with the two-stage transistor amplifier with current feedback that is shown in figure 4.1.

The complete network is redrawn in figure 4.2 to highlight its ac-signal behaviour, before being represented as a structure of 2-port constituent networks as in figure 4.3.

The first point to notice is that to accomplish the representation, a trivial 2-port network, N_9 , without components, has been introduced between ports 2s and 3p, whereas all the other 2-ports contain a single component. Because the complete network and all the constituent networks each have two ports it is convenient to adopt the sign convention for currents which is shown in figure 4.4; the arrow directed from the first port to the second port indicates which port is the second port and thus determines the direction of positive currents.

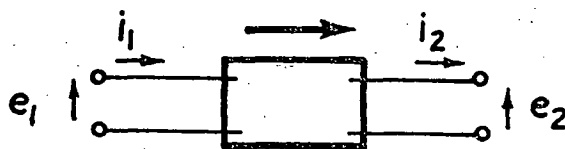


Figure 4.4 Sign convention for voltages and currents of a 2-port network.

The representation of the amplifier by figure 4.3 is valid because

- 1) the networks connected at each port share either a common voltage (p) or a common current (s), and
- 2) each port could be isolated by ideal transformers without affecting the behaviour of the network.

The structure of the complete network is represented diagrammatically by a graph whose nodes correspond to ports of the structure and whose branches correspond to the constituent 2-port networks, as in figure 4.5. The first and

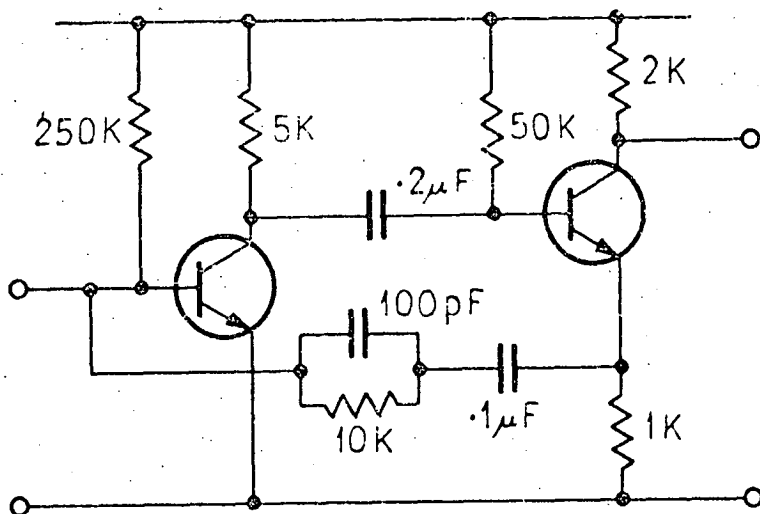


Figure 4.1 Circuit diagram of two-stage transistor amplifier.

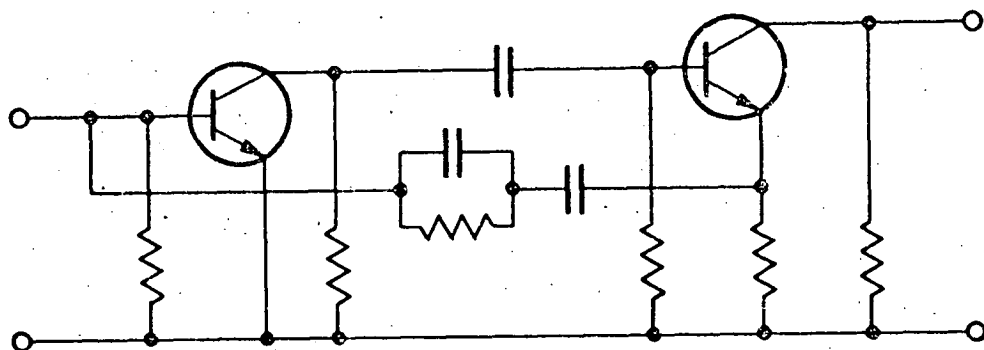


Figure 4.2 Network diagram highlighting small-signal behaviour.

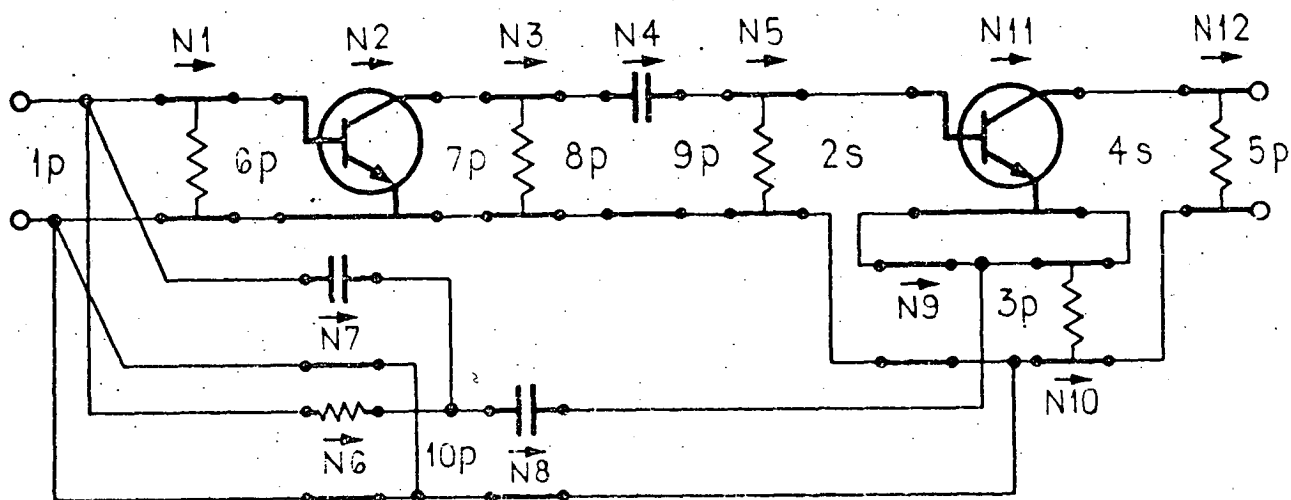


Figure 4.3 Complete network as a structure of 2-port networks.

second ports of the complete network are identified by connecting between them a closing branch which is shown dashed and labelled B0 in the figure. It should be remembered that the arrows identify the first and second ports of the branches, as they do in figure 4.3, but do not indicate the direction of any signal flow.

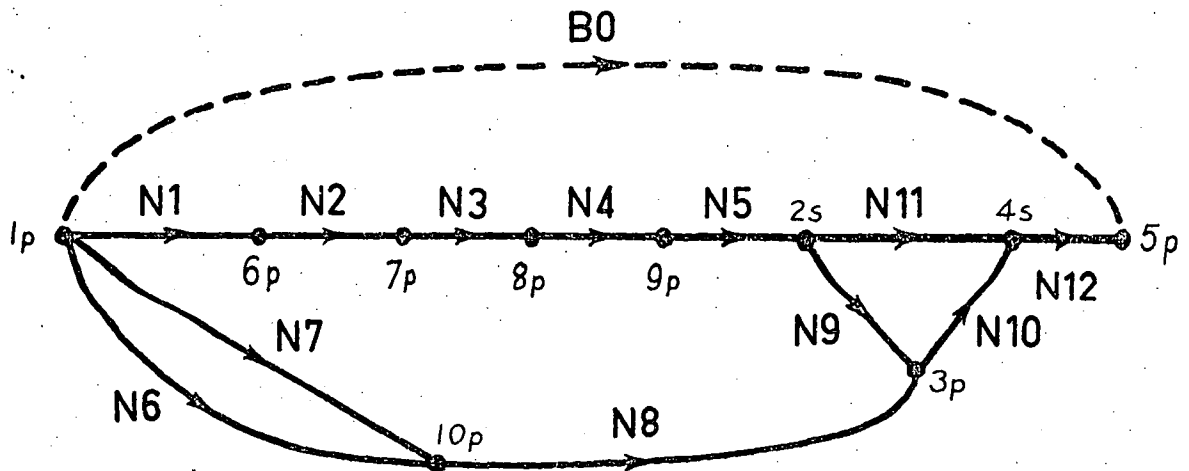


Figure 4.5 Structure graph

The graph of 2-port networks, here called the structure graph, is peculiar to this approach and shows clearly the main signal paths of the network. For the amplifier network it provides a rigorous procedure for representing the main signal path through the amplifier and clearly identifies the two major forms of feedback; the voltage feedback with the emitter resistance of the second stage, and the current feedback around both stages.

If at this stage we assume that a computer is programmed to interconnect the solutions of any number of constituent 2-port networks connected in any structure, and that the solutions or analyses of the constituent networks are available to the computer, then all that remains is to specify how the networks are interconnected. This could be accomplished, for example, simply by listing all branches of the structure graph against the pairs of nodes between which they are connected, as follows:

B0: 1p 5p,	N1: 1p 6p,	N2: 6p 7p,
N3: 7p 8p,	N4: 8p 9p,	N5: 9p 2s,
N11: 2s 4s,	N12: 4s 5p,	N10: 3p 4s,
N9: 2s 3p,	N8: 10p 3p,	N6: 1p 10p,
N7: 1p 10p.		

However, if the solution of all the constituent networks containing a single component were interconnected at one time, as might be implied by this one list of branches, there is no reason to believe that this method of analysis would

by any more efficient than the conventional methods which combine all the components at one time. To appreciate the value of a tearing procedure in terms of computational efficiency, the constituent networks should be combined in substructures to form larger constituent networks, which in turn are combined in new structures, until, after many such stages, the complete network is formed.

4.6 ALGEBRAIC REDUCTION

It is apparent with the amplifier network, as it is with most networks, that the most common (and simplest) substructure combines two 2-port networks to form a third 2-port network. For example, networks N6 and N7 are connected in parallel-parallel, and the resulting network is cascaded with N8. Because the result after each combination is another 2-port network, these operations may be specified by an algebraic expression such as

$$B1 = (N6 \text{ pp } N7) \text{ c } N8 \quad (4.1)$$

where pp and c represent the interconnecting operations of parallel-parallel and cascade respectively. To complete the set, there are three more operations, ps, sp, and ss, representing the respective interconnections of parallel-series, series-parallel, and series-series.

Substructures in another part of the network could be similarly specified by the expression

$$B2 = ((N1 \text{ c } N2) \text{ c } (N3 \text{ c } N4)) \text{ c } N5. \quad (4.2)$$

The order in which the cascade operations are performed does not affect the result, so the brackets in this expression could be deleted. The above expressions are assigned to numbered blocks which serve to identify them either in a subsequent algebraic expression or in a list of branches of a structure graph.

After the constituent networks have been combined according to the expressions 4.1 and 4.2, the complete network is said to be algebraically reduced and is represented by the structure shown in figures 4.6 and 4.7. Although it would be better, at this stage, to analyse the subnetwork consisting of networks N9, N10, N11, B1 and B2, and algebraically cascade the result with network N12, for the sake of brevity all these constituent networks are combined at one time, as specified by the list of branches:

BO: 1p 5p,	B1: 1p 3p,	B2: 1p 2s,
N9: 2s 3p,	N10: 3p 4s,	N11: 2s 4s,
N12: 4s 5p.		

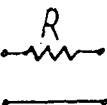





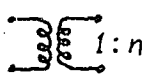
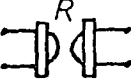
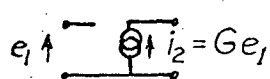



MNEMONIC	NETWORK	POLYNOMIALS					
		1	2	3	4	5	6
SR		1	R	0	1	1	1
TR		1	0	1/R	1	1	1
SC		s	1/C	0	s	s	s
TC		1	0	Cs	1	1	1
SL		1	Ls	0	1	1	1
TL		s	0	1/L	s	s	s
F		1/n	0	0	n	1	1
G		0	R	1/R	0	-1	1
H		0	1	0	0	G	0
U		1	0	0	1	1	1
X		1	0	0	1	-1	-1
I		1	0	0	-1	1	-1

Table 4.1 Basic 2-port networks and their polynomials.

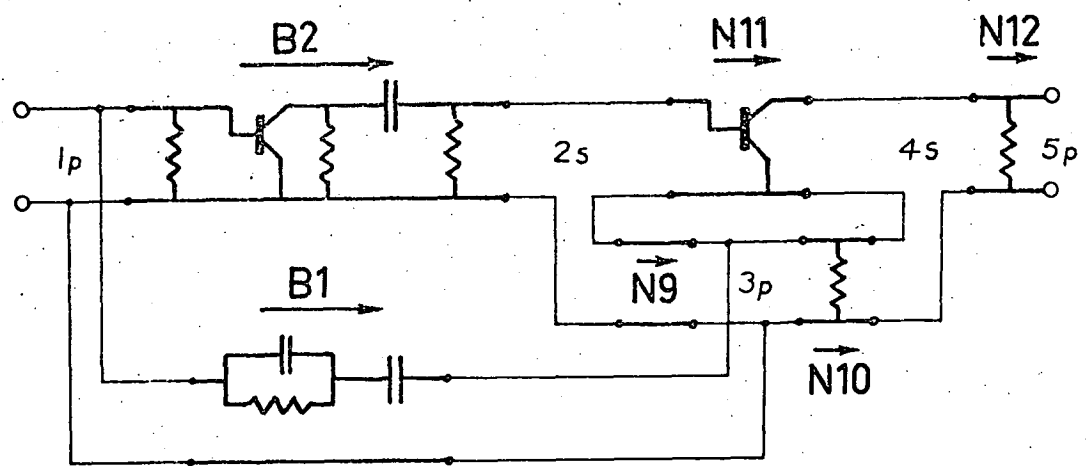


Figure 4.6 Complete network as an algebraically reduced structure.

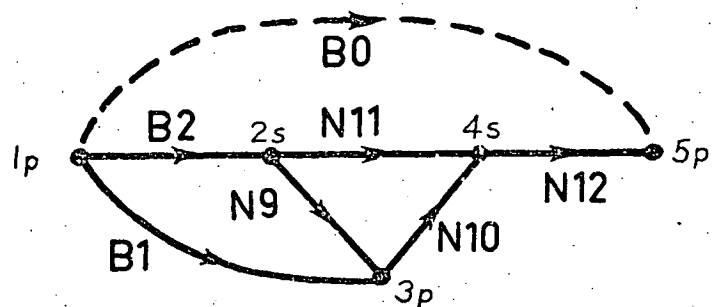


Figure 4.7 Algebraically reduced structure graph.

The computational effort required to analyse a structure of 2-port networks is roughly proportional to the product of the degrees * of the nodes in the structure graph with the closing branch included. For a series branch structure corresponding to the cascade connection of two 2-port networks the node-degree product is 8; for the simple structure with two branches in parallel the node-degree product is 9, and for the structure of figure 4.7 the node-degree product is 162. Hence the total effort required to analyse the two-stage amplifier by analysing separately the seven substructures, specified by the two algebraic network expressions and the list of branches, is proportional to the sum of the node-degree products: $9 + 5 \times 8 + 162 = 211$. This is considerably less than the effort required to analyse the structure of figure 4.5 as a whole, which has a node-degree product of 10,368.

An algebraic network expression is, in general, a concise specification of combinations of 2-port networks which correspond, in the structure graph, to combinations of pairs of branches which are either in "series" or in "parallel". It is interesting to note that algebraic reduction as defined above is therefore analogous to the combination of 2-terminal impedances in series or parallel which simplifies the analysis of complex structures of such devices. Just as the combination of two impedances, either in series or in parallel, presents a very

* the degree of a node is the number of branches connected to it.

simple problem of analysis and is used whenever possible to simplify a more difficult analysis problem, so does the combination of two branches in the structure graph present a comparatively simple problem with the interconnection of their corresponding 2-port networks, and algebraic reduction should therefore be used wherever possible.

The networks with a single component could be specified by both a mnemonic code for the type of the network, and a number for the component value. For example, if T denotes a 2-port network with a single 2-terminal device placed in parallel with both ports, S denotes a 2-port network with a single 2-terminal device in series with both ports, and R and C denote resistance and capacitance respectively, the expressions 4.1 and 4.2 might be rewritten for a computer as

$$B1 = (SR10E3 \text{ pp } SC10OE-12) \text{ c } SC.1E-6 \quad (4.3)$$

and

$$B2 = TR25OE3 \text{ c } N2 \text{ c } TR5E3 \text{ c } SC.2E-6 \text{ c } TR5OE3 \quad (4.4)$$

These expressions illustrate a form of network description which is easy to prepare and can be interpreted by a programmed computer. In comparison with the conventional methods of network description, using lists of branches and nodes, it has the advantage that it is easy to interpret mentally and mistakes are more likely to be recognised. A formal specification of this language for network description is given in section 5.5.

4.7 POLYNOMIALS OF 2-PORT NETWORKS

From chapter 1, the linkage polynomials of a 2-port network are given by

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \frac{1}{N_{12}} \begin{bmatrix} N_2^1 & N_1^2 \\ N_1^1 & N_2^2 \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix}$$

With all networks limited to two ports the sign convention for currents is changed (see figure 4.4) and the notation for polynomials is simplified. The new polynomials are given by

$$\begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \frac{1}{N2} \begin{bmatrix} N4 & -N6 \\ N5 & -N1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \quad (4.5)$$

and

$$\begin{bmatrix} e_1 \\ i_1 \end{bmatrix} = \frac{1}{N5} \begin{bmatrix} N1 & N2 \\ N3 & N4 \end{bmatrix} \begin{bmatrix} e_2 \\ i_2 \end{bmatrix} \quad (4.6)$$

The new and old polynomials are therefore related as follows:

$$\begin{aligned}
 N_1 &= N_{11}^2 \\
 N_2 &= N_{12} \\
 N_3 &= N_{12}^2 \\
 N_4 &= N_{22}^1 \\
 N_5 &= N_{21}^1 = -N_{22}^1 \\
 N_6 &= N_{12}^2 = -N_{11}^2
 \end{aligned}$$

The polynomial identity 2.27 becomes

$$N_3 \cdot N_2 = N_4 \cdot N_1 - N_5 \cdot N_6 \quad (4.7)$$

4.8 ANALYSIS OF 2-PORT NETWORK STRUCTURES

For structures containing only pairs of 2-port networks, as occur in algebraic reduction, a non-topological method of analysis using the set of six polynomials has been known for some time. Expressions relating the polynomials of the complete network to the polynomials of its two constituent networks have been published by Mathaei [35].

If networks A and B are cascaded to produce a network N, the polynomial relationships are:

$$\begin{aligned}
 N_1 &= A_1 \cdot B_1 + A_2 \cdot B_3 \\
 N_2 &= A_1 \cdot B_2 + A_2 \cdot B_4 \\
 N_3 &= A_3 \cdot B_1 + A_4 \cdot B_3 \\
 N_4 &= A_3 \cdot B_2 + A_4 \cdot B_4 \\
 N_5 &= A_5 \cdot B_5 \\
 N_6 &= A_6 \cdot B_6
 \end{aligned} \quad (4.8)$$

If networks A and B are connected in parallel-parallel, the polynomial relationships are:

$$\begin{aligned}
 N_1 &= A_1 \cdot B_2 + A_2 \cdot B_1 \\
 N_2 &= A_2 \cdot B_2 \\
 N_3 &= A_2 \cdot B_3 + A_3 \cdot B_2 + A_1 \cdot B_4 - A_6 \cdot B_5 + A_4 \cdot B_1 - A_5 \cdot B_6 \\
 N_4 &= A_4 \cdot B_2 + A_2 \cdot B_4 \\
 N_5 &= A_5 \cdot B_2 + A_2 \cdot B_5 \\
 N_6 &= A_6 \cdot B_2 + A_2 \cdot B_6
 \end{aligned} \quad (4.9)$$

These two sets of expressions can be derived with the topological analysis algorithm, or by multiplying or adding the appropriate parameter matrices defined by the equations 4.5 and 4.6, and by using the identity 4.7. Algebraic reduction occurs so frequently in the analysis of networks that it pays to program these expressions directly, rather than implicitly with a topological analysis algorithm. They appear in an algorithm to evaluate network expressions which is described in section 5.5.3.

If the topological analysis algorithm has to deal only with 2-port networks, polynomial products contributing to closing polynomials can be calculated according to the equation 1.7 rather than the equation 1.3. The complete algorithm is then summarised as follows:

- a) To the set of real constituent networks add the closing network.
- b) With each port associate a pointer which may be directed into any attached network (real or closing).
- c) Set the polynomials of the closing network to zero. These polynomials will be employed as accumulating sums of products of polynomials of the real constituent networks.
- d) Generate every possible setting of pointers once and only once. For every setting take the following steps:
 - i) Search for all possible pointer loops.
 - ii) Determine the polynomial factors expressed in equation 1.7. This expression will consist of a term which includes among its factors a natural polynomial of the closing network, and, if the closing network is traversed by a pointer loop, another term which includes among its factors a transfer polynomial of the closing network. Evaluate both terms using the polynomials of the real constituent networks and add the resulting products to the appropriate polynomials of the closing network.
- e) At the completion of step (d) calculate the polynomials of the complete network N from the accumulated polynomials of the closing network \bar{N} using the equations

$$N1 = \bar{N}4$$

$$N2 = \bar{N}3$$

$$N3 = \bar{N}2$$

$$N4 = \bar{N}1$$

$$N5 = -\bar{N}6$$

$$N6 = -\bar{N}5$$

(4.10)

The latter equations were derived from the equations 1.6.

Potentially the most difficult and time-consuming step is (d)(i), but in many respects this step is similar to the task of finding the trees of a graph. Indeed, an efficient algorithm for generating the trees of a graph [7] has been described using the same concept of pointers. The principal difference is that in the generation of trees a pointer setting which includes a pointer loop contributes zero to a sum of branch admittance products and is entirely disregarded; in the analysis of a structure graph, however, a pointer loop contributes a difference between two polynomial products which may not necessarily be zero. Although more calculation is required with each pointer setting, because polynomials rather than single numerical quantities must be manipulated, the effort required to generate pointer settings and detect pointer loops of a structure graph is the same as the effort required to generate the trees of a graph of similar complexity.

To illustrate the above algorithm it will be used to derive the expressions 4.9 resulting from the analysis of two networks connected in parallel:

- a) The closing network \bar{N} is included in the structure graph with the constituent networks A and B.
- b) Ports 1 and 2 both have pointers which can be directed into all three networks.
- c) The polynomials $\bar{N}1, \bar{N}2, \dots, \bar{N}6$ of the closing network are set to zero.
- d) All the pointer settings are generated as shown in figure 4.8.

For the first pointer setting there can be no pointer loop and its polynomial expression is $\bar{N}2.A3.B2$. Before proceeding to the next pointer setting, $A3.B2$ is added to the current value of $\bar{N}2$ (initially set to zero in (c)). With the second pointer setting a pointer loop can be drawn and the expression in equation 1.7 becomes $\bar{N}2.(A4.B1 - A5.B6)$. Thus two more terms are added to $\bar{N}2$.

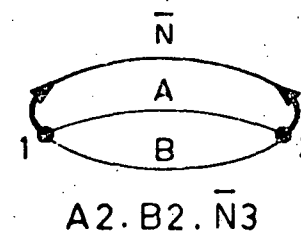
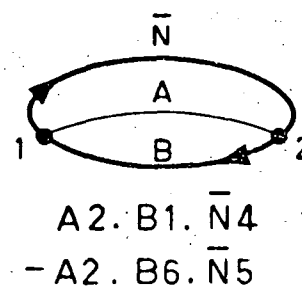
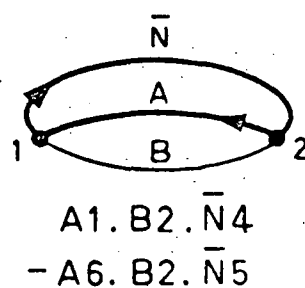
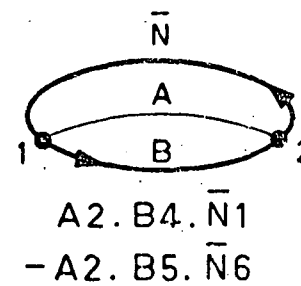
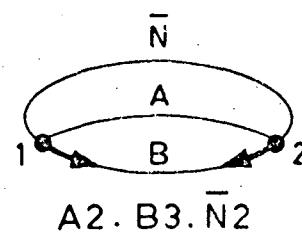
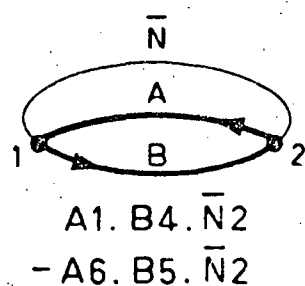
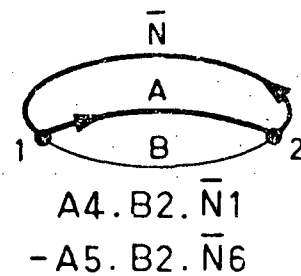
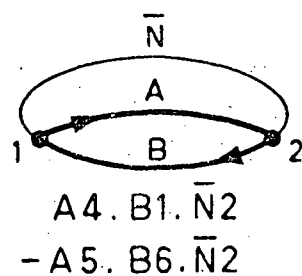
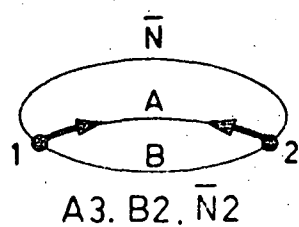


Figure 4.8 Nine pointer settings for a paralleled pair of 2-port networks A and B.

d) (cont.)

When this step is completed the polynomials of the closing network are represented by the expressions:

$$\bar{N}_1 = A_4.B_2 + A_2.B_4$$

$$\bar{N}_2 = A_3.B_2 + A_4.B_1 - A_5.B_6 + A_1.B_4 - A_6.B_5 + A_2.B_3$$

$$\bar{N}_3 = A_2.B_2$$

$$\bar{N}_4 = A_1.B_2 + A_2.B_1$$

$$\bar{N}_5 = -A_6.B_2 - A_2.B_6$$

$$\bar{N}_6 = -A_5.B_2 - A_2.B_5$$

e) The expressions 4.9 follow after using equations 4.10 to relate the polynomials of the complete network to those of the closing network.

4.9 IMPLEMENTING THE ANALYSIS METHOD

After the tearing procedure has been applied to a network as many times as are possible, all the constituent networks contain at most one component and are called basic networks. The analysis of a basic network is generally straightforward, requiring only the assembly of its six polynomials. Because there are only a few different types of basic networks, and they occur frequently in many complete networks, it is convenient to construct a table of their polynomials, as in table 4.1. It is an interesting but irrelevant exercise to find a minimum set of basic networks from which all other basic networks, and hence all networks, can be formed by interconnecting their solutions.

With reference to table 4.1 it should be noted that it is not necessary for a 2-port network to have any particular admittance, impedance, or hybrid matrix representation to be meaningfully characterised by a set of six polynomials. Indeed, an ideal operational amplifier has no such matrix representation, although it does have a forward transmission matrix as in equation (5), and is characterised by the polynomials 0, 0, 0, 0, 1, 0. All the polynomials of a network may be multiplied by some common factor without affecting the characterisation.

The polynomials of a transistor specified by its h parameters are obtained by expressing the parameters as ratios with the common denominator of one, and identifying the polynomials with appropriate numerators and denominators. The six polynomials, in order, are

$$h_{11} h_{22} - h_{12} h_{21}, h_{11}, h_{22}, 1, -h_{21}, h_{12}.$$

Alternatively, if a transistor is modelled by, say, a hybrid-pi network then it is analysed, in the same way as any other network, by applying the tearing procedure until all the passive components and the controlled source belong to separate basic networks.

Most passive networks such as filters with bridged or paralleled T, ladder, or lattice structures ("structures" used here in the conventional sense) have a very simple structure graph, irrespective of the number of components, and can be fully described by a single algebraic network expression. Networks with this property (the author ventures to call them algebraic networks) are analysed with a rapidity which matches that of the methods in the class mentioned first in section 4.3. Even multistage amplifiers with many feedback loops have a comparatively simple structure after algebraic reduction.

The effort required to analyse a structure of 2-port networks is of the same order of magnitude as that required to analyse a similar structure of 2-terminal devices with a topological method. Comparison of a graph of 2-terminal devices representing the network of figure 4.2 with the algebraically reduced structure graph (figure 4.7) of the same network would therefore illustrate one of the advantages of this method over existing topological methods. In terms of computing time, this method imposes a penalty commensurate with a network's complexity (indicated by the node-degree product of its algebraically reduced structure graph) as distinct from a network's size (indicated by the number of components).

Besides judging a computer program by its computing speed, another important consideration is its computer-store requirement. Programs implementing the various features of this approach have been written in ALGOL and run on an Elliott 503 computer. A basic program, which reads data in the form of algebraic network expressions and branch lists, assembles polynomials of basic networks, and calculates the polynomials of a complete network, occupies approximately 4000 locations of 39-bit words. Over 2000 words are left in the main store for work space, which is consumed at the rate of approximately 16 words for every reactive component.

This last fact illustrates another advantage of the tearing procedure: individual components can be assimilated by an algebraic network structure as they are read from the input data file, and only the reactive components consume extra store space because they may increase the degrees of the network polynomials.

Thus the size of networks that can be analysed depends mainly on the number of reactive components; large passive filter complexes with 100 reactive components and over 200 components altogether have been successfully analysed with less than 2000 words of store space. This compares favourably with matrix methods of analysis which consume store space in proportion to the square of the number of nodes or the number of meshes.

Another version of the program * written in FORTRAN has been implemented on a GE 265 time-shared computer which has an available core store for both program and data of less than 5300 words of 20 bits. Although this program will only accept network data in the form of algebraic network expressions, it will analyse networks whose polynomials have degrees of up to 40.

Implementation of the full algorithm for analysing general structures of 2-port networks is not without its problems. At its simplest each pointer setting is treated independently and each polynomial product is formed from the polynomials of the constituent networks. But at the expense of extra primary store, both the speed and accuracy of the algorithm can be improved. First, with a suitable method for generating pointer settings as suggested in section 1.4.2, intermediate polynomial products can be transferred from one pointer setting to the next. Second, in evaluating the expression of equation 1.7, it may be decided before polynomial multiplication begins that, for the j -th pointer loop, $n_j = t_j$, in which case the expression is exactly zero. This decision not only saves time, but—and this is more important—it eliminates a potential source of round-off error which would be introduced if the difference between two large and nominally equal polynomials was calculated.

An optimised algorithm for topological analysis is described in section 5.6.

* This program, called ALENA (Algebraic Linear Electrical Network Analysis), has become widely available through its support by Honeywell as a library program on their national computer time-sharing system.

4.10 NUMERICAL ACCURACY

Use of Laplace transform techniques in a general purpose method of network analysis cannot be contemplated unless the so-called "accuracy" problem is overcome. This problem arises in the evaluation of a polynomial at a particular frequency and occurs when the exact value of the polynomial is many orders of magnitude less than its constant coefficient, for then the difference between two nearly-equal numbers is calculated, and their round-off errors, accumulated during the analysis process, are magnified.

Clearly, the evaluation of a polynomial will be most accurate at the frequency for which the polynomial equals its constant coefficient. Normally this occurs at the zero frequency, $s = 0$, but if the frequency variable s is suitably scaled and the transformation $s^2 = t^2 - 1$ is introduced then the polynomial evaluation will be most accurate at any desired frequency, given by $t^2 = 0$ or $s = j$.

The frequency transformation need only be applied to s^2 and not to s so that every polynomial is represented by the coefficients of the terms t^0 , st^0 , t^2 , st^2 , t^4 , st^4 , etc. But it must be applied throughout the analysis process or the desired information at $s = j$ may be irrecoverably lost among the round-off errors. The transformation is easy to implement in the analysis process for it only requires one modification to the routine for multiplying polynomials. When a product of terms is formed it may be split into two terms as shown:

$$\begin{aligned} ast^i \cdot bst^j &= abs^2 t^{i+j} \\ &= abt^{i+j+2} - abt^{i+j} \end{aligned}$$

The effectiveness of this frequency transformation was investigated in the analysis of an actual filter complex consisting of two band-stop filters and one band-pass filter (block B7 of figure 6.1) containing, in all, 50 reactive components. Between certain ports the complete network was known to exhibit a pronounced band-pass behaviour, but when analysed without a frequency transformation the band behaviour was completely masked by round-off error (see the second and third columns of table 4.2). The analysis was repeated twice with frequency transformations chosen to give the most accurate evaluation at two slightly different frequencies in the pass band, so that the differences in the results would be entirely due to the round-off errors generated through the analysis process. As expected, the results were most accurate in the pass band, but the effect of round-off error was satisfactorily small over the entire frequency

FREQUENCY cycles/sec	CALCULATED RESPONSE, AMPLITUDE, d.b.			
	without frequency transformation		with frequency transformation	
1630	10.714	-7.0398	-24.041	-24.041
1650	8.3458	-4.5865	-18.747	-18.750
1670	11.802	-5.8016	-17.226	-17.227
1680	8.2228	-2.5088	-12.028	-12.028
1690	11.632	-2.8787	-8.0758	-8.0757
1700	12.284	-3.1705	-5.8857	-5.3857
1740	5.2681	.48114	-4.2088	-4.2088
1780	10.687	1.9100	-5.6543	-5.6543
1800	8.5790	3.3182	-11.245	-11.245
1820	7.3544	3.9951	-20.705	-20.707
frequency scale cycles/sec	1740	1741	1740	1741

Table 4.2 Calculated frequency response of a filter complex containing 50 reactive components, illustrating the effect of a frequency transformation.

spectrum, the worst case being a loss of five out of nine significant figures in the response calculation at a frequency on the edge of the pass band (see the fourth and fifth columns of table 4.2).

In general, the accuracy problem is alleviated by the frequency transformation to the extent that, with one analysis, accurate evaluation of response is possible over a band of the frequency spectrum, and in very severe cases, such as a filter with many pass or stop bands, may the analysis have to be repeated several times to cover the entire frequency spectrum.

4.11 POLYNOMIAL REPRESENTATION

Although the term "polynomial" implies that it be represented by a list of real coefficients, it may not be expedient to use this form of representation in the analysis of some types of networks. For example, the problem of allocating store locations for polynomials would be greatly simplified if they were represented by complex numbers, being their values at a particular frequency. This simpler representation becomes an attractive alternative to the frequency transformation discussed in the previous section if the accuracy of the polynomial coefficients has such a critical bearing on the response that the analysis has to be repeated many times with different frequency transformations.

For networks with distributed-parameter subnetworks the complex number representation must be used together with some tearing technique. However, with the approach suggested here, the subnetworks with lumped parameters can be analysed separately using the conventional polynomial representation and only in the last stages, when the distributed parameter subnetworks are connected, must the analysis be performed at each desired frequency.

Indeed, such a combined strategy, in which polynomials are represented by their coefficients in the early stages of analysis and by complex numbers in the final stages, would be ideal for analysing large networks when accuracy was important. Constituent networks could be combined in subnetworks whose polynomials had degrees of less than, say, 10; the polynomials would be evaluated at discrete frequencies, and the analysis completed by combining the subnetworks at these frequencies.

Because the analysis algorithm avoids polynomial division it is also practical to represent polynomials by lists of terms each containing a numerical coefficient and a symbolic product of network parameters. In this way a network can be analysed with its parameters entered either numerically or symbolically.

For the study of network sensitivities, however, the above polynomial representation is not satisfactory because the specification of a large number of variable parameters leads to the proliferation of polynomial terms in unwieldy numbers. For a sensitivity analysis it is considered sufficient to calculate the partial derivatives of the polynomials with respect to each variable parameter. This can be achieved by extending each polynomial coefficient to include a list of first derivatives as well as the constant term, and by modifying the polynomial multiplication routine to avoid the generation of second derivatives.

4.12 CONCLUSION

The approach to linear network analysis outlined in this chapter establishes a modus operandi for computer programs which combine, for the first time, many of the desirable features of existing network analysis methods. Specifically, the rational polynomials of the Laplace transform technique are used for computationally-efficient calculation of frequency response; a network-tearing procedure is used which makes it practical to calculate the polynomial coefficients of large networks; and the analysis process has a topological nature, requiring routines for polynomial multiplication and addition only.

With regard to the wealth of accumulated experience in the application of a large number of computer programs for general circuit analysis it is now inconceivable that one method should be superior in all applications. In setting up computer-aided circuit design facilities the trend is towards providing a varied arsenal of programs, each with unique advantages in some applications. In this context, programs based on the new method provide a more economical and a more complete frequency analysis of most types of filters and equalisers, as well as many types of amplifier circuits. It might also be chosen for the ease with which it handles degenerate devices and active components. In designing such programs consideration must be given to numerous options which cover such aspects as input data format, a library of basic networks and common subnetworks, intermediate storage of polynomials, provision for sensitivity analysis and Monte Carlo methods, polynomial representation, and frequency transformations.

For convenience in calling up the various programs in a design facility an attempt is made to define a common form of circuit description—usually based on a list of branches and the nodes between which they are connected. Unfortunately it is in this respect that analysis programs based on diakoptic methods are at a disadvantage.

The efficiency of such methods, and this method in particular, is dependent on the manner in which the network is torn into its subnetworks. For the more complex networks the alternatives are many, and although the tearing can be programmed the results could be far from optimum. However, it is in analysing such strongly interconnected networks, for which the tearing process is difficult, that this diakoptic method would, inherently, be less efficient than the straightforward matrix methods and therefore would not be recommended.

It is widely recognised that the most exacting demands are made of analysis methods by programs which design circuits using optimisation techniques. The cost of running such programs is strongly related to the efficiency of the analysis method, for in each iteration of the optimisation process the analysis subprogram normally must calculate the response and its derivatives at many frequencies. But, with calculation of the network polynomial and their derivatives in a single analysis, it should be practical to go further than simply calculating the response at fixed frequencies, and determine those frequencies at which the response—either amplitude, phase or delay—has an extreme value. The optimisation process could then be designed to converge on a true equiripple design rather than a least squares approximation at a relatively large number of predetermined frequencies. It is in these applications that this new approach to network analysis shows the greatest promise.

Some of the dissatisfaction with existing analysis programs may be attributed to the belief of their originators that methods which are highly repetitive in nature are "ideally suited to a digital computer". We can admire the elegantly simple formulation and small amount of programming required for the early matrix-inversion, state-space and topological methods—certainly the programs provide a successful alternative for analysing networks which an engineer might otherwise attempt to analyse by himself. One expects, though, that the second generation of programs will make better use of a computer's resources when analysing much larger networks.

ALGORITHMS FOR COMPUTER PROGRAMS

5.1 INTRODUCTION

The successful implementation of the analysis method, which so far has been discussed only in general terms, is critically dependent on the form of the computer algorithm employed and the data structure with which it is associated. Much research work has been directed to the investigation and development of suitable algorithms and data structures, and the results of this work are felt to be important to an appreciation of the analysis method as a whole.

The development of suitable algorithms is subject to three criteria. Of paramount importance is the need to preserve numerical accuracy throughout the analysis process. Of secondary importance are the requirements that the largest communications networks be analysed, and that analysis be economic and convenient to the user.

Unfortunately, the achievement of these goals involves a competition for a limited computer resource—the core store—and a reasonable trade-off must be made between the size of data storage areas and the size of the program. Nonetheless, the core store of the Elliott 503 computer, on which the programs were run, sufficed for the development of programs with near optimum efficiency in their use of both processor time and core store, while allowing the programs to be tested on quite large networks.

This chapter does not attempt to document any particular program but presents those program elements which may be adopted profitably in any implementation of the analysis method.

To avoid ambiguity, algorithms are described in a compiler language. Attention is focused on the storage and manipulation of polynomials as whole entities because it is this aspect which has the greatest influence on the overall efficiency and the accuracy of the method. The other details of the algorithms are included for the sake of completeness and rigour, and are presented in a concise manner which is economic in the use of language and which makes the action of the algorithms as clear as possible. The generation of efficient

machine code is intentionally left to either an optimising compiler, or to a competent programmer who can interpret these algorithms and modify program statements to make best use of a particular compiler.

Throughout this and the subsequent chapter it is understood that all networks are 2-port networks characterised by sets of six polynomials.

5.2 PROGRAMMING LANGUAGE

Computer programs employing this analysis method could conceivably be written in almost any computer language. But in practice, with limited time available, they could not be written in machine code or an assembly language, and it is even doubtful if they could have been developed to their present stage of refinement using FORTRAN.

To maintain algorithms of a complex nature it is essential that they be expressed concisely and that most of the programming details be managed implicitly by the compiler. It is also an advantage that the action or flow of an algorithm be readily comprehensible, and, apart from the liberal use of comments, this can be best achieved with the avoidance of branch statements in the source program. Such is the argument against low-level, flowchart-like languages such as BASIC and FORTRAN.

The two major algorithms constituting the analysis method are, by nature, recursive. This fact alone is sufficient justification for the use of a compiler language — such as ALGOL — which provides a mechanism for the dynamic allocation of storage space during program execution. Without this mechanism, stacks (arrays) of sufficient size must be declared prior to execution, and extra coding must be included so that every time the same code is reentered or completed the variables local to that code are either pushed onto, or pulled from, the appropriate stack. ALGOL is desirable also because of the precise and concise definition of its grammar, for the power of its statements which are relevant to this application, and for its wide acceptance as a programming language.

However, in this application ALGOL is deficient in several important respects. For instance, although not possible in ALGOL, it is convenient to group together the parameters of a particular 2-port network—such as the addresses and degrees of its six polynomials—and regard them as attributes of a single entity. Such an entity is then denoted by a single reference variable and may be included as a member of any one of a number of sets during the analysis process.

Facilities for the dynamic creation, manipulation, and expiration of entities in this way are a natural extension of ALGOL, and have been accomplished with SIMULA, a compiler language designed for the simulation of discrete-event systems. The version used here is essentially that developed by Dahl and Nygaard at the Norwegian Computing Centre and implemented on UNIVAC 1100-series computers in 1964 [50]*.

Because SIMULA is not as widely known as ALGOL some of its basic components are now introduced within the context of the analysis method; other components of the language—e.g. the sequencing set, and operation rules for activities—are not relevant to this application.

A class declaration is, in appearance, like the head of an ALGOL procedure declaration; it introduces an identifier for a class of similar entities and describes the types and number of attributes which determine a particular representative of the class. For example, the declaration

```
"class network (netdegree);
```

```
  integer netdegree;
```

```
  begin integer array npaddress, npdegree 1:6 end;"
```

defines a class of entities; each is known as a "network" and has the attributes of a degree and the addresses and degrees of six polynomials. A particular entity belonging to this class is created by a reference expression such as "new network(10)" and expires when it is no longer referenceable.

* The language used here differs from the referenced language in that the symbol activity is replaced by the more appropriate symbol class, as in the more-recent SIMULA 67.

The creation of an entity involves the allocation and initialisation of storage space for its attributes, as well as the generation of an element which refers to the entity. It is the element rather than the set of attributes which is manipulated, and it may be referenced either by an element variable or as the member of a set. Thus with the declarations

```
"element this network;
```

```
  set basic networks;"
```

the statements

```
"this network:- new network (10);
```

```
  include (this network, basic networks);
```

```
  this network:- first (basic networks);"
```

have the effect of creating a new "network" of degree 10 which is denoted by the variable "this network"; of placing this element last in the set or list of "basic networks"; and of altering the variable to denote, instead, the first element of this set. With many other statements and expressions of this type we gain the benefits of a powerful list-processing facility.

A necessary facility of the language is the connection mechanism by which access is gained to the attributes of a particular entity. For example, if it is desired to assign to the integer variable "adrs" the address of the third polynomial of the "network" which is first in the set of "basic networks", it could be accomplished with the statement

```
"inspect first (basic networks) when network do adrs := npaddress [3] ;".
```

When this statement is executed the entity referred to by the element expression "first (basic networks)" is inspected, and if it is from the "network" class the statement following the "do" is executed and the attributes of the referenced entity are made available to it. The connection verb "inspect" may be replaced by "extract", in which case the connection statement would have the additional effect of removing the referenced element from the set of "basic networks" to which it belonged.

The algorithms presented in this chapter are written in SIMULA, although some licence is taken with the declaration of variables and specification of

procedure parameters. Unless explicitly declared or specified, all identifiers are assumed to be global integer variables or integer parameters. All parameters are assumed to be called by value unless specifically included in a name list. Statements represented by an English description of their action are enclosed in diamond brackets<>.

5.3 DATA STRUCTURE

The scheme adopted for the compact storage of polynomial coefficients during the analysis process is based on the fact that the degree of a complete network is not greater than the sum of the degrees of its constituent networks. Thus, if the polynomials of the constituent networks are stored in contiguous locations of the core store, the polynomials of the complete network may be overlaid in the same locations. It is a simple matter to arrange that for all pairs of subnetworks which are combined algebraically in the evaluation of a network expression, their polynomial coefficients do occupy contiguous locations.

Six rows of a two-dimensional array are reserved solely for the six respective polynomials of all networks. When a new network is introduced into the analysis process its polynomials are assembled from the left of the array, starting in the first vacant column.

When two networks are combined algebraically the vacant right-hand end of the array, starting with the first vacant column from the left, adjacent to the constituent polynomials, is used as temporary storage for six polynomials; the complete polynomials are first accumulated in these temporary areas, and finally assembled in the same locations occupied by the constituent polynomials.

The polynomials of subnetworks occurring in network expressions are therefore overwritten and can take no further part in the analysis. However, if a subnetwork is to be used in subsequent network expressions, its self-defining network expression is assigned to a block, its complete polynomials remain where first assembled in the array, and the polynomials of subsequent expressions are assembled further to the right.

The branches of a structure graph to be analysed by the general topological analysis method refer to blocks already assembled from network expressions in the above manner. Analysis of a structure graph requires extra space for the temporary storage of a large number of intermediate polynomials, and two extra rows are added to the main array solely for this purpose. Single polynomials are stacked in this area during the analysis of a structure, in the same recursive manner that sets of six polynomials are stacked in the first six rows during the evaluation of a network expression.

A list of branches defining a structure graph is introduced as a set of parameters to a basic network and may therefore appear almost anywhere in a network expression. Its complete polynomials are accumulated directly in the first vacant locations from the left, in the first six rows. Most intermediate polynomial products are stored in the extra two rows, but some intermediate polynomials with a special significance are stored at the right-hand end of the first six rows.

A grammar for network expressions and an algorithm for their evaluation are discussed later, in section 5.5.

To facilitate access to the various polynomials by universal utility routines the main array is actually declared with a single dimension. Any polynomial is then located with a single number, defined as the address of that polynomial, which is the address or subscript of the leading coefficient of the polynomial in the main array. The address of a network is defined as a base address from which can be calculated the addresses of its polynomials with a statement of the form

"naddress [px] := base address + px * rl;"

where "rl" is the length of a row in the equivalent two-dimensional array.

A typical arrangement of polynomials in the main array is shown in figure 5.1.

5.4 UTILITY ROUTINES

The following procedures are responsible for all manipulation of polynomials stored in the main array. They are described here only to the extent that their results, and their interface with calling algorithms, are precisely defined. Code for the procedure bodies could be quite simple, but because they perform all the "productive" computation of the analysis process, it may be desired to optimise them as far as possible—even to the extent of writing them in the assembly language of the host computer.

In the following descriptions the expression "pol(adrs,deg)" refers to the unique polynomial whose address is given by "adrs" and whose degree is given by "deg"; "pol(adrs)", on the other hand, refers to the polynomial whose address is given by "adrs" but whose degree is determined by the computation process.

```
procedure polclear (adrs,deg);
```

```
name deg;
```

```
  <depending on the polynomial representation, if the degree is fixed then  
    pol(adrs,deg):=0.0 (i.e. all coefficients are set to zero); otherwise,  
    deg:=-1>;
```

```
procedure polcopy (adrs1,deg1,adrs2,deg2);
```

```
name deg2;
```

```
begin deg2:=deg1;
```

```
  <pol(adrs2):=pol(adrs1,deg1)>
```

```
end;
```

```
procedure poladd (adrs1,deg1,adrs2,deg2,adrs3,deg3,sign);
```

```
name deg3;
```

```
begin if sign<0
```

```
  then <pol(adrs3):=pol(adrs1,deg1)-pol(adrs2,deg2)>
```

```
  else <pol(adrs3):=pol(adrs1,deg1)+pol(adrs2,deg2)>;
```

```
  deg3:=<degree of pol(adrs3)>
```

```
end;
```

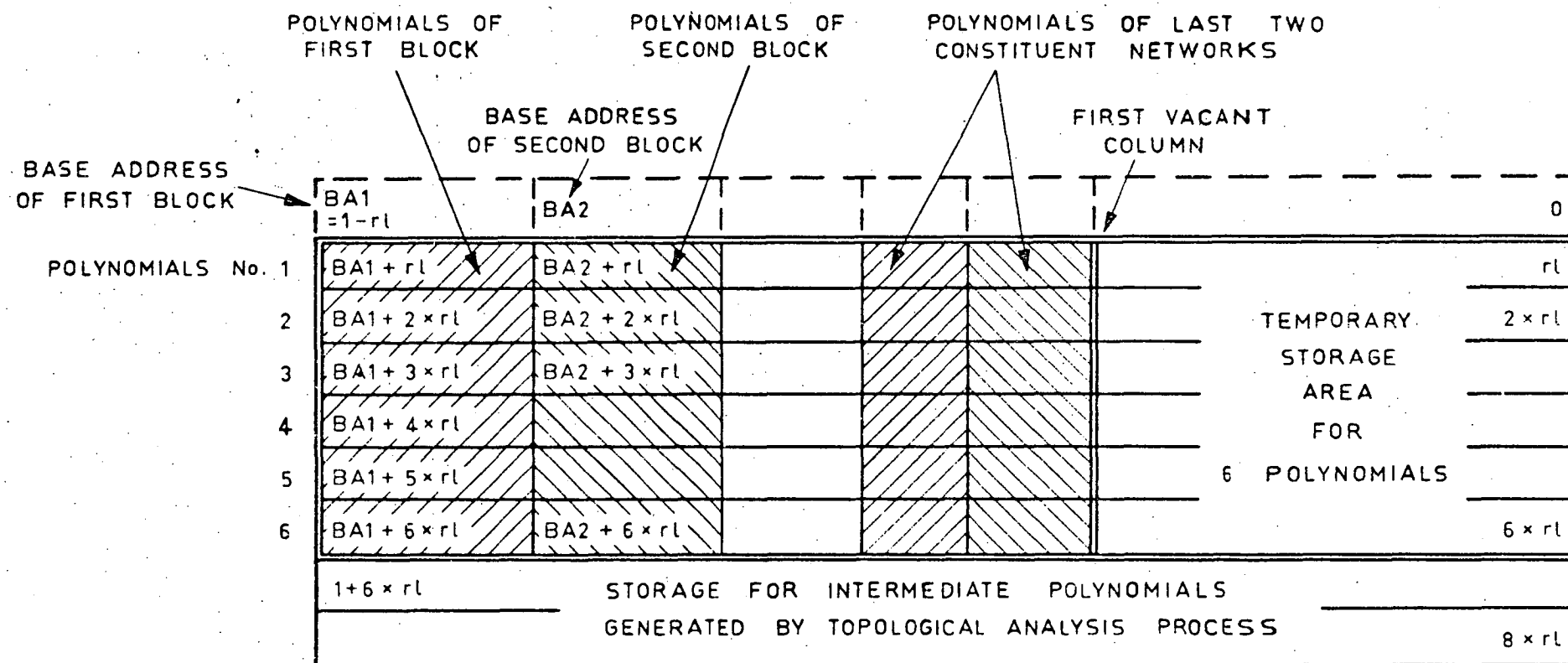


Figure 5.1 Typical arrangement of polynomials in the equivalent two-dimensional array. Arithmetic expressions define the subscripts of corresponding locations in the actual array.

```

procedure polmtply (adrs1,deg1,adrs2,deg2,adrs3,deg3,sign);
name deg3;
begin if sign = 0
    then<pol(adrs3):=pol(adrs1,deg1)*pol(adrs2,deg2)>
    else if sign<0
        then<pol(adrs3):=pol(adrs3,deg3)-pol(adrs1,deg1)*pol(adrs2,deg2)>
        else<pol(adrs3):=pol(adrs3,deg3)+pol(adrs1,deg1)*pol(adrs2,deg2)>;
    deg3:=<degree of pol(adrs3)>
end;

```

The above procedure is used for virtually all polynomial multiplication and addition. The reason for combining these two functions in the one procedure derives from the significant saving in processor time which can be achieved when the degree of a polynomial product equals the degree of the polynomial to which it is to be added.

If polynomials are represented by lists of numerical coefficients the product of a pair of polynomials involves the multiplication of every coefficient of the first polynomial with every coefficient of the second polynomial, and the addition of every such arithmetic product to an appropriate accumulating sum which ultimately becomes a coefficient of the product polynomial. Thus, the process of polynomial multiplication is seen to involve a large element of polynomial addition; in particular, it is noted that if the accumulating sums are not initially set to zero but initialised with the coefficients of the polynomial to which the product is to be added, the desired multiplication and addition is performed simultaneously. The need to add a product of polynomials to an existing polynomial, in situ, occurs frequently in the analysis process, and with the above approach can be achieved with less effort than can the product alone.

This economy of polynomial multiplication and addition (or subtraction) requires that the six polynomials of any network all have the same degree. Although, in theory, this is not always the case, it can be enforced computationally by the inclusion of zero coefficients, without incurring a significant computational penalty. And because the degree of a network equals the degrees of its polynomials, there is the additional benefit that most of the book-keeping associated with the calculation and storage of polynomial degrees can be eliminated.

However, if polynomials are represented in a different form—such as a list of terms each defined by a symbolic product of network parameters, a power of s , and a numerical coefficient—the above economy can not be realised. The "degree" of a polynomial used in these procedures loses its significance as the highest exponent of s but continues to serve the purpose of determining the number of elements, coefficients, or terms in the polynomial, and therefore determines the space that must be allocated in the main array. With the latter form of polynomial representation it is desirable to allow the polynomials of a network to have different "degrees", and all the procedures and algorithms presented in this chapter have been designed to be applied equally-well in either situation. It is necessary to change only the bodies of the procedures comprising these utility routines.

5.5 ALGEBRAIC REDUCTION

The language in which a network is described to an analysis program has a significant bearing on the convenience of that program as an analytical tool. It should impose few constraints on the types of parameters, be concise, "natural", and simple to learn.

As a means of describing a network, an algebraic network expression exhibits the notational economy of a mathematical formula, and, due to the versatility of the many network operations, permits network parameters to be introduced in a wide variety of forms. For example, a transistor may be defined by any lumped parameter model, or any set of H, G, Y, Z , or transmission parameters; it may be defined in either a common base, common emitter, or common collector configuration, and have the configuration changed

algebraically. Another advantage of the network expression is that it can usually be arranged to reflect the natural structure of the network—the manner in which subnetworks are cascaded or paralleled—and is therefore easier to comprehend and check.

Unlike some other methods of network description, a network expression strongly influences the analysis process: in this case, by directing the order in which subnetworks are combined. Thus, the specification of an algorithm to evaluate network expressions depends on their grammatical form. For this reason, and also to establish a standard network-description language which future analysis programs may adopt, a grammar for network expressions is now specified.

5.5.1 Syntax of Network Expressions

The syntax of network expressions is here defined in Backus-Naur Form [53]. The metasyMBOL "::<=" has the English meaning of "is defined as", the metasyMBOL "|" has the meaning of "or", and pairs of diamond brackets <> enclose characters which are to be treated as a unit.

Numbers in the right-hand margin refer to statements which are out of sequence.

0	<assignment statement>	::= <block> = <network>	
1	<block>	::= B <block number>	
1.1	<block number>	::= <integer>	
2	<network>	::= (<network>)	
		<port interchange operator><network>	
		<network><dyadic operator><network>	
		<basic network>	3
2.1	<port interchange operator>	::= r	
2.2	<dyadic operator>	::= <cascade operator> <parallel operator>	
2.2.1	<cascade operator>	::= c	

2.22 <parallel operator> ::= <port interconnection><port interconnection>
 2.221 <port interconnection> ::= <parallel port>|<series port>
 2.2211<parallel port> ::= p
 2.2212<series port> ::= s
 3 <basic network> ::= <block>| 1
 <trivial network>|<simple network>|
 <ideal transformer>|<mutual inductor>|
 <voltage-controlled current>|
 <voltage amplifier>|
 <gyrator>|
 <general network>|
 <network structure>
 3.1 <trivial network> ::= <unit network>|<cross-over network>|
 <current-inversion NIC>
 3.11 <unit network> ::= U
 3.12 <cross-over network> ::= X
 3.13 <current-inversion NIC>::= I
 3.2 <simple network> ::= <component position><component> 4
 3.21 <component position> ::= <series position>|<shunt position>
 3.211 <series position> ::= S
 3.212 <shunt position> ::= I
 3.3 <ideal transformer> ::= F<parameter> 5
 3.4 <mutual inductor> ::= M<parameter><parameter><parameter>
 3.5 <voltage-controlled current>
 ::= H[<degree>,<polynomial>,<polynomial>] 6
 3.6 <voltage amplifier> ::= A[<degree>,<polynomial>,<polynomial>]
 3.7 <gyrator> ::= G<parameter> 5
 3.8 <general network> ::= N[<degree>,<polynomial>,<polynomial>,
 <polynomial>,<polynomial>,<polynomial>,
 <polynomial>] 6
 3.9 <network structure> ::= D[<branch list>]
 3.91 <branch list> ::= <branch>,<closing branch>|
 <closing branch>,<branch>|
 <branch list>,<branch>|
 <branch>,<branch list>

3.911	<branch>	::= <block>:<node part>	1
3.9111	<node part>	::= <node number><port interconnection>	1
		<node number><port interconnection>	2.221
3.91111	<node number>	::= <integer>	
3.92	<closing branch>	::= BO:<node part>	3.9111
4	<component>	::= <basic component> <tuned component> <compound component>	
4.1	<basic component>	::= <resistor> <inductor> <capacitor>	
4.11	<resistor>	::= R<parameter>	5
4.12	<inductor>	::= L<parameter>	5
4.13	<capacitor>	::= C<parameter>	5
4.2	<tuned component>	::= X<parameter><parameter><parameter>	5
4.3	<compound component>	::= <second port termination><network>	2
4.31	<second port termination>	::= <short circuit> <open circuit>	
4.311	<short circuit>	::= Y	
4.312	<open circuit>	::= Z	
5	<parameter>	::= <real number> [<parameter identifier>] <parameter><multiplying operator><parameter>	
5.1	<parameter identifier>	::= <any string of characters not containing]>	
5.2	<multiplying operator>	::= * /	
6	<polynomial>	::= <parameter> <parameter><polynomial>	
6.1	<degree>	::= <integer>	

For programming convenience the formats for <integer> and <real number> comply with the respective requirements for free-format input of integers and floating-point numbers by the particular compiler system which implements the analysis algorithm.

Blank spaces are generally permitted anywhere in a network expression to improve its readability.

If necessary, lower-case characters may be replaced by their corresponding upper-case characters, and square brackets [] may be replaced by round brackets ().

5.5.2 Semantics of Network Expressions

In the above syntax definition a "network" is a 2-port network and a "component" is a 1-port network. A "component" may comprise a single 2-terminal device (a "basic component"), either a series or shunt LCR combination (a "tuned component"), or, in the case of a "compound component", be the first port of a "network" which has its second port terminated in either an open circuit (Z) or a short circuit (Y).

The many symbols that may appear in a network expression are mnemonic and serve as the context by which meanings are attached to network parameters. With this grammar the parameters have the following meanings :

- 3.3 F: secondary to primary turns ratio
- 3.4 M: primary inductance (henries), secondary inductance (henries),
coupling coefficient
- 3.5 H: numerator and denominator polynomials of the transconductance
- 3.6 A: numerator and denominator polynomials of the amplifier gain
- 3.7 G: gyration resistance (ohms)
- 3.8 N: the six polynomials which fully characterise the network
- 4.11 R: resistance (ohms)
- 4.12 L: inductance (henries)
- 4.13 C: capacitance (farads)
- 4.2 SX: inductance (henries), resonant frequency (hertz),
resistance (ohms)— of a series LCR combination
- TX: capacitance (farads), resonant frequency (hertz),
conductance (mhos)— of a parallel LCR combination

The format for a polynomial depends on the form of polynomial representation used within the program. The given definition (metalanguage statement 6) applies when polynomials are represented by a list of numerical coefficients, in which case each polynomial must include a parameter for every coefficient, beginning with the coefficient of the highest power of

s determined by the given "degree".

Circuit diagrams for some basic networks are given in figure 4.1.

5.5.3 Evaluation of Network Expressions

The algorithm which interprets a network expression, assembles the polynomials of basic networks and manipulates them algebraically is shown in figure 5.2. Its right-hand margin contains statement numbers which are referred to in the following discussion.

The data input to the algorithm is in the form of an ordered set of "significant characters" (100) which includes all except the editing characters — blanks, line-feeds, etc — in the network expression, and terminates with a statement delimiter. The data is processed by a single call (501) to the recursive procedure "assemble network".

Local to the main procedure (200) is a procedure for combining two networks algebraically (210), and local to the latter is procedure "pma" (220) whose sole function is to expand a set of three polynomial-index parameters and call (224) the utility procedure "polmtply" with the appropriate parameters. It multiplies polynomial "px1" of the first network with polynomial "px2" of the second network and accumulates the product in the temporary polynomial "px3", according to the parameter "sign".

The addresses of the six temporary polynomials are calculated (230,231) prior to combining the two networks.

If the polynomials of the first network, the polynomials of the second network, and the temporary polynomials are denoted by A,B and T respectively, the statements 233-235 perform the cascade operation by evaluating the polynomial expressions:

<u>set</u> significant characters;	100
<u>boolean</u> expression terminated, closing bracket;	101
<u>integer</u> <u>array</u> tempadrs, tempdeg [1:6];	102
<u>element</u> this block;	103
<u>class</u> network (netdegree);	104
<u>integer</u> netdegree;	105
<u>begin</u> <u>integer</u> <u>array</u> npaddress, npdegree [1:6] <u>end</u> ;	106
<u>procedure</u> assemble network (address of this network, this network, degree of last network);	200
<u>name</u> this network; <u>element</u> this network;	201
<u>begin</u> <u>integer</u> address of second network;	202
<u>element</u> second network;	203
<u>procedure</u> combine the two networks;	210
<u>begin</u> <u>procedure</u> pma(px1,px2,px3,sign);	220
<u>begin</u> <u>inspect</u> second network <u>when</u> network <u>do</u>	221
<u>begin</u> adrs 2:= npaddress[px2];	222
deg 2:= npdegree[px2]	223
<u>end</u> ;	
polmtply(npaddress[px1], npdegree[px1], adrs2, deg2, tempadrs[px3], tempdeg[px3], sign)	224
<u>end</u> polynomial multiplication and addition;	
<u>inspect</u> second network <u>when</u> network <u>do</u> base address:= address of second network + netdegree +1;	230
<u>for</u> j:= 1 <u>step</u> 1 <u>until</u> 6 <u>do</u> tempadrs[j]:= base address + j * <row length of main array>;	231
<u>if</u> <cascade operation is required>	232

Figure 5.2 Algorithm for evaluation of network expressions (Page 1 of 5)

```

then begin pma(1,1,1,0); pma(1,2,2,0); pma(3,1,3,0); pma(3,2,4,0);      233
           pma(2,3,1,1); pma(2,4,2,1); pma(4,3,3,1); pma(4,4,4,1);      234
           pma(5,5,5,0); pma(6,6,6,0)                                   235
end
else begin pl:= 1; p2:= 2; p3:= 3; p4:= 4; sign:= -1;                    236
         if<first ports are connected in series> then                    237
             begin <swap p1 and p3>; <swap p2 and p4>; sign:= -sign end;  238
         if<second ports are connected in series> then                    239
             begin <swap p1 and p2>; <swap p3 and p4>; sign:= -sign end;  240
         pma(pl,p2,p1,0); pma(p4,p2,p4,0); pma(5,p2,5,0); pma(6,p2,6,0);  241
         pma(p2,p1,p1,1); pma(p2,p4,p4,1); pma(p2,5,5,1); pma(p2,6,6,1);  242
         pma(p2,p2,p2,0); pma(2,3,p3,0); pma(3,2,p3,1);                243
         pma(1,4,p3,1); pma(6,5,p3,sign); pma(4,1,p3,1); pma(5,6,p3,sign)  244
         end;
         for j:= 1 step 1 until 6 do                                     245
             begin npaddress[j]:= address of this network + j * <row length of main array>;  246
                 polcopy(tempadrs[j], tempdeg[j], npaddress[j], npdegree[j])  247
             end;
             netdegree:= <maximum of npdegree[j]>                        248
         end combination of the two networks;

extract first(significant characters)                                    300
when <"("> do                                                            301
    begin assemble network(address of this network, this network, 10000);  302

```

<u>if</u> closing bracket	303
<u>then</u> closing bracket:= <u>false</u>	304
<u>else</u> <u>print</u> "TOO MANY OPENING BRACKETS"	305
<u>end</u>	
<u>when</u> <"r"> <u>do</u>	306
<u>begin</u> assemble network(address of this network, this network,0);	307
<interchange the pairs of polynomials{1,4} and{5,6} of "this network">	308
<u>end</u>	
<u>when</u> <"S" or "T"> <u>do</u>	309
<u>begin</u> <u>extract first</u> (significant characters)	310
<u>when</u> <"R", "L", "C", or "X"> <u>do</u>	311
<u>begin</u> <extract and interpret the following parameter(s) which specify a component>;	312
<create a new "network" whose polynomial addresses are based on the	313
"address of this network", and refer to it as "this network">;	314
<assemble the numerator and denominator of the component impedance	315
as the respective polynomials 1 and 3 of "this network">;	316
na:= 1; da:= 3	
<u>end</u>	
<u>when</u> <"Y" or "Z"> <u>do</u>	318
<u>begin</u> assemble network(address of this network, this network, 0);	319
na:= <u>if</u> <second port is terminated with an open circuit (Z)> <u>then</u> 1 <u>else</u> 2;	320
da:= na + 2	321
<u>end</u>	
<u>otherwise</u> <u>print</u> "CHARACTER DOES NOT SPECIFY A COMPONENT";	322

<u>if</u> <component is to be placed in the shunt (T) position>	323
<u>then begin</u> zx:= 2; nx:= 1; dx:= 3 <u>end</u>	324
<u>else begin</u> zx:= 3; nx:= 2; dx:= 1 <u>end</u> ;	325
<u>inspect</u> this network <u>when</u> network <u>do</u>	326
<u>begin</u> npaddress [zx]:= npaddress[5]; npdegree[zx]:= netdegree;	327
npaddress[nx]:= npaddress[na]; npdegree[nx]:= npdegree[na];	328
npaddress[dx]:= npaddress[da]; npdegree[dx]:= npdegree[da];	329
npaddress[4]:= npaddress[5]:= npaddress[6]:= npaddress[1];	330
npdegree[4]:= npdegree[5]:= npdegree[6]:= npdegree[1];	331
polclear(npaddress[zx], npdegree [zx])	332
<u>end</u>	
<u>end</u>	
<u>when</u> <"A", "B", "D", "F", "G", "H", "I", "M", "N", "U", or "X"> <u>do</u>	333
<u>begin</u> <extract and interpret the following characters(if relevant) which specify a basic network>;	334
<create a new "network" whose polynomial addresses are based on the	335
"address of this network", and refer to it as "this network">;	336
<assemble the polynomials of "this network" to represent the basic network>	337
<u>end</u>	
<u>when</u> <statement delimiter> <u>do</u>	338
<u>begin</u> expression terminated:= <u>true</u> ;	339
<u>print</u> "EXPRESSION IS NOT COMPLETE"	340
<u>end</u>	
<u>otherwise print</u> "CHARACTER DOES NOT SPECIFY A BASIC NETWORK";	341

operation:	400
<u>if not</u> (closing bracket <u>or</u> expression terminated) <u>then</u>	401
<u>inspect</u> this network <u>when</u> network <u>do if</u> netdegree < degree of last network <u>then</u>	402
<u>extract first</u> (significant characters)	403
<u>when</u> < "c", "p", or "s"> <u>do</u>	404
<u>begin</u> <extract and interpret the following character (if relevant) which completes	405
the specification of a dyadic network operation>;	406
address of second network:= address of this network + netdegree + 1;	407
assemble network(address of second network, second network, netdegree);	408
combine the two networks;	409
<u>goto</u> operation	410
<u>end</u>	
<u>when</u> <")"> <u>do</u> closing bracket:= <u>true</u>	411
<u>when</u> <statement delimiter> <u>do</u> expression terminated:= <u>true</u>	412
<u>otherwise print</u> "CHARACTER DOES NOT SPECIFY A NETWORK OPERATION"	413
<u>end</u> assembly of network;	
 closing bracket:= <u>false</u> ;	500
 assemble network (<base address of new block>, this block, 10000);	501
 <u>if</u> closing bracket <u>then print</u> "TOO MANY CLOSING BRACKETS";	502

$$T1 = A1.B1 + A2.B3,$$

$$T2 = A1.B2 + A2.B4,$$

$$T3 = A3.B1 + A4.B3,$$

$$T4 = A3.B2 + A4.B4,$$

$$T5 = A5.B5,$$

$$T6 = A6.B6.$$

Statements 241-244 perform the parallel-parallel operation by evaluating the polynomial expressions:

$$T1 = A1.B2 + A2.B1,$$

$$T2 = A4.B2 + A2.B4,$$

$$T5 = A5.B2 + A2.B5,$$

$$T6 = A6.B2 + A2.B6,$$

$$T2 = A2.B2,$$

$$T3 = A2.B3 + A3.B2 + A1.B4 - A6.B5 + A4.B1 - A5.B6.$$

However, if either or both of the ports are connected in series rather than parallel, the effect of cascading with a unit gyrator is first incorporated (237-240). When all the new polynomials have been calculated the polynomial addresses of the first network are calculated (246) and the temporary polynomials are copied into these locations (247), thus effecting the assignments :

$$A1 = T1,$$

$$A2 = T2,$$

$$A3 = T3,$$

$$A4 = T4,$$

$$A5 = T5,$$

$$A6 = T6.$$

Execution of the main procedure begins with the extraction of the next significant character from the input file (300).

If an opening bracket is encountered (301), a recursive call is made to the same procedure (302) and a closing bracket is cancelled (304). A recursive call is also made if a port-interchange operator is encountered, and the desired reversal operation is achieved simply by interchanging two

pairs of polynomials (306-308).

The assembly of a simple network (309) is preceded by the assembly of the numerator (N) and denominator (D) polynomials of a component impedance, which, in the case of a compound component (318), requires another recursive call to the same procedure (319). If the component is placed in the shunt position, the six polynomials become N,O,D,N,N,N; if placed in the series position, they become D,N,O,D,D,D. The assignment of numerator and denominator polynomials is achieved by copying polynomial addresses (327-331) rather than polynomial coefficients.

After assembling a network, and finding (401) that the network expression is terminated neither permanently (with a statement delimiter) nor temporarily (with a closing bracket), the decision to continue or terminate the procedure depends on the degrees of the last two networks to be assembled (402); by not combining two networks if the degree of the second is less than the degree of the first, the procedure exhibits a preference for combining networks of the same degree and thus minimises polynomial manipulation. By calling the procedure with a value of 0 or 10000 in place of the parameter representing the "degree of the last network", this decision mechanism is used to ensure that the procedure terminates either as soon, or as late, as possible.

Apart from a closing bracket (411) or a statement delimiter (412), a network should only be followed by a dyadic network operator (404). In the latter case a base address for a second network is calculated (407), a recursive call is made in order to assemble a second network (408), and the two networks are combined algebraically (409).

The algorithm incorporates no precedence rules — such as the parallel operator before the cascade operator — and the user is encouraged to introduce pairs of brackets whenever there is doubt about the possible execution order of network operations.

There is a clear relationship between the algorithm and the syntax definition for network expressions: the calls to the procedure "assemble network" in algorithm statements 501, 302, 307, 408, and 319 correspond to references to the metavariable "<network>" in their respective syntax-definition statements 0,2,2,2, and 4.3.

Any program which implements this algorithm must, like any interpreter, output unambiguous diagnostic messages whenever a symbol can not be properly interpreted. Examples of diagnostic messages occur in statements 305, 322, 340, 341, and 502. It is also desirable that a program should attempt to recover from data errors — either by skipping spurious characters or by making suitable assumptions — and, as far as possible, continue scanning the data for further errors. However, the algorithm of figure 5.2 has not been encumbered with programming details of this nature.

The algorithm described above is essentially an interpreter rather than a compiler; that is, interpretation of a network expression and analysis of the specified network are concomitant. This simpler approach, which requires only one pass through the input data, is generally satisfactory — but in some applications it may prove costly. For instance, when a data error is discovered, the polynomial manipulation preceding the discovery is wasted. Also, if the evaluation of a network expression should be repeated with different network parameters, as in an iterative network-design program, the interpretation is repeated unnecessarily. In an alternative approach polynomials could be assembled and manipulated with procedures controlled by program switches. A compiler would interpret the input data file, construct a table of fixed and variable network parameters, and compile a sequence of program switch settings and parameter indices.

5.6 TOPOLOGICAL ANALYSIS

5.6.1 Interface with the Algorithm

The analysis algorithm presented in figure 5.3 as the procedure "topologic" requires that the topological data describing the network structure and the parameters of the individual networks be available

in a particular form.

If the structure is defined by a branch list, as defined by statement 3.91 in the syntax definition of network expressions, then each branch, by virtue of its position in the list, has a unique index number "bx" and is linked to corresponding elements in three arrays. Two arrays, "first node" and "second node", simply list the first and second node numbers of each branch, and thus completely determine the network structure in the conventional manner. The third array, "branch reference", is a list of elements referring to "branches" which are representatives of the class declared globally as follows:

```
"class branch (branch degree);
  begin integer array bpaddress, bpdegree, bptag[1:6] end;"
```

The attributes of a "branch" determine the address, degree, and a tag for each of the six polynomials which characterise a 2-port network. The "branch degree" is the maximum of the individual "bpdegree[j]". If a polynomial contains only a constant term and its magnitude is zero or unity, then its tag is set equal to the polynomial; otherwise, its tag is set to +3.

The analysis procedure assumes that each port in the structure is connected in parallel rather than in series. Therefore, if a port of a constituent network is connected (to other networks) in series it must be cascaded with a unit gyrator: if the first port is connected in series, the pairs of branch polynomials {1,3} and {2,4} are interchanged and the sign of the tag of polynomial 5 is changed; if the second port is connected in series, the pairs of polynomials {1,2} and {3,4} are interchanged and the sign of the tag of polynomial 5 is changed. Finally, each transfer polynomial (5 and 6) is compared with its corresponding natural polynomial (4 and 1 respectively). If its tag is ± 3 and, except for a possible sign difference, the polynomial is identical to its corresponding natural polynomial then its tag is changed to either +2 or, if there is a sign difference, to -2.

```

procedure topologic(nodes, branches, closing bx, first node, second node, branch reference, tempadrs, stack address);      100
integer array first node, second node, tempadrs;      101
element array branch reference;      102
begin
  boolean possible to extend path, loop includes closing branch;      110
  boolean array listed branch reversed, pointer set[1: nodes];      111
  integer array listed node, listed branch[1: nodes], node sum of branch, possible pointers into branch[1: branches];      112
  class polynomial(paddress, pdegree, ptag);;      113
  class directed branch(bx, breversed); boolean breversed;;      114
  element unit polynomial, bb, extended product, loop polynomial;      115
  element array closing loop product[1:6];      116
  set term factors, loop natural polynomials, loop transfer polynomials;      117
  set array branch list of node[1: nodes];      118

procedure set pointer(k, n, pathstart, partial product);      200
element partial product;      201
begin integer pptag, nextadrs;      202
  element b;      203

  procedure register polynomial(px,bx);      210
  if bx = closing bx      211
    then begin closing npx:= px; closing tpx:= 0 end      212
    else inspect branch reference[bx]when branch do      213
      begin tftag:= tftag * bptag[px];      214

```

Figure 5.3 Algorithm for topological analysis (Page 1 of 8)

```

    if tftag = 0                                     215
    then goto next branch                             216
    else if abs(bptag[px]) ≠ 1 then include(new polynomial(bpaddress[px], bpdegree[px], bptag[px]), term factors) 217
end;

procedure register path (k);                         220
for k:= k step - 1 until pathstart do register polynomial(if listed branch reversed[k] then 1 else 4, listed branch[k]); 221

procedure adjust tag of all branches connected to this node by(a); 230
for b:- first(branch list of node[n]), suc(b) while exist(b) do 231
    inspect b when directed branch do possible pointers into branch[bx] := possible pointers into branch[bx] + a; 232

element procedure product(polynomial factors, address, degree, tag, sign); 240
set polynomial factors;                             241
begin extract(if empty(polynomial factors) then unit polynomial else first(polynomial factors)) when polynomial co 242
    begin adrs l:= paddress;                          243
        deg l:= pdegree                               244
    end;
if empty(polynomial factors)                         245
then begin if sign = 0                               246
    then polcopy(adrs l, deg l, address, degree)      247
    else poladd(adrs l, deg l, address, degree, address, degree, tag * sign) 248
end
else begin

```

next factor:	<u>extract first</u> (polynomial factors) <u>when</u> polynomial <u>do</u>	250
	<u>begin</u> adrs 2:= paddress;	251
	deg 2:= pdegree	252
	<u>end</u> ;	
	<u>if empty</u> (polynomial factors)	253
	<u>then</u> polmtply(adrs 1, deg1, adrs 2, deg 2, address, degree, tag * sign)	254
	<u>else begin</u> adrs 3:= <u>if</u> adrs 1 = tempadrs[1] <u>then</u> tempadrs[2] <u>else</u> tempadrs [1];	255
	polmtply(adrs 1, deg 1, adrs 2, deg 2, adrs 3, deg 1, 0);	256
	adrs 1:= adrs 3;	257
	<u>goto</u> next factor	258
	<u>end</u>	
	<u>end</u> ;	
	product:- <u>new</u> polynomial(address, degree, <u>if</u> tag = 0 <u>then</u> sign <u>else</u> tag)	259
	<u>end</u> product;	
	<u>inspect</u> partial product <u>when</u> polynomial <u>do</u>	260
	<u>begin</u> pptag:= ptag;	261
	nextadrs:= paddress + pdegree + 1	262
	<u>end</u> ;	
	pointer set [n]:= <u>true</u> ;	263
	listed node [k]:= n;	264
	adjust tag of all branches connected to this node by (-1);	265
	<u>for</u> b:- <u>first</u> (branch list of node [n]), <u>suc</u> (b) <u>while</u> <u>exist</u> (b) <u>do</u>	266
	<u>begin</u> possible to extend path:= <u>false</u> ;	267

Figure 5.3 (page 3 of 8)

tftag:= pptag;	268
<u>clear</u> (term factors);	269
<u>if</u> abs(tftag)≠ 1 <u>then</u> <u>include</u> (partial product, term factors);	270
<u>inspect</u> b <u>when</u> directed branch <u>do</u>	271
<u>begin</u> possible pointers into branch[bx]:= possible pointers into branch[bx] + 1;	272
listed branch[k]:= this bx:= bx;	273
listed branch reversed[k]:= breversed	274
<u>end</u> ;	
<u>for</u> bb:- <u>first</u> (branch list of node[n]), <u>suc</u> (bb) <u>while</u> <u>exist</u> (bb) <u>do</u>	275
<u>inspect</u> bb <u>when</u> directed branch <u>do</u>	276
<u>if</u> possible pointers into branch[bx] = 0 <u>then</u> register polynomial(2, bx);	277
next n:= node sum of branch[this bx] - n;	278
<u>for</u> nx:= k-1 <u>step</u> -1 <u>until</u> pathstart <u>do</u> <u>if</u> listed node[nx]= next n <u>then</u>	279
<u>begin</u> register path(nx -1);	280
<u>if</u> nx = k -1	281
<u>then</u> register polynomial(3, this bx)	282
<u>else</u> <u>begin</u> lnptag:= ltptag:= 1;	283
<u>clear</u> (loop natural polynomials); <u>clear</u> (loop transfer polynomials);	284
<u>for</u> nx:= nx <u>step</u> 1 <u>until</u> k <u>do</u>	285
<u>begin</u> this bx:= listed branch[nx];	286
<u>if</u> listed branch reversed[nx]	287
<u>then</u> <u>begin</u> npx:= 1; tpx:= 6 <u>end</u>	288
<u>else</u> <u>begin</u> npx:= 4; tpx:= 5 <u>end</u>	289
<u>if</u> this bx = closing bx	290

```

then begin closing npx:= npx;                                291
            closing tpx:= tpx;                                292
            loop includes closing branch:= true               293
        end
    else inspect branch reference[this bx] when branch do    294
        begin lnptag:= lnptag * bptag[npx];                  295
            ltptag:= ltptag * bptag[tpx];                      296
            if lnptag = 0 and ltptag = 0 then goto next branch; 297
            if abs(bptag[tpx]) = 2                             298
            then register polynomial(npx, this bx)             299
            else begin if lnptag  $\neq$  0 and abs(bptag[npx])  $\neq$  1 then 300
                include(new polynomial(bpaddress[npx], bpdegree[npx], bptag[npx]),
                    loop natural polynomials);                 301
                if ltptag  $\neq$  0 and abs(bptag[tpx])  $\neq$  1 then      302
                include(new polynomial(bpaddress[tpx], bpdegree[tpx], bptag[tpx]),
                    loop transfer polynomials)                   303
            end
        end
    end;
    if loop includes closing branch                            304
    then begin closing loop product[closing npx]:-           305
        product(loop natural polynomials, tempadrs[5], 0, 0, lnptag);
        closing loop product[closing tpx]:-                   306
        product(loop transfer polynomials, tempadrs[6], 0, 0, ltptag);
    end
end;

```

Figure 5.3 (Page 5 of 8)

```

        loop includes closing branch:= false                                307
    end
    else begin if empty (loop natural polynomials) and empty(loop transfer polynomials) 308
        then goto next branch;                                            309
    loop polynomial:-                                                    310
        product(loop natural polynomials, tempadrs[3],0, 0, lnptag);
    if ltptag  $\neq$  0 then inspect loop polynomial when polynomial do      311
        loop polynomial:-                                              312
            product(loop transfer polynomials, tempadrs[3], pdegree, lnptag, - ltptag);
        inspect loop polynomial when polynomial do                    313
        begin include(loop polynomial, term factors);                    314
            tftag:= tftag * ptag                                         315
        end
    end
    goto next pointer                                                    316
end;
    for nx:= pathstart-1 step-1 until 1 do if listed node[nx] = next n then 317
    begin register path(k);                                              318
        goto next pointer                                              319
    end;
    possible to extend path:= true;                                    320
next pointer:                                                            321
    if k<nodes                                                            322

```



```

then begin if possible to extend path                                323
    then revised path start:= path start                            324
    else begin revised path start:= k + 1;                          325
        for next n:= 1, next n + 1 while pointer set[next n] do 326
            end;
        set pointer(k + 1, next n, revised path start, product(term factors, nextadrs, 0, 0, tftag) 327
    end
else begin if closing tpx = 0                                        328
    then inspect closing polynomial[closing npx]when polynomial do 329
        closing polynomial[closing npx]:- product(term factors, paddress, pdegree, ptag, tftag) 330
    else begin extended product:- product(term factors, tempadrs[4], 0, 0, tftag); 331
        for px:= closing npx, closing tpx do                        332
            inspect closing loop product[px]when polynomial do if ptag ≠ 0 then 333
                begin include(polynomial, term factors);           334
                    include(extended product, term factors);       335
                    lptag:= ptag;                                    336
                    inspect closing polynomial[px]when polynomial do 337
                        closing polynomial[px]:-                    338
                            product(term factors, paddress, pdegree, ptag, tftag * lptag)
                end
            end
        end;
    next branch:                                                    339
        possible pointers into branch[this bx]:= possible pointers into branch[this bx] - 1 340

```

```

    end;
    adjust tag of all branches connected to this node by(+1);
    pointer set[n] := false
end set pointer;

unit polynomial:- new polynomial(stack address, 0, 1);
inspect branch reference[closing bx] when branch do for px:= 1 step 1 until 6 do
    closing polynomial[px]:- new polynomial(bpaddress[px], bpdegree[px], bptag[px]);
for bx:= 1 step 1 until branches do
    begin p:= 0;
        node sum of branch[bx] := first node[bx] + second node[bx];
        if <the three polynomials 3,4 and 5 of this branch are not all zero> then
            begin include(new directed branch(bx, false), branch list of node[first node[bx]]);
                p:= p + 1
            end;
            if <the three polynomials 1,3 and 6 of this branch are not all zero> then
                begin include(new directed branch(bx, true), branch list of node[second node[bx]]);
                    p:= p + 1
                end;
                possible pointers into branch[bx] := p
            end;
        for nx:= 1 step 1 until nodes do pointer set[nx] := false;

    set pointer(1,1,1, unit polynomial)
end topologic;

```

The branch whose index is denoted by "closing bx" is the closing branch. Its polynomials must be set initially to zero, given positive tags, and located at addresses to which the polynomials of the new, complete network are to be assigned. If either or both of its ports are connected in series then its polynomials are interchanged in the same way as those of any other branch. Then the relationship between a complete network and its closing network is taken into account by interchanging the three pairs of polynomials $\{1,4\}$, $\{2,3\}$ and $\{5,6\}$, and by changing the signs of the tags of both the transfer polynomials.

Within the context of algebraic reduction the topological analysis procedure is required to assemble the polynomials of a "basic network". The prior interchanging of closing polynomial addresses described above ensures that at the termination of the topological analysis the accumulated polynomials will represent the basic network and be stored in the correct locations.

The addresses of storage space for up to six temporary polynomials (at the right-hand end of the first six rows of the main array) are given by the array "tempadr_s[1:6]". Intermediate polynomial products are to be stacked behind a unit polynomial of degree zero at the address given by "stack address" (in the last two rows of the main array).

5.6.2 Action of the Algorithm

Throughout the course of the algorithm the three attributes — address, degree and tag — of any polynomial are conveniently manipulated as a single entity declared as a "polynomial" in statement 113. The main procedure commences with the creation of seven such entities representing the unit polynomial which heads the stack of intermediate products (400), and the six closing polynomials (401,402).

The data describing the topology of a structure is more useful in other forms. When given any one node of a branch the other node is readily found (278) using the sum of the two node numbers. It is also convenient to

list all the branches connected to a particular node, and to record which ends of the branches are connected to that node. For this purpose the concept of a "directed branch" is introduced (114); its attributes determine a particular "branch" (by its index "bx") and indicate whether the relevant node, in whose list the directed branch appears, is the first or second node of the branch ("breversed" is "false" or "true", respectively).

Node numbers are summed, directed branches are created, and branch lists are assembled by the statements 405, 407 and 410. Note that if all the polynomials that might be determined by directing the pointer of a node into a branch are zero, that branch is not included in the branch list of that node (406, 409).

Also before the algorithm commences, tags associated with each branch and node must be initialised: it is noted that at this stage of the algorithm none of the pointers are set (413), and it is therefore possible for any branch to have the maximum number (1 or 2) of pointers directed into it (412).

The aim of the algorithm is to generate all pointer settings, and, for each one, calculate a product of polynomials and add it to a closing polynomial as determined by equation 1.7. With only a partial setting of pointers certain factors of equation 1.7 will be determined; their product is calculated, stored as an intermediate polynomial, and referred to as the "partial product".

The algorithm proceeds by setting pointers one at a time and is initiated by a single call of the procedure "set pointer", with parameters specifying that the first pointer to be set is that of the first node, that the current pointer path commenced with the first pointer to be set, and that the partial product of polynomials at this stage is the unit polynomial (414).

The setting of one more pointer determines more factors of equation 1.7. As the factors are determined they are included in the set of "term factors"

and it is only after all the new factors have been included, and it is known that none of them is zero, that the factors are multiplied together and a new "partial product" is formed. This policy is followed throughout the algorithm: no manipulation of polynomials is undertaken until it is assured that such manipulation will be productive. At each setting of a single pointer three different polynomial products may need to be formed and three sets are allocated for this task (117). The tag of a prospective polynomial factor is inspected and if $\neq 1$ the polynomial is not included in the set. If the tag is zero the entire set may be neglected and no further polynomials included. In this way the occurrence of zero or unit polynomials is exploited to the fullest extent in speeding up the algorithm.

The element procedure "product" (240) provides the bridge between the polynomial manipulation requirements of the algorithm and the utility routines of section 5.2. Its function is to multiply together all the polynomials included in a specified set and to either add the resulting polynomial to a specified polynomial or store the product at a specified location. Intermediate products are stored alternately in the space allocated for the first two temporary polynomials (255, 256).

The inclusion of constituent polynomials in the set of "term factors" is performed by the procedure "register polynomial" (210). A tag ("tftag") for the product of members of the set is revised (214) and if it becomes zero (215) the setting of the last pointer is rejected (216). If the polynomial to be registered belongs to the closing branch (211) then a note is made (212) of the closing polynomial to which the full product of constituent polynomials is to contribute.

Throughout the course of the algorithm a list of the nodes whose pointers have been set and a list of the branches to which they are directed are maintained in the arrays "listed node" and "listed branch" (112). The boolean array "listed branch reversed" (111) indicates whether the listed node corresponding to the listed branch is its first or second node.

When the procedure "set pointer" (200) is entered the "pointer set" flag of node n is raised (263) and this node becomes the k -th "listed node" (264). Until the pointer is actually directed to a particular branch the tags of all the branches connected to this node are reduced by one (265). The pointer is then directed in turn to each of the branches connected to this node (266-340).

With each setting of this pointer the set of "term factors" is initialised to include only the "partial product" (268-270). The branch to which the pointer is directed becomes the k -th listed branch (273,274) and its tag is increased by one (272) to cancel the adjustment made in statement 265. All the branches connected to node n are scanned (275) and if it is determined that now no pointers can be directed into one of these branches then its second polynomial is included in the set of "term factors" (277).

Pointers are set in an order which attempts to follow the formation of pointer paths. The current pointer path, which commenced with the k -th listed node, where $k = \text{"pathstart"}$, is extended until it intersects either a previous path or itself. The next node in the current pointer path is indicated by "next n " (278). If "next n " is not among the listed nodes then it is possible to extend the pointer path (320). In this case (323) the polynomials in the set of "term factors" are multiplied together to form a new "partial product" and the procedure "set pointer" is called recursively (327). However, if "next n " is among the listed nodes the current pointer path can not be extended; therefore, before the procedure is called again a search must be made for a new node (326) with which to start a new pointer path (325).

When all pointers have been set the product of "term factors" is added to the appropriate natural closing polynomial (330). However, if the pointer setting was such that the closing branch was traversed by some pointer loop then two different polynomial terms are determined and must be added to the appropriate natural and transfer closing polynomials.

In this case the product of "term factors", which is common to both terms, is calculated and stored in the space allocated to the fourth temporary polynomial, and is referred to as the "extended product" (331). The remaining factors of the two terms were previously calculated by the algorithm (305,306), stored in the space allocated to the fifth and sixth temporary polynomials, and each referred to as a subscripted "closing loop product". For both terms (332), the appropriate "closing loop product" (334) and the common "extended product" (335) are included in the cleared set of "term factors" and their product is added to the appropriate closing polynomial (338).

If the current pointer path terminates because "next n" is found among the listed nodes corresponding to previous pointer paths (317), each branch traversed by the path contributes a natural polynomial to the set of "term factors". These polynomials are registered by a call (318) to the procedure "register path" (220).

If "next n" is found among the listed nodes corresponding to the current pointer path (279) then a pointer loop is detected. The natural polynomials determined by that part of the path which is not included in the loop are registered by statement 280. If the loop includes only the last listed branch (281), it is not a proper pointer loop but simply a reversal of direction; this branch has both its pointers directed into it and therefore contributes its third polynomial to the set of "term factors" (282).

To handle a proper pointer loop the two sets reserved for loop polynomials are cleared (282, 284), the branches traversed by the loop are scanned (285), and the appropriate (287-289) natural and transfer polynomials are included in their respective loop-polynomial sets (301, 303).

It is at this stage that a most important decision is made which affects both the economy and the accuracy of the algorithm. If the corresponding natural and transfer polynomials are identical in magnitude (298), the two

loop products have a common factor; this factor is therefore included directly in the set of "term factors" (299) rather than included in both sets of loop polynomials.

Normally, if the loop does not traverse the closing branch, the "loop natural polynomials" are multiplied together, their product is stored in the space allocated to the third temporary polynomial, and the product is referred to as the "loop polynomial" (310). The "loop transfer polynomials" are then also multiplied together and the resulting product is subtracted from the "loop polynomial" (312). The difference, representing a factor $(n_i - t_i)$ in equation 1-7, is included in the set of "term factors" (314, 315).

But, if both sets of loop polynomials are empty (308) this setting of the last pointer is rejected (309). This situation can only occur if the corresponding natural and transfer polynomials of every branch in the loop are identical in magnitude, and in such a case the two loop products would be equal and their difference $(n_i - t_i)$ would be zero. However, if both products were computed, in practice it is likely that, due to truncation errors, their difference would not be exactly zero. The resulting products could be of sufficient magnitude to render the analysis results meaningless.

If the loop does traverse the closing branch (304), the two products of the "loop natural polynomials" and the "loop transfer polynomials" are stored temporarily (305, 306) until the pointer setting is complete and they become factors (334) in two different terms (332).

Before the pointer of node n is directed to another branch the tag of the current branch is reduced by one (340), to cancel the adjustment made in statement 272. When the pointer has been directed to all the branches connected to this node their tags are increased by one (341) to cancel the original reduction (265), the "pointer set" flag is lowered (342), and the procedure is terminated.

5.6.3 Application of the Algorithm

The algorithm presented here is designed primarily to analyse general structures of general, constituent 2-port networks, but, wherever possible, the occurrence of zero or unit polynomials is exploited to minimise computation. In practice, many trivial networks may need to be introduced to allow a complex network to be represented as a structure of 2-port networks, and the occurrence of zero and unit polynomials is therefore common.

Because of its ability to analyse structures of 2-port networks this algorithm is believed to be a significant advance over the existing topological methods of analysis, which either enumerate trees of a network graph or analyse signal-flow graphs. The power of this algorithm is better appreciated by noting that both network graphs and signal-flow graphs can be transformed, quite simply, into particular types of structure graphs and therefore can be analysed by this one algorithm.

Network Graphs

To find the trees of a graph whose branches are either resistors, unistors or gyrators each branch is modelled as a "simple network" with its component in the series position; each node becomes a parallel interconnection of ports. The equivalent structure graph is then identical to the network graph.

The six polynomials of a branch are $y_{21}, 1, 0, y_{12}, y_{12}, y_{21}$ where y_{12} is the admittance of an equivalent unistor directed from the first to the second port and y_{21} is the admittance of an equivalent unistor directed from the second to the first port, as in figure 5.4. For a resistor, $y_{12} = y_{21}$; for a gyrator, $y_{12} = -y_{21}$.

If node g is the "ground" node as far as the behaviour of the unistors is concerned, the port corresponding to this node must be collapsed. Therefore, if a complete 2-port network is defined by connecting a closing branch from node g to any other node, it is its first polynomial

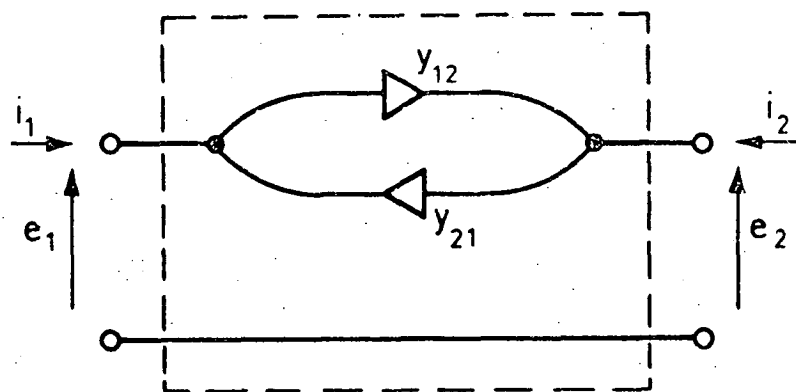


Figure 5.4 An equivalent 2-port network representing the general branch of a network graph.

which will correspond to the branch-admittance-product-sum of the trees of the original network graph.

The analysis algorithm has been tailored specifically to this task of generating the trees of a graph in order to investigate its effectiveness in comparison with other tree-finding schemes.

Because only one polynomial is required, the pointer of the "ground" node is directed permanently into the closing branch and the pointer at the other end of the closing branch is not permitted to be directed into the closing branch. Only one polynomial, the branch admittance, need be associated with each branch and all information needed to distinguish the six polynomials is contained in the polynomial tags.

In practice, polynomial manipulation is eliminated and the branches in a tree are determined by the "listed branch" array; the sign of a branch-admittance-product is determined by the tag "tftag" associated with the set of "term factors", or may be determined from the "listed branch reversed" array.

Because every transfer polynomial is identical to its corresponding natural polynomial, and the third polynomial of every branch is zero, the formation of a pointer loop or the reversal in direction of a pointer path will always result in a zero product. This observation is, of course, consistent with the definition of a tree and allows a major part of the algorithm (280-316) to be greatly simplified. Further, because the second polynomial of every network is unity, all statements related to the adjustment and interpretation of branch tags are eliminated.

A paper describing such a simplified, tree-generating algorithm [7] is appended to the thesis.

In the course of investigations by an independent worker, the algorithm was evaluated in comparison with a representative selection from more than thirty other tree-generating methods [23]. The most promising methods were programmed in assembler for an IBM 360 computer, and used to find the trees of a dozen representative networks of varying size and complexity. In all cases the performance of the program based on this algorithm was close to the best in each particular test, and in five documented tests it generated all trees in the least time.

Signal-Flow Graphs

To analyse a signal-flow graph it is formulated as an electrical network with quantities represented by voltages. Branches become ideal voltage amplifiers with gain equal to the branch transmission and nodes become interconnections of ports.

Because the amplifiers which represent the branches directed away from a particular node all share the same input voltage, representing the quantity at the node, their first ports are connected in parallel. But the branches directed toward a particular node are represented by amplifiers whose output voltages must be added together, and this is accomplished by connecting their second ports in series.

The electrical analogue of the signal-flow graph of figure 5.5 is shown in figure 5.6. Its structure graph (figure 5.7) is similar to the signal-flow graph but, in general, each node of the original graph is replaced by both a series and a parallel node: incoming branches are connected to the series node, outgoing branches are connected to the parallel node, and the two nodes are joined by a "unit network" directed from the series to the parallel node.

When analysed by the algorithm the series ports are converted to parallel ports, the "amplifiers" are converted to voltage-controlled current sources, and the "unit networks" are converted to unit gyrators.

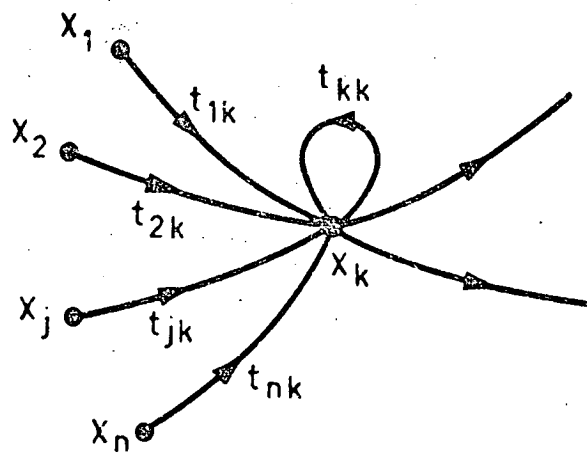


Figure 5.5 Signal-flow graph.

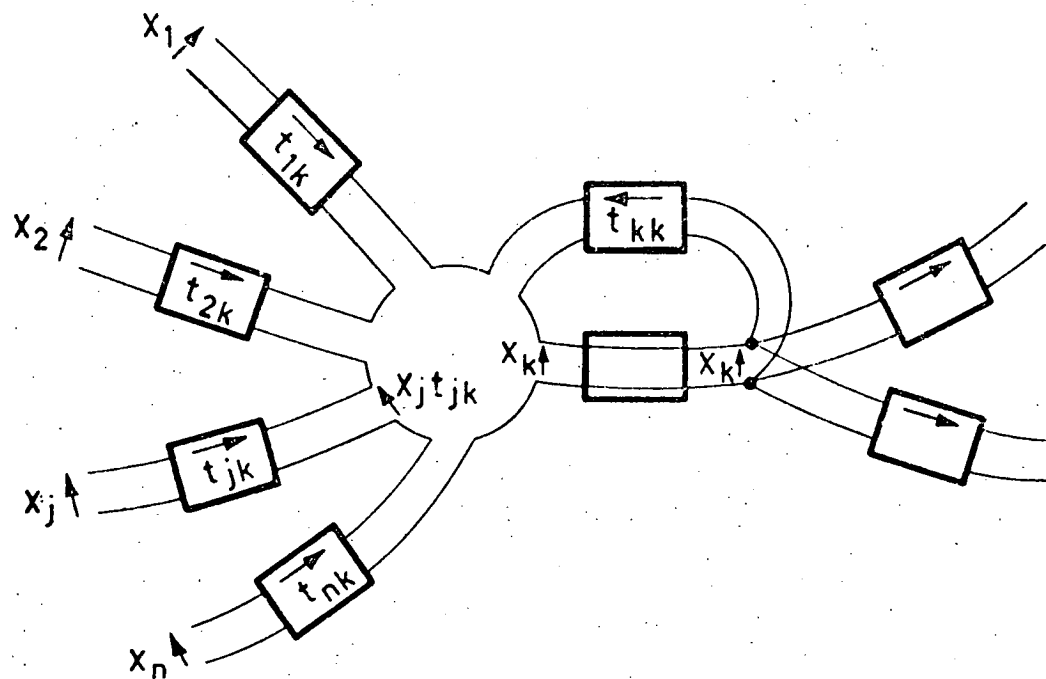


Figure 5.6 Electrical analogue of signal-flow graph.

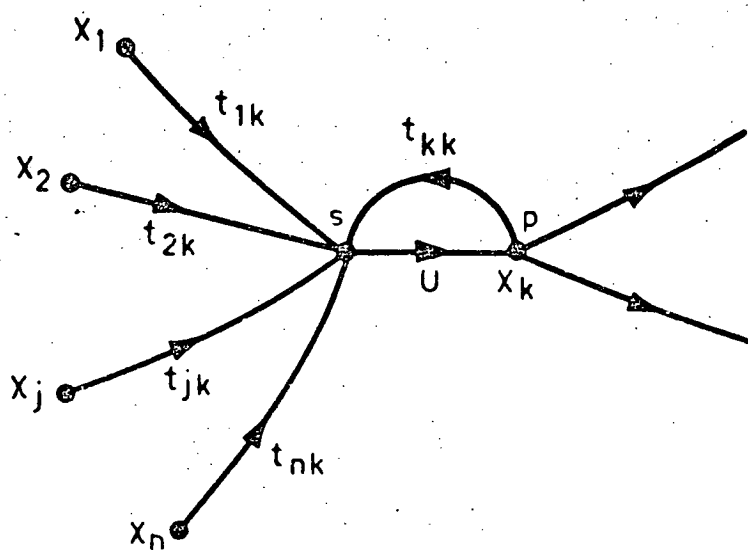


Figure 5.7 Structure graph corresponding to signal-flow graph.

For a transmission branch its polynomials numbered 1,3,4 and 6 are all zero and a pointer need never be directed to its second port; consequently the pointers of all the series ports are directed permanently toward their corresponding parallel ports. Further organisational simplification may be achieved because all "loop natural polynomials" are zero.

If the denominators (polynomial 2) of the branch transmittances are restricted to unity, all statements concerned with the adjustment and interpretation of branch tags can be eliminated (as with network graphs) and all path segments which are not part of some pointer loop will result in a zero product. In this case the algorithm seeks pointer loops only and evaluates the Shannon-Happ formula which expresses the transfer function of a signal-flow graph in terms of its loop transmittances. This method of detecting loops is apparently unique and certainly bears no resemblance to the routine used in a major version of the analysis program NASAP [45].

However, there is little practical justification for adapting the general algorithm specifically to signal-flow graphs. Even if the algorithm is applied to signal flow graphs in its present form there is little unnecessary setting of pointers, no unnecessary manipulation of polynomials, and branch transmissions may be specified as ratios of polynomials. In many applications the existence of the general algorithm may obviate the use of a signal-flow graph to describe the system; the components of a control system, for example, may actually be 2-port devices and the complete system may be described more compactly if represented directly as a structure of 2-port networks. (see section 6.5).

5.7 CONCLUSION

This chapter presented those ingredients which are believed to be essential to any successful implementation of the new approach to linear analysis. Together they permit analysis on two distinct levels.

The lower level is designed to handle the simplest forms of subnetwork interconnections —whether they be series-parallel connections of 2-terminal networks, cascade-parallel connections of 2-port networks, or chains of signal-flow sub-graphs with simple feed-back loops. The language with which simple structures are described has been made highly versatile, largely through the adoption of algebraic concepts which permit the nesting of network-manipulative expressions to an almost unlimited extent. In particular, 2-terminal networks may be converted to 2-port networks and vice versa, and structures of 2-port networks may be analysed on the higher level and the resulting networks manipulated further on the lower level. Consequently the greater economy of analysis on the lower level is available at any stage in the analysis of large networks.

The higher level concerns the analysis of any structure of 2-port networks using the topological method introduced in chapter 1. The computer algorithm embodying this method (section 5.6) is the most important contribution of part II of the thesis. It supersedes the conventional topological methods, which analyse either network graphs or signal-flow graphs, because it not only performs the same type of analysis but offers more convenience to the user and can handle larger networks.

DEMONSTRATION OF COMPUTER PROGRAMS

6.1 INTRODUCTION

The computer algorithms discussed in the last chapter are the culmination of many working computer programs which followed the evolution of ideas and demonstrated the feasibility, or otherwise, of various analytical and programming techniques. The most recent versions, which have been documented and maintained for general use by the Department of Electrical Engineering [8], were used for the demonstrations reported in this chapter.

When coding these programs much effort was given to reducing computation time and saving core space. Although written principally in ALGOL, they include some machine-code instructions; they calculate addresses of polynomial coefficients explicitly rather than with the normal subscripting process; and they do not call procedures recursively. Nevertheless, the reported execution times are indicative of what can be achieved and should be easily bettered with more modern computers.

The coding of these programs has not been documented because of their strong machine-dependence, their divergence from the documented algorithms, and their evolutionary nature - which is responsible for the now-inappropriate choice of identifiers. The grammar of the input language has also been modified slightly since the programs were written, and in the following reports the actual input data has been altered to comply with the syntax rules of section 5.5.1.

The Elliott 503 computer on which the programs were run is a second generation machine. It has hardware for floating-point arithmetic, 8k words of main core store, 16k words of core backing store, a 300 line/min. line printer for output, and a 1000 char/sec. paper tape reader for input. A single-address instruction for integer addition is performed in 7.2 μ S. With 39 bits in a word, floating-point numbers are represented with an accuracy of approximately 9 decimal digits and their magnitude cannot exceed 10^{77} . Programs were run under a compile-and-go operating system and had access to approximately 6k words of the main core store.

Many aspects of the programs are outside the scope of the thesis and are only mentioned briefly here.

In all but the third demonstration the programs calculate network polynomials and then evaluate them to arrive at a frequency response, a transient response, or poles and zeros. Due to the main-store limitations of the computer, polynomial evaluation is performed by a separate program which follows the analysis program and retrieves the polynomials from the backing store. The evaluation program has facilities for (1) cancelling common polynomial factors, using an adaptation of the Euclidean Algorithm; (2) tabulating frequency response, including amplitude, phase and group delay; (3) finding roots of polynomials using Bairstow's method; (4) calculating the residues of poles; and (5) tabulating transient response. A second version of the evaluation program tabulates frequency response and finds the roots of polynomials whose coefficients have been transformed according to the process described in section 4.10.

An early version of the analysis algorithm was developed for repetitive execution within an iterative network-design program. Analysis was directed by a compiled form of the network expression and in each iteration produced the relevant polynomial coefficients and their first derivatives with respect to all the variable parameters. Modification of parameters was determined by a version of the Newton-Raphson process [12] which aimed to realise desired locations for poles and zeros.

This program demonstrated the suitability of the algebraic reduction process in this role, and proved to be a useful tool for network realisation. For example, in one simple application it adjusted two independent parameters of a "poorly designed" oscillator circuit so that it oscillated at a specified frequency.

However, in some applications, and particularly with symmetrical networks, the matrix of partial derivatives tended to be singular and the Newton-Raphson process became unstable. This problem was not pursued at the time but the

recent experience of many people, including the author [9], suggests that the synthesis problem is better approached with an arsenal of general-purpose function-minimisation routines including, for example, steepest-descent and a conjugate-gradient technique.

6.2 LARGE PASSIVE FILTER

The filter analysed in this example has many common subnetworks which, in the program data, are assigned to separate blocks as follows:

B1 = (TC1.245E-6 c TR5E6 c TY(SL6.72E-3 c SR1))

ss TY(SL.796 c SR87) ss (TC.01055E-6 c TR5E8);

B2 = ((SL3.78E-3 c SR.6) pp SZ(TC2.214E-6 c TR3E6))

c ((SL6.3E-3 c SR1) pp SZ(TC1.33E-6 c TR4E6));

B3 = B1 c B2 c (TY(SL.239 c SR26) ss (TC.0352E-6 c TR2E8)) c B2 c B1;

B4 = (TY(SL.951 c SR104) ss (TC.0074E-6 c TR1E9))

c (TY(SL1.098 c SR120) ss (TC.0088E-6 c TR8E8));

B5 = SL.764 c SR83 c SZ(TC.011E-6 c TR4E8);

B6 = B4 c B5 c TY(SL3.16E-3 c SR.5) c TC2.65E-6 c TR2E6 c B5 c B4;

B7 = (B3 c B3) pp B6;

B8 = SR600 c B7 c B7 c TR600 c F2

The corresponding networks are shown in figure 6.1.

The use of compound components such as TY(SL6.72E-3 c SR1) and SZ(TC2.214E-6 c TR3E6), which are equivalent to (TL6.72E-3 ss TR1) and (SC2.214E-6 pp SR3E6) respectively, is necessary to avoid floating-point overflow or underflow during the analysis process. As a general rule it is preferable to introduce small impedances such as series losses in the series position, and large impedances such as shunt losses in the shunt position; their parameters then have the least effect on the magnitudes of all the polynomial coefficients.

The filter was designed to separate a telegraph channel, centred on 1740 Hz, from an audio band. Block B6 is a band-pass filter and B3 is a band-stop filter. The parallel configuration of B7 is meant to be all-pass but some attenuation is expected at the edges of the telegraph band.

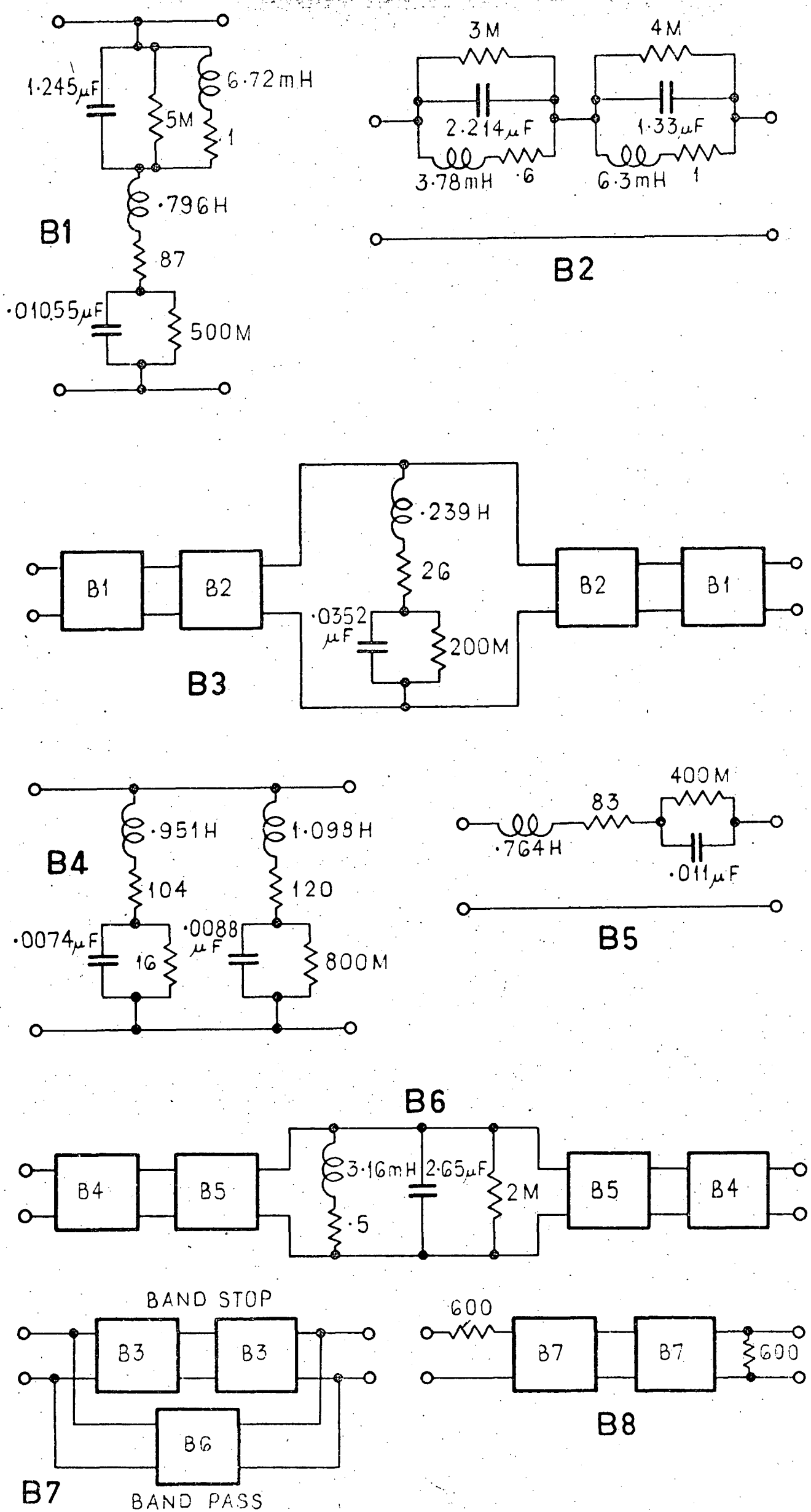


Figure 6.1 Large passive filter and its subnetworks.

For good measure two identical filters are cascaded; both ports are terminated with 600 ohms, and the terminated network is cascaded by an ideal transformer to increase the output voltage by a factor of 2. The transfer voltage ratio (P_5/P_1) of B8 therefore directly indicates the insertion loss of the filter. Altogether, the filter includes 202 components or branches, of which 100 are reactive, and has 102 nodes.

The program execution times were:-

for analysis (HUCC program no. U1049):

Computation	29
console message printing	<u>13</u>
total	42 seconds;

for evaluation of the transformed polynomials (HUCC program no. U1095/2):

computation	43
console message printing	<u>9</u>
total	52 seconds.

Interpretation and evaluation of network expressions was performed as the data was read from paper tape and for most of the time the program was input-bound. In order to assess the accuracy of the results the filter was analysed twice in the 29 seconds, using a frequency transformation with frequency scales of 1740 and 1741 Hz. Calculation of the two sets of polynomials of degree 100 required a total of 81,336 coefficient multiplications. In 43 seconds the polynomial evaluation program printed the coefficients of the numerator and denominator polynomials of the voltage transfer function and tabulated the response at 50 frequencies, for both analyses. This program was output-bound at all times.

Samplings of the results of the two analyses are compared in table 6.1.

FREQUENCY Hz	INSERTION LOSS decibels		PHASE degrees	
1500	7.9952	7.9987	-9.79	-9.79
1550	9.4370	9.1801	-107.42	-106.22
1600	57.041	67.298	-48.07	-176.04
1650	50.715	60.002	-82.75	58.71
1700	11.712	11.713	-144.51	-144.51
1750	8.2670	8.2671	-56.65	-54.65
1800	22.736	22.712	27.66	28.31
1850	59.586	63.370	-7.17	52.16
1900	53.468	63.449	-125.65	-129.07
1950	9.8267	9.8571	112.86	112.99
2000	9.7746	9.7744	29.15	29.15

Table 6.1. Results of two analyses of the large passive filter.

This most severe test of the analysis program clearly indicates the limitations of the method. The results are tolerable within the pass bands but unacceptable at the edges of the telegraph band. However, with the longer word length or hardware double-precision offered by large computers such as the CDC6000 series (60-bit words), ICL System 4, IBM System 360 (64-bit double words), and UNIVAC 1108 (72-bit double words) the results should in this case be acceptable at all points of interest in the frequency band. With only a ten-fold increase in processing speed the calculation of the filter polynomials should be completed in less than a second, and their evaluation at, say, 50 frequencies would take a fraction of a second.

Such projected performance invites comparison with ECAP, the most widely known and oldest of the general circuit analysis programs. Unfortunately, this large filter exceeds by a wide margin the capacity of most versions of ECAP — even though they use at least five times the core store than does this program. The possibility of solving 100 nodal equations at only one frequency in one second of processor time — whether by Crout's method or some sparse matrix technique — is a matter for uncertain conjecture.

6.3 THREE-STAGE 1C AMPLIFIER

A three-stage integrated-circuit amplifier, together with its representation as a structure of 2-port networks and its structure graph, is shown in figure 6.2.

In the program data the three transistors are assigned to separate blocks B1 to B3 for convenience, but blocks B4 to B8 are necessary to describe the branches of the structure graph which appears as a basic network in the expression for the complete network B9:

```

B1 = SR1E3 c ((TC6.3E-12 c TR25E3 c H[ 0,-4E-3,1] )
      pp SC1.5E-12) c SR150 c TC3.4E-12;
B2 = SR450 c ((TC.7E-12 c TR12.5E3 c H[ 0,-8E-3,1] )
      pp SC.6E-12) c SR100 c TC2.7E-12;
B3 = SR500 c ((TC1.2E-12 c TR6.25E3 c H[ 0, -16E-3, 1] )
      pp SC15E-12) c SR20 c TC4.6E-12;
B4 = B3 pp SC [C] ;
B5 = SR6E3 pp SC.16E-12;
B6 = SR3E3 pp SC.08E-12;
B7 = TR150 c ((SR470 c ((SR1E3 c SL [L] ) pp SC25E-12)) pp SR560E3);
B8 = U;
B9 = SR5.6E3 c D[B0:1s4p, B1:1s2p, B2:2p3p,
      B4:3p4p, B5:2p5p, B6:5p3p, B7:5p4p, B8:5pls] ;

```

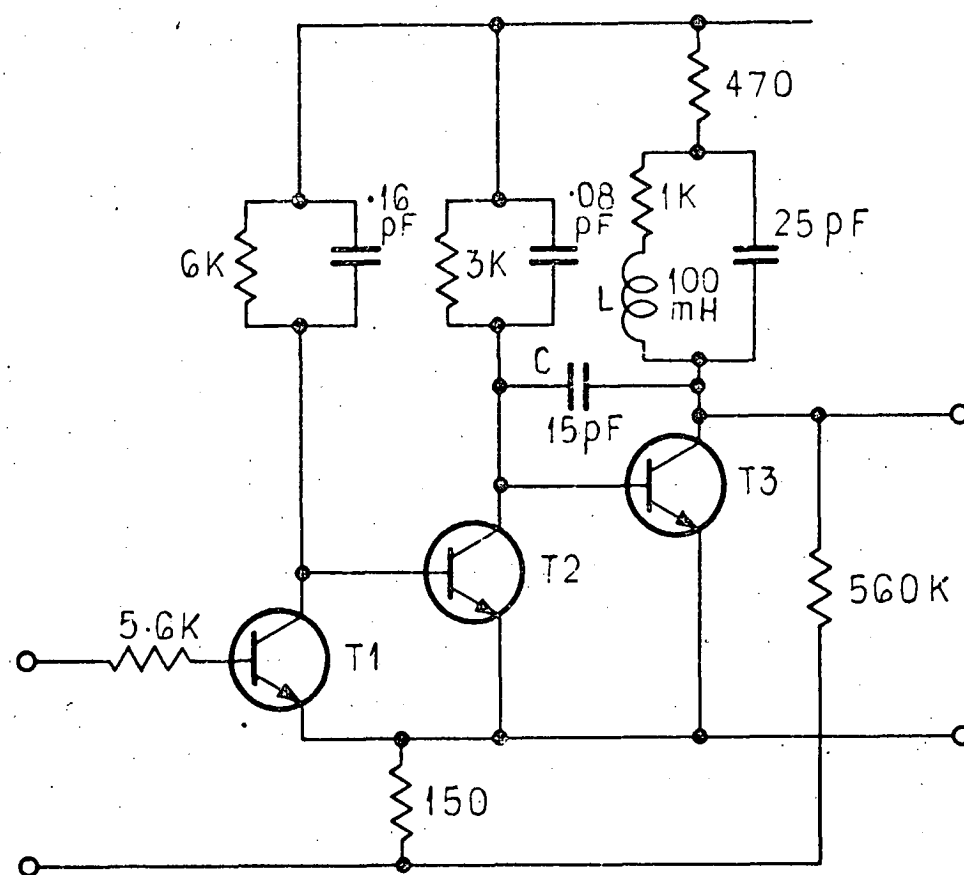
Two parameters, in the expression for B4 and B7, are introduced as identifiers but are subsequently assigned the numerical values:

C = 15E-12;

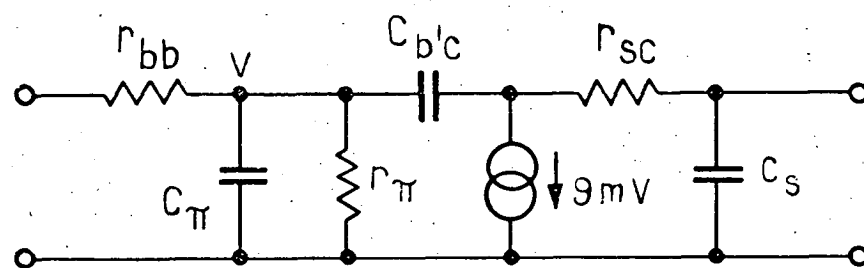
L = 100E3.

This example demonstrates an analysis program designed for parameter sensitivity studies. Parameters introduced by their identifiers are regarded as variables and the analysis program calculates network polynomials and their partial derivatives with respect to the logarithms of each variable parameter; that is, for each polynomial P and variable parameter k it calculates the polynomials

$$\frac{\partial P}{\partial \ln k} = \frac{\partial P}{\partial k} \cdot k .$$



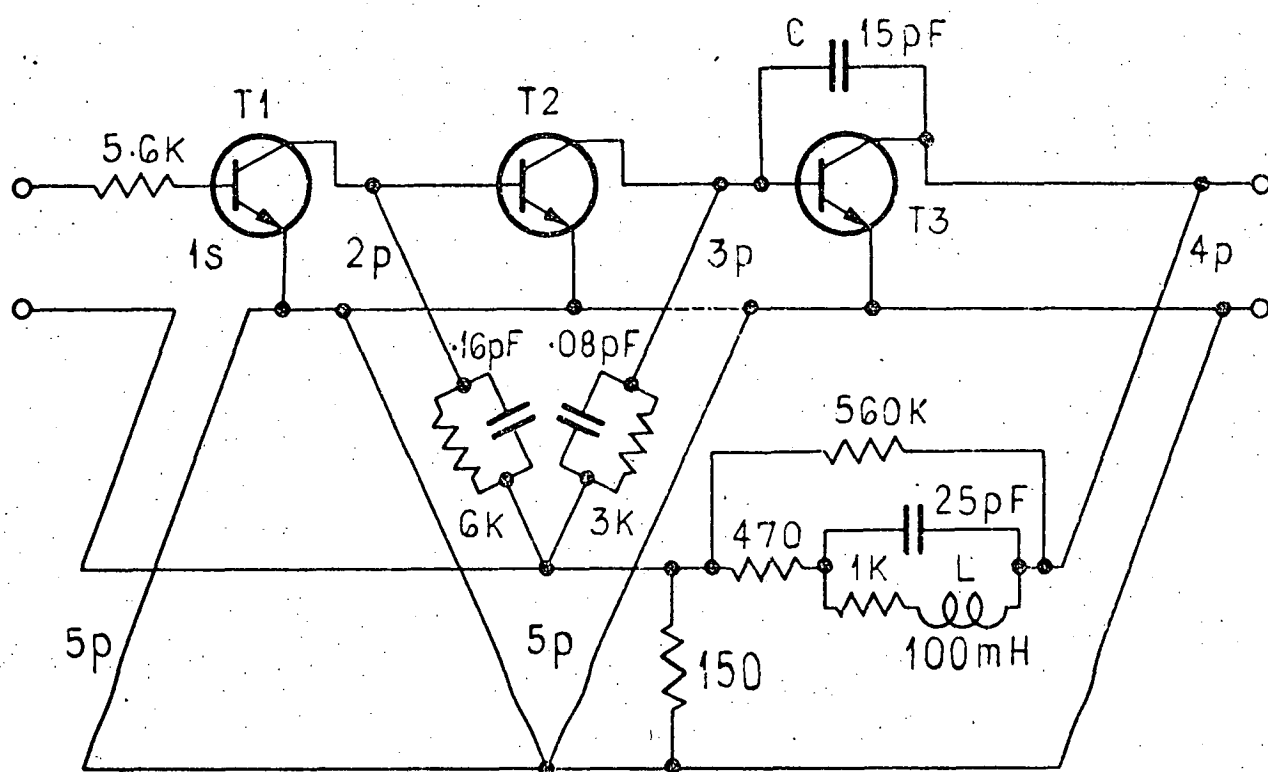
(a)



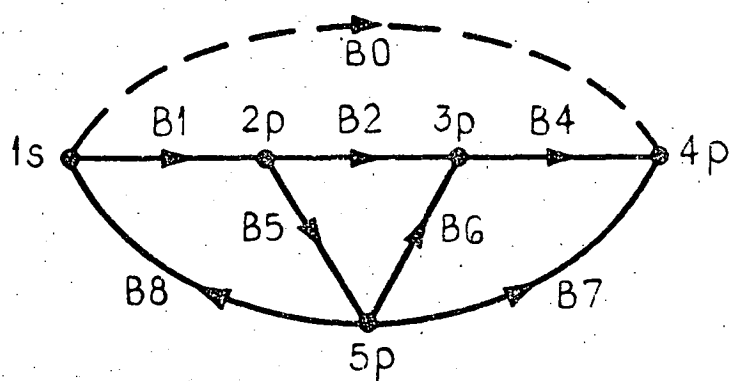
	T1	T2	T3
r_{bb}	1K	450	500
r_{sc}	150	100	20
$C_{b'c}$	1.5 pF	.6 pF	15 pF
C_s	3.4 pF	2.7 pF	4.6 pF
C_{π}	6.3 pF	.7 pF	1.2 pF
r_{π}	25K	12.5K	6.25K
9m	4 mA/V	8 mA/V	16 mA/V

(b)

Figure 6.2 Three-stage IC amplifier : (a) circuit diagram
(b) transistor models



(c)



(d)

Figure 6.2 Three-stage IC amplifier : (c) structure of 2-port networks
(d) structure graph.

To do this it gives a third dimension to the main array of polynomial coefficients by adding extra layers which are simply the partial derivatives of the first layer. An extra layer is added for each variable parameter.

The only other major modifications to the basic program involve (1) changes to the procedures such as "pma" and "product" which call the utility routines, and (2) enlargement of the attributes of each "network" and "branch" to indicate those variable parameters of which it is a function.

The complete network has 13 nodes, 30 components, 3 controlled sources, and polynomials of degree 12.

The program execution times were:-

for analysis (HUCC program no. U1033):

computation	25
console message printing	<u>13</u>
total	38 seconds;

for evaluation of polynomials (HUCC program no. U1095):

computation	13
console message printing	<u>9</u>
total	22 seconds

When calculating polynomials the program was mostly input-bound. Algebraic reduction required only 697 coefficient multiplications, but the topological analysis of the structure graph required 27,620 multiplications.

Although the maximum number of pointer settings—the node-degree-product of the graph—is 324, only 205 settings resulted in nonzero polynomial products and only 221 products were added to the closing polynomials. This saving is due to the occurrence of zero branch polynomials as indicated by the following sets of tags allocated by the program to the branch polynomials:

B1: 3,3,3,3,-3,3
 B2: 3,3,3,3,3,3
 B4: 3,3,3,3,3
 B5: 3,3,0,3,2,2
 B6: 3,3,0,3,2,2
 B7: 3,3,3,3,3,3
 B8: 0,1,1,0, -1,1.

Evaluation of the polynomials was fully output-bound. The poles and zeros of the voltage transfer function were found and the sensitivities of the poles to both variable parameters were calculated.

Sensitivities are calculated with the same routines used for evaluating a normal transfer function. The sensitivity of a polynomial P to a parameter k is obtained by evaluating the ratio of polynomials

$$\frac{\partial P}{\partial \ln k} \bigg/ P \quad \left(= \frac{\partial \ln P}{\partial \ln k} = \frac{\partial P}{\partial k} \cdot \frac{k}{P} \right) .$$

If the Laplace transform is inverted prior to calculating the transient response to an impulse of a system with this pseudo transfer function, the calculated residue r_j of a pole p_j is the pole sensitivity $r_j = - \frac{\partial p_j}{\partial \ln k}$; if, instead, the transient response to a step input is requested then the residues give the sensitivities in the form $r_j = - \frac{\partial \ln p_j}{\partial \ln k}$.

The poles and their sensitivities are given in table 6.2. For example, comparison of the imaginary parts of the dominant pole and its sensitivities confirms that its frequency is almost inversely proportional to the square root of the inductance L ($3.173E5 \approx 0.4 * 6.327E5$).

POLE P	SENSITIVITY $\frac{\partial P}{\partial C} \cdot C$	SENSITIVITY $\frac{\partial P}{\partial L} \cdot L$
-3.622E4 ±6.327E5j	3.343E3 +1.473E4j	4.607E2 +3.173E5j
-8.256E6 ±4.142E7j	4.113E6 +7.919E6j	4.660E3 +3.617E2j
-1.998E8	-1.493E7	-3.195E2
-2.698E8	1.286E8	9.783E1
-1.297E8	1.425E8	-1.066E2
-2.553E9	5.161E5	9.262E1
-8.032E9	7.331E5	4.042E1
-4.065E10	2.750E7	-3.323E4
-4.541E10	1.967E8	7.326E4
-6.622E10	2.145E8	-2.009E4

Table 6.2. The sensitivities to parameters L and C of the natural frequencies of the three-stage IC amplifier with its input short-circuited.

The performance of this program might be assessed by comparison with other topological analysis programs such as CALAHAN [11] and NASAP [52]. Experience with tree-generating methods suggests that, even if the network for the amplifier was reduced to 10 nodes by the combination of branches in series and parallel wherever possible, more processing time than this 25 seconds would be expended in simply generating trees — without computing branch-admittance products, their sums, and their derivatives. Analysis of an equivalent signal-flow graph, which would have approximately twice the number of nodes, would be even more costly.

6.4 GENERAL CONVERTER

The circuit analysed in this example arose in an independent investigation of prototype circuits for generalised impedance converters and active transformers*. Figure 6.3 shows the prototype circuit (without biasing arrangements), its representation as a structure of 2-port networks, and its structure graph.

All four transistors are identical. The model used for a transistor in a common base configuration is shown in figure 6.4(a) and the other configurations are obtained from this network by interconnecting it with trivial networks as shown in figure 6.4(b) and (c).

The network expressions which specify the complete network B7 are:

$$B1 = SR25 \text{ c } (N [0,0,0,0,1,.99,0] \text{ ss } TR100);$$

$$B2 = (X \text{ c } B1) \text{ ps } U;$$

$$B3 = (B1 \text{ c } X) \text{ sp } U;$$

$$B4 = TR[R1] \text{ c } rB3 \text{ c } SR[R2];$$

$$B5 = SR[R3] \text{ c } B1 \text{ c } TR[R4];$$

$$B6 = U;$$

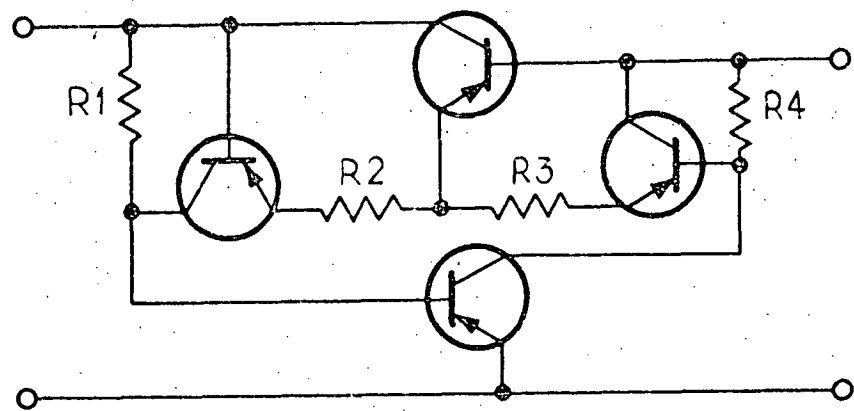
$$B7 = D[B0:1p11p, B2:3p9p, B2:7s5s, B4:2s4s, B5:8s10s,$$

$$B6:1p2s, B6:2s3p, B6:3p4s, B6:4s6p, B6:6p8s,$$

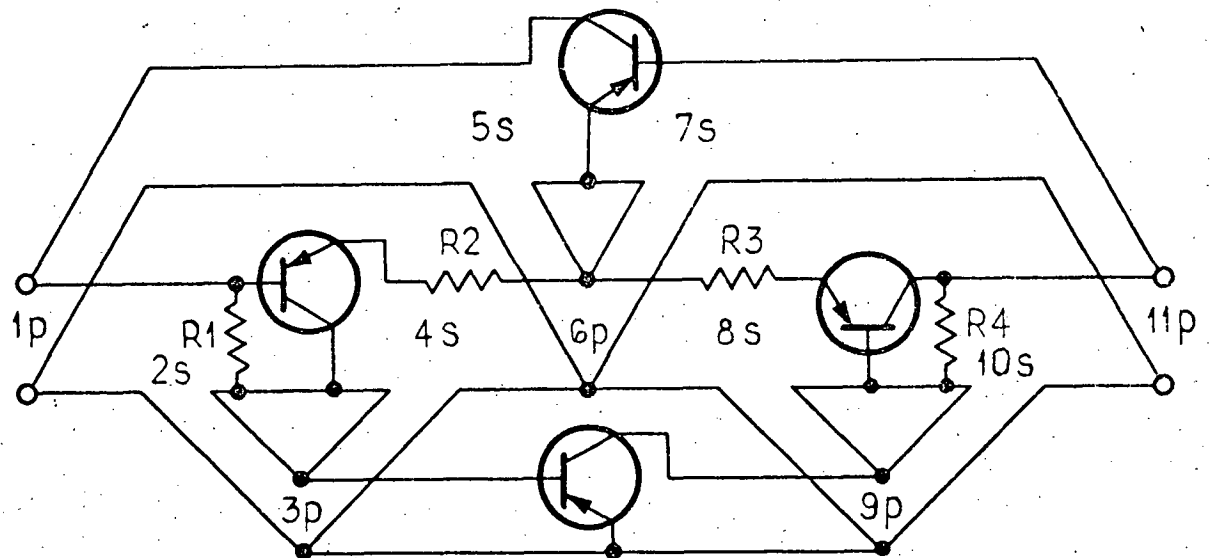
$$B6:8s9p, B6:9p10s, B6:10s11p, B6:1p5s, B6:5s6p,$$

$$B6:6p7s, B6:7s11p] .$$

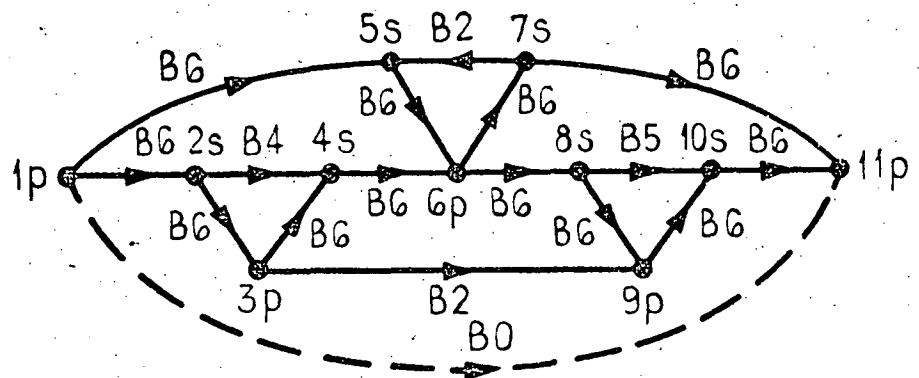
* This investigation was undertaken by Mr. R.S. Crocker, a contemporary post-graduate student in the Department of Electrical Engineering.



(a)



(b)



(c)

Figure 6.3 General converter : (a) prototype circuit
(b) structure of 2-port networks
(c) structure graph

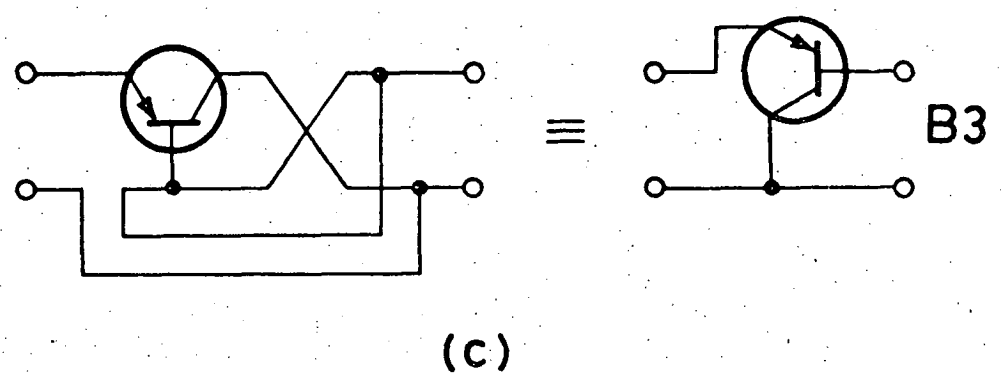
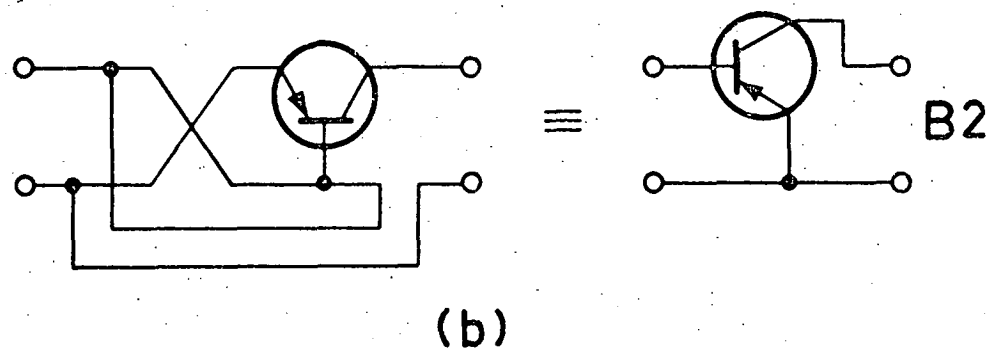
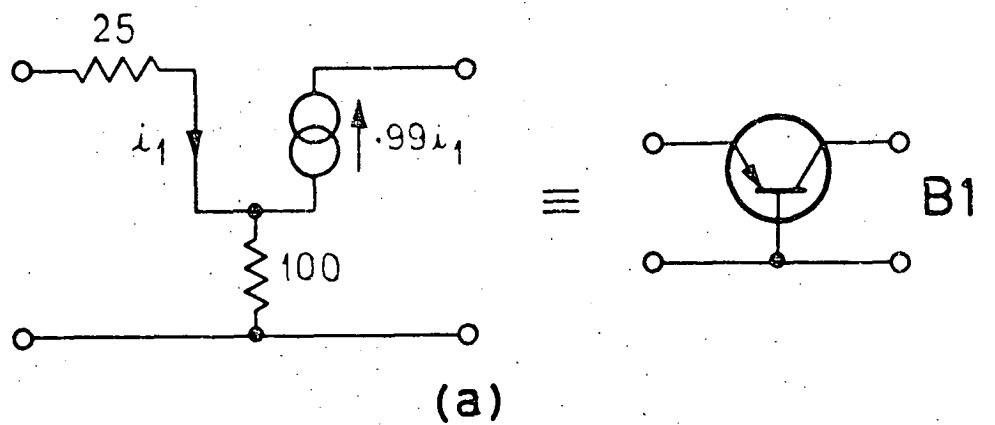


Figure 6.4

- (a) Equivalent circuit for transistor
- (b) Transformation from common base to common emitter
- (c) Transformation from common base to common collector

This example demonstrates a program designed for both numeric and symbolic analysis. The four parameters R1, R2, R3 and R4, introduced as identifiers in the expressions for B4 and B5, are manipulated as symbols rather than numbers throughout the analysis.

A polynomial is represented as a list of pairs of computer words: the first of each pair contains a floating-point numerical coefficient and the second contains an integer whose bit pattern records the exponent of s and the exponents (0 or 1) of up to 25 symbolic parameters.

The execution times for analysis (HUCC program no. U1068) were:

data input and algebraic reduction	5
topological analysis	46
lineprinter output	23
console message printing	<u>4</u>
total	78 seconds.

Algebraic reduction required 33 multiplications and the topological analysis required a further 957 multiplications. The following tags were assigned to the branch polynomials:

B2(3p9p):	0,3,0,3,3,0
B2(7s5s):	3,0,3,0,3,0
B4:	3,3,3,3,2,3
B5:	3,3,3,3,3,0
B6(all branches):	0,1,1,0,-1,1.

The structure graph has a node-degree product of 177147, but, due to the large number of zero branch polynomials, only 178 polynomial products were added to the closing polynomials.

To simplify the printing of polynomials and to make the presentation of results less cumbersome the program associates every symbolic parameter with a unique letter of the alphabet. For this network the program printed the

dictionary

$$A = 1/R_1$$

$$B = R_2$$

$$C = R_3$$

$$D = 1/R_4$$

The inversion of parameters R_1 and R_4 indicates that in the case of resistors introduced in the shunt position the program regards the conductance rather than the resistance as the relevant parameter. When printing polynomials the program is designed to group together the terms with a common exponent of s but in this case the polynomials are independent of frequency. The lineprinter output is shown in figure 6.5.

The results show that if the four variable resistances have values within an order of magnitude of 1000 ohms the six polynomials of the complete network can be roughly approximated by

$$P_1: 1 - CD = 1 - R_3/R_4$$

$$P_2: 0$$

$$P_3: 0$$

$$P_4: 1 - AB = 1 - R_2/R_1$$

$$P_5: (1 - AB).(1 - CB) = (1 - R_2/R_1).(1 - R_3/R_4)$$

$$P_6: 1$$

They confirm that with appropriate selections of parameters the network may be caused to behave either as a transformer, as any one of various types of negative-impedance converter, or as either of two types of controlled source.

6.5 POSITION CONTROL SYSTEM

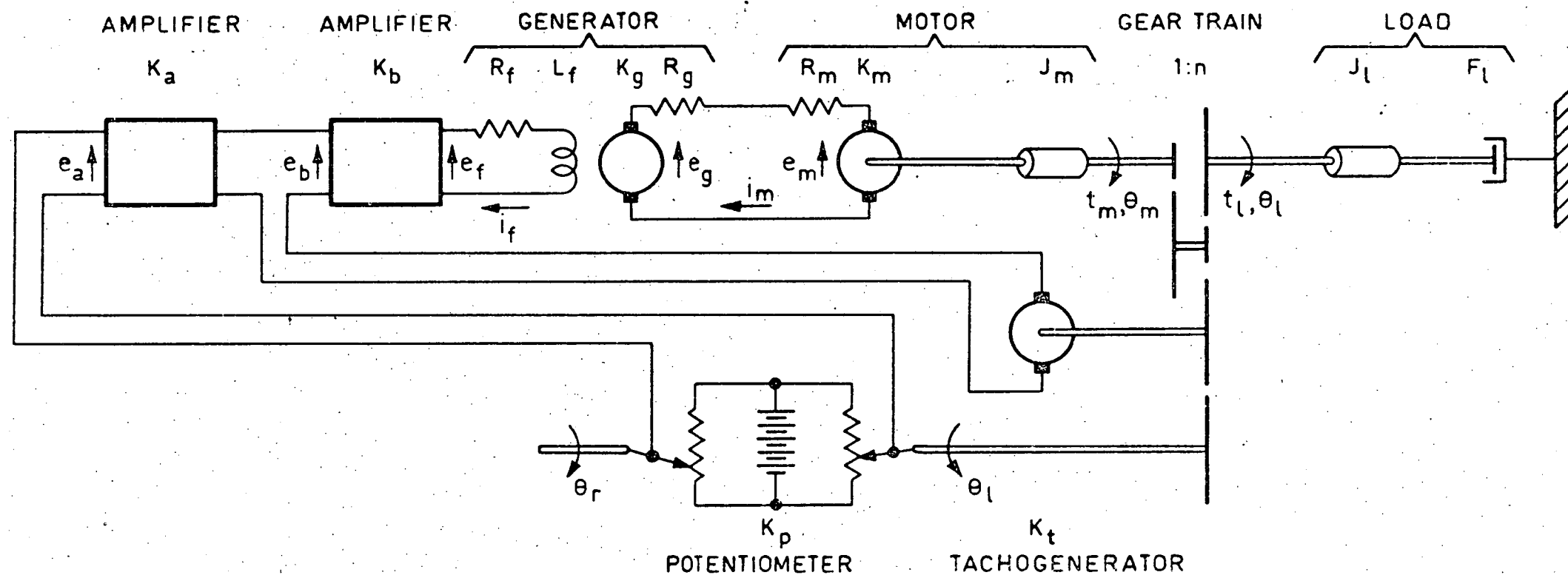
The position control system of figure 6.6 is represented for analysis in two ways: as a signal-flow graph and as a 2-port network. Comparison of the two illustrates the convenience of the new approach.

The signal-flow graph and the system equations it represents are shown in figure 6.7. Although pedantic, the task of establishing the equations is separated from the task of solving them in order to clarify the procedure. A graph of this complexity could be reduced on inspection by successively eliminating nodes of degree 2 and reducing the inner loops, but, as such a process is

GENERAL CONVERTER

POLY 6: +(-9.703970₁₀-09 +2.600054₁₀-11*A -2.522000₁₀-09*D +6.827600₁₀-06*AD -9.800000₁₀-11*CD +2.600000₁₀-09*ACD)

Figure 6.5 Lineprinter output for analysis of general converter.



POTENTIOMETER

TACHOGENERATOR

AMPLIFIER GAIN

AMPLIFIER GAIN

GENERATOR FIELD RESISTANCE

FIELD INDUCTANCE

E.M.F.

ARMATURE RESISTANCE

$$K_p = 1 \text{ VOLT}/10^\circ$$

$$K_t = .01 \text{ VOLTS/R.P.M.}$$

$$K_a = 1$$

$$K_b = 100$$

$$R_f = 50 \Omega$$

$$L_f = 5 \text{ H}$$

$$K_g = 200 \text{ VOLTS/AMP}$$

$$R_g = 24.4 \Omega$$

MOTOR ARMATURE RESISTANCE

BACK E.M.F.

TORQUE

INERTIA

GEAR RATIO

LOAD INERTIA

VISCOUS FRICTION

$$R_m = 24.4 \Omega$$

$$K_e = 1.25 \text{ VOLTS/RAD/SEC}$$

$$K_m = .812 \text{ FT. LB/AMP}$$

$$J_m = 8 \times 10^{-4} \text{ SLUG FT.}^2$$

$$n = 50$$

$$J_l = 1 \text{ SLUG FT.}^2$$

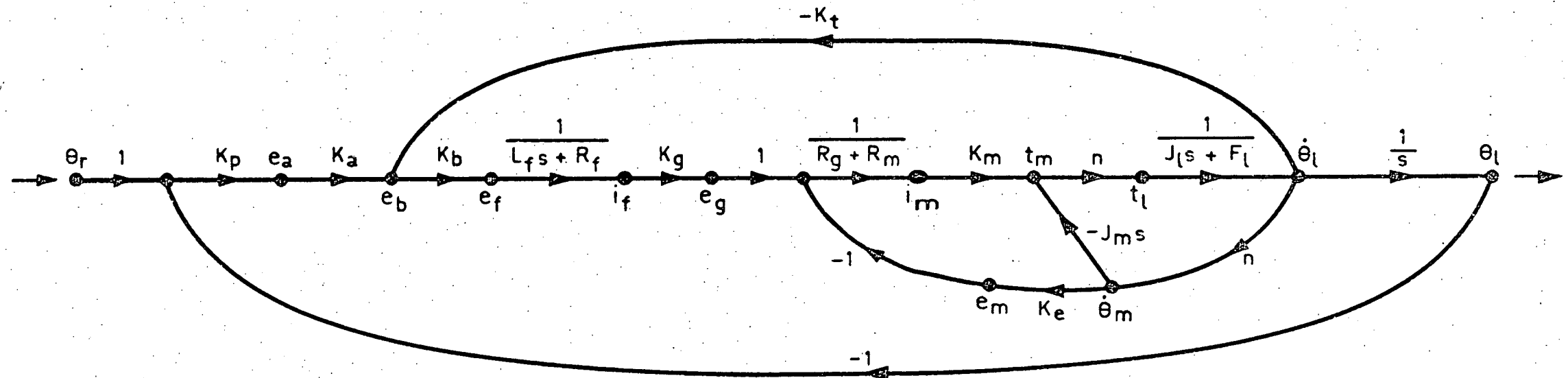
$$F_l = .00143 \text{ FT. LB/RAD/SEC}$$

Figure 6.6 Position control system

REFERENCE POSITION	$\theta_r =$ RADIANS
FIRST AMPLIFIER INPUT	$e_a = (\theta_r - \theta_l) K_p$ VOLTS
SECOND AMPLIFIER INPUT	$e_b = K_a e_a - K_t \dot{\theta}_l$ VOLTS
FIELD VOLTAGE	$e_f = K_b e_b$ VOLTS
FIELD CURRENT	$i_f = e_f / (L_f s + R_f)$ AMPS
GENERATOR E.M.F.	$e_g = K_g i_f$ VOLTS
MOTOR BACK E.M.F.	$e_m = K_e \dot{\theta}_m$ VOLTS

ARMATURE CURRENT	$i_m = (e_g - e_m) / (R_g + R_m)$ AMPS
MOTOR SPEED	$\dot{\theta}_m = n \dot{\theta}_l$ RAD/SEC
MOTOR TORQUE	$t_m = K_m i_m - J_m s \dot{\theta}_m$ FT. LB
LOAD TORQUE	$t_l = n t_m$ FT. LB
LOAD SPEED	$\dot{\theta}_l = t_l / (J_l s + F_l)$ RAD/SEC
LOAD POSITION	$\theta_l = \dot{\theta}_l / s$ RADIANS

(a)



(b)

Figure 6.7 Position control system : (a) system equations, (b) signal-flow graph.

equivalent to algebraic reduction, it is left to the computer program.

The signal-flow graph is represented by a system of voltage amplifiers with the structure shown in figure 6.8. Each amplifier block corresponds to a path segment of the signal-flow graph and is assigned polynomials with one of the following expressions:

$B1 = A[0, 0.1, 2 \times 3.14159/360];$ /*POTENTIOMETER AND FIRST AMPLIFIER*/
 $B2 = A[1, 0, 2E4, 5, 50];$ /*SECOND AMPLIFIER AND GENERATOR*/
 $B3 = A[0, .812, 48.8];$ /*MOTOR TORQUE*/
 $B8 = A[0, -1.25, 1];$ /*MOTOR BACK E.M.F.*/
 $B7 = A[1, -8E4, 0, 0, 1];$ /*MOTOR INERTIA*/
 $B4 = A[1, 0, 50, 1, .00143];$ /*GEAR TORQUE AND LOAD*/
 $B6 = A[0, 50, 1];$ /*GEAR SPEED*/
 $B5 = A[1, 0, 1, 1, 0];$ /*SPEED INTEGRATOR*/
 $B9 = A[0, -.01, 2 \times 3.14159/60];$ /*TACHOMETER*/
 $B10 = A[0, -1, 1];$ /*POSITION FEEDBACK*/

Part of the complete structure has been analysed topologically but the remainder can be analysed by algebraic reduction. However, before blocks B9 and B10 can be combined in series-parallel with other subnetworks, their ports must be interchanged to reverse the branch directions shown in the structure graph. Because of an implied sign convention applying to ports connected in series (the sum of second-port voltages equals the sum of first-port voltages) an interchange of ports must be accompanied by an interchange of terminals of any port which is connected in series. Crossover networks are therefore introduced, and the complete structure is described by the expression

$$\begin{aligned}
 B11 = & (B1 \text{ c } ((B2 \text{ c } D[B0:1s4p, B3:1s2s, B4:2s4p, \\
 & B8:3pls, B7:3p2s, B6:4p3p]) \text{ sp } (X \text{ c } rB9)) \\
 & \text{ c } B5) \text{ sp } (X \text{ c } rB10).
 \end{aligned}$$

In the alternative representation each component of the system is modelled with an electrical 2-port device. Shaft torques and speeds are represented

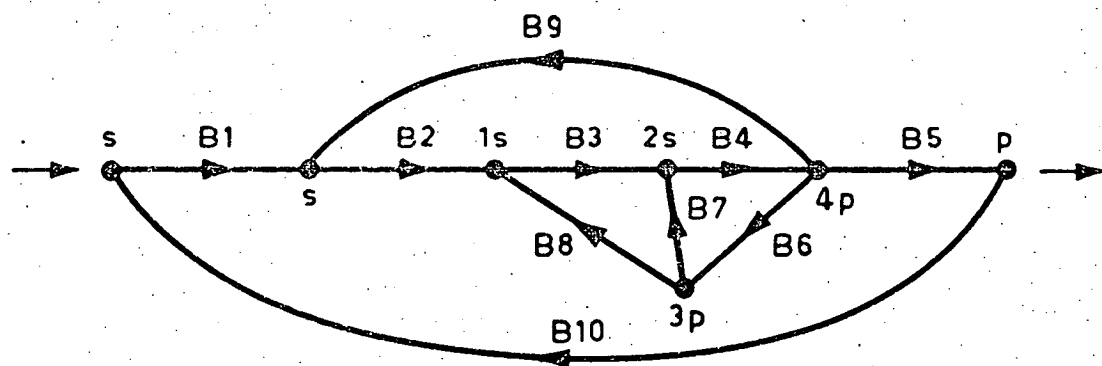


Figure 6.8 Structure graph corresponding to signal-flow graph.

by voltages and currents respectively, shaft inertia is represented by inductance, viscous friction by resistance, and the gear train by an ideal transformer. The resulting network and its structure graph is shown in figure 6.9. It could be specified with only one network expression but in order to clarify the specification of the major components they are here assigned to separate blocks:

B1 = A[0,0.1,2*3.14159/360];	/*POTENTIOMETER*/
B2 = A[0,1,1];	/*FIRST AMPLIFIER*/
B3 = A[0,100,1];	/*SECOND AMPLIFIER*/
B4 = SR50 c SL5 c N[0,0,0,1,0,200,0] c SR24.4;	/*GENERATOR*/
B5 = SR24.4 c N[0,0,.812*1.25,1,0,.812-1.25] c SL8E-4;	/*MOTOR*/
B6 = F50;	/*GEAR TRAIN*/
B7 = SL1 c SR.00143 c N[1,0 0,0 0,1 0,0 0,0 1,0 0];	/*LOAD*/
B8 = N[0,0,0,2*3.14159/60,0,0,-.01];	/*TACHOMETER*/.

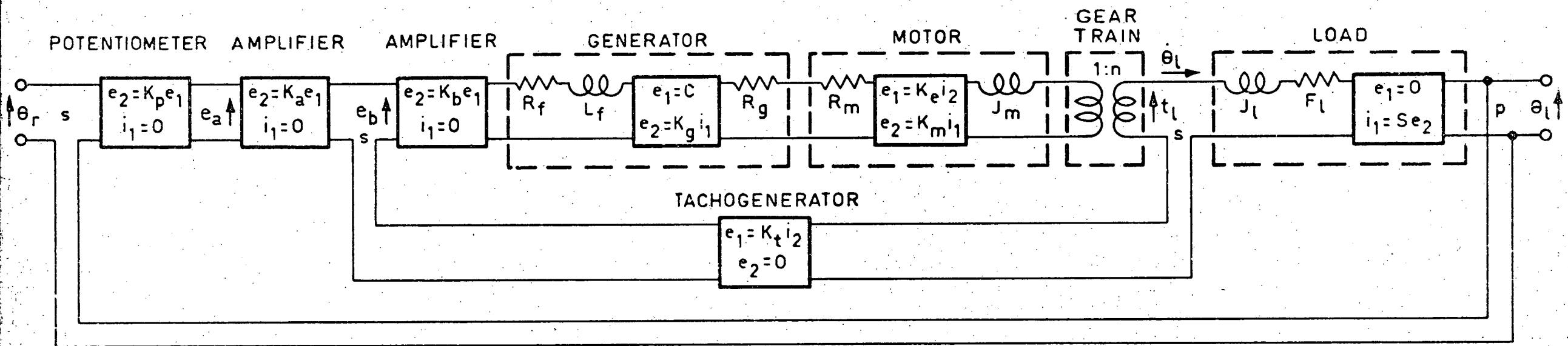
All the parameters of the system and the factors for conversion of units take their place directly in this specification without any preliminary processing.

The complete system is specified by the expression

$$B9 = (B1 \text{ c } B2 \text{ c } ((B3 \text{ c } B4 \text{ c } B5 \text{ c } B6) \text{ ss } B8) \text{ c } B7) \text{ sp } U.$$

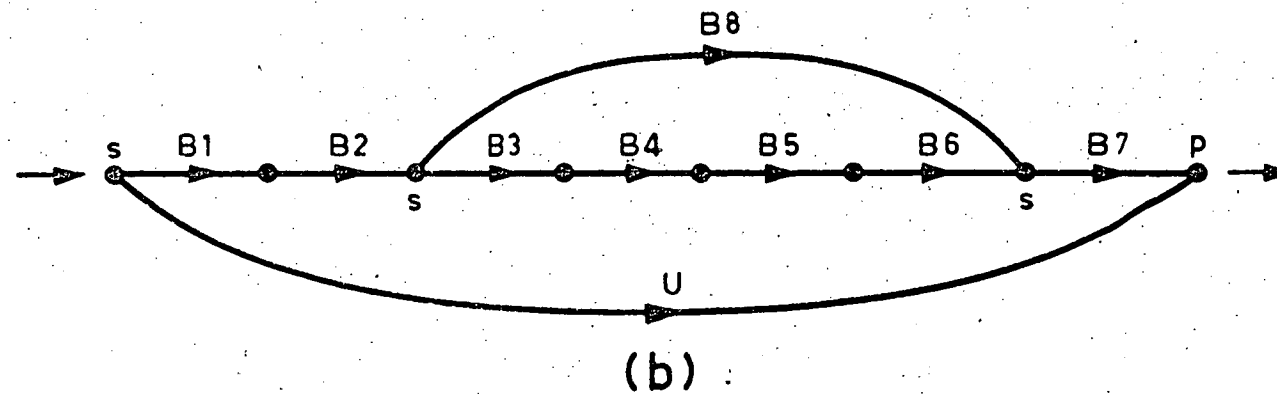
6.6. CONCLUSION

The four examples demonstrate the suitability of the general analysis method in a wide variety of situations. At one extreme it provides an extremely efficient analysis of the largest filters and, at the other, an efficient fully-symbolic analysis of small, strongly-interconnected, active circuits. The efficiency of analysis is due largely to the diakoptic approach inherent in algebraic reduction, while versatility is ensured by the possible introduction of a topological analysis at any stage in an algebraic reduction.



(a)

Figure 6.9(a) Equivalent network of position control system



(b)

Figure 6.9(b) Structure graph of equivalent network

REFERENCES

1. Bashkow, T.R.: "The A Matrix, New Network Concept", IRE Transactions on Circuit Theory, vol. CT-4, pp.117-119, September 1957.
2. Bashkow, T.R.: "Network Analysis", Mathematical Methods for Digital Computers, chap. 26, pp. 280-290, editors A. Ralston and H.S. Wilf, Wiley, 1960.
3. Berry, R.D.: "An Optimal Ordering of Electronic Circuit Equations for a Sparse Matrix Solution", IEEE Transactions on Circuit Theory, vol. CT-18, no. 1, pp. 40-50, January 1971.
4. Branin, F.H. Jr.: "The Relation Between Kron's Method and the Classical Methods of Network Analysis", IRE WESCON Convention Record, Part 2-Circuit Theory, pp. 3-28, 1959.
5. Branin, F.H. Jr.: "Computer Methods of Network Analysis", IEEE Proceedings, vol. 55, no. 11, pp. 1787-1801, November 1967.
6. Brayshaw, G.S.: "Representation of the Ideal Transformer Topological Graph by Means of a New Constraint Operator Q", IEEE Proceedings, vol. 55, no. 3, pp. 454-455, March 1966.
7. Brownell, R.A.: "Growing the Trees of a Graph", IEEE Proceedings, vol. 56, no. 6, pp.1121-1123, June 1968.
8. Brownell, R.A.: "Computer Programs for Linear Network Analysis", University of Tasmania, Electrical Engineering Dept., internal report, September 1968.
9. Brownell, R.A.: "Design of Group Delay Equalising Networks by Function Minimisation", IREE Aust, Abstracts of Circuit Theory Colloquium on Filter & Integrated Circuit Design, pp. 2-5, September 1969.
10. Bryant, P.R.: "The Explicit Form of Bashkow's A Matrix", IRE Transactions on Circuit Theory, vol. CT-9, no. 3, pp.303-306, September, 1962.
11. Calahan, D.A.: "Linear Network Analysis and Realisation Digital Computer Programs: An Instruction Manual", University of Illinois Bulletin, vol. 62, no. 58, February, 1965.

12. Calahan, D.A.: "Computer Design of Linear Frequency Selective Networks", IEEE Proceedings, vol. 53, no. 11, pp. 1701-1706, November 1965.
13. Chen, Wai-Kai: "Topological Analysis for Active Networks", IEEE Transactions on Circuit Theory, vol. CT-12, no. 1, pp. 85-91, March 1965.
14. Coates, C.L.: "General Topological Formulas for Linear Network Functions", IRE Transactions on Circuit Theory, vol. CT-5, pp. 30-42, March 1958.
15. Corrington, M.S.: "Simplified Calculations of Transient Response", IEEE Proceedings, vol. 53, no. 3, pp. 287-292, March 1965.
16. Dahl, O-J. & Nygaard, K.: "SIMULA 67 Common Base Definition", Norwegian Computing Centre, Oslo, June 1967.
17. Deckert, K.L. & Johnson, E.T.: "LISA 360 — A Program for Linear Systems Analysis", IBM Program Information Dept. Hawthorn, New York, 1966.
18. Downs, T.: "Symbolic Evaluation of Transmittances from the Nodal Admittance Matrix", Electronics Letters, vol. 5, pp. 379-380, 7th August, 1969.
19. Downs, T.: "Inversion of the Nodal Admittance Matrix in Symbolic Form", Electronics Letters, vol. 6, no. 3, pp. 74-76, 5th February, 1970.
20. Downs, T.: "Inversion of Nodal Admittance Matrix for Active Networks in Symbolic Form," Electronics Letters, vol. 6, no. 22, pp. 690-691, 29th October, 1970.
21. Dunn, W.R. Jr., & Chan, S.P.: "Topological Formulation of Network Functions Without Generation of K-trees", Proceedings of Sixth Allerton Conference on Circuit and Systems Theory, pp. 822-831, October 1968.
22. Faddeev, D.K. & Faddeeva, V.N.: Computational Methods of Linear Algebra, Freeman & Co., 1963.
23. Fernandez, E.B.: "An Evaluation of Tree Generation Methods", 12th Midwest Symposium on Circuit Theory, Austin, Texas, April 1969.

24. Francis, J.G.: "The Q-R Transformation", Computer Journal, vol. 4, no. 3, pp. 265-271, October 1961, and no. 4, pp. 332-345, January, 1962.
25. Hobbs, E.W.: "Topological Network Analysis as a Computer Program", IRE Transactions on Circuit Theory, vol. CT-6, no. 1, pp. 135-136, March 1959.
26. Idleman, T.E., et al: "SLIC — A Simulator for Linear Integrated Circuits", IEEE Journal of Solid-State Circuits, vol. SC-6, no. 4, pp. 188-203, August, 1971.
27. Ishizaki, Y., et al: "An Algebraic Manipulation Method for Network Analysis", Proceedings of Fourth Allerton Conference on Circuit and Systems Theory, pp. 61-69, October 1966.
28. Kron, G.: "A Set of Principles to Interconnect the Solutions of Physical Systems", Journal of Applied Physics, vol. 24, no. 8, pp. 965-980, August 1953.
29. Kron, G.: Diakoptics: The Piecewise Solution of Large-Scale Systems, MacDonald & Co., 1963.
30. Kuh, E., & Rohrer, R.: "The State-Variable Approach to Network Analysis", IEEE Proceedings, vol. 53, no. 7, pp. 672-686, July 1965.
31. Liou, M.L.: "A Novel Method of Evaluating Transient Response", IEEE Proceedings, vol. 54, no. 1, pp. 20-23, January 1966.
32. MacWilliams, J.: "Topological Network Analysis as a Computer Program", IRE Transactions on Circuit Theory, vol. CT-5, no. 3, pp. 228-229, September 1958, and vol. CT-6, no. 1, p. 136, March 1959.
33. Mason, S.J.: "Feedback Theory— Some Properties of Signal Flow Graphs", IRE Proceedings, vol. 41, no. 9, pp. 1144-1156, September 1953.
34. Mason, S.J.: "Topological Analysis of Linear Non-reciprocal Networks", IRE Proceedings, vol. 45, pp. 829-838, June 1957.
35. Matthaei, G.L.: "Some Simplifications for Analysis of Linear Circuits", IRE Transactions on Circuit Theory, vol. CT-4, no. 3, pp. 120-124, September 1957.
36. Mayeda, W.: "Topological Formulas for Active Networks", International Technical Report, University of Illinois, January 1958.

37. Mayeda, W.: "Reducing Computation Time in the Analysis of Networks by Digital Computer", IRE Transactions on Circuit Theory, vol. CT-6, no. 1, pp. 136-137, March 1959.
38. Morgan, B.S. Jr.: "Sensitivity Analysis and Synthesis of Multi-variable Systems", IEEE Transactions on Automatic Control, vol. AC-11, no. 3, pp. 506-512, July 1966.
39. Muller, D.E.: "A Method for Solving Algebraic Equations Using an Automated Computer", Mathematical Tables Other Aids Computers, vol. 10, pp. 208-215, 1956.
40. Neill, T.B.M.: "Techniques for Circuit Analysis (Part Two)", Computer Aided Design, vol. 2, no. 2, pp. 29-43, Winter 1970.
41. Pike, D.B.: "Linkage Polynomials—The Polynomials of Minor Determinants of Matrices of Linear Lumped Finite Time-Invariant Networks", Ph.D. thesis, University of Sydney, School of Electrical Engineering, June 1968.
42. Pinel, J.F., & Blostein, M.L.: "Computer Techniques for the Frequency Analysis of Linear Electrical Networks", IEEE Proceedings, vol. 55, no. 11, pp. 1810-1819, November 1967.
43. Pottle, C.: "A 'Textbook' Computerized State-Space Network Analysis Algorithm", IEEE Transactions on Circuit Theory, vol. CT-16, pp. 566-568, November 1969.
44. Riordan, R.H.S.: "The Analysis of Multistage Transistor Amplifiers", IREE Aust. Proceedings, vol. 29, no. 3, pp. 70-78, March 1968.
45. Russel, E.C. et al: "Instrumentation of a NASAP Subroutine", IEEE Transactions on Education, vol. E-12, no. 4, pp. 243-250, December 1969.
46. Sandberg, I.W. & So, H.C.: "A Two-Sets-of-Eigenvalues Approach to the Computer Analysis of Linear Systems", IEEE Transactions on Circuit Theory, vol. CT-16, no. 4, pp. 509-517, November 1969.
47. Shipley, R.B. & Coleman, D.: "A New Direct Matrix Inversion Method", AIEE Transactions, Part 1, vol. 78, pp. 568-572, November 1959.
48. Talbot, A.: "Topological Analysis of General Linear Networks", IEEE Transaction on Circuit Theory, vol. CT-12, no. 2, pp. 170-180, June 1965.

49. Tinney, W.F., & Walker, J.W.: "Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization", IEEE Proceedings, vol. 55, no. 11, pp. 1801-1809, November 1967.
50. Univac: "SIMULA Programmers Reference Manual", Univac Division of Sperry Rand Corporation, manual UP-7556, 1967.
51. Wilkinson, J.H.: The Algebraic Eigenvalue Problem, Oxford University Press, 1965.
52. Zobrist, G.W.: "Signal Flow Graphs as an Aid in Network Analysis", IEEE Transactions on Education, vol. E-12, no. 4, pp. 235-242. December 1969.
53. Naur, P. et al: "Report on the Algorithmic Language ALGOL 60", ACM Communications, vol. 3, p. 299, 1960.

E P I L O G U E

General Conclusions

The place of this work in engineering theory and practice is in the gulf between the broad and complex topological theory of electrical networks, and its practical implementation in everyday tools for circuit analysis and design. To define it further, it is concerned only with linear time-invariant networks, and is largely independent of the state-variable approach to this subject.

The work as a whole is built on quite simple concepts. The topological theory of part I is developed from an analytical process which itself is developed from the elementary concept of a tree. Network graphs constructed entirely of simple resistive branches, however, are necessarily reciprocal, and some refinement is necessary to include non-reciprocal networks within the scope of the theory. This is achieved by the conceptual construction of network graphs with unistors—basic branch elements introduced by Mason. It is remarkable that, notwithstanding the initial importance of trees, in the topological analysis algorithm of chapter 5 the concept of a tree has no special significance; rather, it is the concept of a loop that plays the dominant role.

Algebraic reduction is another simple but useful concept that is related to the series-parallel combination of resistors—although, within the context of 2-port networks, it may be recognised as the arithmetic combination of pairs of like network matrices, composed of either the A,B,C,D parameters or any set of hybrid parameters. The concept is developed in two stages: first, with the adoption of a set of six polynomials to characterise the general 2-port network; and second, with a relationship between network polynomials and topological quantities that allows topological analysis methods to be incorporated.

As circuits and systems become more complex, and specifications call for finer tolerances, the digital computer will play an increasingly important role as an analytical tool. Progress in this field is not dependent simply on the development of suitable algorithms, but on the development of better languages for communication at two levels: between the circuit designer and the programmed computer, and between the algorithm writer and the computer.

Analysis and design programs must become more powerful, versatile and easier to use; while to facilitate the development of these programs, some well-structured, scientifically-oriented language such as Algol should be extended to handle the manipulation and analysis of networks in a more natural way. It is anticipated, therefore, that the most significant advances in this direction will be the result of collaboration between the design engineer, the network theorist, and the computer scientist.

To conclude the thesis we shall explore some possible, future applications and developments.

Experience with the several versions of the analysis program has demonstrated the viability of a single program incorporating all the facilities discussed in part II. Transformation of polynomial coefficients at any scaled frequency would be optional, and polynomials would be represented in any of four ways: (1) a fully symbolic representation, with provision for parameters to be introduced either numerically or symbolically; (2) by numerical polynomial coefficients and their partial derivatives with respect to nominated parameters; (3) by numerical polynomial coefficients only; and (4) by their complex values at a nominated frequency. Routines would be included to convert from one form of polynomial representation to another, evaluate polynomials, display frequency response, search for roots, invert Laplace transforms, and display transient response. Such a program would consolidate the practical results of this work in one powerful analytical tool.

The most promising development would be an extension of Algol to include the concept of a 2-port network as a type of variable. Network variables, to which network expressions could be assigned, might be declared with a statement such as

"network block 1, B5, preamplifier, filter;".

The syntax for network expressions defined in chapter 5 could be implemented with few changes; specifically, all blocks would become network variables, and parameters would become arithmetic expressions. Basic networks could be regarded as standard network procedures, and the opportunity would exist to define whatever basic networks were appropriate to a particular application, as in this example for crystal filter analysis:

```

"network procedure series crystal (frequency, C1, CO, resistance);
value frequency, C1, CO, resistance;
real frequency, C1, CO, resistance;
begin real omega;
    omega:= 6.283185 * frequency;
    series crystal:= (SL(1.0/(omega * omega * C1))
        c SC(C1) c SR(resistance)) pp SC(CO)
end;".

```

If this approach is taken, other facilities must also be provided by the language to make it workable. The algebra for networks should be accompanied by an algebra for polynomials—admitting the polynomial operations of addition and multiplication (but not division) and admitting parameters with either a literal or a numerical value. Procedures would be needed to differentiate polynomials with respect to symbolic parameters, to substitute real numbers for symbolic parameters, and to substitute complex numbers for the symbolic frequency parameter. The concept of a 1-port network, or component, as a type of variable would also be useful, and some facility must be provided for the specification of branch lists of structure graphs. Although it is not the objective of this present work to formulate the most desirable structure for an extended language, it is clear that the grammar for network expressions is well suited to such a language and that the other necessary facilities are well within the current state of the compiling art.

The association of a versatile analysis capability with the full language facilities of Algol would greatly assist the development of more sophisticated programs for circuit analysis and design. Two of the most important applications will be mentioned.

In the field of statistical design the most complex relationships between network parameters—as occur, for example, with changes in fabrication-process parameters and with changes in operating temperature—could be expressed succinctly with arithmetic expressions and procedure calls in the place of network parameters. Such versatility is essential for realistic Monte Carlo simulations of circuits operating in various environments. In a program for Monte Carlo analysis the calculation of complete network polynomials could be achieved by a single assignment statement within a controlled loop, and there would

therefore, be no difficulty in interfacing the network analysis task with the remainder of the application program.

In developing iterative design programs, the necessary links between the analysis process and suitable library procedures for function minimisation could be achieved with just a few program statements. It would also be a simple matter to experiment with the error function to be minimised, and so realise circuits that were optimised according to various performance criteria.

If embodied in a compiler for the new language, the methods of algebraic reduction and topological analysis would be transparent to users of the language. An analysis process could be incorporated with only a few program statements, rather than the hundreds of statements currently required to specify the relevant algorithms, and it could be interwoven with other program statements with greater flexibility than is possible with calls to Algol procedures or Fortran subroutines.

With regard to the theory part of the thesis, it is believed that further development of the analysis algorithm introduction in chapter 1 could lead to analysis methods with greater power than those of chapter 5. The admission of networks with up to three ports might require more polynomials to characterise subnetworks but the amount of polynomial manipulation would be significantly reduced. For instance, all the networks discussed in chapter 6 could be constructed by combining 3-port networks two at a time, thus avoiding a topological analysis of their structures.

Research is needed to develop an algebra for 3-port networks, and a satisfactory algorithm for the topological analysis of structures of 3-port networks.

Growing the Trees of a Graph

Abstract—An algorithm is described which generates without duplication and with appropriate sign all the trees of a graph containing directed elements. A path-finding algorithm is extended in an application of Mason's method of expansion of paths.

INTRODUCTION

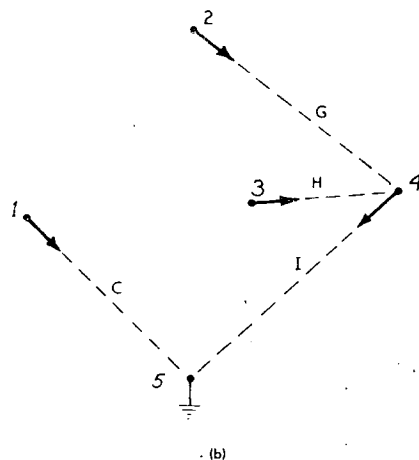
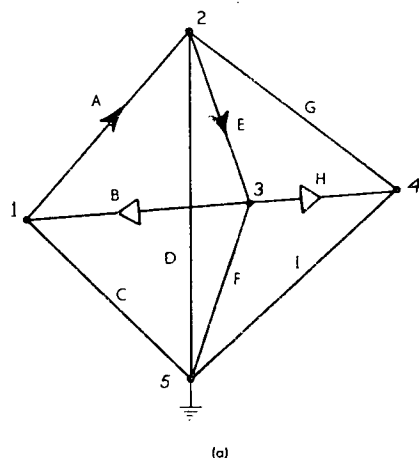
An algorithm for finding, without duplication, all the trees of a graph forms the heart of computer programs to analyze linear networks by topological methods. This letter presents an algorithm which generates all the trees of a graph one at a time and without duplication, and determines the signs of trees for graphs containing directed elements. The three types of elements allowed in the graph are the directed unistor and gyrator¹ [denoted by the black arrow in the example shown in Fig. 1(a)] and the undirected resistor. The sign of a tree will normally be positive except that the sign will be changed for every gyrator which in the tree is directed away from a designated ground node and any tree containing a unistor which is directed away from the ground node will be neglected. The algorithm is based on Mason's method of expansion of paths¹ although similarities will be noticed in many other methods of expanding node determinants, notably that of Tsai.² Paths are generated by a method similar to that recently published by Kroft for finding all the paths through a maze.³ One reason for presenting what may be another version of existing tree-finding algorithms is to show that it requires only a small extension, in the bookkeeping, of a path-finding algorithm.

CONSTRUCTION

The algorithm requires that the graph be specified by lists of the branches which are connected at each node, with the exception of both unistor branches which are not entered in the lists of the nodes to which they are directed and gyrator branches which are entered negatively in the lists of the nodes to which they are directed [as shown in Fig. 1(c)].

To visualize the action of the algorithm, pointers are associated with each node except the ground node and these may be set in the direction of any branch in the list of their associated nodes. The key to the algorithm lies in the observation that every tree corresponds to a unique combination of pointer settings that is determined by tracing, and setting pointers, along the unique paths from every node to the ground node. For example, the combination of pointer settings which corresponds to the tree CGIH of Fig. 1(a) is shown in Fig. 1(b). It follows that every tree will be formed once and only once by generating all the possible combinations of pointer settings. If there are n nodes in the graph a combination of pointer settings will determine a set of $n-1$ or fewer branches which may or may not be a tree, but the number of combinations to be tested is less than all the combinations of $n-1$ branches.

To ensure the generation of all combinations of pointer settings which are likely to determine trees, the pointers are set one at a time in such a way that each pointer is successively directed to all the branches in its node branch list. A pointer is reset either when it completes the formation of a loop or tree, or when the following pointer begins a new cycle through its branch list. The detection of loops is simplified by setting pointers in the order indicated by their direction, i.e., following the formation of paths. After each pointer is set or reset a list, which is headed by the ground node and contains the nodes whose pointers have been set, is updated and searched to determine whether the node indicated by the pointer is included. The search is made in two parts: 1) the nodes in the path currently being traced—the ungrounded nodes—are scanned and if the node is found because a loop is about to be formed, then the last pointer is reset, and 2) the remaining nodes in the list are scanned and if the node is found because the path has terminated at a ground node then all the nodes in the list are considered to be grounded and the next pointer to be set may be chosen from any of the ungrounded nodes. To continue the generation



n	NBL[n]			
1	C	A		
2	G	E	D	-A
3	H	F	-E	B
4	I	G		

(c)

b	A	B	C	D	E	F	G	H	I
BNS[b]	3	4	6	7	5	8	6	7	9

(d)

Fig. 1. (a) Graph. (b) Combination of pointer settings. (c) Node branch lists. (d) Branch node sums.

of a path after a pointer has been set towards a new branch, the node at the other end of this branch must be determined. The search that would be required if the graph were specified only by the node branch lists is avoided by calculating and storing the sums of the node numbers of each branch when the node branch lists are formed [Fig. 1(d)]. The number of the next node will therefore be obtained by subtracting the number of the last node from the node sum of the new branch.

BOOKKEEPING

The flow chart shown in Fig. 2 introduces only sufficient variables and arrays to describe the essential action of the algorithm. Additional variables are required to hold the current values of the length of the node list, the sign of the branch product, and the number of grounded nodes to determine the division between the grounded and ungrounded nodes in the

Manuscript received February 13, 1968.

¹ S. J. Mason, "Topological analysis of linear nonreciprocal networks," *Proc. IRE*, vol. 45, pp. 829-838, June 1957.

² W.-K. Chen, "Topological network analysis by algebraic methods," *Proc. IEE (Correspondence)* (London), vol. 114, pp. 86-88, January 1967.

³ D. Kroft, "All paths through a maze," *Proc. IEEE (Letters)*, vol. 55, pp. 88-90, January 1967.

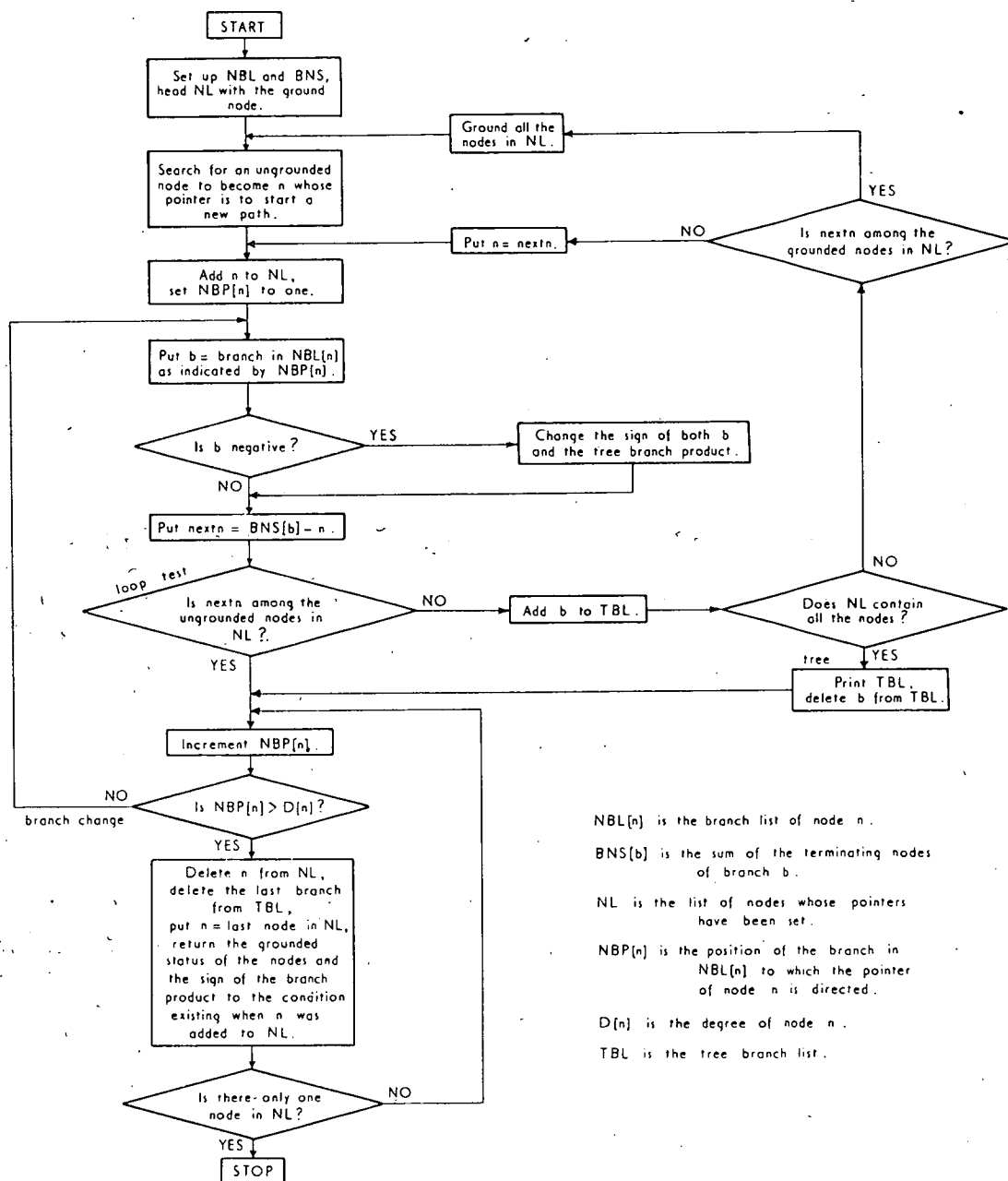


Fig. 2. Flow chart of tree-generating algorithm.

node list. To correct these values after a backtracking step in which a node is deleted from the node list, each successive value of the last two variables must be stored in arrays. For the purpose of choosing an ungrounded node to start a new path a further array is required to indicate which nodes are ungrounded. A run through the flow chart with the example shown in Fig. 1 should require 76 loop tests and 47 branch changes while finding the 40 trees in the following order:

CGIH CGIF -CGIE CGIB CEHI CEFI CEF G CEBI CEBG
 CDHI CDHG CDFI CDFG -CDEI -CDEG CDBI CDBG -CAHI
 -CAHG -CAFI -CAFG CAEI CAEG -CABI -CABG AGIH
 AGIF -AGIE AGIB AEHI AEFI AEFG ADHI ADHG ADFI
 ADFG -ADEI -ADEG ADBI ADBG.

PERFORMANCE

The algorithm has been written in ALGOL and run on an Elliott 503 computer with the graph of a ladder filter containing 20 branches and 10 nodes, an example that was used by MacWilliams and Hobbs to evaluate

their tree-finding algorithms.^{4,5} The actual ground terminal of the filter was chosen as the ground node for the algorithm because it had the largest degree. An upper bound on the number of trees is provided by the number of combinations of 9 branches, 167 960, and—more appropriately for this algorithm—by the number of combinations of pointer settings, i.e., the product of the degrees of the ungrounded vertices, 16 384. The algorithm required 8692 loop tests and 6027 branch changes while finding the 4756 trees. These figures indicate similarities between this algorithm and that of Hobbs which tested 16 384 sets of branches and that of MacWilliams which examined 6028 sets of branches.

A frequent attempt at the formation of a loop will occur when a pointer is set to reverse the direction of a path. Therefore, by commencing the node search with the second-last node to be added to the node list and

⁴ J. MacWilliams, "Topological network analysis as a computer program," *IRE Trans. Circuit Theory (Correspondence)*, vol. CT-5, pp. 228-229, September 1958.

⁵ E. W. Hobbs and F. J. MacWilliams, "Topological network analysis as a computer program," *IRE Trans. Circuit Theory (Correspondence)*, vol. CT-6, pp. 135-136, March 1959.

scanning to the top of the list many loop tests will terminate after only one comparison. With careful attention to such programming details and by keeping the array accessing to a minimum the 4756 trees of the example were found in 29 seconds. This can be compared with times reported by MacWilliams and Hobbs that were of the order of five minutes on an IBM 704 computer.

R. A. BROWNELL
Dept. of Elec. Engrg.
University of Tasmania
Hobart, Tasmania, Australia

Reprinted from the PROCEEDINGS OF THE IEEE

VOL. 56, NO. 6, JUNE, 1968

pp. 1121-1123

COPYRIGHT © 1968—THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

PRINTED IN THE U.S.A.