J. Paul Roth

Minimization over Boolean Trees*

Abstract: An algorithm is provided for what might be termed the general problem of logical design of circuits with one output and no feedback. Given a set $\mathfrak{B}$ of logical building blocks, each with a positive cost, each with one output, and given a Boolean function f the problem is to prescribe a Boolean tree constructed from the available set of building blocks which realizes f and which has a minimum cost. Actually a more general problem involving don't care **conditions is treated. The cost of a Boolean tree shall be the sum of the costs of the building blocks of which it is composed. A special case of this problem is the classical logical problem of finding a functional expression for a given logical function which uses a minimum number of conjunctions, disjunctions and negations. Programmed on an IBM 704 computer, the algorithm is believed to be efficient on problems with eight or less variables.**

Introduction

The problem. We proceed by way of example. Consider the following functional expression for a Boolean function f of 4 variables,

$$\mu\{\textcircled{v}[\,\&(a, b), c], d, \mathrm{v}(e, f)\},$$

where the symbols μ, $\textcircled{v}$, &, v denote, respectively, the MAJORITY, the EXCLUSIVE-OR, the AND, and the 2-variable OR. This functional expression may be represented by the "Boolean tree" of Fig. 1.

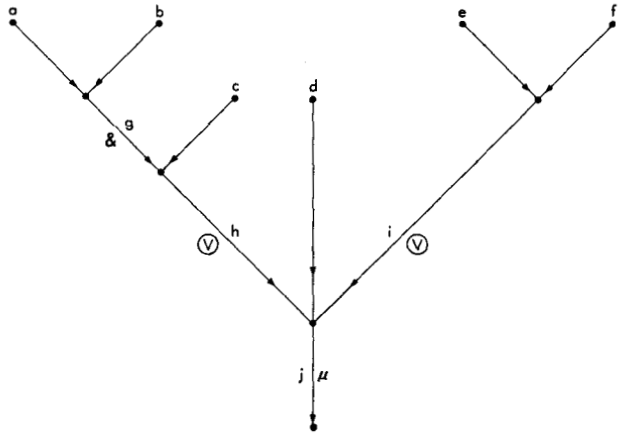

Figure 1 **Boolean tree of** $\mu\{\textcircled{v}[\&(a, b), c], d, \mathrm{v}, (e. f)\}$

*This is Part IV of a series of papers on a theory of logical design of automata.

The Boolean tree may be thought of as a wiring diagram for a logical circuit realizing this function, constructed from a set $\mathfrak{B} = \{\mu, \textcircled{v}, \&, \mathrm{v}\}$ of primitive logical building blocks.

This paper treats the question of finding a functional expression, or Boolean tree, constructed from an arbitrary set $\mathfrak{B}$ of primitive logical building blocks, primitive Boolean functions; more generally, however, this paper solves the following minimization problem.

Let $\mathfrak{B}$ be a set of primitive building blocks, that is, a prescribed set of Boolean functions, with each being associated a positive integer called its *cost.* Let the cost of a functional expression be the sum of the costs of the elements therein. Given a Boolean function f, the problem is to devise an algorithm which will construct a functional expression for f, from the set $\mathfrak{B}$ which has a minimum cost. Actually the problem treated here is a more general one, arising naturally in the design of automata, involving so-called DON'T-CARE conditions.

A special case is the classical logical problem (cf. Hilbert-Bernays) of finding a minimum functional expression for f, constructed from the set consisting of the following three functions: the two-variable AND, the two-variable OR, and the NOT; the cost of each of these expressions is usually considered to be one.

Previous work in the field. As stated by several authors, the bulk of the work in the area of minimization of Boolean functions has been devoted to the problem of finding minimum expressions of the so-called two-level AND-OR type. Another way to describe the same thing: to find the minimum normal form expression for a given Boolean function. Still another equivalent description for the same problem: to find a minimal cover of a cubical complex. A fairly comprehensive list of references for work done in this field is contained in the bibliography of the first three parts of this series. Thus, attention here will be devoted to work on the more general problem.

Muller, Ashenhurst, and Markov have attacked the problem of finding upper and lower bounds for the minimum cost of a decomposition, under various assumptions.

In a series of papers Ashenhurst has devoted himself to finding decompositions for Boolean functions. In his paper in the Proceedings of the International Symposium on the Theory of Switching [2], he considers the problem of disjunctive decompositions. A Boolean function $f(x_1, \cdots, x_n)$ is said to have a disjunctive decomposition $F[y_1, \cdots, y_s, \phi(z_1, \cdots, z_t)]$ if no y equals any z. He describes a method for detecting such decompositions and gives illustrations for functions of six variables. Ashenhurst has several interesting results in this study.

Abhyankar [1] addresses himself to the general problem of finding minimum "sums of products of sums". Here $\mathfrak{B}$ consists of ANDs and ORs. He describes procedures for obtaining such expressions in the case when the complex for the given function consists of two isolated vertices; he also obtains partial results in the case when this complex consists of three isolated vertices. Another way to describe the first problem is to say that the function is such that its complete canonical form consists of exactly two terms and these two terms differ from each other in at least two variables. A similar description is possible for the case when the cubical complex for the function consists of three isolated vertices.

Roth and Wagner [15] give an algorithm for the following specialization of the general problem. The decomposition is allowed to be a disjunction of a set of subdecompositions to be termed Boolean trees, such that within each subdecomposition no input variable appears more than once. Each primitive function has a positive cost and the algorithm gives a minimum over this class of circuits. The methods of the present paper are generalizations of the concepts there developed. Much insight into the present problem was provided by Eric G. Wagner.

In a subsequent paper the author gives an algorithm for the multiple-output problem.

A compact notation for normal form. One of the essential difficulties in dealing with problems in a large number of variables is to develop a compact notation to represent these large functions. The truth-table type of representation of a Boolean function, for example, is not very efficient even for relatively small numbers of variables. The author's mode of representation of a Boolean function is essentially a shorthand notation for a normal form expression. This notation will become clear by means of a simple example. Consider the function $f = ab\bar{c} \vee bc\bar{d}e \vee \bar{a}\bar{e}$. This normal form expression may be represented by the following array of symbols consisting of 0, 1 or x.

a	b	c	d	e	
1	1	0	x	x	$\Leftrightarrow a\, b\, \bar{c}$
x	1	1	0	1	$\Leftrightarrow b\, c\, \bar{d}\, e$
0	x	x	x	0	$\Leftrightarrow \bar{a}\, \bar{e}$.

This array has five columns, one for each variable and three rows, one for each term. The term $ab\bar{c}$ is represented by the first row of this array, namely, $110xx$. The variables a and b appear in the term and are represented by 1's in the columns a and b; c appears negated and is thus represented by a 0 in the c column. Variables d and e do not appear: their absence is indicated by x's appearing in columns d and e. In similar fashion $bc\bar{d}e$ is denoted $x1101$ and $\bar{a}\bar{e}$ by $0xxx0$.

In general, a normal form expression in n variables will be represented by an array with n columns and with as many rows as terms in the normal form. If a variable occurs unnegated in this term, a 1 appears in the column corresponding to this variable. If a variable occurs negated, then a 0 appears in the appropriate column. If the variable does not occur in the term, then an x appears in the appropriate column. In particular, the function f which is identically 1 would be represented by a single row, each column of which is an x. The function f, identically 0, would be represented by an empty array, to be denoted ϕ.

To correlate with the author's terminology, such an array of 1's, 0's and x's is termed a *cube* and the set of all possible cubes corresponding to terms which might be used in a normal form expression for a given Boolean function is termed the *complex* of the function.

It is not sufficient merely to have a compact notation to represent a Boolean function. One must be able to perform all the necessary operations on this array. In other words, for instance, tests for implication, complementation, intersection, et cetera, must be described in terms of a calculus based upon this notation. Section 2 recapitulates such a calculus, based on the "cubical" notation.

One more conception, familiar to logical designers, involves the so-called DON'T-CARE conditions. Let h and f be Boolean functions such that h implies f. We seek a functional expression G representing a Boolean function g such that h implies g and g implies f. The function $\bar{h} \cdot f$ is the DON'T-CARE conditions. The customary problem is the case when $h = f$, but in the design of machines it is not uncommon that conditions arise to whose "outcome" the designer is indifferent: these DON'T-CARE conditions may be utilized to achieve a more economical design. It may be noted that the algorithm which the author gives here utilizes in a conceptual sense the DON'T-CARE conditions.

Description of the algorithm. An indication of the method will be given by consideration of a few examples. Consider the example shown in Fig. 1.

	a	b	c	d	e	f	g	h	i	j	
A										1	
B				x				1	1		$\Pi_i^{\mu dhi}$
				1				x	1		
				1				1	x		
C				x	1	x		1			$\Pi_i^{\mathbf{v}ef}$
				x	x	1		1			
				1	1	x		x			
				1	x	1		x			
				1	x	x		1			
D			0	x	1	x	1				$\Pi_h^{\mathbf{v}cg}$
			1	x	1	x	0				
			0	x	x	1	1				
			1	x	x	1	0				
			x	1	1	x	x				
			x	1	x	1	x				
			1	1	x	x	0				
			0	1	x	x	1				
E	1	1	0	x	1	x					$\Pi_g^{\&ab}$
	0	x	1	x	1	x					
	x	0	1	x	1	x					
	1	1	0	x	x	1					
	0	x	1	x	x	1					
	x	0	1	x	x	1					
	x	x	x	1	1	x					
	x	x	x	1	x	1					
	0	x	1	1	x	x					
	x	0	1	1	x	x					
	1	1	0	1	x	x					

Figure 2 **Table for injection operations of Boolean tree of Figure 1**

This Boolean tree describes a function of six variables. For descriptive purposes each branch of the tree is labeled with a distinctive letter. The lowest branch is labeled j and we may think of j as the "output" variable for the function μ. This is, $j = \mu(d, h, i)$. We shall construct a normal form expression defined by this Boolean tree in iterative fashion. Considered as a function of the single variable j, this function is 1 if and only if j is 1. Thus, referring to the table of Fig. 2, the first row, labeled A, consists of the entry 1 in the column labeled j. Now we do the equivalent of making the substitution $j = \mu(d, h, i)$.

First, however, let us settle on a compact normal form notation for the *majority* element: $\mu(d, h, i) = hi \vee di \vee dh$; this corresponds in the 0-1-x notation to the cubes $x11$, $1x1$, $11x$. Hence, the substitution $j = \mu(d, h, i)$ consists in replacing the row having a single 1 in it, in the j^{th} column, by the three rows labeled B in Fig. 2. The exchange from the row A to the rows B is to be thought of as a transformation $\Pi_i^{\mu dhi}$. This is termed an *injection operator.* Next the equivalent of the substitution $i = \mathbf{v}(e, f)$ is made. Consider, for instance, the first row of the rows B. This is transformed into the first two rows of C according to the following reasoning. The cubical representation used here for $\mathbf{v}(e, f)$ is

$e \quad f$

$x \quad 1$

$1 \quad x.$

Similarly, the second row of B is transformed into the third and fourth rows of C and the last row of B is transformed into the fifth row of C.

In similar fashion the succeeding injection operators are defined, to yield the set of rows E which are a normal form representation for the function described by the Boolean tree of Fig. 1. This normal form is expressed in terms of the "input variables" a, b, c, d, e, f. A precise definition of the injection operators is given in the body of the text. The process of applying a set of injection operators corresponding to some functional expression is in effect a method for transforming a Boolean function given in any functional expression into a normal form expression. To put it another way, we have analyzed the circuit, the Boolean tree of Fig. 1. The problem of synthesis is, of course, more difficult: In the synthesis problem one would be given a normal form expression such as exhibited by the rows E of Fig. 2, and one would be attempting to determine whether or not a Boolean tree such as that of Fig. 1 was an admissible functional expres-

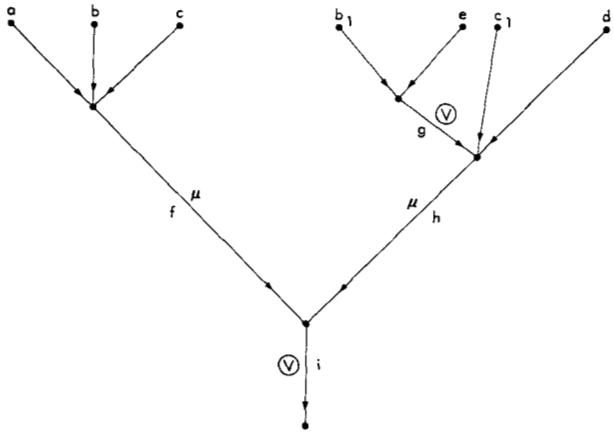

Figure 3 $\textcircled{v}[\mu(a, b, c), \mu(\textcircled{v}(b, e), c, d)]$

sion, that is, constructible from a prescribed set of logical building blocks.

Let us consider one more example, however, as shown in Fig. 3, to illustrate a fundamental difficulty which did not arise in the previous example.

It will be observed in this Figure that the input variables b and b_1 and c and c_1 appear; assume that $b = b_1$ and $c = c_1$. Figure 4 exhibits the successive injection operations defined by the Boolean tree of Fig. 3. It will be observed that this Boolean tree has each branch labeled by a distinct variable. The output branch is labeled i and the function attached thereto is the EXCLUSIVE-OR $\textcircled{v}$. In similar fashion to the previous example, the first set of rows labeled A consists of a 1, in the column designated by the variable i. Now the EXCLUSIVE-OR $\textcircled{v}$ will be designated by the normal form $\bar{f}h \vee f\bar{h}$ or in the 0-1-x notation.

f	h
0	1
1	0

The process of going from row A to rows B corresponds to applying the injection operator Π_i^{vfh}; of going from rows B to rows C, to applying the injection operator $\Pi_f^{\mu abc}$. To describe the next operation, a new phenomenon must be considered. Here we must represent not only the majority function but also the inverse of the majority function, $\bar{\mu} = \bar{b}\bar{c} \vee \bar{a}\bar{c} \vee \bar{a}\bar{b}$. Thus the inverse of the majority

	a	b	c	d	e	f	g	h	i	b_1	c_1	
A									1			
B						0		1				Π_i^{vfh}
						1		0				
	x	0	0					1				$\Pi_f^{\mu abc}$
	0	x	0					1				
C	0	0	x					1				
	x	1	1					0				
	1	x	1					0				
	1	1	x					0				
	x	0	0	1			1				x	$\Pi_h^{\mu dgc_1}$
	x	0	0	x			1				1	
	x	0	0	1			x				1	
	0	x	0	1			1				x	
	0	x	0	x			1				1	
	0	x	0	1			x				1	
	0	0	x	1			1				x	
	0	0	x	x			1				1	
D	0	0	x	1			x				1	
	x	1	1	0			0				x	
	x	1	1	x			0				0	
	x	1	1	0			x				0	
	1	x	1	0			0				x	
	1	x	1	x			0				0	
	1	x	1	0			x				0	
	1	1	x	0			0				x	
	1	1	x	x			0				0	
	1	1	x	0			x				0	
	x	0	0	1	1					0	x	$\Pi_g^{vcb_1}$
	x	0	0	1	0					1	x	
	x	0	0	x	1					0	1	
	x	0	0	x	0					1	1	
	x	0	0	1	x					x	1	
	0	x	0	1	1					0	x	
	0	x	0	1	0					1	x	
	0	x	0	x	1					0	1	
	0	x	0	x	0					1	1	
	0	x	0	1	x					x	1	
	0	0	x	1	1					0	x	
	0	0	x	1	0					1	x	
	0	0	x	x	1					0	1	
	0	0	x	x	0					1	1	
E	0	0	x	1	x					x	1	
	x	1	1	0	0					0	x	
	x	1	1	0	1					1	x	
	x	1	1	x	0					0	0	
	x	1	1	x	1					1	0	
	x	1	1	0	x					x	0	
	1	x	1	0	0					0	x	
	1	x	1	0	1					1	x	
	1	x	1	x	0					0	0	
	1	x	1	x	1					1	0	
	1	x	1	0	x					x	0	
	1	1	x	0	0					0	x	
	1	1	x	0	1					1	x	
	1	1	x	x	0					0	0	
	1	1	x	x	1					1	0	
	1	1	x	0	x					x	0	
	x	0	0	1	1							$\mathcal{C}$
	0	1	0	1	0							
	0	0	x	1	1							
	0	0	1	x	1							
	0	0	1	1	x							
F	x	1	1	0	1							
	1	0	1	0	0							
	1	1	x	0	1							
	1	1	0	x	1							
	1	1	0	0	x							

Figure 4 **Table for injection operations of Boolean tree of Figure 3 followed by consistency operator** $\mathcal{C}$

will be represented by the following array:

$a\ b\ c$

$x\ 0\ 0$

$0\ x\ 0$

$0\ 0\ x.$

The first of the rows B thus transforms into the first three of the rows C, while the second of the rows B transforms into the last three of the rows B. By similar rules one transforms the rows C into the rows D and the rows D into the rows E by the injection operator $\Pi_h^{udgc_1}$ and $\Pi_g^{vb_1e}$.

Now, however, we impose the conditions $b = b_1$, $c = c_1$. Again, this must be performed in the cubical calculus. In effect, we must perform an "intersection" of the columns c and c_1, b and b_1. If we examine the fate of the first two rows of the rows E, as reproduced below in Fig. 5, we will understand the general situation. As indicated in the figure, the first term corresponds to the expression $\bar{b}\bar{c}de\bar{b}_1$. Since b and b_1 both appear negated while c appears negated but c_1 does not appear, this term is equivalent to the term $\bar{b}\bar{c}de$, which in the "complex" of the variables $a\ b\ c\ d\ e$ is represented by the "cube" $x0011$.

Observe, however, the fate of the second "cube," corresponding to $\bar{b}\ \bar{c}\ d\ \bar{e}\ b_1$. Here b appears negated whereas b_1 is not negated. This expression thus corresponds to a contradiction and is hence represented by the empty "cube" ϕ. Applying similar rules to each of the rows E, one deduces the set of rows F described in the complex of variables $a\ b\ c\ d\ e$. Essentially, what we have done is to knock out the inconsistent terms inherent in the normal form expression represented by the rows E. The set of rows F is a normal form representation for the Boolean function in Fig. 3. (See Section 7 for a more efficient method of derivation.)

This example indicates that there would be no telling from looking at the normal form expression described by the rows F that a Boolean tree such as given in Fig. 3 was "admissible" for the function. This expression must somehow have been deduced from the larger "complex" described by the set of rows E. In effect, what must be done to realize the existence of such a functional expression is to adjoin contradictions to the normal form expression. A calculus and algorithm for such a procedure is one of the main results of this paper.

a	b	c	d	e	b_1	c_1			a	b	c	d	e
x	0	0	1	1	0	x	$\leftrightarrow \bar{b}\,\bar{c}\,d\,e\,\bar{b}_1 \leftrightarrow$	x	0	0	1	1	
x	0	0	1	0	1	x	$\leftrightarrow \bar{b}\,\bar{c}\,d\,\bar{e}\,b_1 \leftrightarrow$			ϕ			

Figure 5

Section 1 states the problem. Section 2 develops a calculus for performing logical operations–specifically, a calculus of cubical complexes. Section 3 describes the so-called singular complex, which is essentially the author's device for handling variables which appear more than once in a term, and may be thought of as a systematic procedure for identifying variables and adding the contradictions induced by these identifications. Such a procedure seems to be necessary to recognize certain "economical" functional expressions. Section 4 describes the projection operator which determines for a given primitive function and selection of variables whether or not a factorization is permissible for these choices. For a particular primitive function α and selection of coordinates λ, if the "projection operator acts perfectly" then a factorization exists. A fast procedure for determining whether or not a projection operator acts perfectly is given in this section. The algorithm requires frequent constructions of the Π-operation and the author is indebted to his colleagues Drs. J. H. Griesmer and R. M. Karp for their algorithm for this construction, given in Section 4.4. The procedure is "fast" in that it tests each cube of a cover separately and each individual test is rapid. Section 5 describes projective words which are products of projection operators and, suitably restricted, are in one-to-one correspondence with Boolean trees. (More precisely, it is a correspondence between equivalence classes). Section 6 defines the injection operator, which is a type of inverse of the projection operator. Its use in this paper is mostly confined to proving that the over-all algorithm does, in fact, produce a minimum. This operator, however, has demonstrated its own utility in the analysis of circuits. Section 7 describes an algorithm for computing the Boolean function of any Boolean tree. The method described was suggested independently by Miss Ruth Norby and Charles Stieglitz. A comparison of Fig. 8 with Fig. 4 will give an indication of its relative efficiency. In Section 8 the algorithm for finding a minimum Boolean tree compatible with the design requirements is given. The essential trick is to note that if the cost of a building block is one less than its number of inputs, then the cost of the Boolean tree is monotone with the "degeneracy" necessary in order to make the projective word corresponding to the Boolean tree act "perfectly." A solution having been obtained under this assumption is then used to set a bound on the size of the singular complex,

through which it is necessary to search for more economical solutions. As more economical solutions are obtained, this bound is diminished. Appendix A describes a procedure, adapted from R. Bellman [6], for determining an optimum criterion for termination. This development is due to R. M. Karp. Appendix B contains a short discussion of the efficiency of the algorithm and fast approximations thereto.

1. Statement of the problem

Let $\mathfrak{B}$ be a set of primitive building blocks, that is, a prescribed set of Boolean functions, with each being associated a positive integer called its *cost*. Let the cost of a functional expression or Boolean tree be the sum of the costs of the primitive blocks or primitive functions prescribing it.

Let f and d be Boolean functions (d corresponds to the DON'T-CARE conditions). *Problem*: Find a functional expression F composed of primitives from $\mathfrak{B}$ such that $f \rightarrow F \rightarrow f \vee d$, of minimum cost.

2. The complex of a Boolean function, a calculus of complexes

A calculus of cubical complexes, essential for subsequent developments, is described in this section.

2.1 Let Q^n denote the set of all conjunctions of n literals $a_1, \cdots, a_n$ or their negations, in which each a_i appears at most once, either negated or unnegated: such a conjunction T is called a *term*. Let this term be represented by an ordered set of n symbols $(t_1, \cdots, t_n)$, where: the i^{th} "coordinate" $t_i = 1$ if a_i appears in unnegated form in T; $t_i = 0$ if a_i appears in negated form in T; $t_i = x$ if a_i does not appear at all in T. Such a set $t = (t_1, \cdots, t_n)$ is termed a *cube*. If no $t_i = x$, t is termed a *vertex*, and corresponds to a term in complete disjunctive normal form. The function f identically 1 defines the single cube $xx \cdots x$, n x's; the function g identically 0 corresponds to the *empty cube*, denoted ϕ.

2.2 Since there is a one-to-one correspondence between terms T formed from n literals, and cubes with n coordinates, the set of all such cubes may be identified with the set Q^n, and termed thereby the *n-cube*.

Geometrical cubes have long been familiar to switching theorists. Their representation as an ordered set of symbols 0, 1 or x makes possible the development of a calculus for performing logical operations, suitable for a digital computer and entailing a certain "compactness of information."

2.3 *The complex of a Boolean function.* Let f be a Boolean function of n variables. The set of all terms permissible in any normal form expression for f is termed the *complex* of f; equivalently, it is the set of all terms which imply f. We shall deal with the cubical representation of the complex of f. The set $\mathbf{K}(f)$ of cubes determined by the terms which imply f has the property that if it contains a given cube $(t_1, \cdots, t_n)$ then it contains all of its subcubes or "faces", obtained by replacing some coordinates t_i, which are x, to a 0 or 1:* This set $\mathbf{K}(f)$ is termed the *cubical complex* of f. This may be made the abstract property defining a "cubical complex." (Indeed, it is precisely the same property which is used to define the simplicial complex of combinatorial topology.)

Definition. A *cubical complex* K of Q^n is a subset of Q^n with the property that if c is a cube which belongs to K, then its faces also belong to K. A subset L of K, $L \subset K$, is termed a *subcomplex* of K if L is itself a complex; if L is the cubical complex of a Boolean function g, $L = \mathfrak{K}(g)$, and K, of f, $K = \mathbf{K}(f)$, then $L \subset K$ is equivalent to the proposition $g \rightarrow f$.

2.4 *K-cover of L.* Given complex K and subcomplex L, a *K-cover* of L is a set C of cubes of K such that each vertex of L is a face of some cube of C. This means that for each vertex v of L there is a cube c in C which can be transformed into $v = (v_1, \cdots, v_n)$ by changing each of the coordinates c_i of c, which are x, to the value v_i.

This definition has the following logical interpretation: K, C, and L define Boolean functions f, g, and h respectively — C is a normal form expression for g — such that $h \rightarrow g \rightarrow f$.

If $K = L$, a K-cover is referred to simply as a *cover*.

2.5 *Products of cubes and complexes.* The products of cubes and of complexes introduced in this section provide a basis for a calculus which permits the forming of all operations of the propositional calculus.

Let $c = (c_1, \cdots, c_n)$ and $d = (d_1, \cdots, d_n)$, coordinates c_i and d_i being 0, 1 or x, be cubes of Q^n.

2.5a The *intersection* $c \cap d$ of c and d is defined by the following table

*Cube $s = (s_1, \ldots, s_n)$ is a *face* of cube $t = (t_1, \ldots, t_n)$ if, for each i, $t_i = s_i$ or x.

$\cap$	0	1	x
0	0	ϕ	0
1	ϕ	1	1
x	0	1	x

and the rule

$c \cap d = \phi$, if for any i, $c_i \cap d_i = \phi$,

$c \cap d = (c_1 \cap d_1, \cdots, c_n \cap d_n)$ otherwise.

The intersection of two cubes corresponds logically to forming the product of the corresponding terms. Following set-theoretic conventions, c and d are said to be *disjoint* if $c \cap d = \phi$. The *intersection* of complexes A, B is the complex determined by all cubes $a \cap b$, $a \in A$, $b \in B$, and shall be denoted $A \cap B$.

2.5*b* The *#-product* $c\#d$ of cubes c and d is a particular set of cubes forming a cover of the vertices of c which are not in d. This product is thus a species of differencing operation. Logically speaking, if c and d correspond to terms S and T respectively, then $c\#d$ is a way of forming $S \cdot \bar{T}$.

It is defined explicitly by the following rules:

1) $c\#d = \phi$ if $c \subset d$.

2) $c\#d = c$ if $c \cap d = \phi$; i.e., $c_i \cap d_i = \phi$ for some i.

3) Otherwise, let $i(1), \cdots, i(r)$ be the set of coordinates $i(j)$, for which $c_{i(j)} = x$ and $d_{i(j)} = 0$ or 1. Then $c\#d$ contains r cubes, the j^{th} being $(c_1, \cdots, c_{i(j)-1}, \bar{d}_{i(j)}, c_{i(j)+1}, \cdots, c_n)$, which has all its coordinates equal to those of c, except for the $i(j)^{\text{th}}$, which is 1 if $d_{i(j)}$ is 0 and 0 if $d_{i(j)}$ is 1.
Examples: $x110\#xx10 = \phi$; $xx1x1\#x1x0x = x01x1 + xx111$; $x11x0x\#1x100x = 011x0x + x1110x$; $xxx\#101 = 0xx + x1x + xx0$.

Machinewise, the first two operations are performed first and in parallel—that is, on all coordinates simultaneously.

Given complexes A and B, we use the same symbol, $A\#B$, to denote the complex composed of all cubes of A which are disjoint from cubes of B.

Likewise if C and D are sets of cubes (they may be thought of as covers for K and L, respectively) then $C\#D$ is the set of all products $c\#d$, with c an element of C and d an element of D. Machinewise, however, $C\#D$ is more efficiently programmed on a "subsuming-as-you-go" basis.

2.5*c* The **-product* $c * d$ of cubes c and d is defined by the following table of the coordinate *-product

*	0	1	x
0	0	y	0
1	y	1	1
x	0	1	x

and the rule: if $a_i * b_i = y$ for more than one i, then $a * b = \phi$; if not, then $a * b = (i(a_1 * b_1), \cdots, i(a_n * b_n))$, where $i(0) = 0$, $i(1) = 1$ and $i(x) = i(y) = x$. For example, $x10x * 101x = \phi$; $xx100 * 1xx01 = 1x10x$.

Logically speaking, the *-product of two terms is the "largest" term which implies their disjunction: it is a generalization of Quine's consensus.

*The *-product of complexes* A and B is the complex determined by the set $A * B$ of all *-products of cubes of A with cubes of B. In particular $A * A$ is the unique Boolean complex containing A.

2.5*d* Given complexes A and B, the *union* $A \cup B$ shall denote the complex determined by their set-theoretic union. In general $A \cup B$ will not be Boolean.

The next section develops a generalization of the cubical complex.

3. Singular cubical complexes

3.0 *Motivation.* As indicated by the second example of the Introduction, shown in Fig. 3, it seems necessary to have a systematic procedure for adding contradictions to normal form expressions for a given function in order to recognize the existence of certain "economical" functional expressions. The singular cubical complex may be thought of as a device for realizing such a systematic procedure. (From another point of view, however, the singular complex may be considered as standing in strict analogy to the combinatorial singular complexes of combinatorial topology, with a somewhat different twist.)

Contradictions are here to be thought of as coming about through the identification of certain variables. This identification is defined by means of the "degeneracy map."

3.1 *The degeneracy map.* Let $I_n = \{1, 2, \cdots, n\}$. Let ψ be a mapping of I_{n+r} into I_n such that $\psi(i) = i$ for $i \leq n$; ψ is termed a *degeneracy map*. Its *degeneracy* is r.

3.2 *Complex of inconsistency.* Given a degeneracy map ψ, let $N(\psi)$ denote the complex in Q^{n+r} determined by the following cubes and their faces: for each pair of integers i, j of I_{n+r}, $i \neq j$, for which $\psi(i) = \psi(j)$, $N(\psi)$ contains the cubes: c_{01} whose

i^{th} coordinate is 0 and whose j^{th} coordinate is 1, all other coordinates being x, and; c_{10} whose i^{th} coordinate is 1, j^{th} coordinate 0, all others being x. Now $N(\psi)$—the N stands for nonsense—will be termed the *complex of inconsistency* of ψ. (Cubes of $N(\psi)$ are familiarly referred to as nonsense cubes.)

3.3 A *singular complex* consists of a cubical complex S of Q^{n+r} plus a degeneracy map $\psi{:}I_{n+r} \rightarrow I_n$. The subcomplex $S \cap N(\psi)$ of S is termed the *subcomplex of inconsistency*, while its complement $S\#N(\psi)$ is termed the *subcomplex of consistency*. The *order* of S is $n + r$; its *degeneracy* is r, and its *rank* is n. The cubical complex without degeneracy map, as defined in Section 2 above, may be considered as corresponding to the special case when the degeneracy r is zero, so that ψ is the identity map. In this case the complex will be termed *non-singular*.

3.4 *Consistency operator.* Associated with any degeneracy map are three other mappings. The first, termed the *consistency operator* $\mathcal{C}$, is a map of Q^{n+r} into Q^n: for d an element of Q^{n+r}, $d = (d(1), \cdots, d(n + r))$,

$$\mathcal{C}\, d = \phi \quad \text{if for any } i, \quad \bigcap_{i=\psi(j)} d(j) = \phi,$$

$$\mathcal{C}\, d = \left(\bigcap_{\psi(k)=1} c(k), \cdots, \bigcap_{\psi(p)=n} c(p)\right) \quad \text{otherwise.}$$

Such a map preserves face relations and thus induces a map—this will also be denoted $\mathcal{C}$—of $\mathbf{K}^{n+r}$ into $\mathbf{K}^n$ preserving the algebra of cubical complexes.*

3.5 The second map σ, depending only on the degeneracy of ψ, maps Q^n into Q^{n+r}: for

$$c = (c(1), \cdots, c(n)),$$

let

$$\sigma c = (c(1), \cdots, c(n), x, \cdots, x),$$

the last r coordinates being x. This map commutes with the face operators and thus induces a map $\sigma *$ of $\mathbf{K}^n$ into $\mathbf{K}^{n+r}$.

3.6 The third is a map s of $\mathbf{K}^n$ into $\mathbf{K}^{n+r}$: for K a complex of $\mathbf{K}^n$, let

$$s(K, \psi) = \sigma^*(K) \cup N(\psi);$$

$s(K, \psi)$, together with the degeneracy map ψ, is a singular complex.

3.7 The singular complex $s(K, \psi)$ is called the *complete singular complex* defined by K and ψ. The following properties are easily verified.

*$\mathbf{K}^n$ denotes the set of all cubical complexes in Q^n.

3.8 *Propositions.*

$$\mathcal{C}s(K, \psi) = K,$$

$$A \subset B \Rightarrow \mathcal{C}A \subset \mathcal{C}B,$$

$$\mathcal{C}(A\#B) = \mathcal{C}A\#\mathcal{C}B.$$

3.9 *Example.* Let $K \subset Q^4$ be given by the following cover,

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \left[\begin{array}{cccc} 1 & 0 & x & 1 \\ x & x & 0 & 0 \\ 1 & 0 & 1 & x \end{array}\right. & & & \left.\begin{array}{c} \\ \\ \\ \end{array}\right. \end{array}.$$

Let $\psi{:}I_5 \rightarrow I_4$ be given by $\psi(5) = 1$. Then

$$N(\psi) = \begin{array}{c} 1\ 2\ 3\ 4\ 5 \\ \left[\begin{array}{ccccc} 1 & x & x & x & 0 \\ 0 & x & x & x & 1 \end{array}\right] \end{array}$$

and $s(K, \psi)$ is given by the following cover

$$s(K, \psi) = \begin{array}{c} 1\ 2\ 3\ 4\ 5 \\ \left[\begin{array}{ccccc} 1 & 0 & x & 1 & x \\ x & x & 0 & 0 & x \\ 1 & 0 & 1 & x & x \\ 1 & x & x & x & 0 \\ 0 & x & x & x & 1 \end{array}\right] \end{array}.$$

4. Projection operator

4.0 *Motivation.* A basic subproblem in the general problem of devising an efficient procedure for designing Boolean trees of minimum cost to realize any prescribed function is to devise a *fast* test for whether or not a conjectured factorization "works."

To be specific, suppose f is a Boolean function, $f = f(a_1, \cdots, a_r, b_1, \cdots, b_s)$. Given a primitive $\alpha = \alpha(c_1, \cdots, c_s)$, whether there exists a function G such that

$$f = G[\alpha(a_1, \cdots, a_r), b_1, \cdots, b_s]$$

is seen to be equivalent to determining whether or not f can be rendered in the form

$$\alpha(a_1, \cdots, a_r)P(b_1, \cdots, b_s) \vee$$

$$\bar{\alpha}(a_1, \cdots, a_r)\, Q(b_1, \cdots, b_s) \vee R(b_1, \cdots, b_s), \qquad (1)$$

where P, Q, and R are normal form expressions in the variables $b_1, \cdots, b_s$.

Given DON'T-CARE conditions d, the factorization problem then becomes: to determine whether there

exist normal form expressions P, Q, R of variables $b_1, \cdots, b_s$ such that

$$f \to \alpha P \vee \bar{\alpha} Q \vee R \to f \vee d. \tag{2}$$

The problem is not simply to determine whether or not this single factorization exists, however, but to determine a complete factorization (a Boolean tree) which realizes the given function, possibly involving utilization of the DON'T-CARE conditions. Hence the procedure for determining whether a single factorization works must be set up so as to be convenient for iteration. The procedure to be described is accomplished by means of the cubical calculus.

Let α be a given primitive, $\alpha = \alpha(a_1, \cdots, a_r)$. Let λ denote (the selection of) r integers from I_n; let μ denote (the selection of) the complementary set.

Let K be a cubical complex. Let $\mathbf{K}^t$ denote the set of all cubical complexes in Q^t. Then the projection operator $\Pi_{\alpha\lambda}$ is a mapping of $\mathbf{K}^{r+s}$ into $\mathbf{K}^{s+1}$.

Let f and $f \vee d$ correspond to complexes L, K of Q^n. If there exists an expression of type (1) such that implications (2) hold, so that the conjectured factorization "works," then $\Pi_{\alpha\lambda}$ is said to be (K, L)-*perfect.*

Algebraically, the projection operator or Π-operator is defined as a mapping of cubical complexes into cubical complexes. Products of projection operators, termed *projective words*, are thus definable. Interest in this paper centers on "tree-like" projective words and these correspond to Boolean trees and hence to multiple and complete factorizations.

Since a singular complex has been defined as a pair consisting of a cubical complex S of $\mathbf{K}^{n+r}$ plus a map $\psi: I_{n+r} \to I_n$, it will suffice to define the Π-operator on a cubical complex. In Roth-Wagner [15] it was indicated that the π-operators were independent of the cover on which they were defined. In the present context, however, a considerably more general definition is required; the definition will be intrinsic—that is, independent of the cover.

4.1 *A Cartesian product.* First it is necessary to define a special product of complexes. Let $M \in \mathbf{K}^r$ and $N \in \mathbf{K}^s$ with $t = r + s$. Let λ and μ be one-to-one maps of I_r and I_s, respectively into I_t, with $\lambda(I_r) \cap \mu(I_s) = \phi$. The (λ, μ)-product of M and N is a complex $M \times_{\lambda\mu} N$ of $\mathbf{K}^t$: for each $c \in M$ and $d \in N$, then $M \times_{\lambda\mu} N$ shall contain a cube $c \times_{\lambda\mu} d$, where for each $i \in I_t$,

$$(c \times_{\mu\lambda} d)(i) = c(\lambda^{-1}(i)) \quad \text{if} \quad i \in \lambda(I_r)$$

$$(c \times_{\mu\lambda} d)(i) = d(\mu^{-1}(i)) \quad \text{if} \quad i \in \mu(I_s).$$

4.2 *Abstract definition of* Π*-operator.* Next let $\mathbf{A}^r$ be the set of all pairs of disjoint complexes in Q^r,

$$\mathbf{A}^r = \{(\alpha_0, \alpha_1) \mid \alpha_0 \cup \alpha_1 \subset Q^r, \alpha_0 \cap \alpha_1 = \phi\}.$$

The subset $\mathbf{B}^r$ of pairs (α_0, α_1) of Boolean complexes whose union contains all vertices of Q^r is in one-to-one correspondence with the set of all Boolean functions of r variables. Let $\mathbf{K}^t$ denote the set of all cubical complexes in Q^t. Let $J_t^{r,s}$ denote the set of all pairs (λ, μ) of one-to-one maps of I_r and I_s into I_t with disjoint images.

Let $\alpha = (\alpha_0, \alpha_1) \in \mathbf{A}^r$, $K \in \mathbf{K}^t$ and $(\lambda, \mu) \in J_t^{r,s}$. Then $\Pi(\alpha, K, (\lambda, \mu))$ is a complex of $\mathbf{K}^{s+1}$, whose cubes are defined by the following relations:

$$\Pi(\alpha, K, (\lambda, \mu)) \supset \beta_1 \times 1 \quad \text{if} \quad \alpha_1 \times_{\mu\lambda} \beta_1 \subset K$$

$$\Pi(\alpha, K, (\lambda, \mu)) \supset \beta_0 \times 0 \quad \text{if} \quad \alpha_0 \times_{\lambda\mu} \beta_0 \subset K$$

$$\Pi(\alpha, K, (\lambda, \mu)) \supset \beta \times x$$

$$\text{if} \quad \alpha_1 \cup \alpha_0 \times_{\lambda\mu} \beta \subset K \quad \text{and if} \quad \alpha \in \mathbf{B}^r.$$

Here $\beta_1 \times 1$ denotes an element of Q^{s+1}, defined in the following way:

$$(\beta_1 \times 1)(i) = \beta_1(i) \quad \text{for} \quad i < s + 1$$

$$(\beta_1 \times 1)(i) = 1 \quad \text{for} \quad i = s + 1.$$

A similar definition holds for $\beta_0 \times 0$ and $\beta \times x$.

Note: It is frequently convenient, when iterating Π-operators, to consider that the elements of the image complex in $\mathbf{K}^{s+1}$ are mappings from the set $\{\mu(I_s) \cup s + 1\}$, instead of from I_{s+1} into $\{0, 1, x\}$. Since there is a natural one-to-one correspondence between these two sets, this will cause no difficulty.

The image complex $\Pi(\alpha, K, (\lambda, \mu))$ is thus defined by prescribing the cubes which it contains. If it contains no cubes, then it will be said to form the empty complex of $\mathbf{K}^{s+1}$.

It is convenient to introduce the following notions. Let (λ, μ) be an element of $J_t^{r,s}$ with $t = r + s$. The pair (λ, μ) thus splits any cube c of Q^t into two parts, its "λ-*part*" and its "μ-*part*,"

$$c = (c_\lambda, c_\mu),$$

c_λ being the coordinates of its λ-part, or its λ-*coordinates*, and c_μ the coordinates of its μ-part, or its μ-*coordinates*.

4.3 Let $\alpha = (\alpha_0, \alpha_1)$ and (λ, μ) be fixed, so that Π becomes a function of K alone, $\Pi = \Pi(K)$. Let K be a complex with subcomplex L. Let (c_λ, c_μ) be a cube of L and suppose that $c_\lambda \cap \alpha_i \neq \phi$, for $i = 0$ or 1. The operator Π is said to be (K, L)-*perfect* if for each (c_λ, c_μ) of L, the image $\Pi(K)$ contains the cube $c_\mu \times i$.

	λ	μ	ν		λ	μ	ν	
$M_1 = [\alpha_1] \times_{\lambda\mu} x^3$	1 0 0 1	x x x x x x		Step 1	0 0 1 1	x x x x x x		$M_0 = [\alpha_0] \times_{\lambda\mu} x^3$
C	1 0 0 1 1 1 0 x	x 1 x x x 0 x 0 x 1 x x			1 0 0 1 1 1 0 x	x 1 x x x 0 x 0 x 1 x x		C
$M_1 \# C = N_1$	1 0 0 1	x 0 x 0 x 1		Step 2	0 0 1 1	0 x x x 1 x		$M_0 \# C = N_0$
$(M_1 \# C)_\mu = P_1$		x 0 x 0 x 1		Step 3		0 x x x 1 x		$(M_0 \# C)_\mu = P_0$
x^3		x x x				x x x		x^3
$x^3 \# P_1 = Q_1$		1 1 x x 1 0		Step 4		1 0 x		$x^3 \# P_0 = Q_0$
$Q_1 \times_{\mu\nu} 1$		1 1 x x 1 0	1 1	Step 5		1 0 x	0	$P_0 \times_{\mu\nu} 0$

Figure 6 **Example of Griesmer-Karp construction of Π-operation**

4.4 *Griesmer-Karp construction of* Π*-operation.* The following efficient construction of the Π-operation is due to Griesmer and Karp [8].*

0) Let α be a Boolean function of r variables, determining two disjoint complexes α_0, α_1. Let $[\alpha_0]$, $[\alpha_1]$ be *covers* of these complexes. Let λ, μ be as above. Let K be a complex and C a cover thereof. Then perform the following steps: for $i = 0$ or 1,
1) Form $M_i = [\alpha_i] \times_{\lambda\mu} x^s$ ($x^s = xx \cdots x$, sx's)
2) Form $M_i \# C = N_i$
3) Let $P_i = (M_i \# C)_\mu$ denote the set of μ-parts of $N_i \# C$
4) Form $Q_i = x^s \# (P_i)$
5) Store $Q_i \times_{\mu\nu} i$ in the image.

The above table (Fig. 6) illustrates this procedure for a complex K given by the cover C shown in the second block of rows.

This construction gives the complete answer, so to speak, and admirably does it in terms of covers of the complexes K, α_0 and α_1. (Do not be misled by the fact that in the example the *covers* $[\alpha_0]$ and $[\alpha_1]$ were vertices—it is not necessary in general).

The following is an algorithm for determining whether or not a projection operator is (K, L)-perfect without actually constructing the operation.

4.5 *Test for* (K, L)*-perfection* The next construction enjoys the advantages that are implied in Proposition I below and in the fact that it is defined for covers of the complexes of α.

Proposition I: Let K and L be complexes, K containing L. A Π-operator is (K, L)-perfect if and only if it is (K, c)-perfect for each element c of an L-cover of L.

The following is a restatement of the above proposition in the form on which the construction is based.

Proposition II: Let C be a cover of K, N a subcomplex of K. A Π-operator is $(K, K\#N)$-perfect if and only if, for each c of C, Π is $(K, c\#N)$-perfect.

In the algorithm, N will be the subcomplex of DON'T-CARE cubes plus nonsense cubes.

Proposition III: Let K be a complex and N a subcomplex. Let $c = (c_\lambda, c_\mu)$ be a cube of K. The following algorithm provides an effective test for whether or not a Π-operator Π_α is $(K, c\#N)$-perfect. Here α_1 and α_0 are the complexes of α and are represented by covers $[\alpha_1]$ and $[\alpha_0]$ respectively.

1° Determine whether the following relations hold

$$c_\lambda \cap [\alpha_1] = \phi, \tag{1}$$

$$c_\lambda \cap [\alpha_0] = \phi. \tag{2}$$

Both equalities cannot hold if α corresponds to a Boolean function.

2.01° If Eq. (1) holds and not Eq. (2), then the λ-part of c intersects $[\alpha_1]$ but not $[\alpha_0]$.* Next deter-

**Added in proof*: This procedure works even better if one carries along covers C_0 of K_0 and C_1 of K, where K_0 is the "care-complex" in the complement of K and where the $\#$-procedure is replaced by appropriate intersections. An "intersection" procedure, due to E. G. Wagner, for forming the projection operator seems for certain applications to be more efficient than this construction: cf. IBM Research Report SR-103.

*The designation 2.01°, the decimal part of it .01, is to indicate that the first formula (1) is ϕ (hence the first digit is 0) and that the second (2) is not ϕ (hence the second digit is 1). Similarly for 2.10, 2.00, et cetera.

mine whether

$$[\alpha_1] \subset c_\lambda; \tag{2.01}$$

if it is, then Π_α is (K, c)-perfect* and hence, *a fortiori*, Π_α is $(K, c\#N)$-perfect.

3.01° On the other hand, if $[\alpha_1] \not\subset c_\lambda$, then the following test determines whether or not the remaining elements of C can "make up" for what c itself cannot do:

$$[[\alpha_1] \times_{\lambda\mu} (c\#N)_\mu]\#C = \phi. \tag{3.01}$$

The factor $(c\#N)$ gives the subcomplex of c, not in N, and $(c\#N)_\mu$ denotes the μ-part of this set of cubes. The (λ, μ)-product $[[\alpha_1] \times_{\lambda\mu} (c\#N)_\mu]$ represents then the subcomplex that must be covered by C in order that Π_α be $(K, c\#N)$-perfect; this condition shall hold, clearly, if and only relation (3.01) is satisfied.†

2.10° If relation (2) holds but (1) does not, then proceed with steps 2.10° and 3.10°, which are the same as 2.01° and 3.01°, with $[\alpha_1]$ replaced by $[\alpha_0]$.

2.00° If neither (1) nor (2) hold, then c_λ has cubes in common with both α_1 and α_0. Consequently, both sets of tests—2.01° possibly followed by 3.10°; and 2.10° possibly followed by 3.10°—must be performed. In other words, in this case, for Π_α to be $(K, c\#N)$-perfect, the proposition

$$[(2.01) \vee (3.01)] \ \& \ [(2.10) \vee (3.10)]$$

must be true.

This algorithm may be represented by the following flow chart (Fig. 7).

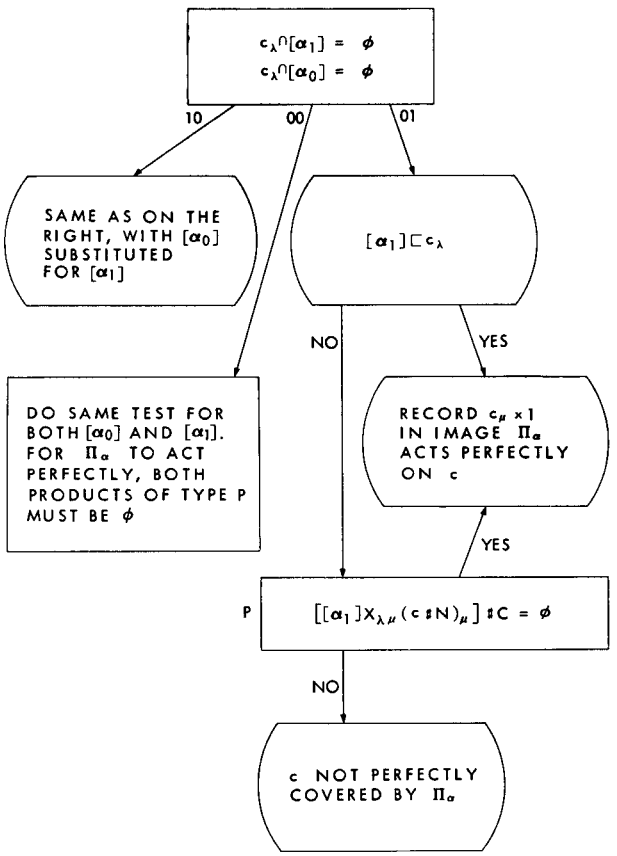

Figure 7 **Flow chart for test whether Π_α acts perfectly on a cube $c \in C$.**

4.6 *Remark.* In the operation of the over-all algorithm to find a minimum Boolean tree, described in Section 8, I visualize that the above #-construction will first be used to determine whether or not a given Π-operator is $(K, K\#N)$-perfect by applying the test to each cube of a cover C of K (except, of course, to those cubes of C which are elements of N) until one c is found for which Π_α is not $(K, c\#N)$-perfect or until all elements of C are tested. As soon as a Π-operator is found not to be $(K, K\#N)$-perfect, it is dropped from consideration. On the other hand, if it is found to be $(K, K\#N)$-perfect, then the Griesmer-Karp method will be applied to ascertain the image of N under Π_α, to complete the computation of the Π-operation. Thus most of the time only the #-construction will be used.

*Attention is here restricted to the case when α is Boolean, $\alpha \in B^r$.

†This form of that test is due to William Boyle, IBM Data Systems Division, Poughkeepsie.

5. Projective words

Projective words are products of projection operators and correspond, under suitable restrictions, to multiple decompositions. In order to define these products suitably, however, it is necessary to extend the domain of the projection operator.

In Section 4.2 the Π-operator was defined as a mapping $\Pi: \mathbf{A}^r \times \mathbf{K}^t \times \mathbf{J}_t^{r,s} \to K^{s+1}$, with $r + s = t$. More generally let the domain of Π be the union

$$\mathfrak{D}' = \bigcup_{\substack{0<r \\ 0\leq s}} \mathbf{A}^r \times \mathbf{K}^t \times \mathbf{J}_t^{r,s}.$$

Thus the range of Π will be the union of all $\mathbf{K}^{s+1}$.

It is sufficient for the purposes of this paper to consider only pairs (λ, μ) of maps which are order-preserving, that is, if $i < j$ then $\lambda(i) < \lambda(j)$, and similarly for μ. Under this assumption the selection of the function λ uniquely determines μ. With this assumption, the domain $\mathfrak{D}'$ may be modified by changing the last factor $J_t^{r,s}$ to $J^r = \{\lambda | \lambda: I_r \to I_t,$ λ order-preserving$\}$. Let

$$\mathfrak{D} = \bigcup_{\lambda(r) \leq t} \mathbf{A}^r \times \mathbf{K}^t \times J^r$$

denote this new domain.

Now let $\alpha \in \mathbf{A}^r$ and $\lambda \in J^r$ be considered fixed. Then $\Pi(\alpha, K, \lambda)$ becomes a function $\Pi_{\alpha\lambda}$ of K alone. Thus $\Pi_{\alpha\lambda}$ defines a mapping

$$\Pi_{\alpha\lambda}: \cup \mathbf{K}^t \to \cup \mathbf{K}^u, \quad \lambda(r) \leq t, \quad \lambda(r) - r + 1 \leq u,$$

with the property $\Pi_{\alpha\lambda}|\mathbf{K}^t \subset \mathbf{K}^{t-r+1}$. To simplify notation let $\mathbf{K} = \bigcup_t \mathbf{K}^t$ and let $\Pi_{\alpha\lambda}$ be extended over $\mathbf{K}$ by the agreement that for all $K \in K^t$ for $t < \lambda(r)$, $\Pi_{\alpha\lambda}K = \phi$. Let $\alpha \in \mathbf{A}^r$, $\lambda \in J^r$, $\beta \in \mathbf{A}^s$, $\omega \in J^s$, with $\lambda I^r \cap \omega I^s = \phi$. The *product* $\Pi_{\alpha\lambda} \cdot \Pi_{\beta\omega}$ can then be defined as $\Pi_{\beta\omega}$-followed-by-$\Pi_{\alpha\lambda}$, with its domain restricted to the range of $\Pi_{\beta\omega}$. A *projective word* is a product of projection operators. Strictly speaking, however, the product, $\Pi_{\alpha\lambda}\Pi_{\beta\mu}$, actually means $\Pi_{\beta\mu}$, followed by $\Pi_{\alpha\lambda}$, restricted to the range of $\Pi_{\beta\mu}$. As with the projection operator, the domain and range of a projective word is $\mathbf{K}$.

By the way of motivation, since both the domain and range of a projection operator is now $\mathbf{K}$, the set of all cubical complexes, we are now set up to iterate these maps. Thus the result of the application of a Π-operator on a complex is again a complex, which means that the new factorization problem comes out in the same general format as the old.

In dealing with projection operators, particularly as factors in a projective word, it is convenient to designate the "new" coordinate introduced; for example, in the definition of the Π-operator in 4.2 this new coordinate is $s + 1$. In Roth-Wagner [15] this new coordinate was used as a superscript and termed the "*output coordinate.*" Thus, for example, if $\lambda(1) = 1, \cdots, \lambda(r) = r$ then the operator $\Pi(\alpha, K, \lambda)$ might be written as

$$\Pi^{s+1}_{\alpha 12 \cdots r}$$

The subscripts $1, 2, \cdots, r$ are termed *input coordinates.* A projective word

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

shall be termed *acyclic* if it satisfies the condition:

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

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

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: (a) 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, and (b) $y(i) \notin A_{i-1}$.

For the remainder of the paper the term projective word will always be understood to mean acyclic projective word. It is said to be *tree-like* (Roth-Wagner) if in addition to condition (1) it also satisfies the second condition:

(2) 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 one-to-one correspondence is established in Roth-Wagner [15] between equivalence classes of Boolean trees and equivalence classes of projective words.

6. Injective words

Injection operators and their products, injective words, to be defined in this section, are used principally for proofs and certain constructions. They are directly useful, however, in analyzing logical designs. In particular, they have been fruitful in the theory of diagnosis of machine failures. They are types of inverses of projection operators and projective words.

6.1 Let λ, μ, ν be one-to-one mappings of I_r, I_{s-1}, I_1 respectively into I_t, for $r + s < t$, with pairwise disjoint images. Actually λ, μ, ν are considered as a *set* of maps into I_t, one set for each t. Furthermore, it is convenient to consider the domain of λ as being a subset I'_s of I_t, of cardinality s, rather than I_s, and similarly for μ and ν. Let F denote the set of all such maps. The *injection operator* is a map

$$\Pi^*: \mathbf{A}^r \times \mathbf{K}^s \times F \to \mathbf{K}^t$$

defined as follows: for $\alpha = (\alpha_0, \alpha_1) \in \mathbf{A}^r$, $K \in \mathbf{K}^s$, $(\lambda, \mu, \nu) \in F$,

$$\Pi^*(\alpha, K, (\lambda, \mu, \nu)) \supset c \times_{\lambda\mu} \alpha_0 \quad \text{if} \quad c \times_{\nu\mu} 0 \in K$$

$$\Pi^*(\alpha, K, (\lambda, \mu, \nu)) \supset c' \times_{\lambda\mu} \alpha_1 \quad \text{if} \quad c' \times_{\nu\mu} 1 \in K$$

$$\Pi^*(\alpha, K, (\lambda, \mu, \nu)) \supset c'' \times_{\lambda\mu} x^r \quad \text{if} \quad c'' \times_{\nu\mu} x \in K,$$

where $x^r = xx \cdots x$, rx's. From the definition it follows that the image of such a triple is a cubical complex of $\mathbf{K}^t$.

In the use of Π^*, usually μ will be determined by λ and ν; μ will usually be an order-preserving map of I_{s-1} into the complement of $\lambda I_r \cup \nu I_1$.

Let the arguments α and (λ, μ, ν) be fixed: let

$$\Pi^{\alpha\lambda}_{\nu}: \mathbf{K}^s \to \mathbf{K}^t$$

denote the corresponding mapping. *Par un abus de langage,* $\Pi^{\alpha\lambda}_{\nu}$ will be referred to as the injection operator. If K is Boolean, then so is $\Pi^{\alpha\lambda}_{\nu}(K)$.

Since the elements λ, μ, ν of F were considered as sets of mappings, one for each t, the domain and

range of $\Pi_{\nu}^{\alpha\lambda}$ may be extended over $\mathbf{K} = \bigcup_t \mathbf{K}^t$, $\Pi_{\nu}^{\alpha\lambda}: \mathbf{K} \to \mathbf{K}$.

6.2 *Example.* Let $\alpha \in \mathbf{K}^2$, with

$$\alpha_0 = \begin{array}{c} \begin{matrix} 1 & 2 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array} \qquad \alpha_1 = \begin{array}{c} \begin{matrix} 1 & 2 \end{matrix} \\ \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \end{array}$$

Let K be a complex of $\mathbf{K}^4$, prescribed by the rows c_1, c_2, c_3 of the following array:

	1	2	3	5
c_1	1	x	x	0
c_2	0	x	1	x
c_3	x	x	0	0

Let $\nu(1) = 3$, $\lambda(1) = 6$, $\lambda(2) = 7$. Finally, with μ determined by λ and ν, $\mu(1) = 1$, $\mu(2) = 2$, $\mu(3) = 5$. Then $\Pi_{\nu}^{\alpha\lambda}(K)$ is given by the cover consisting of the rows of the following matrix:

	1	2	5	6	7
	1	x	0	x	x
	0	x	x	1	0
	0	x	x	0	1
	x	x	0	0	0
	x	x	0	1	1

6.3 *Proposition.*

$$\Pi_{\alpha\lambda}^{\nu}(\Pi_{\nu}^{\alpha\lambda}(K)) = K, \quad \text{and} \quad \Pi_{\nu}^{\alpha\lambda}(\Pi_{\alpha\lambda}^{\nu}(K)) \subset K.$$

6.4 *Injective word.* The product of injection operators $\Pi_{\nu}^{\alpha\lambda}$, $\Pi_{\sigma}^{\beta\omega}$, if defined, is termed an *injective word* and is of course a map of $\mathbf{K}$ into $\mathbf{K}$. A product of several injection operators is also termed an *injective word.* There is a clear one-to-one correspondence between projective and injective words, obtained by interchanging the roles of subscripts and superscripts.

6.5 The definitions of the properties *acyclic* and *tree-like* for injective words are defined as for projective words, with the roles of superscripts and subscripts interchanged.

7. The singular and nonsingular complex of a tree-like word

7.1 In Roth-Wagner [15] it was shown how a Boolean tree defines a nonsingular cubical complex $\mathbf{K}(T)$. Let Π^* be a tree-like injective word. Let [1] denote the cubical complex consisting of the single vertex 1. The singular complex $s(\Pi^*)$ is defined to be $\Pi^*([1])$, the image of Π^* acting on [1]. This complex is singular if some of the "initial" coordinates of Π^* (see Roth-Wagner) are repeated. The degeneracy map ψ is defined by the identifications, if any, among the input labels of Π^*. If Π is the projective word dual to Π^*, then its singular complex $s\Pi$ shall also be $\Pi^*([1])$.

7.2 A tree-like injective or projective word Π^* or Π define also a nonsingular complex, by application of the consistency operator $\mathfrak{C}$,

$$\mathbf{K}(\Pi^*) = \mathfrak{C}s(\Pi^*)$$

$$\mathbf{K}(\Pi) = \mathfrak{C}s(\Pi).$$

7.3 *Proposition*: Let T be the Boolean tree of a tree-like projective word Π. Then $\mathbf{K}(T) = \mathbf{K}(\Pi)$. Similarly for an injective word Π^*.

7.4 *Proposition*: If Π is (K, L)-perfect and $\Pi(K) = [1]$ then $\Pi^*([1]) \supset L$ and conversely.

7.5 It is considerably more advantageous, especially from a programming point of view, however, to compute the nonsingular complex of a Boolean tree by the following procedure,* which in effect removes redundancies and inconsistencies as soon as they arise. For an injection operator $\Pi_c^{\alpha a_1 \cdots a_r}$, let $P_c^{\alpha a_1 \cdots a_r}$ denote the operation of $\Pi_c^{\alpha a_1 \cdots a_r}$ followed by the consistency operator $\mathfrak{C}$ followed by the subsuming operation $\sum$:

$$P_c^{\alpha a_1 \cdots a_r} = \sum \mathfrak{C}\Pi_c^{\alpha a_1 \cdots a_r}.$$

For Π^* an injective word, let P^* be the mapping derived from Π^* by replacing each one of its factors $\Pi_c^{\alpha a_1 \cdots a_r}$ by the appropriate $P_c^{\alpha a_1 \cdots a_r}$.

Proposition: $\mathbf{K}(\Pi^*) = P^*([1])$.

Figure 8 exhibits this method as applied to the Boolean tree of Fig. 3, and is to be compared with the more lengthy table of Fig. 4.

8. The algorithm

8.1 First a precise statement of the problem will be given. Let K be a nonsingular cubical complex and L a subcomplex. The problem is to find, in the class of all Boolean trees T constructed using primitive components from $\mathfrak{B}$ such that $K \supset \mathbf{K}(T) \supset L$, one of minimum cost.

*Suggested independently also by Miss Ruth E. Norby, IBM Research Laboratory, and Mr. Charles Stieglitz, IBM General Products Division Development Laboratory.

a	b	c	d	e	f	g	h	i	
								1	
					0		1		P_i^{vfh}
					1		0		
x	0	0					1		P_f^{uabc}
0	x	0					1		
0	0	x					1		
x	1	1					0		
1	x	1					0		
1	1	x					0		
x	0	0	1			1			$P_h^{udgc_1}$
0	x	0	1			1			
0	0	x	1			1			
0	0	1	x			1			
0	0	1	1			x			
x	1	1	0			0			
1	x	1	0			0			
1	1	x	0			0			
1	1	0	x			0			
1	1	0	0			x			
x	0	0	1	1					$P_g^{vcb_1}$
0	1	0	1	0					
0	0	x	1	1					
0	0	1	x	1					
0	0	1	1	x					
0	1	1	0	1					
x	0	1	0	0					
1	1	x	0	1					
1	1	0	x	1					
1	1	0	0	x					

Figure 8 **The "P-operations" for Figure 3**

8.2 The algorithm will now be described. Instead of dealing with the subcomplex L of K, it is more convenient to deal with its complement in K,

$$K\#L = M.$$

Thus, one seeks a Boolean tree T, of minimum cost, such that $K \supset \mathbf{K}(T) \supset K\#M$.

Let C be a cover of K and D a cover of M.

1° The first step is to determine whether or not K may be embedded in a lower dimensional cube, that is, whether or not any of its coordinates are redundant. Coordinate j is redundant if and only if for each cube c of a cover C of K the product $c'\#C = \phi$, where c' is the cube obtained from c by changing its j^{th} coordinate to x.

2° Assume then that no coordinate of K is redundant. Select a projective word $\Pi_{\alpha\lambda}$. Determine whether $\Pi_{\alpha\lambda}$ is $(K, K\#M)$-perfect, by use of the #-construction of Section 4.

If $\Pi_{\alpha\lambda}$ is $(K, K\#M)$-perfect, then use the Griesmer-Karp construction of the Π-operation on M to find the total images of K and M under $\Pi_{\alpha\lambda}$. Let K_2, M_2 denote these images, $K_2 = \Pi_{\alpha\lambda}(K)$, $M_2 = \Pi_{\alpha\lambda}(M)$: note carefully that M_2 is the image of $\Pi_{\alpha\lambda}$ *restricted* to M. This procedure is then repeated, with K_2, M_2 substituted for K, M.

If $\Pi_{\alpha\lambda}$ is not $(K, K\#M)$-perfect another projective word $\Pi_{\beta\nu}$ is tried on $(K, K\#M)$.

This procedure is repeated until either K has been reduced to a complex in $\mathbf{K}^1$ (i.e., until the image complex has only one coordinate) or until, in effect, all projective words have been tried on K. Of course, it is not necessary to try all projective words since, for example, if Π is the product of two projective words, $\Pi = \Pi_2 \cdot \Pi_1$, and if Π_1 is not $(K, K\#M)$-perfect, then, of course, neither is Π. That is, if $\Pi_{\alpha\lambda}$ is found to be not $(K, K\#M)$-perfect then *no* tree with α, λ selected will "work"—one trial stands for millions.

Consider first the case when the cost of any primitive element of $\mathfrak{B}$ is one less than the number of its inputs. Then if a projective word Π is found which is $(K, K\#M)$-perfect and reduces K to a complex of $\mathbf{K}^1$, i.e., to a complex of order 1, then this projective word, and its corresponding Boolean tree, is minimal. (The proof is given below.)

3° Suppose then that no projection operator is $(K, K\#M)$-perfect. One next constructs complete singular complexes $s(K, \psi)$ of degeneracy 1. Let S_1 denote the complete singular complex defined by K and ψ. Let R_1 denote the complex formed by the union of $s(M, \psi)$ and $N(\psi)$.

One then seeks, as in step 2°, a projective word Π which is (1) $(S_1, S_1\#R_1)$-perfect and (2) reduces S_1 to a complex with only one coordinate, that is, to a complex in $\mathbf{K}^1$. If no satisfactory Boolean tree had been found in step 1° and, as stated above, the cost of any primitive element of $\mathfrak{B}$ is its number of inputs, less one, then a projective word which does satisfy conditions (1) and (2) is a minimum. (Proof below).

All singular complexes of degeneracy 1 are investigated until either they are exhausted or until one is found for which a projective word satisfies the two requirements.

If none exist, one investigates singular complexes $s(K, \psi)$ of degeneracy 2 in similar fashion, et cetera, until a smallest degeneracy d is reached for which a "satisfactory" complex and projective word are found. The proof of the effectiveness of the algorithm is given by the following theorem.

8.3 *Theorem.* Let K be a nonsingular cubical complex and M a subcomplex. Let the cost of a component of $\mathfrak{B}$ be its number of inputs, less one. Let S_p, R_p be complete singular complexes defined by K, M and degeneracy mapping ψ, of degeneracy p. Let Π be a projective word which is $(S_p, S_p\#R_p)$-

perfect and for which $\Pi(S_p) = [1]$ and let p be the smallest degeneracy for which such a Π exists. Then Π defines a Boolean tree T of minimum cost and satisfies the requirement

$$K\#M \subset \mathbf{K}(\Pi) \subset K.$$

Proof. First it will be established that if Π is $(S_p, S_p\#R_p)$-perfect, and $\Pi(S_p) = [1]$ then it satisfies the condition (1) as above. Now for Π to be $(S_p, S_p\#R_p)$-perfect implies by Proposition 7.4, that

$$S_p\#R_p \subset \Pi^*([1]) = s(\Pi).$$

Hence, by the propositions of 3.8

$$K\#M = \mathcal{C}(S_p)\#\mathcal{C}(R_p) = \mathcal{C}(S_p\#R_p) \subset \mathcal{C}s(\Pi) = \mathbf{K}(\Pi).$$

Hence

$$K\#M \subset \mathbf{K}(\Pi).$$

On the other hand, since

$$[1] = \Pi(S_p),$$

by Proposition 6.2,

$$s(\Pi) = \Pi^*([1]) = \Pi^*(\Pi S_p) \subset S_p.$$

Hence

$$\mathbf{K}(\Pi) = \mathcal{C}s(\Pi) \subset \mathcal{C}S_p = K,$$

so that the above conditions are indeed satisfied. The only fact remaining to be established is that no other Π' satisfying these conditions has lower cost. But this is not possible since the singular complex corresponding to Π' would have then lower degeneracy than the degeneracy d of Π, contrary to hypothesis that d was the smallest such. *Q.E.D.*

Next remains the case when the cost of a primitive element of $\mathfrak{B}$ is *not* the number of its inputs, less one.

8.4 *Case where cost is not the number of inputs, less one.* The procedure for constructing a minimum word must be augmented by the following step.

4° Let C be the cost of a tree which satisfies the specifications of the problem. Let g be the ratio, minimum over all elements of $\mathfrak{B}$, of the cost of a block divided by the number of its inputs less 1. Then one need search only for complexes whose degeneracy is $\leq [C/g] - n$ where n is the rank of the complex in which K is originally embedded. In Appendix *A*, R. M. Karp gives a best determination, based on Bellman's dynamic programming, for the necessary size of the degeneracy.

Appendix A—Optimum criterion for termination

The problem of determining when the algorithm of Section 8 terminates reduces to the following:

Let $\mathfrak{B}$ consist of elements $E_1, \cdots, E_n$, where E_i has cost c_i and has s_i inputs, $i = 1, \cdots, N$.

Problem: What is the largest number $O(k)$ of inputs a tree T of cost k may have, where T is constructed from $\mathfrak{B}$? $O(k)$ is computed by the following recurrence scheme, adapted from a method due to Bellman, cited in Dantzig [6]

$$O(k) = -\infty, \qquad k < 0$$

$$O(0) = 1$$

$$O(k) = \max_i [(s_i - 1) + O(k - c_i)],$$

$$k > 0, \qquad i = 1, \cdots, N.$$

Related is the determination of the minimum cost $C(m)$ a tree with m inputs may have; $C(m)$ is given by the following relations:

$$C(m) = -\infty, \qquad m < 0,$$

$$C(1) = 0,$$

$$C(m) = \min_i [c_i + C(m - (s_i - 1))], \qquad m > 0.$$

This method was communicated to the author by Dr. R. M. Karp.

Appendix B—Concerning the efficiency of the algorithm and of fast approximations

In the absence of experimental tests on the speed of the algorithm, in the form of program-computed problems, it is hazardous to make any definite statements as to the efficiency of an algorithm. (Hand computations, however, have been promising.) In its favor let it be said that: (1) The single test for whether or not a given projection operator is (K, L)-perfect is actually equivalent to a very large number of other tests: In essence it determines whether or not *any* tree exists which realizes the function and has α as one of its logical blocks with inputs λ; (2) the speed of the Griesmer-Karp construction depends essentially on the number of cubes of a cover and *not* on the number of vertices; (3) the inconsistencies due to adding redundant variables is expeditiously handled by the nonsense cubes $N(\psi)$.

Only a program will test the commercial effectiveness of the algorithm!

Past experience has indicated that it is easy to develop "fast" approximate methods to a given algorithm. (See Ewing, Roth, Wagner [7].) One mode of procedure for the present algorithm would be to proceed as follows: Try a single projective word on a particular set of coordinates. If it does not act (K, L)-perfectly, then add suitable redundant

coordinates to "force" it to act (S, M)-perfectly where S and M are the singular complexes derived by this additional redundancy. Proceed in this manner until a satisfactory Boolean tree has been so obtained. In particular, one might start out with rather large building blocks, large in the sense of being a function of a rather large number of variables. It is certain that experimental results will give indications as to effective modes of procedure of this type. The algorithm should be programmed in subroutine form so that each particular subalgorithm can be examined to develop appropriate approximations thereto.

Another development which will be reported elsewhere and which promises to add greatly to the speed of the algorithm is a generalization of Ashenhurst's procedure for detection of decompositions, to be precise: given complexes K and L with K containing L, the problem is to determine whether there exists any primitive α such that Π_α is (K, L)-perfect. The work involved in this test is roughly of the same order of magnitude as that which determine whether or not a projective word Π_α, with α specified, acts (K, L)-perfectly. In particular, suppose that the number of primitives, each of r variables, is of the order of 50. Such a test as here outlined would, of course, enormously improve the efficiency of the program.

Bibliography

1. Abhyankar, Shreeram, *IRE Transactions on Electronic Computers* **8**, 3-8 (1959).
2. Ashenhurst, Robert L., "The Decomposition of Switching Functions," *Proceedings of an International Symposium on the Theory of Switching*, April 2-5, 1957. The Annals of the Harvard Computation Laboratory XXIX, Harvard University Press, Cambridge, Massachusetts, 1959, pp. 74-116.
3. Bellman, R. (unpublished paper).
4. Curtis, H. Allen, *Journal for the Association for Computing Machinery* **6**, 245-258 (1959).
5. Curtis, H. Allen, *Journal for the Association of Computing Machinery* **6**, 538-547 (1959).
6. Dantzig, G. B. (1959), Draft of Chapter XXVI, "Discrete Variable Extremum Problems," from *Linear Programming and Extensions* (to appear).
7. Ewing A. C., Roth, J. P., and Wagner, E. G., "Algorithms for Logical Design," presented at the AIEE Fall General Meeting, Chicago, Illinois, October 11-16, 1959; IBM Research Report RC-133.
8. Griesmer, J. H., and Karp, R. M. (unpublished paper).
9. Hilbert, D., and Bernays, P., *Grundlagen der Matematik*, Berlin: Julius Springer, Berlin, vol. 1, 1934.
10. Markov, A. A., *Journal for the Association of Computing Machinery* **5**, 331-334 (1958).
11. Muller, D. E., *IRE Transactions on Electronic Computers* **EC-5**, No. 1, 15-19 (1956).
12. Quine, W. V., *American Mathematical Monthly*, **62**, 627-631 (1955).
13. Roth, J. Paul, *Transactions of American Mathematical Society*, **88**, 301-326 (1958). See also ECP 56-02, Institute for Advanced Study, April, 1956.
14. Roth, J. Paul, "Algebraic Topological Methods, in Synthesis," *Proceedings of an International Symposium on the Theory of Switching*, 2-5 April 1957. The Annals of the Harvard Computation Laboratory XXIX, Harvard University Press, Cambridge, Massachusetts, 1959, pp. 57-73. This paper earlier appeared as an IBM Research Report RC-11, June 1957.
15. Roth, J. P. and Wagner, E. G., *IBM Journal* **4**, 326-344 (1959).
16. Roth, J. Paul, "Synthesis of Logical Systems with Many Outputs" (To appear).

Received October 19, 1959.