MACHINE ALGORITHMS FOR BOOLEAN MATRICES

by

DALE ROBERT COMSTOCK

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

MASTER OF SCIENCE

June 1963

APPROVED:

[redacted]

Professor of Mathematics

In Charge of Major

[redacted]

Chairman of Department of Mathematics

[redacted]

Chairman of School Graduate Committee

[redacted]

Dean of Graduate School

Date thesis is presented July 18, 1962

Typed by Carol Baker

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION	1
II.	BOOLEAN ALGEBRA	3
III.	BOOLEAN MATRICES	8
IV.	CONTACT NETWORKS	15
V.	THE ANALYSIS OF CONTACT NETWORKS	22
VI.	MACHINE ALGORITHMS FOR CONTACT NETWORKS	31
	BIBLIOGRAPHY	43
	APPENDIX	45

MACHINE ALGORITHMS FOR BOOLEAN MATRICES

CHAPTER I

INTRODUCTION

The complex circuitry of the modern electronic digital computer has resulted in detailed studies of methods of analysis and synthesis of switching networks. In many of these studies the desirable outcome is that of simplifying the circuit, so that the final circuit will be as economical as possible.

In the past ten years the concept of Boolean matrices has been used with success in switching network theory.

The purpose of this paper is to discuss some of the general theory of Boolean matrices, illustrate their application to contact networks and formulate machine algorithms for answering the question: Is network A equivalent to network B ? That is, suppose network A has been redesigned to yield a simpler network B. The simplification may have been done by trial and error or any of the many other methods studied in recent years. We mechanize a method for showing the equivalence of the two networks.

The first section of this paper defines a Boolean algebra and lists some of its elementary properties. The second section

presents Boolean matrices and their properties. The next section illustrates the application of Boolean matrices to contact networks. The last section presents some machine algorithms for network analysis with Boolean matrices. An appendix is included which discusses two special cases.

CHAPTER II

BOOLEAN ALGEBRA

Definition 2. 1. A set B together with two binary operations, (+) and (·), is a Boolean algebra provided that the following postulates hold for elements $a, b, c \in B$:

P_1. The operations (+) and (·) are commutative, i. e. $a + b = b + a$ and $a \cdot b = b \cdot a$.

P_2. There exist in B distinct identity elements 0 and 1 relative to the operations (+) and (·), respectively, i. e. there exist elements $0, 1 \in B$ such that $a + 0 = a$ and $a \cdot 1 = a$.

P_3. Each operation is distributive relative to the other, i. e. $a + (b \cdot c) = (a + b) \cdot (a + c)$ and $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

P_4. For each $a \in B$, there exists an $a' \in B$ such that $a + a' = 1$ and $a \cdot a' = 0$. The element a' is called the dual of a.

The above definition of a Boolean algebra is due to E. V. Huntington (4).

Some of the basic properties of a Boolean algebra are given

by the following theorem:

Theorem 2. 1. For each $a, b, c \in B$, we have

i. 0 and 1 are unique elements of B;

ii. $a + a = a$ and $a \cdot a = a$;

iii. $a + 1 = 1$ and $a \cdot 0 = 0$;

iv. $a + a \cdot b = a$ and $a \cdot (a + b) = a$;

v. $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$;

vi. $(a')' = a$;

vii. $a + b = (a' \cdot b')'$ and $a \cdot b = (a' + b')'$;

viii. $a + a' \cdot b = a + b$.

For a proof of the above theorem and a discussion of other properties of a Boolean algebra see (11).

Definition 2. 2. The order relation $\leq$ (less than or equal to) is defined on B as follows: for each $a, b \in B$, $a \leq b$ if and only if $a + b = b$.

Theorem 2. 2. The relation $\leq$ partially orders the set B, i. e. for any $a, b, c \in B$, we have

i. $a \leq a$;

ii. if $a \leq b$ and $b \leq a$, then $a = b$;

iii. if $a \leq b$ and $b \leq c$, then $a \leq b.c$.

Theorem 2.3. For $a, b \in B$, if $a \leq b$, then $a \cdot c \leq b \cdot c$ and $c \cdot a \leq c \cdot b$ for any $c \in B$.

Proof. By P_3 and P_4 of Definition 2.1, we have $a \cdot c + b \cdot c = (a + b) \cdot c$. But, if $a \leq b$, then $a + b = b$ by Definition 2.2. Hence, $a \cdot c + b \cdot c = b \cdot c$, which is equivalent to $a \cdot c \leq b \cdot c$ by Definition 2.2. $c \cdot a \leq c \cdot b$ follows from P_1.

Perhaps the simplest Boolean algebra, denoted by B_0, consists of the two elements 0 and 1. Addition (+), multiplication ($\cdot$), and duality (') are defined by the tables

+	0	1
0	0	1
1	1	1

$\cdot$	0	1
0	0	0
1	0	1

a'	a
0	1
1	0

.

Now to the Boolean algebra B_0, we adjoin the variables $x_1, x_2, \ldots, x_m$ whose common domain is B_0, extending to them the postulates and definitions previously made.

Definition 2.3. A function $f(x_1, x_2, \ldots, x_m)$ built up from the variables $x_1, x_2, \ldots, x_m$ and from the elements of B_0 by a finite number of applications of the operations (+), ($\cdot$), and (') will be called a Boolean function of the m variables $x_1, x_2, \ldots, x_m$.

Definition 2.4. The set of all Boolean functions of the m variables $x_1, x_2, \ldots, x_m$ will be called the Boolean algebra B_m.

This particular Boolean algebra has been extensively applied in switching network theory. Except for the more general results concerning Boolean matrices, this paper will be devoted entirely to the Boolean algebra B_m.

As a consequence of the definition of Boolean functions and the equality of functions, we have for any $f, g \in B_m$, $f = g$ if and only if f and g have the same value when $x_1, x_2, \ldots, x_m$ take on any set of values from B_0.

The previous remark gives rise to the so-called "identification problem" of the Boolean algebra B_m, i.e. given two functions of B_m, are they equivalent? Several solutions to this problem exist. An obvious solution to the problem is to evaluate the two functions for every set of values of the variables $x_1, x_2, \ldots, x_m$. This requires a large amount of computation if m is very large. A second method is to perform an equivalence transformation on the function which transforms it into a unique canonical form. Then we identify the equivalence of two functions by inspection of their representations in canonical form.

In this paper we shall show the equivalence of two functions by the second method mentioned above.

The canonical form that we have selected has come to be known as Quine's canonical form. It is formed in the following

manner:

i. transform the function into an equivalent polynomial representation;

ii. if β_1 and β_2 are two monomials of this polynomial such that $\beta_2 \leq \beta_1$, then delete β_2 from the polynomial;

iii. if $x_1 \cdot \beta_1$ and $x_1' \cdot \beta_2$ are two monomials of the polynomial such that $\beta_1 \cdot \beta_2 \neq 0$ and $\beta_1 \cdot \beta_2$ is not less than or equal to any monomial of the polynomial, then add $\beta_1 \cdot \beta_2$ to the polynomial;

iv. if the monomials x_1 and x_1' ($i = 1, 2, \ldots, m$) occur in the polynomial, then the canonical form is the Boolean constant 1.

In view of the distributive postulates and Theorem 2.1, vii, any Boolean function can be transformed into a polynomial. Of course, such a transformation may not yield a unique result. Laxdal (5) has shown that after a finite number of applications of ii, iii, and iv, the polynomial representation of a Boolean function has been transformed into Quine's canonical form.

CHAPTER III

BOOLEAN MATRICES

Definition 3.1. Let B be an arbitrary Boolean algebra and let M be the set of all n x n matrices, where each element of the matrix is an element of B. We make the following definitions where $A = (a_{ij})$, $B = (b_{ij})$, ... are elements of M (i, j = 1, 2, ... , n):

i. $A + B = (a_{ij} + b_{ij})$;

ii. $A \cdot B = (a_{ij} \cdot b_{ij})$;

iii. $A' = (\Sigma_{ij})$ where $\Sigma_{ij} = a'_{ij}$;

iv. $Z = (\Sigma_{ij})$ where $\Sigma_{ij} = 0$;

v. $I = (\Sigma_{ij})$ where $\Sigma_{ij} = 1$;

vi. $A = B$ if and only if $a_{ij} = b_{ij}$.

Theorem 3.1. The set M together with the two operations (+) and (·) as defined in Definition 3.1 is a Boolean algebra.

Proof. The theorem is established immediately since the elements of M satisfy the postulates P_1, P_2, P_3, and P_4 of Definition 2.1.

Theorem 3.1 is due to the Russian mathematician A. G. Lunts (8). As a consequence of Theorem 3.1, Boolean matrices

have all the properties of a Boolean algebra and, in particular, Theorem 2.1 holds for Boolean matrices.

Definition 3.2. For $A, B \in M$, $A \leq B$ if and only if $a_{ij} \leq b_{ij}$ for $i, j = 1, 2, \ldots, n$.

Theorem 3.2. For $A, B \in M$, $A \leq B$ if and only if $A + B = B$.

Proof. If $A \leq B$, then $a_{ij} \leq b_{ij}$ for $i, j = 1, 2, \ldots, n$. Hence, by Definition 2.2, $a_{ij} + b_{ij} = b_{ij}$ for every i, j, and so $A + B = B$. If $A + B = B$, then $a_{ij} + b_{ij} = b_{ij}$ for $i, j = 1, 2, \ldots, n$. Hence, by Definition 2.2, $a_{ij} \leq b_{ij}$ for every i, j and so $A \leq B$.

As in ordinary matrix theory over a field, we make the following definitions:

Definition 3.3. For $A, B \in M$, we define

i. $AB = (\Sigma_{ij})$ where $\Sigma_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$;

ii. $E = (\Sigma_{ij})$ where $\Sigma_{ij} = 1$ if $i = j$ and $\Sigma_{ij} = 0$ if $i \neq j$;

iii. $|A| = \sum a_{1s_1} \cdot a_{2s_2} \cdots a_{ns_n}$ where the summation is taken over all permutations $(s_1, s_2, \ldots, s_n)$ of $(1, 2, \ldots, n)$. $|A|$ is called the determinant of A;

iv. if A_{ij} is the submatrix of A obtained by deleting the ith row and the jth column (which intersect in the element a_{ij}) of A, then $|A_{ij}|$ is termed the minor of a_{ij} in A;

v. $A^T = (\Sigma_{ij})$ where $\Sigma_{ij} = a_{ji}$ is called the transpose of A;

vi. $\overline{A} = (\Sigma_{ij})$ where $\Sigma_{ij} = \left|A_{ji}\right|$ is called the adjoint of the matrix A.

Notice the difference between the matrix multiplication of Definition 3.3, i and the Boolean multiplication of Definition 3.1, ii. In the sequel AB will mean matrix multiplication and $A \cdot B$ will mean Boolean multiplication.

The Boolean matrix E appears as the identity element for matrix multiplication, i.e. $AE = EA = A$ for each $A \in M$.

The definition of $|A|$ differs from the ordinary definition of determinant only with respect to the Kronecker symbol, which, of course, would have no meaning in a Boolean algebra.

With respect to matrix multiplication many of the properties of ordinary matrices carry over into Boolean matrices. For a proof of these results, any standard text on matrices can be consulted.

Theorem 3.3. For $A, B, C \in M$, $A(B + C) = AB + AC$ and $(A + B)C = AC + BC$.

Theorem 3.4. For $A, B, C \in M$, $(AB)C = A(BC)$.

Theorem 3.5. For some $A \in M$, $AB \neq BA$.

Theorem 3.6. For $A, B \in M$, $(AB)^T = B^T A^T$.

Theorem 3.7. For $A \in M$, $|A| = |A^T|$.

Theorem 3.8. For $A \in M$, we have

i. $|A|$ is not changed by interchanging rows and columns;

ii. $|A|$ is not changed by the interchange of two rows (or columns);

iii. if all the elements of a row (or column) have a common factor, then it may be taken outside the determinant sign;

iv. If the elements of an arbitrary row (or column) are the sum of two terms, then the determinant appears as the sum of two determinants;

v. $|A|$ may be expanded with respect to the elements of an arbitrary row or column, i.e.

$$|A| = \sum_{k=1}^{n} a_{ik} \cdot |A_{ik}| = \sum_{k=1}^{n} a_{kj} \cdot |A_{kj}|$$

where $|A_{ik}|$ is a minor of the element a_{ik} in A.

We prove some properties of Boolean matrix algebra not encountered in ordinary matrix algebra.

Theorem 3.9. For $A \in M$, $(A^T)' = (A')^T$.

Proof. Let $A = (a_{ij})$ and $A' = (a'_{ij})$. Then $(A')^T = (\Sigma_{ij})$ where $\Sigma_{ij} = a'_{ji}$. But $A^T = (a_{ji})$ so $(A^T)' = (\Sigma_{ij})$ where $\Sigma_{ij} = a'_{ji}$. Hence, $(A^T)' = (A')^T$.

Theorem 3.10. For $A, B \in M$, $(A + B)^T = A^T + B^T$ and $(A \cdot B)^T = A^T \cdot B^T$.

Proof. Let $A = (a_{ij})$ and $B = (b_{ij})$. Then $A + B = (a_{ij} + b_{ij})$, so that $(A + B)^T = (a_{ji} + b_{ji})$. Now $A^T = (a_{ji})$ and $B^T = (b_{ji})$ so $A^T + B^T = (a_{ji} + b_{ji})$. Hence, $(A + B)^T = A^T + B^T$. To show $(A \cdot B)^T = A^T \cdot B^T$, we see that $(A \cdot B)^T = (a_{ij} \cdot b_{ij})^T = (a_{ji} \cdot b_{ji})$. But, also $A^T \cdot B^T = (a_{ji} \cdot b_{ji})$. Hence, the proof is complete.

Theorem 3.11. For $A, B \in M$, if $A \leq B$, then $AC \leq BC$ and $CA \leq CB$ for any $C \in M$.

Proof. $AC + BC = (A + B)C$ by Theorem 3.3. Hence, by hypothesis and Theorem 3.2, $AC + BC = BC$. Thus, $AC \leq BC$. $CA \leq CB$ if $A \leq B$ is established in the same manner. Notice that $CA \leq CB$ does not follow from $AC \leq BC$, since the operation of forming the matrix product is not commutative.

Thoerem 3.12. For $A \in M$, if $E \leq A$, then $E \leq A^2 \leq A^3 \leq A^4 \leq \cdots$.

Proof. This result follows immediately by repeated application of Theorem 3.11, i.e. $E \leq A$ implies $AE = A \leq A^2$, $A \leq A^2$ implies $A^2 \leq A^3$, etc.

For the next theorem we introduce the notation $A^* = \sum_{k=1}^{\infty} A^k$.

Theorem 3.13. For $A, B \in M$, if $E \leq A$ and $E \leq B$, then $(A + B)^* = (A^* B^*)^*$.

Proof. By Theorem 3.12, $A \leq A^*$. Further $E \leq B^*$. Applying Theorem 3.11, we have $AB^* \leq A^*B^*$ and $A \leq AB^*$. Hence, $A \leq A^*B^*$. In a similar manner $B \leq A^*B^*$. Thus, $A + B \leq A^*B^*$ and so $(A + B)^* \leq (A^*B^*)^*$. Further, $A \leq A + B$ and $B \leq A + B$ implies $A^* \leq (A + B)^*$ and $B^* \leq (A + B)^*$. Thus, $A^*B^* \leq (A + B)^*$. Hence, $(A^*B^*)^* \leq ((A + B)^*)^* = (A + B)^*$. Combining $(A + B)^* \leq (A^*B^*)^*$ and $(A^*B^*)^* \leq (A + B)^*$, we have the desired result.

This result was stated and proved for matrices over a semiring by Yoeli (13).

Theorem 3.14. For $A, B, C \in M$, $A(B \cdot C) \leq (AB) \cdot (AC)$ and $(A \cdot B)C \leq (AC) \cdot (BC)$.

Proof. Let $A = (a_{ij})$, $B = (b_{ij})$ and $C = (c_{ij})$. Then $A(B \cdot C) = (\Sigma_{ij})$ where $\Sigma_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \cdot c_{kj}$ for $i, j = 1, 2, \ldots, n$. Now $AB = (\sum_{k=1}^{n} a_{ik} \cdot b_{kj})$ and $AC = (\sum_{k=1}^{n} a_{ik} \cdot c_{kj})$, so that $(AB) \cdot (AC) = (\beta_{ij})$ where

$$\beta_{ij} = \left\{\sum_{k=1}^{n} a_{ik} \cdot b_{kj}\right\} \cdot \left\{\sum_{k=1}^{n} a_{ik} \cdot c_{kj}\right\} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \cdot c_{kj} + R.$$

Hence $\Sigma_{ij} + \beta_{ij} = \beta_{ij}$ for $i, j = 1, 2, \ldots, n$. Thus, by Theorem 3.2, $A(BC) \leq (AB) \cdot (AC)$.

CHAPTER IV

CONTACT NETWORKS

We now introduce the concept of a network and indicate how networks can be described in terms of Boolean matrices.

Definition 4.1. A contact network is a directed graph consisting of n terminals (or nodes) $v_1, v_2, \ldots, v_n$ with exactly one branch b_{ij} $(i \neq j)$ from each v_i to each other v_j, each branch being weighted by an element w_{ij} of the set B_m (see Definition 2.3-2.4) of Boolean functions of the m variables $x_1, x_2, \ldots, x_m$.

The variables $x_1, x_2, \ldots, x_m$ take on the role of contacts in our network with their values coming from B_0. When two or more contacts always open and close simultaneously, we denote them by the same circuit variable. Thus, we do not distinguish between occurrences of a contact and the contact itself as a circuit variable. Thus, in Figure 1, x' of branch $b_{1,4}$ may be a different contact than x' of branch $b_{3,5}$, but they operate simultaneously. We use the convention in which 1 stands for a closed circuit or contact and 0 for an open circuit. The parallel connection between contacts x and y is represented by $x + y$ while their series connection is represented by $x \cdot y$. In practice $\cdot$ is usually omitted. Thus

$x \cdot y$ becomes xy.

Definition 4.2. A directed path of a contact network of length r from v_i to v_j is a sequence of r branches of the form $b_{ik_1}, b_{k_1k_2}, \ldots, b_{k_{r-1}j}$. If $i = j$, the path is called closed, otherwise open. If $i, k_1, k_2, \ldots, k_{r-1}, j$ are all distinct, the path is called proper. An open path which is not proper is redundant.

Definition 4.3. The weight w of a directed path is called the conductance of the path and is defined as the product of its branch weights, i.e. $w = w_{ik_1} \cdot w_{k_1k_2} \cdot \ldots \cdot w_{k_{r-1}j}$.

Theorem 4.1. The maximal length of any proper path of a network of n terminals is n - 1 [see (13)].

Proof. Consider an open path p of length $r > n - 1$. It has the form $b_{ik_1}, b_{k_1k_2}, \ldots, b_{k_{r-1}j}$. There are $(r-1) + 2 = r + 1$ subscripts here. But since $r + 1 > n$, this must mean not all of the subscripts $i, k_1, k_2, \ldots, k_{r-1}, j$ are distinct. Hence, p is not a proper path.

Theorem 4.2. If w is the weight of an open path p from v_i to v_j, then there exists a proper path from v_i to v_j with weight w* such that $w \leq w^*$ [see (13)].

Proof. Let p have the form $b_{ik_1}, b_{k_1k_2}, \ldots, b_{k_{r-1}j}$. Since p is redundant (if not, there is nothing to prove), there must be a coincidence, say $k_q = k_p$. Then p has the form $b_{ik_1}, \ldots, b_{k_{q-1}k_q}, b_{k_qk_{q+1}}, \ldots, b_{k_{p-1}k_p}, b_{k_pk_{p+1}}, \ldots, b_{k_{r-1}j}$. We can omit some of the branches of p to obtain a new path of the form $b_{ik_1}, \ldots, b_{k_{q-1}k_p}, b_{k_pk_{p+1}}, \ldots, b_{k_{r-1}j}$. Continuing to eliminate some of the branches, we finally obtain a proper path p*. By repeated application of Theorem 2.3, we have $w \leq w^*$ where w^* is the weight of the proper path p*.

In a contact network without diodes, the conductance from v_i to v_j is the same as that from v_j to v_i. In actual practice, a great number of diode elements are used in the network to prevent an unwanted (sneak) circuit and to reduce the number of contacts required. A diode element is any device that allows current to pass in one direction only.

For a given n-terminal contact network A we will associate an n x n matrix, where the i, j^{th} entry is the conductance of a path from v_i to v_j.

Definition 4.4. The n x n Boolean matrix $A = (a_{ij})$ such that $E \leq A$ where $a_{ij} = w_{ij}$ $(i \neq j)$, is called the <u>immediate conductance matrix.</u>

In our definitions for conductance matrices, note that $a_{ii} = 1$ ($E \leq A$), since a terminal is always connected to itself by a closed circuit.

If in the network A, diode elements are not present, then the conductance matrix A will be symmetric, i.e. $a_{ij} = a_{ji}$. The matrix A does not define uniquely an n-terminal network. For example, the three terminal networks of Figure 1 have the same immediate conductance matrix. This example is due to Hohn and Schissler (2).

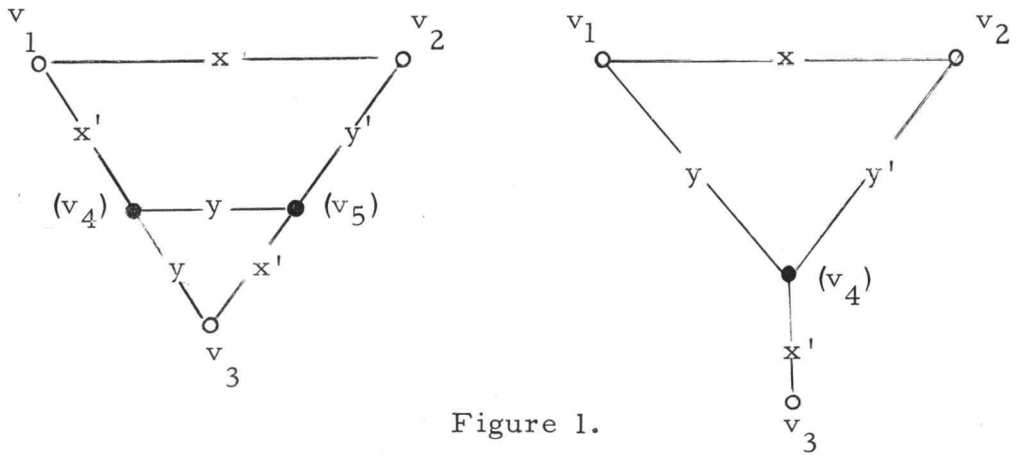

Figure 1.

$$\begin{bmatrix} 1 & x & x'y \\ x & 1 & x'y' \\ x'y & x'y' & 1 \end{bmatrix}$$

Definition 4.5. Let v_i and v_j be any two distinct terminals ($i \neq j$) of a contact network and a_{ij} be the corresponding immediate conductance. Then the complete conductance c_{ij} is the sum of the conductances of all paths from v_i to v_j.

By the definition above, for the complete conductance, we may write $c_{ij} = \sum a_{ik_1} a_{k_1k_2} \cdots a_{k_mj}$ where $k_1, k_2, \ldots, k_m$ runs through all possible subsets of the terminals $1, 2, \ldots, i-1, i+1, \ldots, j-1, j+1, \ldots, n$ for $m = 0, 1, 2, \ldots, n-2$. a_{ij} corresponds to $m = 0$.

Definition 4.6. The $n \times n$ Boolean matrix $C(A) = (c_{ij})$ is called the characteristic of the network A, where $c_{ij} = 1$ if $i = j$ and c_{ij} is the complete conductance if $i \neq j$. A is the immediate conductance matrix of the network.

For the network of Figure 2, we have

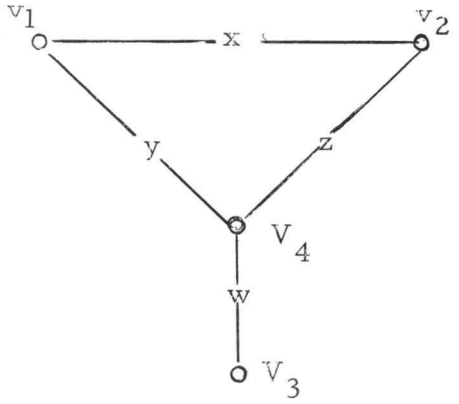

Figure 2.

$$A = \begin{bmatrix} 1 & x & 0 & y \\ x & 1 & 0 & z \\ 0 & 0 & 1 & w \\ y & z & w & 1 \end{bmatrix}$$

$$C(A) = \begin{bmatrix} 1 & x+yz & yw+xzw & y+xz \\ x+yz & 1 & wz+xyw & z+xy \\ yw+xzw & zw+xyw & 1 & w \\ y+xz & z+xy & w & 1 \end{bmatrix} .$$

The characteristic C(A) of a network with immediate conductance matrix A in a certain sense characterizes the network. That is, each c_{ij} represents all proper paths from v_i to v_j. This leads us to make a definition for the equivalence of two networks.

Definition 4.7. Two n-terminal networks with immediate conductance matrices A and B are <u>equivalent</u> (written $A \sim B$) if and only if C(A) = C(B).

For the networks A and B of Figure 1, C(A) = C(B). Hence, $A \sim B$ and the network B, requiring less hardware for its realization, may be substituted for A.

With a given n=terminal network, we can associate another

type of Boolean matrix. We select and number certain nodes in the network n+1, n+2, ... , n+k. We call these nodes non-terminal to distinguish them from the terminal nodes 1, 2, ... , n of the network. The non-terminal nodes are so chosen that between any of the n+k terminals of the network there appears at most a single contact or group of single contacts in parallel. Further, we assume every contact is included in the connection between some pair of terminals.

Definition 4.8. Let p_{ij} be the conductance between terminals v_i and v_j such that $p_{ij} = 0$ if there is no conductance and 1 if there is a short circuit, but otherwise p_{ij} is the symbol denoting a single contact or a sum of such symbols. The (n+k) x (n+k) matrix $P = (p_{ij})$ is called the primitive conductance matrix of the network (2).

In the second network of Figure 1, if we select a single non-terminal node in the obvious way and number it 4, we obtain the primitive conductance matrix

$$P = \begin{bmatrix} 1 & x & 0 & y \\ x & 1 & 0 & y' \\ 0 & 0 & 1 & x' \\ y & y' & x' & 1 \end{bmatrix}$$

CHAPTER V

THE ANALYSIS OF CONTACT NETWORKS

The fundamental problem of the analysis of a contact network involves writing a conductance matrix corresponding to the network and then determining the corresponding characteristic. In this section, we present some algebraic methods of analysis developed by the Russian mathematician A. G. Lunts about 1950. Many of the theorems of this section will be presented without proof since they are readily available [see (2), (7), (8), (13)]. These methods with some modifications will be used in describing a computer oriented procedure for analyzing contact networks.

The first method of analysis arises from the concepts of determinant and adjoint for Boolean matrices [see Def. 3.3].

Theorem 5.1. If A is an n x n immediate conductance matrix corresponding to some contact network, then $C(A) = \overline{A}$, i.e. $C_{ij}(A) = |A_{ji}|$.

Proof. See Lunts (7) [also see (13)]. Yoeli generalizes this result to matrices over a semiring.

The second method of analysis stems from Theorem 4.1

and the following theorem.

Theorem 5.2. If A is an n x n immediate conductance matrix, then $A^{n-1} = A^n = A^{n+1} = \ldots$.

Proof. See Lunts (7).

Theorem 5.3. If A is an n x n conductance matrix and C(A) the corresponding characteristic, then $C(A) = A^{n-1}$.

Proof. See Lunts (7).

As an immediate consequence of Theorems 5.1, 5.2, and 5.3, we have the following corollary.

Corollary. If A is an n x n conductance matrix, and $\bar{A}$ is the matrix adjoint of A, then $A\bar{A} = \bar{A}A = \bar{A}$.

In regard to Theorem 5.3, because of the behavior of the entries of A, it may happen that $C(A) = A^r$ for $r < n-1$. In the case where $a_{ij} \in B_m$, this occurs frequently.

In determining C(A) in the manner of Theorem 5.3, we can shorten the process by forming $AA = A^2$, then $A^2A^2 = A^4$, continuing until $A^{2^k} = A^{n-1}$. This will occur when $2^k \leq n - 1$. In other words, $k = [\ln (n-1)/ \ln 2] + 1$ where $[a]$ is the greatest integer not greater than a.

Consider the matrix product $AA = A^2$ where $a_i{}^2{}_j = \sum_{k=1}^{n} a_{ik} \cdot a_{kj}$. Notice that we must preserve the matrix A until the formation of A^2 is complete. We would like to transform A into C(A) without

having to maintain a second matrix. We can do this by replacing each a_{ij} by $\sum_{k=1}^{n} a_{ik} \cdot a_{kj}$ before proceeding to the next entry. We process the elements of A beginning with $a_{1,1}$, proceeding through the first row, then the second row, continuing through each row from left to right, until the last. Let $A^{(1)}$ be the result of such a transformation on A. Since a_{ij} is one of the terms of $\sum_{k=1}^{n} a_{ik} \cdot a_{kj}$, we will have $A^2 \leq A^{(1)}$. Denoting by $A^{(2)}$ the result of transforming $A^{(1)}$ into itself in the same manner, we will have $A^4 \leq A^{(2)}$. Continuing, we will have $A^{n-1} \leq A^{(k)}$ when $k = [\ln (n-1) / \ln 2] + 1$. Hence, since the maximum length of a proper path in an n-terminal network is n - 1, we have the following result.

Theorem 5.4. If A is an n x n immediate conductance matrix, then $C(A) = A^{(k)}$, where $k = [\ln (n-1)/ \ln 2] + 1$, and the elements of $A^{(p)}$ are formed by replacing each $a_{ij}^{(p-1)}$ of $A^{(p-1)}$ by $\sum_{k=1}^{n} a_{ik}^{(p-1)} a_{kj}^{(p-1)}$.

For the k described in Theorem 5.4, an interesting result occurs. It appears that k could be less than $[\ln (n-1)/ \ln 2] + 1$. In fact, by direct computation, we have shown that for $n \leq 6$,

$k = 2$, while $[\ln 5 / \ln 2] + 1 = 3$. For large n, considerable saving may occur in the number of iterations we must transform A. For large n, we could maintain $A^{(p-1)}$ while computing $A^{(p)}$, then compare $A^{(p-1)}$ and $A^{(p)}$. If $A^{(p-1)} = A^{(p)}$, then $A^{(p)} = C(A)$ and no further processing is required. This, of course, would have the disadvantage of requiring storage for an auxiliary matrix.

The method of analysis of Theorem 5.4 will be used in the procedures for machine analysis of networks. It has the advantage of not requiring a duplicate matrix or additional temporary storage locations.

Another method of analysis of a contact network depends on the notion of the characteristic function (7).

Definition 5.1. The characteristic function of the n x n conductance matrix A is the Boolean function $f_A(x_1, x_2, \ldots, x_n)$ given by

$$f_A(x_1, x_2, \ldots, x_n) = \sum_{i,j=1}^{n} a_{ij} x_i x_j'$$

where $A = (a_{ij})$.

Theorem 5.5. If A and B are n x n conductance matrices of two contact networks, then $C(A) \leq C(B)$ if and only if $f_A(x_1, x_2, \ldots, x_n) \leq f_B(x_1, x_2, \ldots, x_n)$.

The proof of the above result depends on the following two

lemmas. For a proof of the theorem and the lemmas see (7).

Lemma 5.1. $f_A(x_1, x_2, \ldots, x_n) = f_{C(A)}(x_1, x_2, \ldots, x_n)$.

Lemma 5.2. $f_A(C_{\beta 1}(A), C_{\beta 2}(A), \ldots, C_{\beta n}(A)) = 0$ for arbitrary $\beta = 1, 2, \ldots, n$.

Thus the characteristic function characterizes the operation of the network just as the characteristic matrix does. To show A and B are equivalent networks we need only establish that $f_A(x_1, x_2, \ldots, x_n) = f_B(x_1, x_2, \ldots, x_n)$.

The characteristic function can also be used for removal of non-terminal nodes of a primitive conductance matrix. Consider a non-terminal node v_r in a network with k non-terminal nodes and n terminal nodes. We want to remove v_r in such a manner that the characteristic of the network on the remaining terminals will be the same. The procedure was first developed by Lunts (7) [see also(2)]. Conductances a_{ir} and a_{rj} provide a path from v_i to v_j with conductance $a_{ir} \cdot a_{rj}$. We replace the conductances of the given network with others such that between each pair of terminals v_i and v_j $(i, j \neq r)$, there appears conductances corresponding to $a_{ij} + a_{ir} \cdot a_{rj}$, and then remove all conductances a_{ir} and a_{rj} between the non-terminal node v_r and the other terminals of the network. Thus, we remove the non-terminal node v_r, but we retain the same complete conductance on each pair of remaining terminals.

Definition 5.2. The greatest lower bound of a Boolean function $f(x)$ is a variable which is less than or equal to $f(x)$ and is greater than or equal to any variable which is less than or equal to $f(x)$.

Theorem 5.5. The greatest lower bound of a Boolean function, $f(x)$, is $f(0) \cdot f(1)$.

Proof. Since the range of the variable x is the set $\{0, 1\}$, we can write $f(x) = xf(1) + x'f(0)$. Applying Theorem 2.1, iii, and Definition 2.1, P_2, we have

$$\begin{aligned} f(x) &= xf(1)[1 + f(0)] + x'f(0)[1 + f(1)] \\ &= xf(1) + x'f(0) + f(0)f(1) \qquad \text{(Def. 2.1, } P_3, P_4) \\ &= f(x) + f(0)f(1). \end{aligned}$$

Hence, by Def. 2.2, $f(0)f(1) \leq f(x)$ and so $f(0)f(1)$ is a lower bound. To show $f(0)f(1)$ is the greatest lower bound, let b be any lower bound of $f(x)$. Then, $b \leq f(0)$ and $b \leq f(1)$. Applying Theorem 2.3 and the transitive property of the relation $\leq$ (Theorem 2.2, iii), we have $b \leq f(0)f(1)$. Thus, $f(0)f(1)$ is the greatest lower bound.

Definition 5.3. The removal of x from $f(x)$ by replacing $f(x)$ by $f(0)f(1)$ is called the exclusion of the variable x from $f(x)$ and will be denoted by $(Ex)f(x) = f(0)f(1)$.

Lunts (7) has shown that if we exclude the variable x_n from the characteristic function $f_A(x_1, x_2, \ldots, x_n)$, we shall obtain a

characteristic function of some n - 1 terminal network $f_B(x_1, x_2, \ldots, x_{n-1})$ where $b_{ij} = a_{ij} + a_{in}a_{nj}$ $(i, j = 1, 2, \ldots, n-1)$. In other words,

$$\text{if } A = \begin{bmatrix} a_{11} & a_{12} \cdots a_{1} \\ \cdot & \cdot \cdots \cdot \\ \cdot & \cdot \cdots \cdot \\ \cdot & \cdot \cdots \cdot \\ a_{n1} & a_{n2} \cdots a_{nn} \end{bmatrix} \text{, we have}$$

$$B = \begin{bmatrix} a_{11} & a_{12} \cdots a_{1,n-1} \\ \cdot & \cdot \cdots \cdot \\ \cdot & \cdot \cdots \cdot \\ \cdot & \cdot \cdots \cdot \\ a_{n-1,1} & \cdot \cdots a_{n-1,n-1} \end{bmatrix} + \begin{bmatrix} a_{1n} \\ a_{2n} \\ \cdot \\ \cdot \\ \cdot \\ a_{n-1,n} \end{bmatrix} \cdot \begin{bmatrix} a_{n1} \cdots, a_{n,n-1} \end{bmatrix}$$

Furthermore, the following theorem holds:

Theorem 5.6. In the exclusion of the terminal v_n from the n-terminal network A, the elements of the characteristic corresponding to the network B, formed from the remaining terminals, remains unchanged, i.e. $C_{ij}(A) = C_{ij}(B)$ for $i, j = 1, 2, \ldots, n-1$.

Proof. See (7).

Thus, given a primitive conductance matrix of n + k nodes (n terminal and k non-terminal), we can obtain the immediate conductance matrix by repeated application of the exclusion process until no non-terminal nodes remain.

Since for a two terminal network the immediate conductance matrix is its characteristic, we can obtain $C_{ij}(A)$ by excluding successively the terminals v_n, v_{n-1}, $\cdots$, $v_{i+1, i-1}$, $\cdots$, v_{j+1}, v_{j-1}, $\cdots$, v_1.

We illustrate the exclusion of non-terminal nodes 4 and 5 of the first network of Figure 1. As primitive conductance matrix, we have

$$P = \begin{bmatrix} 1 & x & 0 & x' & 0 \\ x & 1 & 0 & 0 & y' \\ 0 & 0 & 1 & y & x' \\ x' & 0 & y & 1 & y \\ 0 & y' & x' & y & 1 \end{bmatrix}.$$

Excluding the terminal v_5, we have

$$\begin{bmatrix} 1 & x & 0 & x \\ x & 1 & x'y' & 0 \\ 0 & x'y' & 1 & y \\ x' & 0 & y & 1 \end{bmatrix} .$$

Excluding the terminal v_4, we have

$$A = \begin{bmatrix} 1 & x & x'y' \\ x & 1 & x'y' \\ x'y' & x'y' & 1 \end{bmatrix} ,$$

which is the immediate conductance matrix shown in Figure 1.

CHAPTER VI

MACHINE ALGORITHMS FOR CONTACT NETWORKS

In the preceding sections the general concept of a contact network has been introduced and Boolean matrix methods of analysis of such networks have been discussed. We now describe procedures for a digital computer for Boolean matrices as applied to switching network analysis. The language of these procedures will be ALGOL 60, the International Algorithmic Language (9).

The problem of representing Boolean functions (elements of B_m) in a digital computer has been studied in detail by Witcraft (12). We sketch briefly the representation presented in his paper. For a detailed description consult Chapter IV of his paper.

The computer's arithmetic unit is binary with a word length of at least m bits where m is the number of contact variables in the network. The computer further has the usual capabilities of forming the logical sum and product of two words, forming the one's complement of a word, shifting the bits of a word and counting the number of shifts.

The machine representation of Boolean functions is

restricted to polynomials. This is not a great restriction, since from the distributive postulates (Def. 2.1, P_3) and the duality theorems (Thm. 2.1, vi and vii) of a Boolean algebra, it follows that any Boolean function can be expressed as a polynomial.

Consider the polynomial $P = \beta_1 + \beta_2 + \ldots + \beta_k$ where β_i is a monomial. P will be represented in the machine as a sequence of word pairs $(M_{\beta_1}, M^*_{\beta_1}), \ldots, (M_{\beta_k}, M^*_{\beta_k})$ where each word pair corresponds to a monomial β_j. Each monomial β_j may have up to m variables representing contacts in the network, so for each of $x_1, x_2, \ldots, x_m$, we indicate whether x_i is a factor of β_j or x_i' is a factor of β_j or neither x_i nor x_i' is a factor of β_j by the following correspondence:

i. $[M_{\beta_j}]_i = 1$ and $[M^*_{\beta_j}]_i = 0$ if x_i is a factor of β_j;

ii. $[M_{\beta_j}]_i = 1$ and $[M^*_{\beta_j}]_i = 1$ if x_i' is a factor of β_j;

iii. $[M_{\beta_j}]_i = 0$ and $[M^*_{\beta_j}]_i = 0$ if neither x_i nor x_i' is a factor of β_j.

For example, suppose m = 4 and we want to represent the polynomial $x_1x_2' + x_1x_2x_3$. Then, the machine representation will be either (1100, 0100), (1110, 0000) or

(1110, 0000), (1100, 0100),

the ordering of the set of word pairs being immaterial.

In addition, associated with each sequence of word pairs

representing the terms of a polynomial is a location word (P, m). P is the memory location of the first monomial, and m is the number of monomials in the polynomial.

In case the polynomial is a Boolean constant, i.e. 0 or 1, the location word will contain that information.

The user of the subroutines for handling Boolean polynomials thus, need only refer to the location words in writing a program. Hence, our procedures for Boolean matrices will only require two pieces of information for each entry in the matrix, namely, the storage location of the polynomial and the number of monomials of the polynomial.

We now describe briefly the code procedures which we use for manipulating Boolean polynomials. For a detailed description of these routines see (12).

BPP (P1, m1, P2, m2, P3, m3);

comment BPP is a code procedure which forms the product of two Boolean polynomials in machine representation. On entry, the location words (P1, m1) and (P2, m2) of the operands and P3, the location at which the product is to be stored, must be provided. On exit, the location word, (P3, m3) of the product, is available.

BPA (P1, m1, P2, m2, P3, m3);

comment BPA is a code procedure which forms the sum of two Boolean polynomials in machine representation. Entry and exit data are as described above for BPP.

QCF (P1, m1, P2, m2);

comment QCF is a code procedure which forms Quine's canonical form (see p. 6) of a Boolean polynomial in machine representation. On entry, the location word (P1, m1) of the operand and P_2, the location at which the canonical form is to be stored must be provided. On exit, the location word of the result, (P2, m2), is available.

BPE (P1, m1, P2, m2, I);

comment BPE is a code procedure which checks two Boolean polynomials in machine representation for equality. The two polynomials must be in Quine's canonical form (QCF). On entry, the location words (P1, m1) and (P2, m2) of the operands must be provided. On exit, the indicator I has a logical value, i.e. true or false, depending on whether P1 = P2 or P1 $\neq$ P2.

We assume there is sufficient random access storage available in which to store the procedures and all of the data.

This assumption is generally not true, so that a limitation depending on the size of the memory unit of the computer would have to be placed on either the length of the polynomials in each entry or on the number of terminals in the network. No input or output procedures are included. For a description of input-output routines consult (12).

We describe four procedures (subroutines) for computer manipulation of Boolean matrices as applied to contact networks. In these procedures it will be assumed that the integer arrays A and a corresponding to the storage locations and the number of monomials, respectively, for each entry in the matrix are already stored in the computer memory. L is the first memory location available for storing results, and further results are stored in sequence in locations L+1, L+2, ... n is the number of terminals in the network.

The first procedure, identified by CHAR, transforms an nth order conductance matrix A into its characteristic C(A) according to Theorem 5.4. In addition CHAR forms Quine's canonical form of each entry in C(A). Thus, on exit, C(A) can be compared with C(B) for the equivalence of A and B as given by Definition 4.7.

The second procedure described below is identified by

CHAR FUN. Let $A = (a_{ij})$ be an n x n conductance matrix. Let $X = (x_{ij})$ where $x_{ij} = x_i \cdot x'_j$. CHAR FUN forms the characteristic function of the matrix A by forming $A \cdot X$ (Boolean multiplication, not matrix multiplication), and summing all the elements in the result. In addition, the characteristic function is placed in canonical form. Thus, on exit, f_A is available for comparison with some f_B (previously determined) to see if A is equivalent to B as given by Theorem 5.5 and Definition 4.7.

The third procedure, identified by EX excludes the terminal v_r from a network by replacing a_{ij} by $a_{ij} + a_{ir} \cdot a_{rj}$ for $i, j \neq r$. Thus, EX forms an (n-1) x (n-1) matrix B from an n x n matrix A, so that $C_{ij}(A) = C_{ij}(B)$ for $i, j = 1, 2, \ldots, n-1$. EX is used for removing non-terminal nodes.

The last procedure has the identifier EQ. It compares two Boolean matrices of the nth order for equality. Each entry of the matrices must be in the same canonical form before activating the procedure. Thus, if C(A) and C(B) are the characteristics corresponding to two networks A and B, then EQ answers the question, is $A \sim B$? The answer will be in the form of a Boolean logical value, i.e. true or false.

MACHINE ALGORITHMS

```
procedure CHAR (A, a, L, n); value n;
integer array A, a; integer L, n;
begin integer i, j, k, s, m, t;
      integer array P[1:3], p[1:3];
      m := entier (ln(n-1)/ ln(2)) + 1;
      s := L;
      t := 0;
   LL: t := t + 1;
      for i := 1 step 1 until n do
      for j := 1 step 1 until n do
      begin P[1] := s;
            BPP (A[i,1], a[i,1], A[1,j], a[1,j], P[1], p[1]);
            s := s + 2 x p[1];
            for k := 2 step 1 until n do
            begin P[2] := 1;
              BPP (A[i,k], a[i,k], A[k,j], a[k,j], P[2], p[2]);
              s := s + 2 x p[2];
              P[3] := 1;
              BPA (P[1], p[1], P[2], p[2], P[3], p[3]);
              s := s + 2 x p[3];
```

```
                P[1] := P[3];
                p[1] := p[3];
            end
            A[i,j] := P[3];
            a[i,j] := p[3];
        end
        if t < m then go to LL else
        for i := 1 step 1 until n do
        for j := 1 step 1 until n do
        begin P[1] := s;
            QCF (A[i,j], a[i,j], P[1], p[1]);
            s := s + 2 x p[1];
            A[i,j] := P[1];
            a[i,j] := p[1];
            s := s + 2 x p[1];
        end
        L := s;
end CHAR
```

```
procedure CHAR FUN (A, a, X, x, F, f, L, n); value n;
integer array A, a, X, x; integer F, f, L, n;
begin integer i, j, k, s;
      integer array P[1:3], p[1:3];
      s := L;
      P[1] := s;
      BPP (A[1,1], a[1,1], X[1,1], x[1,1], P[1], p[1]);
      s := s + 2 x p[1];
      for i := 1 step 1 until n do
      for j := 2 step 1 until n do
      begin P[2] := s;
            BPP (A[i,j], a[i,j], X[i,j], x[i,j], P[2], p[2];
            s := s + 2 x p[2];
            P[3] := s;
            BPA (P[1], p[1], P[2], p[2], P[3], p[3];
            s := s + 2 x p[3];
            P[1] := P[3];
            p[1] := p[3];
      end
      F := 1;
      QCF (P[3], p[3], F, f);
      s := s + 2 x f;
```

```
        L := s;

end CHAR FUN
```

```
procedure EX (A, a, L, r, n); value r, n;
integer array A, a; integer r, n;
begin integer i, j, s;
      integer array P[1:2], p[1:2];
      s := L;
      for i := 1 step 1 until r-1, r+1 step 1 until n do
      for j := 1 step 1 until r-1, r+1 step 1 until n do
      begin P[1] := s;
            BPP (A[i,r], a[i,r], A[r,j], a[r,j], P[1], p[1]);
            s := s + 2 x p[1];
            P[2] := s;
            BPA (A[i,j], a[i,j], P[1], p[1], P[2], p[2]);
            s := s + 2 x p[2];
            A[i,j] := P[2];
            a[i,j] := p[2];
      end
end EX
```

```
procedure EQ (A, a, B, b, I, n); value n;
integer array A, a, B, b; Boolean I; integer n;
begin integer i, j;
        I := true
        for i := 1 step 1 until n do
        for j := 1 step 1 until n do
        begin BPE (A[i, j], a[i, j], B[i, j], b[i, j], I);
              if ¬ I then go to LL
        end
LL: end EQ
```

BIBLIOGRAPHY

1. Baker, James J. A note on multiplying Boolean matrices. Communications of the Association for Computing Machinery 5:102. 1962.

2. Hohn, F.E. and L.R. Schissler. Boolean matrices and the design of combinational relay switching circuits. The Bell System Technical Journal 34:177-202. 1955.

3. Hohn, F.E., S. Seshu and D.D. Aufenkamp. The theory of nets. Transactions of the Institute of Radio Engineers EC-6:154-161. 1957.

4. Huntington, E.V. Sets of independent postulates for the algebra of logic. Transactions of the American Mathematical Society 5:288-309. 1904.

5. Laxdal, A.L. A mechanization of Quine's canonical form. Master's thesis. Corvallis, Oregon State College, 1959. 39 numb. leaves.

6. Luce, R. Duncan. A note on Boolean matrix theory. Proceedings of the American Mathematical Society 3:382-388. 1952.

7. Lunts, A.G. Algebraic methods of analysis and synthesis of contact networks. Akademiia Nauk SSSR Izvestia, Ser. Matematika 16:405-426. 1952. (Tr. by H.E. Goheen, Dept. of Mathematics, Oregon State University, Corvallis, Oregon.)

8. Lunts, A.G. The application of Boolean matrix algebra to the analysis and synthesis of relay contact networks. Akademiia Nauk SSSR Doklady 70:421-423. 1950.

9. Naur, Peter (ed.), et al. Report on the algorithmic language ALGOL 60. Communications of the Association for Computing Machinery 3:299-314. 1960.

10. Warshall, Stephen. A theorem on Boolean matrices. Journal of the Association for Computing Machinery 9:11-12. 1962.

11. Whitesitt, J. Eldon. Boolean algebra and its applications. Reading, Massachusetts, Addison-Wesley, 1961. 182 p.

12. Witcraft, D.A. The mechanization of logic II. Master's thesis. Corvallis, Oregon State College, 1960. 108 numb. leaves.

13. Yoeli, M. A note on a generalization of Boolean matrix theory. The American Mathematical Monthly 68:552-557. 1961.

APPENDIX

TWO SPECIAL CASES

Suppose A is a conductance matrix where for each a_{ij} ϵA either $a_{ij} = 1$ or $a_{ij} = 0$. Thus, there are no contacts in the network. There are only closed and open circuits. Baker (1) and Warshall (10) report the use of this type of Boolean matrix in the analysis of flow charts and in handling other types of problems in program topology.

Definition 1. Let $A = (A_{ij})$ be an n x n Boolean matrix where $A_{ij} = 0$ or 1. Define B by the following algorithm:

```
begin
comment The logical values true and false instead of their
common representation by 1 and 0, respectively, are used. ∨ and
∧ correspond to + and · ;
integer i, j, k; Boolean array A, B[1:n, 1:n];
for i := 1 step 1 until n do
for j := 1 step 1 until n do
begin if A[j, i] then
        for k := 1 step 1 until n do
        A[j, k] := A[j, k] ∨ A[i, k];
```

```
end
for i := 1 step 1 until n do
for j := 1 step 1 until n do
B[i,j]:= A[i,j];
end
```

Theorem 1. Let A be transformed into B by the above algorithm. Then B = C(A).

Proof. See (10).

An apparently similar algorithm is described without proof in a short note by Baker (1). It is actually quite different and, in fact, more laborious.

Definition 2. Let $A = (A_{ij})$ be an n x n Boolean matrix where $A_{ij} = 0$ or 1. Define B by the following algorithm:

```
begin
integer i, j, k; Boolean array A, B[1:n,1:n];
for i := 1 step 1 until n do
for j := 1 step 1 until n do
        B[i,j] := A[i,j];
LL: for i := 1 step 1 until n do
    for j := 1 step 1 until n do
        if B[i,j] then
```

```
        for k := 1 step 1 until n do
            B[i,k] := B[i,k]∨B[j,k];
    for i := 1 step 1 until n do
    for j := 1 step 1 until n do
        if ¬(B[i,j]≡A[i,j]) then
            begin for i := 1 step 1 until n do
                  for j := 1 step 1 until n do
                      A[i,j] := B[i,j];
                  go to LL;
            end
    end
```

Theroem 2. Let A be transformed into B by the above algorithm. Then B = C(A).

Proof. Consider the statement B[i,k] := B[i,k]∨B[j,k] in the algorithm above. This is equivalent to replacing B_{ik} by $B_{ik} + B_{jk}$ for k = 1, 2, ... , n provided $B_{ij} = 1$. The only way a change can occur in B_{ik} is for B_{jk} to be 1. But, if $B_{jk} = 1$ (since $B_{ij} = 1$), we have a path from v_i to v_k through v_j, i.e. $B_{ij} \cdot B_{jk} = 1$. Hence, B_{ik} should be set to 1. Obviously, the result of such a transformation forms a new matrix greater than or equal to the original matrix (since we replace 0 by 1, but never 1 by 0). Hence, when one

complete pass through the matrix has been made without changing it, we must have all possible paths through the network.