J. Paul Roth
E. G. Wagner

Algebraic Topological Methods For the Synthesis of Switching Systems Part III*
Minimization of Nonsingular Boolean Trees

Abstract: An algorithm is given for solving a general problem in combinational switching-circuit minimization theory. The circuits considered consist of a disjunction (OR-ing together) of trees of any set of logical elements, with the restriction that in any given tree no input appears more than once. To each logical element is attached a positive cost. A method is presented for designing a minimum-cost circuit of this variety for any given logical function. Two parallel treatments are given, one viewing it as an abstract mathematical problem, the other considering it as an engineering problem.

Introduction

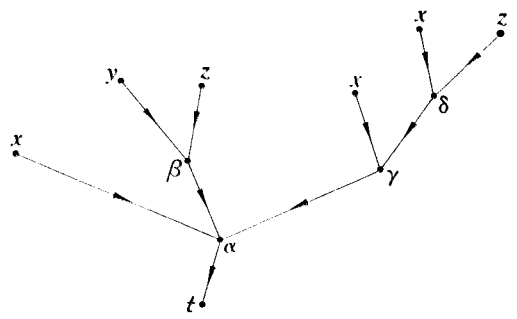
This paper is a continuation of efforts (Refs. [1, 2]) to develop a theory for the effective design of automata. In previous publications, algorithms for designing switching circuits of the so-called "normal form" were devised. These "extraction" algorithms have been programmed on the IBM 704 and have been useful to some logical designers. The kind of circuit that they design, nevertheless is of quite a special variety. This paper makes a rather strong generalization of this class, for which generalized extraction algorithms are still applicable.

Description of the problem

A few preliminaries are required. The "functional expression"

$$t=\alpha[x, \beta(y, z), \gamma(x, \delta(x, z))] \qquad (1)$$

can be represented by the following "functional tree" (circuit block diagram):



The Greek letters α, β, γ, δ denote functions, and the Latin letters x, y, z, t the variables. In the functional expression (1), α is a function of three variables; in the graph (circuit) the node (circuit element) labelled α has three branches (wires) directed toward it, and the opposite ends of these branches carry the labels of the arguments (inputs) of α; the first argument is x, the second is $\beta(y, z)$, which is itself a function of y and z, et cetera. Conversely, every such functional tree (circuit) completely defines some functional expression. Note that the graph has no closed loops—it is a tree, of a rather special variety, as is precisely defined in Section 1.1.

The expression (1) gives t as a composite function, but ultimately it is a function of the three independent variables x, y, z; thus $t=T(x, y, z)$.

In general, drawing from an allowable set $\mathfrak{B}$ of "primitive" functions $\alpha, \beta, \cdots$, it is possible to write T in many different ways as a composite function drawn from the set $\mathfrak{B}$.

In this paper the functions are all Boolean, that is, the arguments and values of the function are either 0 or 1.

The following may be considered as the general problem of design of single-output combinational circuits:

*Part I of this series was published in *Transactions of the American Mathematical Society* (Ref. [1]). Part II will appear in *The Proceedings of the International Symposium on the Theory of Switching* (Ref. [2]).

Given a set of allowable functions (circuit elements), given a function T of n variables, to find a functional tree (circuit) for T using primitive functions (circuit elements) from $\mathfrak{B}$, which uses the least possible number of these primitive functions (elements). More generally, we weight each function according to its "cost" and seek to minimize the total cost. More precisely, the problem is to devise an algorithm so that, given any T, the algorithm will automatically produce a minimum tree. This algorithm must be effective in quite a practical way. To convey the kind of practicality involved, let it be required that the algorithm, when programmed on the IBM 704, should be capable of solving, say, an 8-variable problem with eight hours of machine time. These requirements rule out any kind of exhaustive procedure, even for extremely small problems.

To meet these stringent requirements, we find it expedient to specialize the problem. An expression is said to be *nonsingular* if no variable appears more than once. We deal with disjunctions of nonsingular Boolean expressions, which are of the following form:

$$\mathrm{v}(\phi_1, \phi_2, \cdots, \phi_k) ,$$

where v is a k-variable "or" or "disjunction" and each ϕ is, by itself, a nonsingular Boolean expression. For instance

$$\mathrm{v}[\alpha(x, y), \beta(x, z)]$$

is a disjunction of nonsingular Boolean expressions. Any "normal-form" expression, e.g., abc v acd v efz is a disjunction of nonsingular Boolean expressions.

In this paper the extraction algorithm is applied to find minima over this subclass of Boolean expressions. This problem may be given a logical twist. A disjunction of nonsingular Boolean trees is a rather strong generalization of Quine's normal form. The fundamental formulas of Quine, where no literal appears twice, is generalized to a Boolean tree, with the corresponding restriction of nonsingularity. Just as Quine takes disjunctions of normal formulas, we take disjunctions of nonsingular Boolean trees. There is one further element of generalization, which is *essential* to the understanding of the extraction algorithm—it corresponds to the fact that we include don't-care conditions: If K corresponds to a truth function f and L to a truth function h, then the problem is to find a disjunction of nonsingular Boolean trees, of minimum cost, whose output is g such that $h \Rightarrow g \Rightarrow f$.

References [8] through [10] are concerned with minimization over the class of normal forms. It is of interest that Urbano and Mueller, as well as we, approach the synthesis problem from a combinatorial topological point of view. An algorithm for the general single-output problem described above, minimization that is over the class of *all* Boolean expressions, is given in Ref. [5]. Finally an algorithm for the general multiple-output problem, involving no feedback, is given in Ref. [6].

One of the difficulties in writing a paper in this field is that it is addressed to two classes of individuals — engineers and mathematicians—who are generally separated rather widely in training, orientation and language. Our attempted escape from the dilemma implied is to write two complementary versions, one for mathematicians, with algorithms, constructions and proofs given in abstract and general form; the other for engineers, supplying physical motivation and, for a significant special case, working out the complete details of the algorithm.

In Chapter I, the mathematical version, cubical complexes and their relation to Boolean functions are described in an introductory section. Functional expressions and Boolean trees are treated in Section 1. The question of production of expressions for a given function is treated in Section 2. In Section 3, the minimization problem is described. In Section 4, a generalized extraction algorithm, applicable to the present problem, is given. The efficiency of the algorithm is considered in Section 5.

In Chapter II, the engineering version, Section 1 introduces the form of circuitry under consideration and shows how each such circuit can be represented by a functional expression (composite function). Section 2 shows how the output function of a circuit may be represented by matrices, one giving the input combinations for which it is on (the *on-matrix*), the other giving the combinations for which it is off (the *off-matrix*). A method is developed whereby from the on-matrix we can "generate" circuits which, when they are "OR-ed" together, will result in a circuit with the desired output function. Section 3 shows how we may "extract" from the circuits developed by the method of Section 2 a set which, when "OR-ed" together, will realize the desired output function at minimum cost. Section 4 gives, in flow chart form, a complete algorithm for a special case of this kind of circuitry, followed by an example.

Chapter I
Mathematical Version

J. Paul Roth

Cubical complexes and Boolean functions

A brief resumé is given of the calculus of cubical complexes and its relation to Boolean functions. A complete description is given in Ref. [1].

Let $Q^n = \{(u_1, \cdots, u_n) \mid u_i = 0, 1 \text{ or } x\}$; Q^n will be termed the *n-cube*. An element $(u_1, \cdots, u_n)$ of Q^n is termed a *cube*, its *dimension* is the number k of u_i equal to x, its *codimension* is $n-k$. A cube of dimension k is termed a *k-cube*. (Both Q^n and its element $(xx \cdots x)$ then are termed the n-cube, but this will cause no confusion.) 0-cubes are called *vertices*. Let V^n be the set of all vertices of Q^n: $V^n = \{(u_1, \cdots, u_n) \mid u_i = 0 \text{ or } 1\}$. The cube $c' = (v_1, \cdots, v_n)$ is termed a *face* of $c = (u_1, \cdots, u_n)$ if $u_i = v_i$ or x for all i. A *cubical complex* K is a subset of Q^n having the property that if c belongs to K then every face

of c belongs to K. If c and c' belong to K and c' is a face of c then c is a *coface* of c'. The *intersection* of cubes $a=(a_1, \cdots, a_n)$ and $b=(b_1, \cdots, b_n)$ is defined by the following matrix

$\cap$	0	1	x
0	0	ϕ	0
1	ϕ	1	1
x	0	1	x

and the rule

$a \cap b = \phi$ if for any i, $a_i \cap b_i = \phi$,

$a \cap b = (a_1 \cap b_1, \cdots, a_n \cap b_n)$ otherwise.

A cubical complex K is termed a *Boolean complex* if it has the following property: If K contains all the vertices of a cube c then it contains c itself. If C is a set of cubes, let $\mathfrak{K}(C)$ denote the Boolean complex determined by the vertices of C; $\mathfrak{K}(C)$ will be referred to as the Boolean complex determined by C.

Let f be a Boolean function of n variables, i.e., f is a map of V^n into V^1. Such a map can be uniquely extended to a map f_* of Q^n into Q^1 by the rule: if all the vertices of a cube c of Q^n are mapped into 1 by f, let $f_*c=1$, and similarly for maps into 0; if neither of these, let $f_*(c)=x$. Then $f_*^{-1}(1)$ (as well as $f_*^{-1}(0)$) determines a *Boolean complex* $\mathfrak{K}(f)$. There is in fact a one-to-one correspondence between Boolean functions and Boolean complexes. A set C of cubes of K, for which all vertices of K are faces of some cubes of C, is termed a *cover* of K.

1. Functional expressions and Boolean trees

• *1.1 Functional expressions*

Let f be a Boolean function of n variables $a_1, \cdots, a_n$. The elements a_i will be termed the initial variables. Let $\mathfrak{B}$ be a set of Boolean functions $\alpha, \beta, \cdots, \omega$ to be termed the *bag* of primitive functions. The arguments for these functions are unspecified. As many copies of each of these functions are available as are required for any given construction.

A functional $\mathfrak{B}$-expression for f, or more simply, a *functional expression* for f is a formula for f, written as an explicit composite function of primitive functions from the bag $\mathfrak{B}$. That is, a functional expression $E(f)$ has the form

$$z=\alpha(y_1, \cdots, y_p),\ \alpha \in \mathfrak{B}, \text{ or } z=a_i, \qquad 0<i\leqslant n,$$

where

$$y_i=\beta_i(x_{i1}, \cdots, x_{iq(i)}),\ \beta_i \in \mathfrak{B}, \text{ or } y_i=a_j, \qquad 0<j\leqslant n,$$

and (1)

$$x_{ik}=\gamma_{ik}(w_{ik1}, \cdots, w_{ikr(ik)}),\ \gamma_{ik} \in \mathfrak{B}, \text{ or } x_{ik}=a_j,\ 0<j\leqslant n,$$

$$\cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots$$

$$b_{ik\cdots m}=\omega_{ik\cdots m}(a_{ik\cdots 1}, \cdots, a_{ik\cdots ms(ik\cdots m)}),$$

$$\omega_{ik\cdots m} \in \mathfrak{B} \text{ or } b_{ik\cdots m}=a_t, \qquad 0<t\leqslant n.$$

For instance $z=\alpha[a_1, \beta(a_2, \gamma(a_1, a_3)), \partial(a_1, \lambda(a_1, a_4))]$ is a functional expression. A functional expression is termed *nonsingular* if no initial variable appears more than once.

• *1.2 Trees*

Let T be a *tree*, a linear graph containing no cycles. Let each branch of T be directed, and in such a way that at each node (vertex) excepting one, exactly one branch is directed away from the node. For the exceptional node, let no branch be directed away from it. "Tree" will henceforth always mean a tree which can be and is directed in this sense.

A node of a directed graph T, and in particular of a tree, may be placed in one of the three categories: 1°) All the branches are directed away from it. This will be termed an *input node* of T. 2°) All the branches are directed toward it. This will be termed an *output node* of T. 3°) It is neither 1° nor 2°, and then it will be called an *interior node* of T. It will be assumed that T is nontrivial, that is, T consists of at least one node. An algebraic structure will now be erected on the tree T.

• *1.3 Boolean trees*

1) Let $\hat{u}, \hat{v}, \hat{w}, \cdots$ denote the nodes (or vertices) of T. Let the branches of T be labelled in the following manner: Let $\hat{u}\hat{v}$ denote the branch connecting node $\hat{u}$ to node $\hat{v}$, which is directed from $\hat{u}$ to $\hat{v}$.

2) For $\hat{v}$ a node of T let $P(\hat{v})$ denote the set of nodes $\hat{w}$ such that $\hat{w}\hat{v}$ is a (directed) branch of T; $P(\hat{v})=\{\hat{w} \mid \hat{w}\hat{v} \in T\}$. Let $P(\hat{v})$ be termed the *input nodes* of $\hat{v}$.

3) From the definition of T, it follows that each $\hat{w}$ in T, except for the output node of T, belongs to exactly one $P(\hat{v})$.

4) Simply order each $P(\hat{v})$; for $P(\hat{v})=\{w_1, \cdots, w_k\}$, let $\hat{w}_1<\hat{w}_2<\cdots<\hat{w}_k$ signify the simple order introduced. If $\hat{w}_q$ is the qth term in this order, let it be referred to as the qth *input node of* $\hat{v}$.

5) To each input node $\hat{\imath}$ of T attach an initial variable $a(\hat{\imath})$. To each other node $\hat{v}$ of T, having k input nodes, attach a primitive function α of k variables: $\alpha=\alpha_{\hat{v}}(x_1, \cdots, x_k)$; let the qth argument x_q of α be the function or variable attached to the qth input node of $\hat{v}$.

6) A *Boolean tree* is then a triple $(T, <, \mathfrak{A})$ consisting of a tree T, a set $<$ of simple orders among the input nodes of each node, and a set $\mathfrak{A}$ of assignments of initial variables or Boolean functions from $\mathfrak{B}$ to the nodes of T. A tree is termed *nonsingular* if $a(\hat{\imath})=a(\hat{\jmath})$ implies $\hat{\imath}=\hat{\jmath}$. *Par un abus de terminologie,* we shall denote a Boolean tree as well by T.

7) *T defines a Boolean function,* the output function of T, attached to its output node, in the following way. Let the *degree* of a node $\hat{v}$ be the length of the longest path from an input node of T to $\hat{v}$. (The length of a path is the number of branches in the path.) An input node of T has degree 0. Since each tree is assumed to be nontrivial

it has at least one node. Let T_1 be a tree with exactly one node; since this node satisfies the definition of an input node, some variable is attached to this node—let it be a_i. This node is also an output node and in this case the output function attached to this node is the function $f=a_i$. Having disposed of the trivial case, let T be a tree with more than one node, and let $\hat{v}$ be a node of T of degree 1. Let $\hat{\imath}_1, \cdots, \hat{\imath}_k$ be its input nodes, simply ordered by subscript. Then $a(\hat{\imath}_1), \cdots, a(\hat{\imath}_k)$ being the initial variables attached to these nodes, and $\alpha_{\hat{v}}(x_1, \cdots, x_k)$ being the primitive function attached to $\hat{v}$, let $x_1=a(\hat{\imath}_1), \cdots, x_k=a(\hat{\imath}_k)$. By these identifications, $b_{\hat{v}}=\alpha_{\hat{v}}(a(\hat{\imath}_1), \cdots, a(\hat{\imath}_k))$ becomes a Boolean function of the initial variables $a_1, \cdots, a_n$. Let such an identification procedure be made for all nodes of degree 1.

Let $\hat{w}$ be a node of degree 2 and let $\hat{u}_1, \cdots, \hat{u}_r$ be its input nodes, simply ordered by subscript. Let $\beta_{\hat{w}}(y_1, \cdots, y_r)$ be the function attached to $\hat{w}$. Let $c(\hat{u}_1), \cdots, c(\hat{u}_r)$ be the function attached to these nodes: each $c(\hat{u}_s)$ is equal to some a_j; or else it is equal to some $\alpha_s(a(\hat{\imath}_1), \cdots, a(\hat{\imath}_k))$, depending upon whether $\hat{u}_s$ is a node of degree 0 or degree 1. Let $y_1=c(\hat{u}_1), \cdots, y_r=c(\hat{u}_r)$. Thus $\hat{w}$ becomes associated with a function whose arguments are variables of degree 0 or 1. The definition proceeds inductively to the output node of T, the node of highest degree, to define a function f_T of the initial variables, termed the *output function* of T.

Let $K(T)$ denote the Boolean complex defined by f_T. Note that a nonsingular tree has as its output function a nonsingular functional expression.

• *1.4 Boolean tree of a functional expression*

Consider the functional expression (1) of 1.1. To the expression $\alpha(y_1, \cdots, y_p)$ corresponds the Boolean tree, having p input nodes $\hat{u}_1, \cdots, \hat{u}_p$, or nodes of degree 0, simply ordered by subscript, and one output node $\hat{u}_0$, with branches $\hat{u}_1\hat{u}_0, \cdots, \hat{u}_p\hat{u}_0$. To $\hat{u}_1, \cdots, \hat{u}_p$ is attached $y_1, \cdots, y_p$ respectively; to $\hat{u}_0$ is attached α. Likewise each $\beta_i(x_{i1}, \cdots, x_{iq(i)})$ as a functional expression in its own right admits of the same kind of representation.

To represent the expression

$$\alpha[y_1, \cdots, y_{i-1}, \beta_i(x_{i1}, \cdots, x_{iq(i)}), y_{i+1}, \cdots, y_p],$$

form the following simplicial map of $T[\alpha(y_1, \cdots, y_p)] \cup T(\beta_i(x_{i1}, \cdots, x_{iq(i)}))$ obtained by mapping $\hat{v}_0$ on $\hat{u}_i$, mapping each other vertex (node) on itself. Let such an identification be made for each i between 0 and $p+1$. Similarly in this new structure let such an identification be made between the nodes to which are attached the variables x_{ik} and the nodes labelled γ_{ik} in their corresponding Boolean trees. This process is continued up to $b_{ik\cdots m}$, until all functions used in (1) are accounted for. Thus is defined the Boolean tree of a functional expression.

• *1.5 Equivalence of Boolean trees*

Each vertex $\hat{v}$ of T has associated with it, by construction, a Boolean function of the initial variables: let $f_{\hat{v}}$ denote this function.

Boolean trees S and T shall be considered as *equivalent* if there exists a one-to-one map ϕ of vertices of S onto vertices of T such that:

(1) if $\hat{v}\hat{w} \in S$, then $\phi\hat{v}\phi\hat{w} \in T$ (simplicial map);

(2) if $\hat{v} < \hat{w}$ in S, then $\phi\hat{v} < \phi\hat{w}$ in T;

(3) for each $\hat{v} \in S$, $f_{\hat{v}} = f_{\phi\hat{v}}$.

Functional expressions E and E' shall be considered *equivalent* if $T(E)$ and $T(E')$ are equivalent in the sense defined. Incidentally, it is always possible to represent a Boolean tree by a planar graph such that if $\hat{v} < \hat{w}$ then $\hat{v}$ is to the left of $\hat{w}$. . . .

2. Production of expressions for a given function

How does one find appropriate expressions for a given function f, short of some exhaustive search procedure? An answer to this question is implicit in the projection operators and projective words introduced in this section. The injection operators and injective words are varieties of inverses of these, and are used in certain proofs and constructions.

• *2.1 The projection operator* $\Pi^{y}_{\beta y_1 \cdots y_r}$

Let β be a function of r variables $y_1, \cdots, y_r$ belonging to $\mathfrak{B}$. Let the following two matrices represent the vertices of Q^r, the first mapped on 1, the second mapped on 0 by β, each column being associated with exactly one of the variables $y_1, \cdots, y_r$; let $s+t=2^n$:

$$\beta^{-1}(1) = [\beta] = \begin{array}{c} y_1 \cdots y_r \\ \left[\begin{array}{ccc} b_{11} & \cdots & b_{1r} \\ \cdots & \cdots & \cdots \\ b_{s1} & \cdots & b_{sr} \end{array}\right] \end{array};$$

$$\beta^{-1}(0) = [\beta^*] = \begin{array}{c} y_1 \cdots y_r \\ \left[\begin{array}{ccc} b^*_{11} & \cdots & b^*_{1r} \\ \cdots & \cdots & \cdots \\ b^*_{t1} & \cdots & b^*_{tr} \end{array}\right] \end{array}.$$

Let C be a set of cardinality s, of cubes of the n-cube Q^n. Let C be represented by an $s \times n$ matrix $[C]$ shown below. Each row corresponds to a cube of C. To each column is attached a variable. The columns labelled $y_1, \cdots, y_r$ appear, for convenience, as the first r columns, and those labelled $z_1, \cdots, z_q$ as the last q columns, but the definition of the action of the projection operator on $[C]$ does not in any way depend upon the order in which the columns appear.

$$[C] = \begin{array}{c} y_1 \cdots y_r \; z_1 \cdots z_q \\ \left[\begin{array}{ccc|ccc} c_{11} & \cdots & c_{1r} & d_{11} & \cdots & d_{1q} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{s1} & \cdots & c_{sr} & d_{s1} & \cdots & d_{sq} \end{array}\right] \end{array} \quad \begin{array}{l} q+r=n \\ c_{ij}, d_{kl}=0, 1 \text{ or } x \end{array}$$

The definition of the projection operator $\Pi^{y}_{\beta y_1 \cdots y_r}$ will be given in stages. The subscripts $y_1, \cdots, y_r$ are termed *input labels* and the superscript y, the *output label*. It is assumed that $y=\beta(y_1, \cdots, y_r)$. If 1), each of the columns labelled y_i, $1 \leqslant i \leqslant r$, of $[C]$ equals the corresponding

column of some row permutation of $[\beta]$ and 2), $d_{1k}=d_{jk}$ $1\leqslant k\leqslant q$, $1<j\leqslant s$, then

$$[\Pi^{y}_{\beta y_1\cdots y_r}C]=\overset{y\ z_1\quad z_q}{[1, d_{11}\cdots d_{1q}]}.$$

That is, the cube $\Pi^{y}_{\beta y_1\cdots y_r}C$ is represented by the $1\times(q+1)$ matrix $[\Pi^{y}_{\beta y_1\cdots y_q}C]$, with indicated column labels. If both conditions are not met and $s\neq t$ then $\Pi^{y}_{\beta y_1\cdots y_r}C$ is not defined.

Let E be a set of cardinality t of cubes of Q^n. Let E be represented by the $t\times n$ matrix,

$$[E]=\begin{matrix} \\ \\ \\ \end{matrix}\overset{y_1\ \cdots\ y_r\ z_1\ \cdots\ z_q}{\begin{bmatrix} e_{11}\ \cdots\ e_{1r}\ f_{11}\ \cdots\ f_{1q} \\ \cdots\cdots\quad \cdots\cdots \\ e_{t1}\ \cdots\ e_{tr}\ f_{t1}\ \cdots\ f_{tq}\end{bmatrix}}.$$

If 1°), each column labelled y_i of $[E]$ equals the corresponding column of some row permutation of $[\beta^*]$, and 2°), $f_{1k}=f_{jk}$ for $1\leqslant k\leqslant q$ and $1<j\leqslant t$, then

$$[\Pi^{y}_{\beta y_1\cdots y_r}E]=\overset{y\ z_1\quad z_q}{[0, f_{11}\cdots f_{1q}]}.$$

That is, the cube $\Pi^{y}_{\beta y_1\cdots y_r}E$ in Q^{q+1} is represented by the $1\times(q+1)$ matrix $[\Pi^{y}_{\beta y_1\cdots y_r}E]$ with column labels indicated. If both conditions are not met and $s\neq t$, then $[\Pi^{y}_{\beta y_1\cdots y_r}E]$ is not defined. If $s=t$ and neither 1) and 2) nor 1°) and 2°) are met, then $\Pi^{y}_{\beta y_1\cdots y_r}E$ is not defined. Finally, let G be a set of m cubes of Q^n represented by an $m\times n$ matrix $[G]$ with column labels $y_1,\cdots,y_r$, $z_1,\cdots,z_q$. For m less than s and t, $\Pi^{y}_{\beta y_1\cdots y_r}G$ is not defined. For m greater than s or t, $\Pi^{y}_{\beta y_1\cdots y_r}G$ is the set of images of $\Pi^{y}_{\beta y_1\cdots y_r}$ acting on all subsets, of cardinality s or t, of G.

Let Π_β denote $\Pi^{y}_{\beta y_1\cdots y_r}$. Let c and c' be cubes of $\Pi_\beta G$. Let $\Pi_\beta^{-1}(c, G)$ denote the inverse image of c under Π_β. Then

$$c\cap c'=\phi\Rightarrow\Pi_\beta^{-1}(c, G)\cap\Pi_\beta^{-1}(c', G)=\phi.$$

Note also that for a Π-operation to act nontrivially on a set of cubes, it is necessary that no column labelled with a y shall contain an x. (A more general definition given in Ref. [5] will allow such a circumstance, however.)

Consider the following example: let

$$[C]=\begin{matrix} \\ 1\\ 2\\ 3\\ 4\end{matrix}\overset{a\ b\ c\ d}{\begin{bmatrix} 1&0&1&x\\ 0&1&1&x\\ 1&1&x&1\\ 0&0&x&1\end{bmatrix}}\qquad [\beta]=\begin{bmatrix}0&1\\1&0\end{bmatrix},\qquad [\beta^*]=\begin{bmatrix}1&1\\0&0\end{bmatrix}.$$

Then

$$[\Pi_{\beta a\bar{b}}\ C]=\begin{matrix} \\ 1\\ 2\end{matrix}\overset{e\ c\ d}{\begin{bmatrix}1&1&x\\0&x&1\end{bmatrix}}.$$

We shall say that a projection operator Π_β *acts perfectly* on a set C of cubes if each cube of C is contained in the inverse of some element in the image $\Pi_\beta C$.

• 2.2 Projective words

Under what conditions is the product of Π-operators defined? A necessary and sufficient condition that the product of Π-operators $\Pi_\alpha=\Pi^{y}_{\alpha y_1\cdots y_p}$ and $\Pi_\beta=\Pi^{z}_{\beta z_1\cdots z_q}$ be defined, in either order, is that the sets $\{y_1,\cdots,y_p\}$ and $\{z_1,\cdots,z_q\}$ of input labels be disjoint. For suppose, with no loss of generality, that $y_1=z_1$. Suppose, again without loss of generality, that Π_α acts perfectly on a set C of cubes of Q^n. Then $[\Pi_\alpha C]$ has no column labelled y_1, for each column of $[C]$ has a label different from that of each other column. Therefore Π_β is not defined in $[\Pi_\alpha C]$. Consequently, if the sets of labels of Π_α and Π_β are not disjoint, then the product is not defined. The converse is clear. If, further, the sets $\{y_1,\cdots,y_p, y\}$ and $\{z_1,\cdots,z_q, z\}$ of labels, input and output, of Π_α and Π_β are disjoint, $\Pi_\alpha\Pi_\beta=\Pi_\beta\Pi_\alpha$. A product of projection operators is called a *projective word.*

• 2.3 The injection operator

Let c be a single cube of Q^p, represented by the following $1\times p$ matrix,

$$[c]=\overset{z_1\cdots z_p}{[c_1\cdots c_p]},$$

with column labels $z_1,\cdots,z_p$. Let β be a function with $\beta^{-1}(1)=[\beta]$ and $\beta^{-1}(0)=[\beta^*]$ as exhibited in (2), with column labels $y_1,\cdots,y_r$. The *injection operator* $\Pi^{\beta y_1\cdots y_r}_{z_i}$ acting on c will be defined; $y_1,\cdots,y_r$ are termed the *input labels* and z_i the *output label,* with $z_i=\beta(y_1,\cdots,y_r)$:

$$\Pi^{\beta y_1\cdots y_r}_{z_i}\overset{z_1\ \cdots\ z_p}{[c_1\ \cdots\ c_p]}=$$

$$\begin{cases}\begin{matrix} \\ 1\\ \\ s\end{matrix}\overset{z_1\ \cdots\ z_{i-1}\ y_1\ \cdots\ y_r\ z_{i+1}\ \cdots\ z_p}{\begin{bmatrix} c_1\cdots c_{i-1}\ b_{11}\cdots b_{1r}\ c_{i+1}\cdots c_p\\ \cdots\cdots\quad\cdots\cdots\quad\cdots\cdots\\ c_1\cdots c_{i-1}\ b_{s1}\cdots b_{sr}\ c_{i+1}\cdots c_p\end{bmatrix}} & \text{if } c_i=1\\ [c_1\cdots c_{i-1}\ x\ \cdots\ x\ c_{i+1}\cdots c_p] & \text{if } c_i=x\\ \begin{matrix} \\ 1\\ \\ t\end{matrix}\overset{z_1\ \cdots\ z_{i-1}\ y_1\ \cdots\ y_r\ z_{i+1}\ \cdots\ z_p}{\begin{bmatrix} c_1\cdots c_{i-1}\ b^*_{11}\cdots b^*_{1r}\ c_{i+1}\cdots c_p\\ \cdots\cdots\quad\cdots\cdots\quad\cdots\cdots\\ c_1\cdots c_{i-1}\ b^*_{t1}\cdots b^*_{tr}\ c_{i+1}\cdots c_p\end{bmatrix}} & \text{if } c_i=0\end{cases}$$

If the cube c has no coordinate labelled z_i, then $\Pi^{\beta y_1\cdots y_r}_{z_i}[c]$ is not defined. If $c^{(1)},\cdots,c^{(k)}$ is a set of cubes of Q^p then $\Pi^{\beta y_1\cdots y_r}_{z_i}$ acting on this set is defined to be the set $\{\Pi^{\beta y_1\cdots y_r}_{z_i}c^{(1)},\cdots,\Pi^{\beta y_1\cdots y_r}_{z_i}c^{(k)}\}$. The product of $\Pi^{\alpha a_1\cdots a_r}_{a}$ and $\Pi^{\beta b_1\cdots b_s}_{b}$, for $a\neq b$, acting on a set C of cubes, is defined by the equation

$$\Pi^{\alpha a_1\cdots a_r}_{a}\Pi^{\beta b_1\cdots b_s}_{b}[C]=[\Pi^{\alpha a_1\cdots a_r}_{a}[\Pi^{\beta b_1\cdots b_s}_{b}C]].$$

This product is commutative when $a \neq b$. The product of several injection operators is defined by induction. Such a product is termed an *injective word.*

• *2.4 Boolean trees and projective words*

There is a clear one-to-one correspondence between the set of all injective words and the set of all projective words.

A projective word

$$\Pi = \Pi^{y(s+1)}_{\alpha(s)y(s,1)\cdots y(s,r(s))} \cdots\cdots \Pi^{y(2)}_{\alpha(1)y(1,1)\cdots y(1,r(1))}$$

is termed *tree-like* if it satisfies the following two conditions:

(1) Let $A_0 = \{a_1, \cdots, a_n\}$, the set of initial variables,

$$A_i = A_{i-1} \cup \{y(i)\}, \quad 0 < i \leqslant s.$$

In words, A_i is the set of all input and output labels up to the $(i-1)$st term in Π. The first condition is that each subscript $y(i, j)$ belong to the set A_i; this means that each input label shall be either an initial variable or else an output label from an "earlier" projection.

(2) The second condition is that each superscript, excepting one, appear exactly once as a subscript; the exceptional superscript shall not appear as a subscript. It follows that this exceptional superscript is $y(s+1)$.

A tree-like injective word is similarly defined.

Then there is a clear one-to-one correspondence between functional expressions and tree-like words, either projective or injective. Conversely, in view of this correspondence and Section 1.2, with each tree-like word is associated a Boolean tree, this association justifying the term tree-like.

For the remainder of the paper the term *word* will be understood to mean *tree-like word.*

Two projective words are considered equivalent if their corresponding trees are equivalent as defined in Section 1.5; similarly for injective words.

3. Problem of minimization over class of nonsingular Boolean trees

A Boolean tree is *nonsingular* if no initial variable is attached to more than one input node. Otherwise it is *singular.* Let $T_v = T_v(a_1, \cdots, a_r)$ be a Boolean tree whose output function is the r-variable OR function $v(a_1, \cdots, a_r)$. Let $T_1, \cdots, T_r$ be nonsingular Boolean trees.

The disjunction of $T_1, \cdots, T_r$ will now be constructed. For each tree T_i, let $\hat{u}_0(i)$ denote its output node. Paste this node $\hat{u}_0(i)$ on the ith input node of T_v, for each i. The resulting structure is termed the *disjunction* of the trees $T_1, \cdots, T_r$. In general the disjunction of nonsingular Boolean trees is singular. The so-called normal form of Quine corresponds to a special case of this.

With each function α in the bag $\mathfrak{B}$ let there be associated a positive integer $\mu(\alpha)$ called the *cost* of α. The *cost* $\mu(T)$ *of a Boolean tree* T shall be the sum of the costs of the primitive functions attached to T. Let T be a Boolean tree and $\mathrm{K}(T)$ the cubical complex of T. If L is a subcomplex of $\mathrm{K}(T)$, then T is said to *cover* L.

The minimization problem. Given complexes K and L, with K containing L, to find, among the class of disjunctions of nonsingular Boolean trees such that $Q^p \supset K \supset \mathrm{K}(T) \supset L$, one of minimum cost $\mu(T)$.

4. A general covering problem and the extraction algorithm

In Ref. [2], two algorithms are given for the following *covering problem:* Let K be a cubical complex and L° a subset of its vertices. A K-cover of L° is a set C of cubes of K such that each vertex of L° lies in some cube of C. The cost of C is the sum of the codimensions of the cubes of C. These algorithms are termed the *extraction algorithm* and the *local extraction algorithm.*

An abstraction of the covering problem, and of the extraction algorithm for solving this problem, will be given in this section. The problem described in Section 3, as well as that described in the preceding paragraph, is a special case of this general covering problem.

• *4.1 A general covering problem*

Let V and W be sets with V containing W. Let G be a set and Γ a map of G into the set of all subsets of V. If v belongs to the set $\Gamma(g)$, for g in G, then g is said to *cover* v. A *cover* C *of* W is a set of members of G such that each member of W is covered by some member of C.

With each member g of G let there be associated a positive integer $\mu(g)$ called the *cost* of g. The cost of a cover is the sum of the costs of the members of C.

Problem: Find a cover C of W having minimum cost. Clearly for the problem to have a solution it is necessary that $\Gamma(G) \supset W$.

• *4.2 The general extraction algorithm*

The general extraction algorithm proceeds in iterative fashion. The procedure can be carried out over covers of a single element of W, one at a time, as in the local extraction algorithm of Ref. [2]; or it can be carried out over all of G at one shot, as in the extraction algorithm. The algorithm will be given in steps.

1°) Let $W_1 = W$ and $G_1 = \{g \mid \Gamma(g) \cap W \neq \phi\}$.

2°) A member g' of G_1 is said to be less-than (with-respect-to-W_1) a member g of G_1 if

$$\Gamma(g') \cap W_1 \subseteq \Gamma(g) \cap W_1, \qquad \mu(g) \leqslant \mu(g'), \tag{1}$$

with equality not holding in both relations; if equality does hold in both relations let one of these be decreed, by lexicographical order, less-than (with-respect-to-W_1) the other. Let $g' <_1 g$ denote that g' is less-than g (with-respect-to-W_1). The relationship $<_1$ is a partial order. If $g' <_1 g$ this implies that there exists a minimum cover *not* containing g', for g covers at least as much as g' and costs no more. Members of G_1 which are maximal under the partial order $<_1$ are said to be W_1-*elementary.*

The first step in the algorithm is to find the W_1-elementary members of G_1: let $Z_1 = Z(G_1, W_1)$ denote this

set. In the local version of the algorithm it is only necessary to find the subset of these which cover a given w of W_1.

3°) A member e of Z_1 is a *W_1-extremal* if there is an element w of W_1 which is covered by e but by no other member of Z_1. Let $E_1=E(G_1, W_1)$ denote the set of W_1-extremals of $Z(G_1, W_1)$. Clearly every minimum W_1-cover contains E_1.

The pair (G, W) are defined as being reducible if $E(G, W)\neq\phi$; otherwise irreducible. The reducible case will first be treated.

4°) *The case when (G_1, W_1) is reducible.* Let $M_0=E_1$. Let $(G_1, W_1)\supset(G_2, W_2)$ mean that $G_1\supset G_2$ and $W_1\supset W_2$. Having formed pairs $(G_1, W_1)\supset(G_2, W_2)\supset\cdots\supset(G_r, W_r)$, each pair assumed to be reducible with $W_1\neq\phi$, we define, for elements of G_r, the relation less-than (with-respect-to-W_r) exactly as in 2°, with G_r and W_r replacing G_1 and W_1, with one amendment: if equality holds in both relationships

$$\Gamma(g')\cap W_r\subseteq\Gamma(g)\cap W_r\,,\qquad \mu(g)\leqslant\mu(g')\,,\qquad \text{for } g, g'\in G_r \tag{1*}$$

and if, in the pair (G_{r-1}, W_{r-1}), g' was less-than g (with-respect-to-W_{r-1}) then let g' be less-than g (with-respect-to-W_r) also. In other words, in case of a "tie" the "inherited" relationship if it exists shall be maintained. If equality holds in both relationships (1*) and if no less-than relationship exists at an earlier stage then let one be chosen, by lexicographical order, to be less-than the other (with-respect-to-W_r). Let $g'<_r g$ denote that g' is less-than g (with-respect-to-W_r). Clearly $<_r$ is a partial order. Elements of G_r which are maximal under this partial order are said to be *W_r-elementary*. Let $Z_r=Z(G_r, W_r)$ denote the set of W_r-elementary members of G_r.

A member e of Z_r is termed a *W_r-extremal* if there is an element w of W_r covered by e but by no other member of Z_r. Let $E_r=E(G_r, W_r)$ denote the set of W_r-extremals of G_r. By hypothesis $E_r\neq\phi$. Let

$$\left.\begin{aligned} M_r&=M_{r-1}+E_r\\ W_{r+1}&=W_r-\Gamma(E_r)\\ G_{r+1}&=Z_r-\{g\,|\,g\in G_r \text{ and } \Gamma(g)\cap W_{r+1}=\phi\}\,. \end{aligned}\right\} \tag{2}$$

Proposition: Given the pair (G, W) and map $\Gamma: G\to W$ there exists a minimum cover containing M_r.

Proof: For at each stage i, $0<i\leqslant r$, after the elimination of the elements non-maximal under the partial order $<_i$, it was known that every minimum cover of $W_i\subset W$ must contain E_i, so that the choice of E_i was forced. Q.E.D.

Corollary: If $W_{r+1}=\phi$, then M_r is a minimum cover for the original problem. This follows from the above proposition and the fact that M_r *is* a cover.

Let (G, W) be a pair. Consider the sequence

$$(G, W)\supset(G_1, W_1)\supset\cdots\supset(G_r, W_r)\supset(\phi, \phi)\,,\qquad \text{with}$$

$$W_{i+1}=W_i-\Gamma(E_i)\,,$$

$$G_{i+1}=Z(G_i, W_i)-\{g\,|\,\Gamma(g)\cap W_{i+1}=\phi\}\,.$$

If for some i, $W_i\neq\phi$ but $W_{i+1}=\phi$, then (G, W) is said to have a terminating sequence and is said to be *completely reducible*. The above corollary then disposes of the case when (G, W) is completely reducible. The general case will be considered next.

5°) *The case when (G, W) is not completely reducible.* Suppose that the sequence associated with (G, W) does not terminate, that is, for some $r>0$, $W_r\neq\phi$ but $E_r=\phi$, so that

$$W_{r+1}=W_r-\Gamma(\phi)=W_r\,.$$

The extraction algorithm treats this case by solving two problems, each simpler than the original. The procedure is termed *branching*.

Branching: Let z_0 be an element of G_r. The first problem seeks a minimum cover which *contains* z_0; the second, one that does not. Whichever cover has lower cost is clearly a minimum for the original problem. The two problems are defined in parallel columns below, the one on the left being the problem from which a minimum cover containing z_0 is sought. Let

$W^1_{r+1}=W_r-\Gamma(z_0)$	$W^\circ_{r+1}=W_r$
$G^1_{r+1}=Z(G_r, W_r)-\{z\,\|\,\Gamma(z)\cap W^1_{r+1}=\phi\}$	$G^\circ_{r+1}=G_r-\{z_0\}$
$M_r=M_{r-1}+\{z_0\}$	$M^\circ_r=M_{r-1}$

The branching procedure changes the irreducible problem (G_r, W_r) into problems $(G^\circ_{r+1}, W^\circ_{r+1})$ and (G^1_{r+1}, W^1_{r+1}) where the cardinality of both G°_{r+1} and G^1_{r+1} is less than that of G_r.

Proposition: If (G, W) is irreducible, then the cardinality of G is at least 3.

The proof that the extraction algorithm produces a minimum cover proceeds as follows: By the above proposition we start off with a problem (G, W) with the cardinality of G equal to 3. The branching operation reduces this problem to two reducible problems which are solved by steps 2°, 3°, 4° above: the solution for each of these problems which has the smaller cost is a solution for the original problem. Assume next that for all irreducible problems for which the cardinality of $Z(G, W)$ is less than k the algorithm produces a minimum. Let then (G, W) be an irreducible problem with the cardinality of G being k. As noted above for the two problems into which the branching operation transforms it, the cardinalities of the resulting G's is less than k. Hence by induction hypothesis the algorithm produces a minimum for these problems. The minimum for the original problem is the one of these with lower cost.

• *4.3 A special case of 4.1*

This is the problem described in Section 3. Here V is the cubical complex K, W is the subset L° of vertices of L and G is the class of all disjunctions of nonsingular Boolean trees T for which $\mathrm{K}(T)$ is contained in K. Practical augmentation of the algorithm for this problem is discussed in Chapter II, the engineering version.

5. Estimate of efficiency of algorithm

Crude comparisons show that this algorithm is enormously more effective than some systematic exhaustive procedure. We proceed with an estimate of the expected number of nontrivial Π-operations that must be computed for a given problem. Let K be a Boolean complex in the n-cube. Let $\mathfrak{B}$ be the bag of functions. To simplify computational difficulties, assume that each element in $\mathfrak{B}$ is a function of r variables. If c is the cardinality of $\mathfrak{B}$, then the number of possible Π-operations, when the functions of $\mathfrak{B}$ are non-symmetric, is $c \cdot C_r^n$.

Let K be given by a $v \times n$ matrix $[K]$ of 0's and 1's, the list of vertices of K. Let α be represented by an $s \times r$ matrix $[\alpha]$ of 0's and 1's (representing $\alpha^{-1}(1)$). Throughout let $r+q=n$ and $s+t=2^r$. Let the columns of K be labelled $a(1), \cdots, a(r)$, $b(1), \cdots, b(q)$. With no loss of generality let Π_α operate on the first r coordinates of $[K]$, $\Pi_\alpha = \Pi_{\alpha a(1) \cdots a(r)}$. Then let K be divided into blocks of rows, each row of a block having the same b-coordinates. The expected number of rows in a block can be shown to be $w=v2^{-q}$, and hence the expected number of blocks is 2^q. Next, within a given block, the probability that Π_α shall act nontrivially on this block is $C_s^u 2^{-qs}$. Thus the probability that Π_α shall act nontrivially on K is the product P of the number of blocks, 2^q, and $C_s^w 2^{-qs}$: $P = C_s^w 2^{q(1-s)}$. In particular when $n=8$, $r=4$, $v=2^7$ then $s=8$, $q=4$ and $w=8$, and thus this probability is 2^{-28}. For $n=6$, $r=2$, $v=2^5$, $s=2$, $P=2^{-4}$. For $n=5$, $r=2$, $s=2$, $v=16$, $P=2^{-3}$. Thus it appears from these sample figures that really a very small fraction of possible Π-operations are nontrivial. Consequently only an infinitesimal fraction of the total projective words are considered.

The efficiency of the algorithm is bolstered by another factor: by the $<$-operation, which is described in some detail in the engineering version (see also Ref. [2]), many of the partially formed words are almost immediately discarded; see example in Section 3.3, Chapter II.

Chapter II
Engineering Version

E. G. Wagner

1. Nonsingular circuits

• *1.1 Logical elements and nonsingular circuits*

When we design logical circuits we are given some collection or "bag" of logical elements or functions that we may use and some restrictions on the form of the circuits that we may design with these elements. In this paper there will be no restrictions on the contents of the bag of logical elements beyond those imposed by the desire to be able to realize any logical function and the restriction that all the elements have positive cost. Examples of logical elements and various notations, therefore, are given in Table 1. We restrict ourselves to circuits in which each element drives only one other. We define a *nonsingular circuit* as one in which no input variable occurs more than once. The algorithm described in this paper produces for a given function a single-output combinational circuit with the above restriction, consisting of a disjunction ("OR-ing" together) or conjunction ("AND-ing" together) of nonsingular circuits.

Thus in Fig. 1, the circuit in Fig. 1a is nonsingular; Fig. 1b is not, as the input variable x_1 occurs twice. The circuit shown in Fig. 1c is not of the form considered in this paper, since the OR circuit drives two other elements.

A disjunction of nonsingular circuits consists of a number of nonsingular circuits that drive a common OR circuit. (Since the different nonsingular circuits may have some inputs in common, a disjunction of nonsingular circuits is in general not itself nonsingular.)

Similarly, a *conjunction of nonsingular circuits* is a circuit consisting of a number of nonsingular circuits driving a common AND circuit. A very simple example of a disjunction of nonsingular circuits is circuits in normal form. The nonsingular circuits in that case consist of single AND circuits. Examples of nonsingular circuits and disjunctions of nonsingular circuits are shown in Fig. 2.

In speaking of nonsingular circuits or disjunctions of nonsingular circuits we shall use terms such as, "circuit inputs," "circuit output," "wire," or "output element," in accordance with common engineering usage.

• *1.2 Representation of nonsingular circuits by composite functions*

Let T be a nonsingular circuit or disjunction of nonsingular circuits with n input variables $a_1, \cdots, a_n$. We may represent T by a unique composition of functions from the bag of functions associated with our bag of logical elements. Indeed we may associate such a composite function with every wire in the circuit. If the chosen wire is an input wire then the function is exactly the input variable on that wire. Thus in Fig. 3 the function f associated with the upper-left-hand input wire is $a_1 = a_1$. On the other hand, if the chosen wire is not a circuit input, then it is the output wire of some logical element within the circuit; let the function of this logical element be $\alpha(x_1, \cdots, x_r)$ where each of the variables $x_1, \cdots, x_r$ is associated with one of the input wires on the logical element. If the input wire associated with x_1 is connected

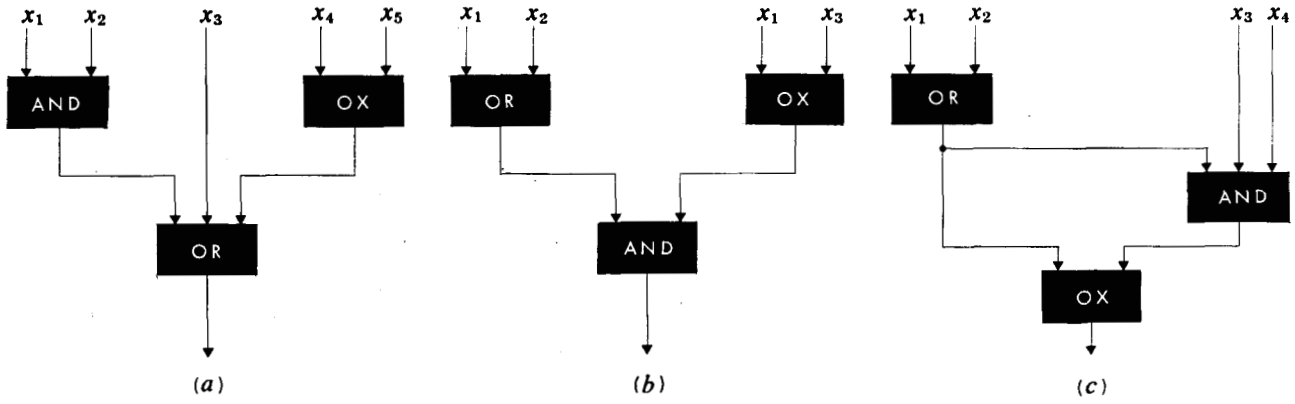


Figure 1 **Examples of combinational circuits. Fig. 1a is of type considered in this paper; 1b and 1c are not.**

a) $\mathrm{v}(\&(x_1, x_2), x_3, \textcircled{v}(x_4, x_5))$
b) $\&(\mathrm{v}(x_1, x_2), \textcircled{v}(x_1, x_3))$
c) $\textcircled{v}(\mathrm{v}(x_1, x_2), \&(\mathrm{v}(x_1, x_2), x_3, x_4))$

Table 1 **Symbols used for circuit elements in Sections 1, 2 and 3.**

Name of Element	*Block Diagram*	*Logical Function*	*On-, Off-Matrices** *On*	*Off*	*Notation for Function*
t-input OR circuit	x_1 $x_2 \cdots x_t$ OR	$x_1 \mathrm{v} x_2 \mathrm{v} \cdots \mathrm{v} x_t$	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$ (here $t=2$)	$[\,0\ 0\,]$	$\mathrm{v}_t(x_1, \cdots, x_t)$
t-input AND circuit	x_1 $x_2 \cdots x_t$ AND	$x_1 \cdot x_2 \cdots x_t$	$[\,1\ 1\,]$ (here $t=2$)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$	$\&_t(x_1, \cdots, x_t)$
2-input EXCLUSIVE-OR circuit	x_1 x_2 OX	$\bar{x}_1 \cdot x_2 \mathrm{v} x_1 \cdot \bar{x}_2$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$	$\textcircled{v}(x_1, x_2)$
inverter circuit	x_1 I	$\bar{x}_1$	$[\,0\,]$	$[\,1\,]$	$\nu(x_1)$

*Defined in Section 2.

with some input variable a_j, then we replace x_i in α by a_j; if the input wire associated with x_i is the output of some other logical element with function $\beta(y_1, \cdots, y_s)$ then we replace x_i in α by $\beta(y_1, \cdots, y_s)$. We do this for all i from 1 to r. In each case where x_i was replaced by a function rather than an input variable, we treat that function in the same manner as we treated α. We continue in this manner until we have a composite function in terms of the input variables $a_1, \cdots, a_n$. Thus in Fig. 3 we get the function $\textcircled{v}(\mathrm{v}(a_1, a_2), a_3)$ at the output wire of the EXCLUSIVE-OR, and the function $\&(\textcircled{v}(\mathrm{v}(a_1, a_2), a_3), \mathrm{v}(a_4))$ at the output of the circuit. The output functions are also given on the other circuits shown so far as examples. It will be noted that when the circuit is nonsingular, then no input variable occurs more than once in the corresponding composite function. We call such composite functions where each input variable occurs at most once *nonsingular composite functions*. There is a one-to-one correspondence between nonsingular circuits and nonsingular composite functions [See Section 1 of the Mathematical Version]. In this chapter we shall identify the two terms; that is, although we shall speak of nonsingular circuits, we shall most often represent them by the corresponding nonsingular composite functions.

We shall in general restrict ourselves to disjunctions of nonsingular circuits, although the algorithms given will work equally well for both the disjunctive and conjunctive cases.

2. Generation of nonsingular circuits

• *2.1 On- and off-matrices of circuits*

Let us consider a circuit consisting of a disjunction of nonsingular circuits, with n input variables. When an input is "on" (pulsed, driven, fed into), we say it has value "1"; when it is "off," it has value "0." Similarly the output has values "1" and "0." We shall describe the operation of the circuit by an $m \times n$ matrix of 1's and 0's called the *on-matrix* of the circuit. Each of the n columns of the on-matrix corresponds to one of the n input variables of the circuit; each column is labeled with its corresponding input variable. Each of the m rows represents one of the m combinations of inputs for which the circuit input is "on," i.e., is "1." We also describe the operation of the circuit by an *off-matrix,* a matrix whose rows represent the combination of inputs for which the circuit output is "off." In particular, the contents of the bag $\mathfrak{B}$ of logical elements can also be considered as nonsingular circuits; and hence on- and off-matrices can be defined for each logical element. The on- and off-matrices of the logical elements given in Fig. 1 are shown in the fourth column. For simplicity of presentation, "don't-care" conditions are not considered here; cf. Chapter I.

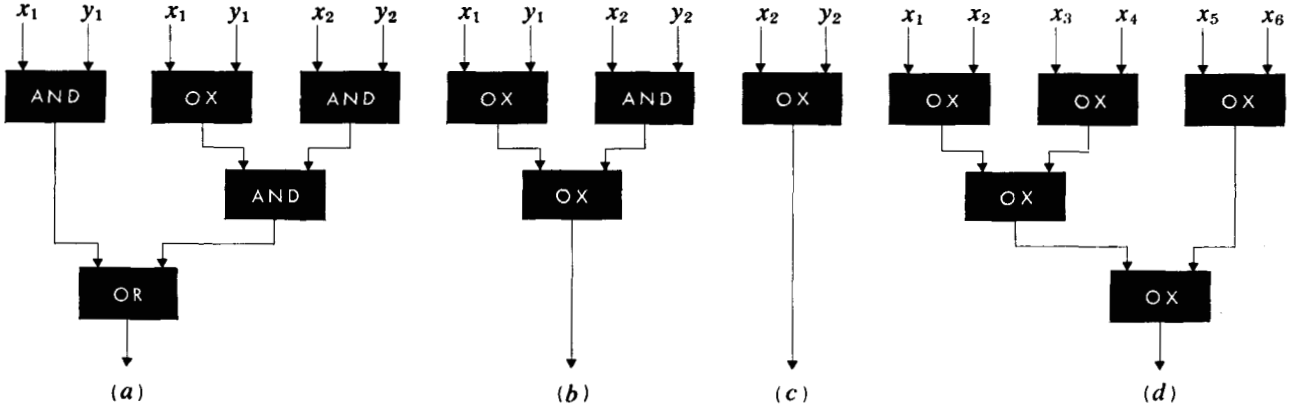
• *2.2 From circuit to on-matrix*

We shall now give a means for developing the on-matrix of a nonsingular circuit by inspecting the nonsingular circuit. The reverse process, the creation of a nonsingular circuit from its on-matrix, will be central to our circuit-designing algorithms. This reverse process will allow us to develop from an $m \times n$ matrix of 1's and 0's a nonsingular circuit, or a circuit which is a disjunction of nonsingular circuits, whose on-matrix will be the given matrix.

We can associate with every logical element appearing in the nonsingular circuit the on-matrix corresponding to that element. As the inputs of an element are the outputs of other elements or circuit inputs, the labels of columns of the on-matrix are the output functions of the element (or circuit inputs) that feed it. Thus in Fig. 3, the AND

Figure 2 **Examples of nonsingular circuits and disjunctions of nonsingular circuits:**

a) Carry output of two-bit adder, $\mathrm{v}(\&(x_1, y_1), \&(\textcircled{v}(x_1, y_1), \&(x_2, y_2)))$
b) Most significant sum bit output of two-bit adder, $\textcircled{v}(\textcircled{v}(x_1, y_1), \&(x_2, y_2)$
c) Least significant sum bit output of two-bit adder, $\textcircled{v}(x_2, y_2)$
d) A parity-bit checker, $\textcircled{v}(\textcircled{v}(\textcircled{v}(x_1, x_2), \textcircled{v}(x_3, x_4)), \textcircled{v}(x_5, x_6))$



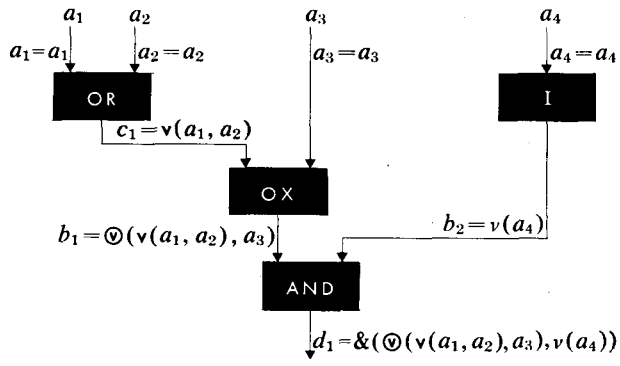


Figure 3 **Nonsingular circuit with output functions shown at each level.**

circuit can be regarded as a nonsingular circuit with input variables $⊛(v(a_1, a_2), a_3)$ and $v(a_4)$. It is then representable by the on-matrix

$$\begin{matrix} d_1 & & b_1 & b_2 \\ [\ 1\ ] & = & [\ 1 & 1\ ] \end{matrix},$$

where $b_1=⊛(v(a_1, a_2), a_3)$ and $b_2=v(a_4)$.

Considering all the elements in Fig. 3 we have:

$$\begin{matrix} c_1 & & a_1 & a_2 \\ [\ 1\ ] & = & \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{matrix} \qquad \begin{matrix} b_1 & & c_1 & a_3 \\ [\ 1\ ] & = & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{matrix}$$

$$\begin{matrix} b_2 & & a_4 \\ [\ 1\ ] & = & [\ 0\ ] \end{matrix} \qquad \begin{matrix} d_1 & & b_1 & b_2 \\ [1] & = & [\ 1 & 1\ ] \end{matrix},$$

where b_1 and b_2 are as above and $c_1=v(a_1, a_2)$.

To get the on-matrix of the total circuit we combine these matrices in accordance with the structure of the circuit. In effect this amounts to a substitution of variables. We replace a single variable e by a function α of variables $d_1, \cdots, d_r$; where $e=1$ we substitute the on-matrix of α; where $e=0$ we substitute the off-matrix of α. Thus, in the circuit shown in Fig. 3, substituting in the matrix of the EXCLUSIVE-OR and removing parentheses, we get:

$$\begin{matrix} & c_1 & a_3 & b_2 \\ \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} 1\right] & = & \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{matrix},$$

thereby expressing b_1 in terms of c and a_3. Similarly, we replace b_2 by a_4, we express c_1 in terms of a_1 and a_2 to obtain

$$\begin{matrix} a_1 & a_2 & a_3 & a_4 \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix},$$

which is the on-matrix for the circuit of Fig. 3.

Let us now consider a circuit that is a disjunction of nonsingular circuits rather than a single nonsingular circuit. Each of the individual nonsingular circuits in the disjunction can be expanded to an on-matrix by the method employed above. For example the carry-bit circuit shown in Fig. 2a consists of two nonsingular circuits which have matrices:

$$A=\begin{matrix} x_1 & y_1 \\ [\ 1 & 1\ ] \end{matrix} \quad \text{and} \quad B=\begin{matrix} x_1 & y_1 & x_2 & y_2 \\ \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{matrix}.$$

We can combine these two matrices into a single four-variable on-matrix for the whole circuit if we expand the first one to include the variables x_2 and y_2. Then combining the two matrices we get as the on-matrix of the whole circuit:

$$\begin{matrix} & x_1 & y_1 & x_2 & y_2 \\ A\left\{\right. & 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 & 0 \\ & 1 & 1 & 0 & 1 \\ & 1 & 1 & 0 & 0 \\ B\left\{\right. & 1 & 0 & 1 & 1 \\ & 0 & 1 & 1 & 1 \end{matrix}.$$

It happens that in this particular example the on-matrices of the two nonsingular circuits were disjoint (that is, they had no rows in common).

• *2.3 From on-matrix to nonsingular circuit*

We shall now consider the reverse process, getting from an on-matrix K to a circuit T with on-matrix K.

We will first show how to develop from an $m \times n$ on-matrix K, an n-input nonsingular circuit T whose on-matrix is made up of rows of K. The on-matrix of T will not necessarily contain all the rows of K. In the next section we will extend this procedure so that it also develops the nonsingular circuits with less than n-inputs whose on-matrices are made up of rows of K.

The procedure is essentially the reverse of that explained in Section 2.2. Above we would replace a column labeled $e=\alpha(a_1, \cdots, a_r)$ by columns labeled $a_1, \cdots, a_r$. We now attempt to replace the columns labeled $a_1, \cdots, a_r$ by a single one labeled $e=\alpha(a_1, \cdots, a_r)$, by "finding occurrences" of the on- and off-matrices of α in the columns labeled $a_1, \cdots, a_r$. If we succeed in this replacement, we then get a new matrix where e replaces $a_1, \cdots, a_r$ and we operate in a similar manner on it. Eventually we will get down to a 1×1 unit matrix whose column label will be a composite function corresponding to some nonsingular circuit with an on-matrix whose rows are contained in K.

We shall now give a semiformal description of the procedure. Let the initial on-matrix K have columns labeled $a_1, \cdots, a_r, b_1, \cdots, b_q$ where $n=r+q$. We now attempt to introduce the r-variable function α from our bag of functions in place of the variables $a_1, \cdots, a_r$, replacing these with a new column $e=\alpha(a_1, \cdots, a_r)$. The first step is to partition the on-matrix K by permuting (rearranging)

the rows so that K is broken into blocks such that rows i and j are in the same block if and only if $b_{ik}=b_{jk}$ for $k=1,\cdots,q$ where b_{ik} and b_{jk} are the values of the b_k component of rows i and j respectively. We call this operation *partitioning* K *on* $a_1,\cdots,a_r$. This operation corresponds to the removal of the parentheses in Section 2.2.

For example, given the following matrix, we partition it on a_1, a_2.

$$\begin{array}{c} \begin{matrix} a_1 & a_2 & b_1 & b_2 & b_3 \end{matrix} \\ \left[\begin{array}{ccccc} 1&1&1&1&1\\ 1&1&1&0&0\\ 1&0&1&1&1\\ 1&0&0&0&1\\ 0&1&1&1&1\\ 0&1&0&0&1\\ 0&0&1&0&0 \end{array}\right] \end{array} = \begin{array}{c} \begin{matrix} a_1 & a_2 & b_1 & b_2 & b_3 \end{matrix} \\ \left[\begin{array}{ccccc} 1&1&1&1&1\\ 1&0&1&1&1\\ 0&1&1&1&1\\ \hline 1&0&0&0&1\\ \hline 0&1&0&0&1\\ \hline 1&1&1&0&0\\ 0&0&1&0&0 \end{array}\right] \end{array} \begin{matrix} \\ \\ \\ \text{Block 1} \\ \\ \text{Block 2} \\ \\ \text{Block 3} \end{matrix}$$

Let us assume that we have m blocks $B_1,\cdots,B_k,\cdots,B_m$. Let $b_{k1},\cdots,b_{kq}$ denote the values of the variables $b_1,\cdots,b_q$ in block B_k. For example in Block 2 above, $b_{21}=0, b_{22}=0, b_{23}=1$. From the partitioned matrix we attempt to form a new matrix with columns e, $b_1,\cdots,b_q$ where $e=\alpha(a_1,\cdots,a_r)$ and α is an r-variable function from our bag of functions. If the kth block of the partitioned matrix contains a submatrix in the $a_1,\cdots,a_r$ columns which contains all the rows of the on-matrix of $\alpha(a_1,\cdots,a_r)$ then $1, b_{k1},\cdots,b_{kq}$ is a row in the new matrix. For example, Block 1 above contains the on-matrix $\left[\begin{array}{cc}0&1\\1&0\end{array}\right]$ of the EXCLUSIVE-OR function, so 1, 1, 1, 1 is a row of the new matrix. If the kth block contains a submatrix which contains the off-matrix of $\alpha(a_1,\cdots,a_r)$, then $0, b_{k1},\cdots,b_{kq}$ is a row of the new matrix. For example, Block 3 above contains the off-matrix $\left[\begin{array}{cc}1&1\\0&0\end{array}\right]$ of the EXCLUSIVE-OR function so 0, 1, 0, 0 is a row of the new matrix. If the kth block B_k of the partitioned matrix contains neither the on- nor the off-matrix of $\alpha(a_1,\cdots,a_r)$ then no row is produced from B_k for the new matrix. After we have considered all blocks for the function $\alpha(a_1,\cdots,a_r)$ we have a new matrix which we denote as $K^1=\Pi^e_{\alpha a_1\cdots a_r}K$. For example using the above on-matrix,

$$\Pi^e_{\textcircled{v}a_1a_2}K=\begin{array}{c}\begin{matrix}e&b_1&b_2&b_3\end{matrix}\\\left[\begin{array}{cccc}1&1&1&1\\1&0&0&1\\0&1&0&0\end{array}\right]\end{array}$$

$$\text{on-matrix of } \textcircled{v}(a_1,a_2)=\left[\begin{array}{cc}1&0\\0&1\end{array}\right]$$

$$\text{off-matrix of } \textcircled{v}(a_1,a_2)=\left[\begin{array}{cc}1&1\\0&0\end{array}\right]$$

or if $\alpha(a_1,a_2)=\mathrm{v}(a_1,a_2)$ then

$$\Pi^e_{\mathrm{v}a_1a_2}K=\begin{array}{c}\begin{matrix}e&b_1&b_2&b_3\end{matrix}\\\left[\begin{array}{cccc}1&1&1&1\\0&1&0&0\end{array}\right]\end{array}.$$

If we get a matrix $\Pi^e_{\alpha a_1\cdots a_r}K$ with no rows, that is, the null matrix, this means that there is no nonsingular composite function f containing $\alpha(a_1,\cdots,a_r)$, where the circuit corresponding to f has an on-matrix contained in K.

Given $K^1=\Pi^e_{\alpha a_1\cdots a_r}K$ we may treat this matrix just as we treated K, that is, we partition it on some of its columns $c_1,\cdots,c_s$ and then form some new matrix

$$\Pi^f_{\beta c_1\cdots c_s}K^1=\Pi^f_{\beta c_1\cdots c_s}\Pi^e_{\alpha a_1\cdots a_r}K\,.$$

If this matrix has at least one row we may continue operating on it. If we reach a point where our matrix is reduced to a 1×1 unit matrix, then the nonsingular function that is the column variable of this matrix corresponds to some nonsingular circuit whose on-matrix is a submatrix of K. It can be shown that if we do not allow "double negation," that is, if we do not replace some column variable b by $v(b)$ and then later replace this by $v(v(b))$, we will always reach either a 1×1 unit matrix or a null matrix in a finite number of steps. This amounts to restricting ourselves to circuits in which we do not allow two inverter circuits to be placed in series. With this restriction, a matrix K with n columns can always be reduced to a unit matrix or a null matrix in $3n-1$ steps or less.

We will now give an example where a matrix K is reduced to a unit matrix by an appropriate series of "Π-operations." The example is the reverse of the example in Section 2.2. There we obtained the on-matrix from the circuit; now we obtain a circuit from the matrix. Let:

$$K=\begin{array}{c}\begin{matrix}a_1&a_2&a_3&a_4\end{matrix}\\\left[\begin{array}{cccc}1&0&0&0\\0&1&0&0\\\hline 1&0&0&0\\\hline 0&0&1&0\end{array}\right]\end{array}$$

partitioning on a_1a_2 as indicated.

Choosing $\alpha(a_1,a_2)=\mathrm{v}(a_1,a_2)$ we get:

$$\Pi^c_{\mathrm{v}a_1a_2}K=\begin{array}{c}\begin{matrix}c&a_3&a_4\end{matrix}\\\left[\begin{array}{ccc}1&0&0\\0&1&0\end{array}\right]\end{array}=K^1 \qquad c=\mathrm{v}(a_1,a_2)\,.$$

Then continuing,

$$\Pi^{b_1}_{\textcircled{v}ca_3}K^1=\begin{array}{c}\begin{matrix}b_1&a_4\end{matrix}\\\left[\begin{array}{cc}1&0\end{array}\right]\end{array}=K^2, \qquad b_1=\textcircled{v}(\mathrm{v}(a_1,a_2),a_3)\,,$$

then

$$\Pi^{b_2}_{\mathrm{v}a_4}K^2=\begin{array}{c}\begin{matrix}b_1&b_2\end{matrix}\\\left[\begin{array}{cc}1&1\end{array}\right]\end{array}=K^3 \qquad b_2=v(a_4)\,,$$

and lastly

$$\Pi^d_{\&b_1b_2}K^3=\begin{array}{c}\begin{matrix}d\end{matrix}\\\left[\begin{array}{c}1\end{array}\right]\end{array} \qquad d=\&\{\textcircled{v}[\mathrm{v}(a_1,a_2),a_3],v(a_4)\}\,.$$

• *2.4 Nonsingular circuits from submatrices*

We shall later find it necessary to be able to develop from an on-matrix K all nonsingular circuits whose on-matrices

are made up of rows from K. The operations described in the preceding section are almost sufficient for doing this. As described, however, they will produce only the n-input nonsingular circuits, where K has n variables. But as the example given in Fig. 2a shows, it is possible for there to be nonsingular circuits, whose on-matrices cover parts of K which have fewer than n inputs. Referring to the example of Fig. 2a we see that although K has four variables, the nonsingular circuit consisting of the AND circuit has only two inputs, namely x_1 and y_1, and its output is consequently independent of x_2 and y_2. This nonsingular circuit can be represented by an on-matrix whose only columns are x_1 and y_1. What we must do, then, is to generate nonsingular circuits not only from K but also from those submatrices of K that are independent of certain column variables of K. Let us denote the submatrix of K that is independent of the r columns $i(1), \cdots, i(r)$ by the symbol $K_{i(1)\cdots i(r)}$. We may form $K_{i(1)\cdots i(r)}$ by means of the following rules. Given K we form K_i as follows: let $q_1, \cdots, q_i, \cdots, q_n$ be a row of K; then $q_1, \cdots q_{i-1}, q_{i+1}, \cdots, q_n$ is a row of K_i if and only if there is another row $q'_1, \cdots, q'_i, \cdots, q'_n$ in K such that $q_j = q'_j$ for $j \neq i$, and if $q_i = 1$ then $q'_i = 0$, and vice versa. We then form K_{ij} from K_i by the same means but by operating on K_i rather than K, and so forth until we reach $K_{i(1)\cdots i(r)}$. For example, if

$$K = \begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array},$$

$$\text{then } K_2 = \begin{array}{c} \begin{matrix} a_1 & a_3 & a_4 & a_5 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array}, \quad K_{2,4} = \begin{array}{c} \begin{matrix} a_1 & a_3 & a_5 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \end{array}.$$

If we form all possible submatrices $K_{i(1)\cdots i(r)}$ of K by these rules and then operate on them by the methods of the preceding section, trying all partitionings and all the functions from our bag of functions, then we will have produced all the nonsingular circuits that have on-matrices consisting of rows from K (with the restriction against "double inversion"). We shall denote the set of nonsingular circuits without double inversions whose on-matrices are made up of rows of K as G (or $G(K)$), and call $G(K)$ the set of *nonsingular circuits generated from* K. Although trying all partitions and all functions from the bag of functions appears to be an extremely exhaustive procedure, it will be found that many possibilities lead to null matrices and only a fairly small percentage of all nonsingular circuits of n and less inputs will actually be produced. Furthermore, as will be shown in Section 3, in certain cases it is not necessary to generate all of G.

• 2.5 Scopes

It is convenient to use a special notation to indicate which rows of the on-matrix of a circuit are contained in or *covered by* the on-matrix of one of nonsingular circuits that make up its realization as a disjunction of nonsingular circuits. List the m rows in the on-matrix of the circuit in descending numerical order viewing them as binary numbers. Then we can say that a given nonsingular circuit T generated from K covers the ith, jth, $\cdots$, and kth rows of an on-matrix K where $i, j, \cdots, k$ are numbers between 0 and $m+1$ when the on-matrix of T consists of the $i, j, \cdots, k$th rows of K. For example in the second example given in Section 2.2, the first nonsingular circuit covers the 1st, 2nd, 3rd, and 4th rows; the second nonsingular circuit covers the 5th and 6th rows. A convenient way to indicate this is to associate with each nonsingular circuit T an m-component vector $S(T)$ called its *scope*, whose ith bit is 1 if and only if the on-matrix of this nonsingular circuit contains (when fully expanded) the ith row of the on-matrix of the complete circuit and is "0" otherwise. Thus in the above example of the scope of the nonsingular circuit, "$\&(x_1, y_1)$" is 111100, and the scope of "$\&(\textcircled{v}(x_1, y_1), \&(x_2, y_2))$" is 000011.

Just as we speak of the scopes of nonsingular circuits so can we speak of the scopes of the matrix K and of any matrix

$$\Pi^e_{\alpha a \cdots b} \cdots \Pi^d_{\beta f \cdots g} K_{i(1)\cdots i(r)} = K'$$

developed from K. The scope of K is, of course, an m-component vector with all components 1. To find the scope of

$$K' = \Pi^e_{\alpha a \cdots b} \cdots \Pi^d_{\beta f \cdots g} K_{i(1)\cdots i(r)}$$

we expand this matrix back to an on-matrix consisting of rows from K by the methods given in Section 2.2 and develop its scope just as we did for nonsingular circuits; that is, if the on-matrix developed from K' contains the ith row of K, then the ith component of the scope of K' is "1" and otherwise it is "0." The on-matrix of K' contains the rows of any nonsingular circuit that can be generated from K' by further operations. Thus the scope of any nonsingular circuit generated from K' has a "1" in the ith coordinate only if the scope of K' has a "1" in the ith coordinate.

If the scope $S(T)$ of a nonsingular circuit T generated from K has a 1 in the ith coordinate, then T *covers the ith row of* K. For a set of nonsingular circuits $T_1, \cdots, T_r$ generated from K, we define a *joint scope* $S(T_1, \cdots, T_r)$ to be a vector whose ith component is 1 if at least one of $S(T_1), \cdots, S(T_r)$ has a 1 in the ith coordinate. If $S(T_1, \cdots, T_r) = (1, 1, \cdots, 1)$, we say that $T_1, \cdots, T_r$ form a *cover of* K. Clearly if $T_1, \cdots, T_r$ form a cover of K, then the disjunction of $T_1, \cdots, T_r$ is a circuit with on-matrix K. If our bag is such that we can realize any function with it, then $G(K)$ is always a cover of K.

3. The extraction algorithm

• 3.1 The definition of cost

To each element in our bag of logical elements we attach some positive integer called its *cost*. In practice this cost would be arrived at by considering such matters as cost of construction, maintenance, power consumption, et cetera. The cost of a nonsingular circuit is defined as the sum of the costs of all the elements in it, plus, if desired, some additional amount to "pay" the cost of "OR-ing" it to the other nonsingular circuits in the disjunction.

The *cost* of a complete circuit (disjunction of nonsingular circuits) is then defined as the sum of the costs of the nonsingular circuits which form it. Equivalently the cost of a cover of an on-matrix K is the sum of the cost of the nonsingular circuits that form it.

In the examples in this section we assume that our bag consists of all the two-input logical elements, and that all of them are of cost 1. This special case has the property that any nonsingular circuit with n inputs can be shown to have a cost of $n-1$. This property will allow us to give some compact examples, for reasons which will appear later. In particular, we can say that the cost of any nonsingular circuit generated from the $p\times(n-r)$ submatrix $K_{i(1)\cdots i(r)}$ of K has cost $n-r-1$.

• 3.2 The minimization problem

Given an on-matrix K, develop a circuit T of minimal cost consisting of a disjunction of nonsingular circuits, such that T has on-matrix K.

We have already given part of the solution of this problem. Namely, we have given a method for getting the set G of all the nonsingular circuits that cover parts of a given on-matrix K. But as examples show, the number of nonsingular circuits developed is far in excess of the number needed to cover K. What is needed, then, is some way to extract a cover of minimal cost from the set G of nonsingular circuits generated from K. The *extraction algorithm* does exactly this. Proof that the algorithm does produce a minimum follows as a special case of the proof of the validity of the general extraction algorithm given in Section 4 of the Mathematical Version.

• 3.3 The algorithm

Three operations are employed in the extraction algorithm: the less-than operation, the determination of extreme nonsingular circuits, and the branching operation. We shall explain the first two of these operations separately and then combine them into an algorithm, which will contain the third operation.

The less-than operation is an operation on covers of an on-matrix K that removes unnecessary "removable" members of the cover. We first define the *less-than relationship*, a relationship between members of a cover.

Let T and T' be two nonsingular circuits in any cover Q of K made up of members from the set G of nonsingular circuits generated from K. Let $S(T)$ and $S(T')$ denote their respective scopes, $\mu(T)$ and $\mu(T')$ their respective costs. We shall write $S(T)\subseteq S(T')$ if the ith coordinate of $S(T)$ is "1," then so is the ith coordinate of $S(T')$, and we say that the scope of T is covered by the scope of T'. Referring back to the meaning of scope we see that this means that circuit T is on only if circuit T' is on. We now say that T is *less-than* T', or T' is *greater-than* T, if and only if $S(T)\subseteq S(T')$ and $\mu(T)\geqslant\mu(T')$; in such a case we will write $T<T'$. This means that $T<T'$ if and only if T is on only when T' is on and T costs at least as much as T'. Clearly then, since T does nothing that is not done by T' at less or equal cost, we can always use T' in place of T in our solution. Thus if we remove T from the cover Q getting a new cover Q', then Q' is less costly than Q. The *less-than operation* on a cover Q then consists of comparing the members of Q by the less-than relationship, removing those that are less than others as we go along. That is, if we find that $T<T'$, then we remove T from Q and do not compare it with any other members remaining in Q. If we perform the less-than operation on $G(K)$ arriving at a cover Z, it can be shown that Z will contain at least one minimal cover of K.

A systematic way to carry out the less-than operation will be given in the algorithm. In the special case we are using for our examples, we actually employ the less-than operation during the generation of G, thus keeping G as small as possible at all times. Namely, as mentioned before, the cost of an n-input nonsingular circuit in this special system is $n-1$, and so any nonsingular circuit generated from an n-variable on-matrix will have a cost of $n-1$. Now if we consider the scope $S(K')$ of a matrix

$$K'=\Pi^{e}_{\alpha ab}\cdots\Pi^{d}_{\beta fg}K_{i(1)\cdots i(r)}$$

we know that it covers the scope of any of the nonsingular circuits that can be formed from it. It follows then that any nonsingular circuit generated from K' will have cost $n-r-1$ and scope covered by $S(K')$ and if we have already formed a nonsingular circuit T such that $S(K')\subseteq S(T)$ and $n-r-1\geq\mu(T)$, then T will be greater than any of the nonsingular circuits that we could possibly develop from K' and so there is no need to generate them at all. In such a case we write $K'<T$. (If on the other hand $S(K')\supset S(T)$ or $n-r-1<(T)$, we cannot conclude anything.) This application of the less-than rule during the generation of G will usually save considerable time and effort. We note that, since the cost is directly related to the number of variables, it pays to generate the nonsingular circuits from the submatrices $K_{i(1)\cdots i(r)}$ of K with the smallest number of inputs (largest r) first, as these have the lowest cost.

The determination of extreme nonsingular circuits: If we wish to find the minimal cover contained in some given cover Q of K we can often immediately find certain elements of Q called *extreme nonsingular circuits* that must be in any minimum contained in Q. Specifically let T be a nonsingular circuit in a cover Q of K. Suppose that the ith bit of $S(T)$ is "1" and that the ith bit of $S(T'')$ is 0 for all T'' other than T in Q. Then T is said to be *extreme in* Q. Now clearly since there must be some nonsingular circuit in the disjunction that is "on" for the ith combination of input signals if the on-matrix of the cir-

cuit is to be K, it follows that if T is extreme then T must be in the solution.

We shall now show how the less-than operation and the determination of extreme nonsingular circuits can be employed to find minimal circuits.

We take the set G of nonsingular circuits generated from an on-matrix K and perform the less-than operation on it (this may have been done during the generation of G). This gives us a new cover Z' of K. We then determine the extreme nonsingular circuits of Z'. Say that $T_1, \cdots, T_p$ are extreme, and let their joint scope be $S(T_1, \cdots, T_p)$. Since the input combinations denoted by $S(T_1, \cdots, T_p)$ are covered by $T_1, \cdots, T_p$ we need no longer consider these combinations in our calculations. We therefore eliminate these components from our scopes (rather than actually remove them it is often more convenient to consider these coordinates to be 0 in all remaining scopes), and continue then with a reduced problem. The removal of $S(T_1, \cdots, T_p)$, as the following example (Table 2) indicates, may make some nonsingular circuits cover nothing and thereby unnecessary, or may make one circuit T less-than another T' where before neither one was less-than the other. T_e is extreme (1st coordinate) so the list of scopes reduces as indicated in Table 3. Now $S(T') \subset S(T)$ and $\mu(T) = \mu(T')$ so that $T' < T$. Furthermore T'' covers nothing (not already covered). Thus we may eliminate both T' and T''. Thus the list reduces to T and so T is extreme. The final solution must be the disjunction of T_e and T. As the example indicates, the less-than operation and the identification of extreme nonsingular circuits used repeatedly one after the other will sometimes suffice to allow us to extract a solution from G. It may happen, however, that we will eventually reach a point where no remaining element of G is less-than any other and there are no extreme nonsingular circuits. We then must employ the *branching* operation.

Suppose that the less-than operation and the identification of extreme nonsingular circuits will carry us no further, and let T be one of the remaining nonsingular circuits in G. Now clearly either T is in a final minimal solution or it is not, so what we can do is to try both possibilities and choose the better. That is, we arbitrarily pick some member T from what remains of G and first treat the problem as if T were an extreme nonsingular circuit and get a solution; we then again find a solution treating T as if it were less-than some other nonsingular circuit remaining in G (i.e., we remove T from G). One

Table 2 **Hypothetical example showing configuration of scopes and costs.**

Circuits	*Scopes*	*Costs*	
T_e	1 0 1 1 1 0 1 1 0	5	
T	0 1 1 1 0 1 0 0 1	3	$T \not< T'$
T'	0 1 0 0 0 1 1 1 1	3	$T' \not< T$
T''	0 0 1 1 1 0 0 0 0	2	

Table 3 **Reduction of Table 2 after first step of algorithm.**

Circuits	*Scopes*	*Costs*	
T	0 1 0 0 0 1 0 0 1	3	$T' < T$
T'	0 1 0 0 0 1 0 0 1	3	
T''	0 0 0 0 0 0 0 0 0	2	

of these two solutions will certainly be a minimal solution for the original problem. Of course, we may have to branch on more than one nonsingular circuit in order to get a solution, but as long as we work to the end, a minimum will be achieved.

The working out of all the branching combinations involves far fewer steps than an exhaustive examination of solutions, as is indicated by the example at the end of the paper. (Cf. also Section 5 of Mathematical Version.)

The next section gives a more complete outline of the algorithm in flow-chart form.

4. Block diagram of extraction algorithm for a special case, plus an example

• *4.1 Block diagram of extraction algorithm for a special case*

In this section we present, in block form, the complete algorithm for the special case when the bag of logical elements contains all the two-input logical elements and all elements have the same cost, as is the case for core circuits.

In the block diagrams of Fig. 4:

1) The ten nontrivial 2-input logical functions in our bag $\mathfrak{B}$ are denoted $\alpha_1, \cdots, \alpha_h, \cdots, \alpha_{10}$ (see Section 4.2 for a listing of these functions).

2) Let r denote the number of columns (or column variables) of any matrix on which we are performing a Π-operation. Let the columns be labeled (numbered) from 1 to r from left to right; this allows us to describe the performance of the Π-operations in a systematic manner.

• *4.2 Example*

In the following example of the application of the above algorithm we have as our bag of functions:

$\alpha_1 = \mathrm{v}(x_1, x_2) = x_1 \mathrm{v} x_2$

$\alpha_2 = \textcircled{v}(x_1, x_2) = \bar{x}_1 x_2 \mathrm{v} x_1 \bar{x}_2$

$\alpha_3 = \&(x_1, x_2) = x_1 x_2$

$\alpha_4 = \theta(x_1, x_2) = \bar{x}_1 x_2$

$\alpha_5 = \psi(x_1, x_2) = x_1 \bar{x}_2$

and their negations designated $\bar{\mathrm{v}}$, $\bar{\textcircled{v}}$, $\bar{\&}$, $\tilde{\theta}$, $\tilde{\psi}$, respectively. Except for the trivial functions, which need not

be considered, this is the set of all functions of two variables. We will not show the complete working out of the problem but only the data at various points in the algorithm. The problem is a randomly chosen one made up from the first four columns and first eight rows of [4] considering odd digits as '1's and even digits as '0's.

$$K=\begin{matrix}a&b&c&d\end{matrix}\begin{bmatrix}1&1&1&1\\1&1&1&0\\1&1&0&1\\1&1&0&0\\1&0&0&1\\1&0&0&0\\0&1&1&1\\0&1&0&0\end{bmatrix}$$

$$K_a=\begin{matrix}b&c&d\end{matrix}\begin{bmatrix}1&1&1\\1&0&0\end{bmatrix}$$

$$K_d=\begin{matrix}a&b&c\end{matrix}\begin{bmatrix}1&1&1\\1&1&0\\1&0&0\end{bmatrix}$$

$$K_b=\begin{matrix}a&c&d\end{matrix}\begin{bmatrix}1&0&1\\1&0&0\end{bmatrix}$$

$$K_c=\begin{matrix}a&b&d\end{matrix}\begin{bmatrix}1&1&1\\1&1&0\end{bmatrix}$$

$$K_{bd}=\begin{matrix}a&c\end{matrix}[\,1\;0\,]$$

$$K_{cd}=\begin{matrix}q&v\end{matrix}[\,1\;1\,]$$

After we operate on all of these matrices according to the algorithm[12] we will get the circuits in G indicated in Table 4.

Table 4 **Nonsingular circuits developed from matrix in Section 4.2.**

Nonsingular Function		*Scope*	*Cost*
1	$\&(a, b)$	1 1 1 1 0 0 0 0	1
2	$\psi(a, c)$	0 0 1 1 1 1 0 0	1
3	$\psi(a, \theta(b, c))$	1 1 1 1 1 1 0 0	2
4	$\&(\&(b, c), d)$	1 0 0 0 0 0 1 0	2
5	$\psi(\psi(b, c), d)$	0 0 0 1 0 0 0 1	2
6	$\theta(\circledv(c, d), b)$	1 0 0 1 0 0 1 1	2
7	$\psi(\psi(\mathrm{v}(a, b), c), d)$	0 0 0 1 0 1 0 1	3
8	$\&(\&(\mathrm{v}(a, c), b), d)$	1 0 1 0 0 0 1 0	3
9	$\theta(\circledv(\mathrm{v}(a, c), d), b)$	1 0 1 0 0 0 1 1	3
10	$\psi(\theta(\circledv(a, c), b), d)$	0 1 0 0 0 0 0 1	3
11	$\theta(\circledv(\circledv(a, c), d), b)$	0 1 1 0 0 0 1 1	3
12	$\psi(\theta(\theta(a, c), b), d)$	0 1 0 1 0 0 0 1	3
13	$\theta(\circledv(\theta(a, c), d), b)$	0 1 0 1 0 0 1 1	3
14	$\&(\&(\mathrm{v}(a, d), b), c)$	1 1 0 0 0 0 1 0	3
15	$\theta(\circledv(\mathrm{v}(a, d), c), b)$	1 1 0 0 0 0 1 1	3
16	$\psi(\theta(\theta(a, d), b), c)$	0 0 1 1 0 0 0 1	3
17	$\theta(\circledv(\theta(a, d), c), b)$	0 0 1 1 0 0 1 1	3
18	$\&(\mathrm{v}(\&(c, d), a), b)$	1 1 1 1 0 0 1 0	3

We now perform the LESS-THAN operation of the above list, getting Table 5.

Table 5 **Reduction of Table 4 by means of LESS-THAN operation.**

Circuit Number	*Scope*	*Cost*	
1	1 1 1 1 0 0 0 0	1	
2	0 0 1 1 1 1 0 0	1	
3	1 1 1 1 1 1 0 0	2	
6	1 0 0 1 0 0 1 1	2	
7	0 0 0 1 0 1 0 1	3	
9	1 0 1 0 0 0 1 1	3	List A
11	0 1 1 0 0 0 1 1	3	
13	0 1 0 1 0 0 1 1	3	
15	1 1 0 0 0 0 1 1	3	
17	0 0 1 1 0 0 1 1	3	
18	1 1 1 1 0 0 1 0	3	

None of these circuits is extreme so we branch on Circuit 1. We first try the case where we assume Circuit 1 is in the solution. Treating Circuit 1 as an extremal, we change those scope components that it covers to "0" and then perform the LESS-THAN operation, giving us a new list (Table 6).

Table 6 **Reduction of Table 5, branching with Circuit 1 as an extremal.**

Circuit Number	*Scope*	*Cost*	
2	0 0 0 0 1 1 0 0	1	
6	0 0 0 0 0 0 1 1	2	List B
7	0 0 0 0 0 1 0 1	3	

Now both Circuit 2 and Circuit 6 are extreme, and together with Circuit 1 we see that they cover everything, so we have a cover of K consisting of Circuits 1, 2, and 6 with a cost of 4.

We now try the case where we assume that Circuit 1 is not in the solution, so we delete Circuit 1 from List A. There are still no extremals so we branch on Circuit 2. We first try the case where we assume that Circuit 2 is in the solution; treating Circuit 2 as extreme and performing the LESS-THAN operation we get the list in Table 7.

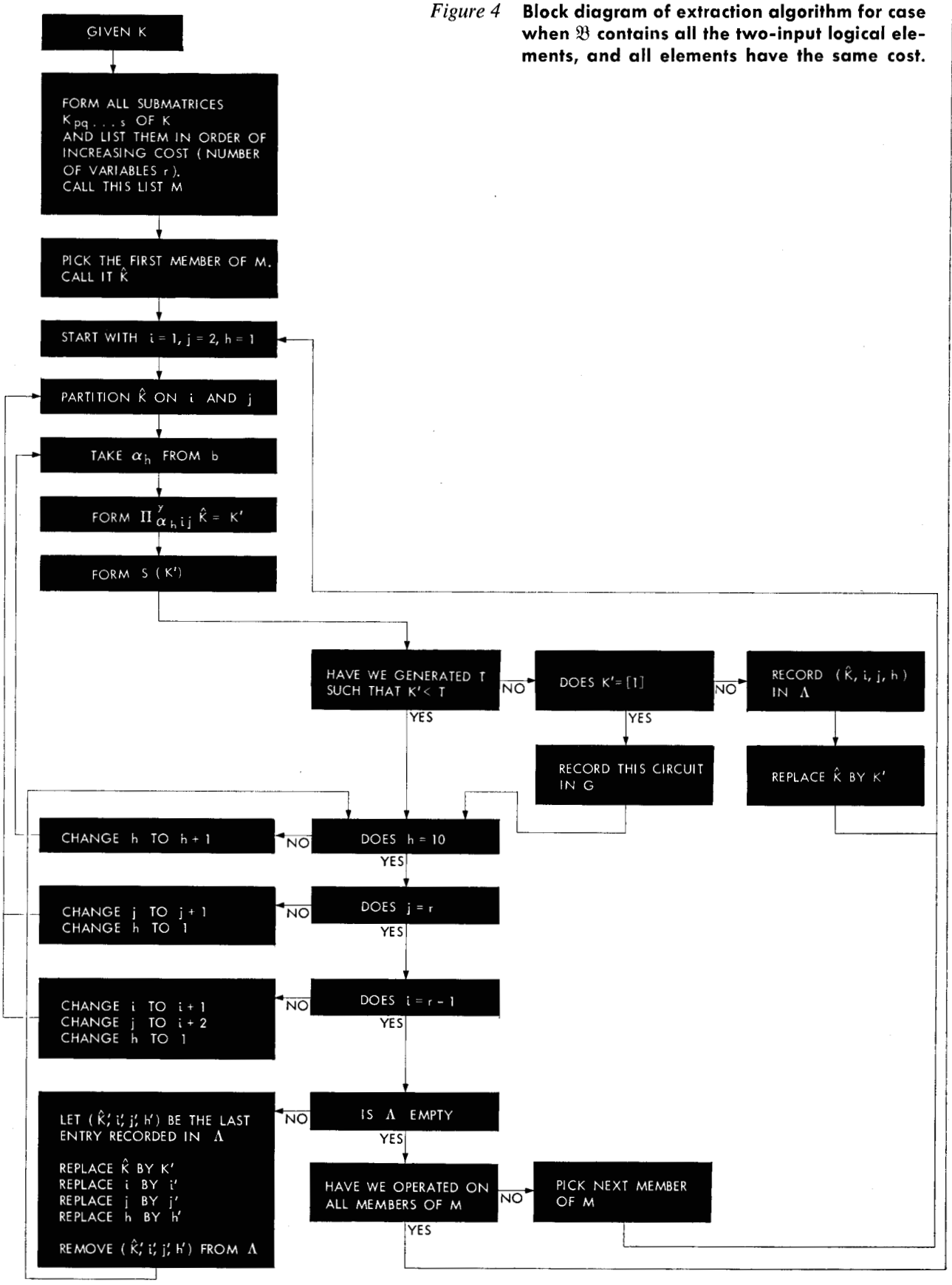
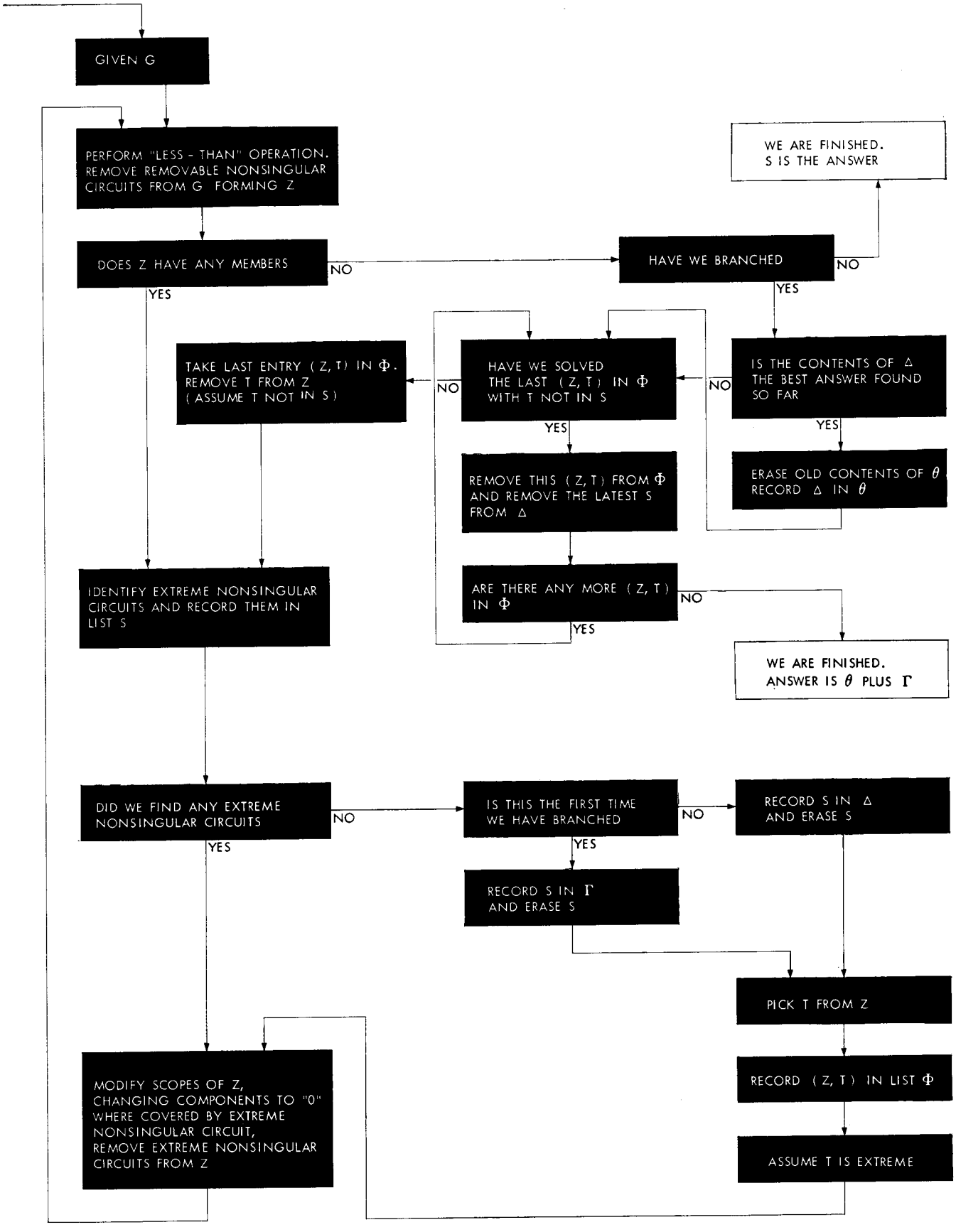


Figure 4 **Block diagram of extraction algorithm for case when $\mathfrak{B}$ contains all the two-input logical elements, and all elements have the same cost.**


GIVEN G
PERFORM "LESS - THAN" OPERATION. REMOVE REMOVABLE NONSINGULAR CIRCUITS FROM G FORMING Z
DOES Z HAVE ANY MEMBERS
NO
YES
HAVE WE BRANCHED
NO
YES
WE ARE FINISHED. S IS THE ANSWER
IS THE CONTENTS OF Δ THE BEST ANSWER FOUND SO FAR
NO
YES
ERASE OLD CONTENTS OF θ RECORD Δ IN θ
HAVE WE SOLVED THE LAST (Z, T) IN Φ WITH T NOT IN S
NO
YES
TAKE LAST ENTRY (Z, T) IN Φ. REMOVE T FROM Z (ASSUME T NOT IN S)
REMOVE THIS (Z, T) FROM Φ AND REMOVE THE LATEST S FROM Δ
ARE THERE ANY MORE (Z, T) IN Φ
NO
YES
WE ARE FINISHED. ANSWER IS θ PLUS Γ
IDENTIFY EXTREME NONSINGULAR CIRCUITS AND RECORD THEM IN LIST S
DID WE FIND ANY EXTREME NONSINGULAR CIRCUITS
NO
YES
IS THIS THE FIRST TIME WE HAVE BRANCHED
NO
YES
RECORD S IN Δ AND ERASE S
RECORD S IN Γ AND ERASE S
PICK T FROM Z
RECORD (Z, T) IN LIST Φ
ASSUME T IS EXTREME
MODIFY SCOPES OF Z, CHANGING COMPONENTS TO "0" WHERE COVERED BY EXTREME NONSINGULAR CIRCUIT, REMOVE EXTREME NONSINGULAR CIRCUITS FROM Z

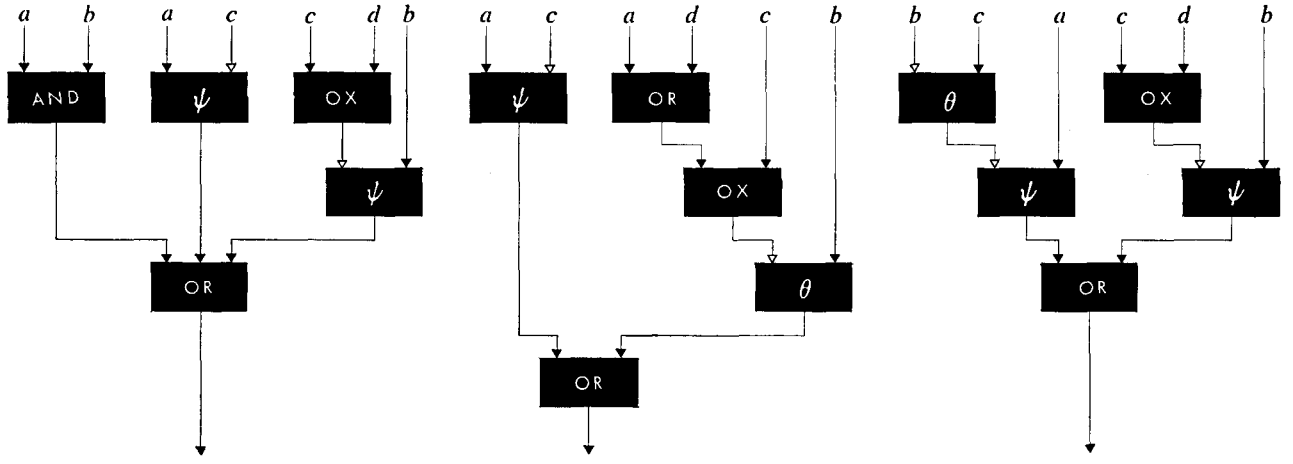


Figure 5 **Three minimal solutions of example in Section 4.2.**

Table 7 **Reduction of Table 5, branching with Circuit 1 deleted.**

Circuit Number	*Scope*	*Cost*	
3	1 1 0 0 0 0 0 0	2	List C
6	1 0 0 0 0 0 1 1	2	
15	1 1 0 0 0 0 1 1	3	

Again we have no extremals so we branch on Circuit 3. Taking the case where Circuit 3 is in the solution, the LESS-THAN operation gives us

Table 8 **Reduction of Table 7 by branching.**

Circuit Number	*Scope*	*Cost*
6	0 0 0 0 0 0 1 1	2

Circuit 6 is clearly extreme and combined with Circuits 2 and 3, which we had already assumed, on this branch, to be in the solution, gives us a cover of K. Thus Circuits 2, 3, and 6 give us a cover of cost = 5.

Now assuming Circuit 3 is not in the solution, we see by inspection of List C that then Circuit 15 is extreme, and we get a solution consisting of Circuits 2 and 15 with cost = 4. Now assuming Circuit 2 is not in the solution, we see by inspection of List A with Circuit 1 removed, that Circuit 3 is extreme, giving, after the LESS-THAN operation, the same as Table 8. So Circuit 6 is again extreme, and we have a cover consisting of Circuits 3 and 6 with a cost of 4.

We have thus found three minimal solutions at cost=4. They are shown in Fig. 5.

References

1. J. Paul Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems. I," *Transactions of American Mathematical Society*, **88,** 301-326 (July, 1958). See also ECP 56-02, Institute for Advanced Study, April, 1956.
2. J. Paul Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems. II," *Proceedings of the International Symposium on the Theory of Switching,* Harvard University, April 2, 1957. To appear.
3. W. V. Quine, "A Way to Simplify Truth Functions," *American Mathematical Monthly,* **62,** 627-631 (1955).
4. The Rand Corporation, *One Million Random Digits with 100,000 Normal Deviates,* The Free Press, Glencoe, Ill. (1955).
5. J. Paul Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems. IV. Singular Boolean Trees." (To appear.)
6. J. Paul Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems. V. The Multiple Output Problem." (To appear.)
7. The Staff of the Harvard Computation Laboratory: *Synthesis of Electronic Computing and Control Circuits,* Harvard University Press, 1951.
8. M. Karnaugh, "The Map Method for Synthesis of Combinational Logic Circuits," *AIEE Transactions,* **72,** 1, 593-598 (1953).
9. E. J. McCluskey, Jr., "Minimization of Boolean Functions," *Bell System Technical Journal,* **35,** 1417-1444 (November, 1956).
10. R. H. Urbano and R. K. Mueller, "A Topological Method for the Determination of the Minimal Forms of a Boolean Function," *IRE Transactions* **EC-5,** 126-132 (September, 1956).
11. D. E. Muller, "Complexity in Electronic Switching Circuits," *IRE Transactions* **EC-5,** No. 1, 15-19 (March, 1956).
12. Dr. Gerard Salton (Harvard University) has brought to our attention the omission from Table 4 of four functions that result from partitioning K on Columns 3 and 4. In particular the function $\theta(\psi(\textcircled{v}(c, d), a), b)$ with scope (11110011) was omitted. Functions 9, 11, 13, 15, 17 and 18 are less-than this new function and should be eliminated from Table 5. The first and third solutions shown in Fig. 5 remain, but the middle one is replaced by the disjunction of Function 2 and the new function.

Received July 17, 1958