Chapter 1

COMPLETE SETS OF LOGIC PRIMITIVES

AMAR MUKHOPADHYAY

I. INTRODUCTION 1
II. ITERATIVELY CLOSED SYSTEM OF FUNCTIONS 3
III. CHARACTERIZATION OF A WEAK COMPLETE SET OF LOGIC PRIMITIVES 5
IV. REDUCTION THEOREMS 7
V. THEOREM OF POST 13
VI. BASES AND SIMPLE BASES 15
VII. ALMOST COMPLETE SETS OF LOGIC PRIMITIVES 16
APPENDIX. PROOF OF THEOREM 7.1 21
REFERENCES 26

ABSTRACT. This chapter is concerned with the problems of building up arbitrarily complex combinational switching circuits by using an interconnection of a set of simpler combinational circuits called primitives. A new and simplified proof of Post's theorem on completeness is presented. Some of the recent results on simple bases by Shetopal and on almost complete sets of logic primitives by Kobayashi are also given.

I. INTRODUCTION

This chapter is concerned with the problem of building up arbitrarily complex combinational switching circuits by using an interconnection of a set of simpler combinational circuits called primitives. We assume that the circuit is to handle binary inputs $x_1, x_2, \ldots, x_n$ each of which can have values 1 or 0, and the output of the circuit is a logic function (switching function, Boolean function) $f(x_1, \ldots, x_n)$ of the inputs. To avoid logical inconsistencies, the primitives are assumed to be interconnected to form a well-formed combinational circuit (Burks and Wright, 1953) which is defined recursively as: (i) each
primitive is a logic circuit; (ii) if $N_1$ and $N_2$ are two distinct logic circuits, then identification of some of the inputs of $N_2$ with an output of $N_1$ is a logic circuit; (iii) identification of any number of inputs of a logic circuit which are not outputs results in a logic circuit; (iv) each input of the primitive is connected to some input $x_i$ ($1 \leq i \leq n$) of the circuit or to constant signals 1 or 0 or to the output of another primitive; the circuit has only one output which is not connected to any primitive input.

One should note that the interconnection rules do not allow closed feedback loops to exist within the circuit. Incorporation of closed feedback loops typically produces a sequential rather than a combinational circuit, but there are circuits which are combinational even with closed feedback loops. This problem will be treated in Chapter II.

(A set of primitives is said to be strong complete if any arbitrary logic function $f(x_1, \ldots, x_n)$ can be realized as the output of a logic circuit which contains a finite number of primitives from the given set and whose inputs are identified with the set of variables $x_1, \ldots, x_n$. The definition implies that, in particular, the constant functions $f(x_1, \ldots, x_n) = 1$ and $f(x_1, \ldots, x_n) = 0$ must also be producible.)

(If the requirement for the realizability of constant functions is removed and if it can be assumed that the constants 1 and 0 can be applied to the inputs whenever necessary, the set of primitives is said to be weak complete.)

Note that a strong complete set of primitives is weak complete but the converse is not necessarily true.

An example of a strong complete set of primitives is provided by the well-known canonic expansion of a logic function (Post, 1921; Shannon, 1949).

$$f(x_1, \ldots, x_n) = \sum \hat{x}_1 \hat{x}_2 \cdots \hat{x}_n f(\varphi_1, \varphi_2, \ldots, \varphi_n)$$

(1)

where $\hat{x}_i = x_i$ or $\bar{x}_i$ (the negation or complement of $x_i$), $\varphi_i = 1$ if $\hat{x}_i = x_i$, $\varphi_i = 0$ if $\hat{x}_i = \bar{x}_i$, $1 \leq i \leq n$, and $\sum$ denotes the extended logical sum operation. The canonic expansion says that the set of primitives consisting of the 2-input logical AND operation, the 2-input logical OR operation\(^1\), and the single input negation or NOT operation can be used to synthesize any arbitrary logic function including the constant functions 0 and 1 (since $x_i + \bar{x}_i = 1$ and $x_i \bar{x}_i = 0$) and, therefore, form a strong complete set of primitives. In fact, (AND, NOT) or (OR, NOT) form a strong complete set since OR can be synthesized using AND and NOT, and AND can be synthesized using

---

\(^1\) Since AND, OR, and EXCLUSIVE OR operations are associative, $k$-input AND operation, $k$-input OR operation, and $k$-input EXCLUSIVE OR operation for $k > 2$ can be realized using only 2-input AND, 2-input OR, and 2-input EXCLUSIVE OR operations, respectively,
OR and NOT as

\[ x_1 x_2 = (\overline{x_1} + \overline{x_2}) \]  \hspace{1cm} (2) \\
\[ x_1 + x_2 = \overline{x_1 x_2} \]  \hspace{1cm} (3)

An example of a weak complete set of logic primitives is provided by the well-known complement-free ring sum canonic expansion (Zhegalkin, 1927; Muller, 1954; Reed, 1954)

\[ f(x_1, \ldots, x_n) = \sum a_i x_{j_1} x_{j_2} \cdots x_{j_k} \]  \hspace{1cm} (4)

where \( a_i \) is 0 or 1, \( 0 \leq i \leq 2^n - 1 \), \( 0 \leq k \leq n \), \( j_i < j_{i+1} \) and the set of variables \( (x_{j_1}, x_{j_2}, \ldots, x_{j_k}) \) denotes a subset of \( k \) variables (this subset is the empty set when \( k = 0 \)) out of the input variables \( (x_1, \ldots, x_n) \) and \( \sum \) denotes the extended logical EXCLUSIVE OR operation. This expansion says the set of primitives consisting of the AND operation and the 2-input EXCLUSIVE OR operation form a weak complete set but not a strong complete set since the constant function \( a_0 = 1 \) cannot be realized using these operations.

Each primitive in the logic circuit may be looked upon as performing a mathematical operation on its inputs. Thus, the function of the entire circuit might be interpreted as performing a mathematical composition with operations corresponding to the primitives. Thus, completeness of logical primitives is mathematically equivalent to logical universality of the set of operations by which it is possible to represent any arbitrary two-valued function of a finite number of two-valued variables as a composition or well-formed formula involving the basic operations and the logic arguments \( x_1, \ldots, x_n \) and possibly the constant arguments 1 and 0. A complete solution to this problem was presented by Post (1941).

Although Post’s theorem has been known to the switching theorists by word of mouth, its proof never appeared in any switching theory textbooks. One of the reasons for this may be that Post’s original proof involves the development of a full hierarchy of an “iteratively closed system of two-valued mathematical logic” using a language and terminology which is not commonly used by switching theorists. The proof of Post’s theorem by Yablonskii (1958) is not readily available to the English-reading worker. In this chapter we propose to give a simplified proof of Post’s theorem on completeness. We shall then give some of the recent results on simple bases by Sestopal (1961) and on almost complete sets of logic primitives by Kobayashi (1967). The reference by Ibuki, Naemura, and Nozaki (1963) came to the author’s notice after this chapter was written.

II. ITERATIVELY CLOSED SYSTEM OF FUNCTIONS

Basic to the understanding of Post’s theorem is the concept of an “iteratively closed system of functions.” Following Post, we shall consider only a contracted system of functions in which all \( n \)-variable logic functions are
assumed to have arguments $x_1, \ldots, x_n$. Given some property $P$, a finite set of functions $F = \{f(x_1, \ldots, x_n)\}$ is said to be iteratively closed with respect to the property $P$ if any function obtained as a composition of functions in $F$ belongs in $F$ and hence preserves the property $P$. Some of the relevant properties of logic functions which we shall use quite often in later developments are:

PROPERTY 1 (P1) (Monotonicity). Let $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ be two Boolean $n$-tuples where $a_i$ and $b_i$ ($1 \leq i \leq n$) are binary constants 0 or 1. We write $A \leq B$ if for all $i$, $a_i \leq b_i$, where $0 \leq 0, 0 \leq 1$, and $1 \leq 1$. Let $f(A)$ denote the value of the function $f(x_1, \ldots, x_n)$ when $x_i = a_i$. Similarly, we define $f(B)$. The function $f(x_1, \ldots, x_n)$ is said to be monotonic if for all $A$ such that $f(A) = 1$, it is true that $f(B) = 1$ whenever $A \leq B$. Monotonic functions are more commonly known in switching theory literature as positive unate functions (see Chapter II).

PROPERTY 2 (P2) (Linearity). A function $f(x_1, \ldots, x_n)$ is said to be linear if its canonic expansion given by Eq. (4) has the form

$$f(x_1, \ldots, x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \cdots \oplus a_n x_n$$  \hspace{1cm} (5)

where $a_i$ ($0 \leq i \leq n$) is 0 or 1 and $\oplus$ denotes the 2-input EXCLUSIVE OR operation.

PROPERTY 3 (P3) (Self-Duality). A function $f(x_1, \ldots, x_n)$ is said to be self-dual if complementing its inputs $x_1, \ldots, x_n$ results in the complementary function, i.e.,

$$f(x_1, \ldots, x_n) = f(\bar{x}_1, \ldots, \bar{x}_n)$$  \hspace{1cm} (6)

PROPERTY 4 (P4) (Zero Preservation). A function $f(x_1, \ldots, x_n)$ is said to be a function preserving zero if

$$f(0, 0, \ldots, 0) = 0$$  \hspace{1cm} (7)

PROPERTY 5 (P5) (One Preservation). A function $f(x_1, \ldots, x_n)$ is said to be a function preserving one if

$$f(1, 1, \ldots, 1) = 1$$  \hspace{1cm} (8)
The proof of the following theorem will be left as an exercise.

**THEOREM 2.1.** The set of \( n \)-variable logic functions having the single property \( P_I \) (\( 1 \leq I \leq 5 \)) forms an iteratively closed system of functions with respect to \( P_I \).

Stated in other words, the above theorem says that a monotonic function of monotonic functions is a monotonic function, a linear function of linear functions is a linear function, etc. The classification of all 1-variable and 2-variable functions according to the Properties \( P_1 \) through \( P_5 \) is given in Table 1. In this table each function occupies a row and if it does not possess

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
<th>( P_4 )</th>
<th>( P_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\text{NOT} (\overline{x}_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>\text{AND} (x_1 \cdot x_2)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\text{OR} (x_1 + x_2)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\text{NAND} (\overline{x}_1 \cdot \overline{x}_2)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>\text{NOR} (\overline{x}_1 \cdot \overline{x}_2)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>\text{IMP} (x_1 \cdot x_2, \overline{x}_1 + x_2)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>\text{NIMP} (x_1 \cdot x_2, \overline{x}_1 \cdot x_2)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>\text{EXOR} (x_1 \oplus x_2)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>\text{EQUIV} (x_1 \oplus x_2, x_1 \oplus x_2)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

the property \( P_I \), 1 is entered in the \( P_I \) column position for the row; otherwise the entry is 0.

**III. CHARACTERIZATION OF A WEAK COMPLETE SET OF LOGIC PRIMITIVES**

In this section, the necessary and sufficient conditions for a set of logic primitives to be weak complete will be given. The following two theorems have been adopted from Glushkov (1963).
THEOREM 3.1. If the constant functions 0 and 1 are available, the operation of negation can be synthesized by means of any nonmonotonic logic function.

Proof: Let us define a relation \( A \leftarrow B \) between any two Boolean \( n \)-tuples \( A = (a_1, \ldots, a_n) \) and \( B = (b_1, \ldots, b_n) \) if there exists an index \( j (1 \leq j \leq n) \) such that \( a_j = 0 \) and \( b_j = 1 \) and \( a_i = b_i \) for all \( i \neq j, 1 \leq i \leq n \). Let \( f(x_1, \ldots, x_n) \) be nonmonotonic. This means that there exist Boolean \( n \)-tuples \( A \) and \( B \) where \( A \leq B \) such that \( f(A) = 1 \) and \( f(B) = 0 \). Thus, there must exist \( n \)-tuples \( A_0, A_1, A_2, \ldots, A_k \) such that \( A = A_0 \leftarrow A_1 \leftarrow A_2 \leftarrow \cdots \leftarrow A_k = B \) for some \( k, 1 \leq k \leq n \). Therefore for some \( t, 0 \leq t < k \), there must exist \( A_t \) and \( A_{t+1} \) such that \( f(A_t) = 1 \) and \( f(A_{t+1}) = 0 \). Since \( A_t \leftarrow A_{t+1} \), let \( A_t = (a_1^t, a_2^t, \ldots, a_{t-1}^t, 0, a_{t+1}^t, \ldots, a_n^t) \) and \( A_{t+1} = (a_1^t, a_2^t, \ldots, a_{t-1}^t, 1, a_{t+1}^t, \ldots, a_n^t) \) for some \( s, 1 \leq s \leq n \). This means that \( f(a_1^t, a_2^t, \ldots, a_{t-1}^t, x_s, a_{t+1}^t, \ldots, a_n^t) = \bar{x}_s \), which shows that negation can be synthesized.

THEOREM 3.2. If the constant functions 1 and 0 are available, the AND and OR operations can be synthesized by means of a nonlinear function.

Proof: Let \( f(x_1, \ldots, x_n) \) be a nonlinear function. Then the canonical expansion of \( f \) given by Eq. (4) must contain at least one term of the form \( x_i x_j p \), where \( 1 \leq i, j \leq n, i \neq j \) and \( p \) is a logical product of some of the variables from \( (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n) \) or \( p = 1 \). Let us pick up one of such terms containing the fewest number of variables in \( p \). Assign value 1 to all the variables in \( p \) so that \( p = 1 \) and assign value 0 to all of the remaining variables but \( x_i \) and \( x_j \). Then \( f \) is reduced to a function \( h(x_i, x_j) \) given by

\[
h(x_i, x_j) = a_0 \oplus a_i x_i \oplus a_j x_j \oplus x_i x_j
\]

(9)

The expressions for \( h \) for eight different assignments of values to \( a_0, a_i, \) and \( a_j \) are shown in Table II. If \( (a_0, a_i, a_j) = (0, 0, 0) \), we get an AND operation \( x_i x_j \). If \( (a_0, a_i, a_j) = (0, 1, 1) \), we get an OR operation \( x_i + x_j \). In all other cases, in view of the classification of Table I, \( h \) is nonmonotonic, so that negation can be synthesized by means of \( h \); thus it is possible to synthesize either the AND or the OR operation. Corollary 1 follows from the above proof.

COROLLARY 3.1. If the operation of negation is available, the operation of AND or OR can be synthesized by means of a 2-variable nonlinear function.
TABLE II
\[ h(x_i, x_j) \text{ for Different Values of } (a_0, a_i, a_j) \]

<table>
<thead>
<tr>
<th>(a_0)</th>
<th>(a_i)</th>
<th>(a_j)</th>
<th>(h(x_i, x_j))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(x_i x_j)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(x_i \oplus x_i x_j = \bar{x}_i x_j)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(x_i \oplus x_i x_j = x_i \bar{x}_j)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(x_i \oplus x_i \oplus x_i x_j = x_i + x_j)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(1 \oplus x_i x_j = \bar{x}_i + \bar{x}_j)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(1 \oplus x_i \oplus x_i x_j = x_i + \bar{x}_j)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(1 \oplus x_i \oplus x_i x_j = \bar{x}_i + x_j)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(1 \oplus x_i \oplus x_i x_j = \bar{x}_i \bar{x}_j)</td>
</tr>
</tbody>
</table>

**THEOREM 3.3** (Weak Completeness Theorem). A set of functions is weak complete if and only if the set contains at least one nonmonotonic function and at least one nonlinear function.

**Proof:** The necessity of the conditions follows from Theorem 2.1. Assuming the availability of constants 1 and 0, we can synthesize the negation operation by using the nonmonotonic operation (by Theorem 3.1) and either the AND or the OR operation by using the nonlinear operation (by Theorem 3.2) and hence from Eqs. (1)–(3), it follows that the set is weak complete.

**IV. REDUCTION THEOREMS**

In this section we shall derive a set of theorems which will give the effect of identifying (i.e., connecting together) some of the inputs of a primitive so as to reduce it to a primitive of a smaller number of variables with the preservation of certain properties. Theorem 4.2 is new and all other theorems have been obtained by Shestopal (1961), wherein proofs are either not given or given in the form of hints or sketches. We shall give complete proofs. These theorems will be used in Section V to prove the theorem of Post.

**THEOREM 4.1.** If \(f(x_1, \ldots, x_n)\) is a nonmonotonic function of \(n > 3\) variables, then it can be reduced to a nonmonotonic function of not more than three variables by identifying some of the inputs.
**Proof:** Since \( f(x_1, \ldots, x_n) \) is nonmonotonic, we can assume, without loss of generality, that there exist product terms \( p_1 = x_1 x_2 \cdots x_k \bar{x}_{k+1} \bar{x}_{k+2} \cdots \bar{x}_{n-1} x_n \) and \( p_2 = x_1 x_2 \cdots x_k \bar{x}_{k+1} \bar{x}_{k+2} \cdots \bar{x}_{n-1} x_n \), for some \( 0 \leq k \leq n-1 \), such that \( f = 0 \) when \( p_1 = 1 \) and \( f = 1 \) when \( p_2 = 1 \). If \( k > 0 \), identify \( x_1, \ldots, x_{k-1} \) and \( x_k \) with the variable \( u \) and \( x_{k+1}, \ldots, x_{n-2} \) and \( x_{n-1} \) with the variable \( v \). Then \( f(x_1, \ldots, x_n) \) is reduced to a 3-variable function \( F(u, v, x_n) \) which is nonmonotonic. If \( k = 0 \), identify \( x_1, \ldots, x_{n-2} \) and \( x_{n-1} \) with the variable \( u \). Then \( f(x_1, \ldots, x_n) \) is reduced to a 2-variable function \( G(u, x_n) \) which is also obviously nonmonotonic.

**Theorem 4.2.** All 3-variable non-self-dual nonlinear functions can be reduced by identification of variables to either 2-variable nonlinear functions or nontrivial 2-variable linear functions.

**Proof:** The theorem is proved by exhaustion. Table III gives a list of all

**Table III**

<table>
<thead>
<tr>
<th>((d_1, d_2, d_3))</th>
<th>NUMBER OF FUNCTIONS</th>
<th>REPRESENTATIVE FUNCTION</th>
<th>IDENTIFICATION OR COMMENT</th>
<th>THE REDUCED FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>((3, 3, 1))</td>
<td>2</td>
<td>(x_1 x_3, x_2 x_3, x_1 x_2, x_3 x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3, x_4)</td>
</tr>
<tr>
<td>((3, 3, 0))</td>
<td>2</td>
<td>(x_1 x_3, x_2 x_3, x_1 x_2, x_3 x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_2, x_3)</td>
</tr>
<tr>
<td>((3, 2, 1))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((3, 2, 0))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((3, 1, 1))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((3, 1, 0))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((3, 0, 1))</td>
<td>2</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((3, 0, 0))</td>
<td>2</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>Linear</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 3, 1))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 2, 0))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 2, 1))</td>
<td>12</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 1, 1))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 0, 1))</td>
<td>12</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>(x_2 = x_3)</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 0, 0))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>None</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 1, 0))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>None</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 0, 1))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>None</td>
<td>(x_1, x_2, x_3)</td>
</tr>
<tr>
<td>((2, 0, 0))</td>
<td>6</td>
<td>(x_1 x_2 x_3, x_1 x_2 x_3, x_1 x_2 x_3)</td>
<td>None</td>
<td>(x_1, x_2, x_3)</td>
</tr>
</tbody>
</table>
TABLE III (continued)

<table>
<thead>
<tr>
<th>(2, 0, 0)</th>
<th>6</th>
<th>$x_1 x_2$</th>
<th>Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 3, 2)</td>
<td>6</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_1 = x_2$</td>
</tr>
<tr>
<td>(1, 3, 0)</td>
<td>6</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_2 = x_3$</td>
</tr>
<tr>
<td>(1, 2, 1)</td>
<td>12</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_1 = x_2$</td>
</tr>
<tr>
<td>(1, 2, 0)</td>
<td>6</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_2 = x_3$</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>12</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_1 = x_2$</td>
</tr>
<tr>
<td>(1, 1, 0)</td>
<td>6</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>None</td>
</tr>
<tr>
<td>(1, 0, 1)</td>
<td>6</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_1 = x_3$</td>
</tr>
<tr>
<td>(0, 2, 1)</td>
<td>2</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_2 = x_3$</td>
</tr>
<tr>
<td>(0, 2, 0)</td>
<td>2</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>Self-dual</td>
</tr>
<tr>
<td>(0, 1, 1)</td>
<td>6</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_2 = x_3$</td>
</tr>
<tr>
<td>(0, 1, 0)</td>
<td>6</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>$x_1 = x_2$</td>
</tr>
<tr>
<td>(0, 0, 1)</td>
<td>2</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>None</td>
</tr>
<tr>
<td>(0, 0, 0)</td>
<td>2</td>
<td>$x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$</td>
<td>Linear</td>
</tr>
</tbody>
</table>

*a* When the function is self-dual, all identifications lead to the single-variable function $x_1$ or $\bar{x}_1$.

Three variable functions along with self-dual and linear functions. A class of functions in this table is designated by a triplet $(d_1, d_2, d_3)$ in the first column of the table where $d_1$, $d_2$, and $d_3$ denote that any function in this class will have $d_1$ terms which are single variable ($0 \leq d_1 \leq 3$), $d_2$ terms which are products of two variables ($0 \leq d_2 \leq 3$), and $d_3$ terms which are products of three variables ($0 \leq d_3 \leq 1$) in its canonical expansion of Eq. (4). Column three of this table gives a representative function in this class and any function which can be obtained from the representative function by permutation of variables or by function complementation will have the same triplet $(d_1, d_2, d_3)$. Note that the triplet $(d_1, d_2, d_3)$ is not a unique set of invariants for the class. In some cases, two classes have the same triplet and they are designated as $(d_1, d_2, d_3)_1$ and $(d_1, d_2, d_3)_2$. The second column gives the

2 The problem of classification of switching functions will be treated in further depth in Chapters IV and V.
number of functions in the class. The fourth column gives one of the possible identifications of the variables which reduce the function according to the theorem or contains a remark. The resulting reduced functions are shown in the fifth column. Note that the nonlinear function whose class designations are $(2, 3, 0)$ and $(0, 3, 0)$ are not reducible according to the theorem and the functions in these classes are self-dual. All other nonlinear 3-variable functions are non-self-dual. Thus, the theorem is proved.

**THEOREM 4.3.** All 4-variable nonlinear functions can be reduced to nonlinear functions of three variables by identification of inputs.

**Proof:** Consider a 4-variable nonlinear function $f(x_1, x_2, x_3, x_4)$. Let us express $f$ as

$$f(x_1, x_2, x_3, x_4) = x_4 g(x_1, x_2, x_3) \oplus h(x_1, x_2, x_3) \quad (10)$$

where

$$g(x_1, x_2, x_3) = f(x_1, x_2, x_3, 1) \oplus f(x_1, x_2, x_3, 0) \quad (11)$$

$$h(x_1, x_2, x_3) = f(x_1, x_2, x_3, 0) \quad (12)$$

First, assume $g = 0$. Then $h$ must itself be nonlinear and $f$ is trivially a 3-variable nonlinear function. Then, assume $g = 1$. Since $f$ is nonlinear, $h$ must be nonlinear and identification of $x_4$ with any one of the variables $x_1, x_2,$ or $x_3$ keeps the resulting 3-variable function nonlinear. Now, assume $g$ is not 0 or 1. Let us identify some of the variables in $(x_1, x_2, x_3)$ such that $g$ is reduced to either a nonlinear function of two variables or a nontrivial linear function of two variables or a single variable or complement of a single variable. If $g$ is nonlinear, such a reduction is always possible as is shown in Table III. Also, if $g$ is nonconstant linear, such a reduction is obviously possible. Since $h$ does not involve the variable $x_4$, such an identification would reduce $f$ to a 3-variable nonlinear function because of the presence of the term $x_4 g$ independent of the effect of this identification on the function $h$.

**THEOREM 4.4.** Nonlinear functions of $n > 3$ variables can be reduced to nonlinear functions of not more than three variables by identification of inputs.

**Proof:** Theorem 4.3 is a special case of this theorem for $n = 4$. The theorem will now be proved by induction. Assume that all nonlinear functions of $k > 3$ variables can be reduced to nonlinear functions of not more than $(k - 1)$
variables by identifying inputs. We then prove that all nonlinear functions of 
\((k + 1)\) variables can be reduced to nonlinear functions of not more than \(k\) 
variables by identifying inputs. We have
\[
f(x_1, \ldots, x_{k+1}) = x_{k+1}g(x_1, \ldots, x_k) \oplus h(x_1, \ldots, x_k)
\]  
(13)

where
\[
g = f(x_1, \ldots, x_k, 1) \oplus f(x_1, \ldots, x_k, 0)
\]  
(14)
\[
h = f(x_1, \ldots, x_k, 0)
\]  
(15)

First assume \(g\) is linear. If \(g = 0\), \(h\) must be a nonlinear function of \(k\) variables 
since \(f\) is nonlinear. If \(g = 1\), identification of \(x_{k+1}\) with any one of the vari-
ables \((x_1, \ldots, x_k)\) will reduce \(f\) to a nonlinear function of \(k\) variables. If \(g\) is 
a nonconstant linear function, we can always identify some variables in 
\((x_1, \ldots, x_k)\) such that \(g\) becomes a linear function of less than \(k\) variables. 
Thus \(x_{k+1}g\) becomes a nonlinear function of less than \((k + 1)\) variables. 
Since \(x_{k+1}\) does not occur in \(h\), \(f\) becomes a nonlinear function of less than 
\((k + 1)\) variables independent of the effect of the identification on \(h\). Now, 
suppose \(g\) is nonlinear. By induction hypothesis, \(g\) can be reduced to a nonlinear 
function of not more than \((k - 1)\) variables. Again, \(x_{k+1}g\) becomes a nonlinear function of at most \(k\) variables and since \(x_{k+1}\) does not occur in \(h\), 
\(f\) becomes a nonlinear function of not more than \(k\) variables.

**Theorem 4.5.** If \(f(x_1, \ldots, x_n)\) is a non-self-dual function of \(n > 2\) 
variables, then it can be reduced to a non-self-dual function of two variables 
by identifying its inputs.

**Proof:** Since \(f\) is non-self-dual, there exist two fundamental products 
\(p_1 = \bar{x}_1\bar{x}_2 \cdots \bar{x}_k\) and \(p_2 = \bar{x}_1\bar{x}_2 \cdots \bar{x}_n\) where \(\bar{x}_i\) is \(x_i\) or \(\bar{x}_i\) such that if \(f = 1\) 
when \(p_1 = 1\), \(f\) is also 1 when \(p_2 = 1\) or if \(f = 0\) when \(p_1 = 1\), \(f\) is also 0 when 
\(p_2 = 1\). Assume first that \(f = 1\). Without loss of generality, let \(p_1 = x_1x_2 \cdots x_k\) 
\(\bar{x}_{k+1}\bar{x}_{k+2} \cdots \bar{x}_n\) where \(1 \leq k \leq n\). Identify \(x_1, x_2, \ldots, x_k\) to \(X\) and \(x_{k+1}, \ldots, x_n\) to \(Y\). Then \(f(x_1, \ldots, x_n)\) is transformed to a function \(F(X, Y)\) whose truth 
table looks like

<table>
<thead>
<tr>
<th>(X)</th>
<th>(Y)</th>
<th>(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>(\phi_1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(\phi_2)</td>
</tr>
</tbody>
</table>
where $\varphi_1$ and $\varphi_2$ are each 0 or 1. When $\varphi_1 = \varphi_2 = 1$, $F(X, Y) = 1$, which is a non-self-dual function. For all other values of $\varphi_1$ and $\varphi_2$, $F(X, Y)$ is reduced to a nontrivial 2-variable function and hence a non-self-dual function (see Table I). For the special case when $p_1 = x_1 x_2 \cdots x_n$, identify the first $k$ variables $x_1, x_2, \ldots, x_k$ ($1 \leq k < n$) with $X$ and $x_{k+1}, x_{k+2}, \ldots, x_n$ with $Y$. Then $f(x_1, \ldots, x_n)$ is transformed to a function $G(X, Y)$ whose truth table is

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\varphi_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$\varphi_1$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Again, if both $\varphi_1$ and $\varphi_2$ are 1, $G = 1$. For all other values of $\varphi_1$ and $\varphi_2$, $G$ is a nontrivial 2-variable function and hence a non-self-dual function. In a similar way, it can be proved that $f$ can be reduced to a 2-variable non-self-dual function, assuming $f = 0$ for both $p_1$ and $p_2$.

**Theorem 4.6.** A nonzero (one) preserving function $f(x_1, \ldots, x_n)$ of $n > 1$ variables remains a nonzero (one) preserving function under any identification of inputs.

The proof of this theorem is simple and is left as an exercise.

Before concluding this section, we shall prove another interesting theorem attributed to Yablonskii (1958). This theorem and its corollary will also be used to prove the theorem of Post in the next section.

**Theorem 4.7.** Every operation that does not preserve zero also either does not preserve one or is non-self-dual.

**Proof:** Suppose $f(x_1, \ldots, x_n)$ does not preserve zero and does preserve one, i.e., $f(0, 0, \ldots, 0) = 1$ and $f(1, 1, \ldots, 1) = 1$, which means that $f$ is non-self-dual since complementing the inputs does not complement the function value. Next, suppose that $f(x_1, \ldots, x_n)$ does not preserve zero and is self-dual. Then, $f(0, 0, \ldots, 0) = 1$ and $f(1, 1, \ldots, 1) = 0$, which implies that $f$ does not preserve one.

**Corollary 4.1.** The operation of negation or the constant function 1 can be synthesized from a nonzero preserving operation.
Proof: Let \( f(x_1, \ldots, x_n) \) be the nonzero preserving operation. Identify all its inputs to the single input \( x_1 \). If \( f \) is nonzero preserving and also non-one preserving, then the output function after identification is obviously \( \bar{x}_1 \). If \( f \) is nonzero preserving and non-self-dual, it means that \( f(0, 0, \ldots, 0) = 1 \) and \( f(1, 1, \ldots, 1) = 1 \), so that the output is a constant function 1 after identification.

V. THEOREM OF POST

THEOREM 5.1. A set of functions is strong complete if and only if it contains (1) at least one function not zero preserving, (2) at least one function not one preserving, (3) at least one non-self-dual function, (4) at least one nonlinear function, (5) at least one nonmonotonic function.

Proof: The necessity of these conditions follows from Theorem 2.1. To prove sufficiency, we shall show constructively that it is possible to synthesize the negation operation and either the AND or OR operation if the conditions of the theorem are satisfied.

Let us consider the primitive which is nonzero preserving. We have to consider two cases according to Theorem 4.7.

Case A: The nonzero preserving primitive is also non-self-dual. By Corollary 4.1, we can synthesize the constant function 1 out of this primitive by identifying all its inputs. This constant 1 function is then connected to all the inputs of the nonone preserving primitive of the set yielding a constant 0 output. Thus, the constant functions 1 and 0 have been produced. Now, we can use the nonmonotonic primitive to synthesize the negation operation (Theorem 3.1) and the nonlinear primitive to synthesize either the AND or the OR operation (Theorem 3.2). Hence the given set of primitives is strong complete.

Case B: The nonzero preserving primitive is also nonpreserving one. The proof in this case should proceed in the following steps.

Step 1. By identifying all the inputs of the primitive, the negation operation can be synthesized (Corollary 4.1).

Step 2. Consider now the nonlinear primitive. This primitive must produce a function of more than one variable since all single variable functions are linear. If this is a 2-variable function, we can synthesize either the AND or the OR operation since the negation operation has already been synthesized in Step 1 (Corollary 3.1). The constant functions 1 and 0 can be synthesized by using the relations \( 1 = x_1 + \bar{x}_1 \) and \( 0 = x_1 \bar{x}_1 \), respectively.
Step 3. Now, suppose the nonlinear primitive produces a 3-variable function. We have to consider three subcases according to Theorem 4.2.

Subcase 1: The 3-variable nonlinear function can be reduced to a 2-variable nonlinear function by identification of appropriate inputs. We can then synthesize either AND or OR as in Step 2.

Subcase 2: The 3-variable nonlinear function can be reduced to a nontrivial 2-variable linear function (see the classes (3, 3, 0) and (1, 3, 0) in Table III). With the use of the negation operation synthesized in Step 1, we can now obtain either 1 or 0 functions since $1 = x_i \ominus \overline{x_i}$ and $0 = x_i \circ \overline{x_i}$. Applying the negation operation again, we can obtain $0(1)$ from $1(0)$, and hence both the constant functions have been obtained. We can now use another copy of the original nonlinear primitive and synthesize the AND or OR operations (Theorem 3.2).

Subcase 3: The 3-variable nonlinear function is self-dual and is not reducible as in Subcases 1 and 2 (see classes (2, 3, 0) and (0, 3, 0) in Table III). We need help from the non-self-dual primitive in the set which we have not used so far. If this non-self-dual primitive produces a function of $n > 2$ variables, we reduce it such that the function is a 2-variable non-self-dual function by identifying inputs (Theorem 4.5). We have to consider three sub-subcases: (a) The reduced non-self-dual function is a constant function $l(0)$. We use the negation synthesized in Step 1 and then use another copy of the original nonlinear function in the set to synthesize either the AND or OR operation. (b) The reduced non-self-dual function is a nontrivial 2-variable linear function. We proceed as in Subcase 2 in Step 3 to synthesize 1 and 0 and either AND or OR. (c) The reduced non-self-dual function is a 2-variable nonlinear function. We proceed as in Step 2 to synthesize 1 and 0 and either the AND or OR operation.

Step 4. Now, suppose the nonlinear primitive in the set produces a nonlinear function of more than three variables. Then we reduce it to a nonlinear function of not more than three variables (Theorem 4.4). If the reduced function is a 2-variable nonlinear function, then we proceed as in Step 2. If the reduced function is a 3-variable nonlinear function, we reduce it further by identifying inputs such that the resulting function is: (i) A nonlinear 2-variable function. Then we proceed as in Step 2. (ii) A nontrivial linear function of 2 variables. Then we proceed as in Subcase 2 in Step 3. (iii) A self-dual function. If the original nonlinear function from which this self-dual function has been derived is also self-dual then we pick up the non-self-dual primitive in the set and proceed as in Subcase 3 in Step 3. But if the original nonlinear function is also non-self-dual, or if there is any other non-self-dual primitive in the set, we can work with either one of them as in Subcase 3 in Step 3 and synthesize constant functions 1 and 0 and either the AND or OR operation.

This exhausts all possible cases and the theorem is proved.
VI. BASES AND SIMPLE BASES

A strong (weak) complete set of logic primitives is called a strong (weak) basis if no proper subset of it forms a strong (weak) complete set. While it can be easily seen that the total number of bases of any kind is infinite, the maximum number of primitives in a basis is bounded. According to Theorems 4.7 and 5.1, a strong basis can contain no more than four primitives; also, since there are nonlinear monotonic functions the maximum number of primitives in a weak basis is two.

A basis is called a simple basis if it is impossible to replace some of the primitives in the set by functions obtained by identifying some of the inputs of one or more primitives of the basis without destroying the completeness property.

Given a property P, a function is said to be simple (Shostop, 1961; Kautz, 1966) with respect to P if any identification of its inputs destroys P for the function. Functions simple with respect to at least one of the properties mentioned in the theorem of Post (Theorem 5.1) will be called simple functions in this section. It follows from the definition that a simple basis can contain only simple functions.

It is interesting that the number of simple functions is finite, as a result of which it follows that the number of simple bases is also finite. To see this, we note the following:

1) From Theorem 4.1, functions simple with respect to nonmonotonicity must be nonmonotonic functions of at most three variables. By exhaustion, it can be shown that the simple nonmonotonic functions are: NOT, IMP, NIMP, EXOR, EQUIV, the 3-variable representative functions in Table III whose class designations are (3, 2, 0), (3, 1, 1), (3, 0, 0), (2, 3, 0), (2, 2, 1), (2, 2, 1), (2, 1, 0), (2, 0, 1), (1, 3, 1), (1, 2, 0), (1, 2, 0), (1, 1, 1), and the functions which can be obtained from these by permutation of inputs only.

2) From Theorem 4.4, functions simple with respect to nonlinearity 3 functions of at most three variables. These are: AND, OR, IMP, NIMP, NOR, NAND, and all the 3-variable functions in Table III whose class designations are (3, 3, 0), (2, 3, 0), (1, 3, 0), and (0, 3, 0).

3) From Theorem 4.5, functions simple with respect to non-self-duality are 0, 1, AND, OR, NAND, and NOR.

4) From Theorem 4.6, functions simple with respect to nonzero (one) preservation are 1 and \( \bar{x} \) (0 and \( \bar{x} \)).

To obtain all possible simple and weak bases it is only necessary to take all possible irredundant combinations of simple nonmonotonic and simple nonlinear functions. Similarly, to obtain all possible simple and strong bases
it is only necessary to take all possible irredundant combinations of simple functions—preserving the five properties of Theorem 5.1. Simple bases having single functions are NOR and NAND; one having a maximum of four primitives is \{0, 1, x_1, x_2, x_1 \oplus x_2 \oplus x_3\}. This basis was also mentioned by Loomis and Wyman (1965) as an example of a complete set.

EXERCISES

1. Determine the strong and weak basis consisting of logic primitives of at most two variables.

2. (Shestopal) Show that there are 48 simple bases; two bases consisting of one function, 22 bases consisting of two functions, 21 bases consisting of three functions, three consisting of four functions.

VII. ALMOST COMPLETE SETS OF LOGIC PRIMITIVES

Given a complete set, an arbitrarily complex combinational function may need potentially infinite number of copies of the primitives for its realization. Kobayashi (1967) posed the following interesting question: Given a set of primitives \( F = \{f_1, \ldots, f_p\} \) which is not complete, if a set of primitives \( G = \{g_1, g_2, \ldots, g_q\} \) can be added to \( F \), is it possible to realize any arbitrary function by using each of \( g_i \) a bounded number, say \( k_i \) times, and using each of \( f_j \) a potentially infinite number of times, if necessary? If this is possible, \( F \) will be called an almost complete set of logic primitives.

The problem of almost complete set of logic primitives seems to have originated from a more intriguing problem in switching theory, called the “minimal-NOT” problem. This problem will be discussed in great detail in Chapter II, but we will briefly mention the following: in Chapter II, Fig. 3 gives a circuit for complementing three inputs by using only two NOT operations. Note that the \( T \)-elements in this circuit can be replaced by circuits consisting of only AND and OR primitives. In an arbitrary combinational circuit using AND, OR, and NOT primitives, by applying this “three-NOT-to-two-NOT” transformation repeatedly, it should be possible to get an equivalent circuit consisting of a large number of AND and OR primitives and at most two NOT primitives. One is therefore tempted to conclude that (AND, OR) is an almost complete set of primitives. The fallacy in the above argument is that the “three-NOT-to-two-NOT” transformations might create closed feedback loops in the circuit and the behavior of the circuit may
correspond to a sequential circuit rather than a combinational circuit. Markov (1958) proved that \( m \) NOT primitives are sufficient for generating the complements of \( 2^m - 1 \) input variables in a circuit without any closed feedback loops. Hence, any function of \( n = 2^m - 1 \) variables may be realized with \( m \) NOT's.

But, here the number of NOT primitives grows with an increase in the number of input variables and hence is not bounded. Huffman proves in Chapter II that a single NOT primitive is sufficient to synthesize any arbitrary combinational function if and only if closed feedback loops are allowed in the network. Thus, \{AND, OR\} is almost an almost complete set! It seems that Kobayashi's general formulation of the problem has been motivated from the above observation.

A complete characterization of an almost complete set of logic primitives is not known, but Kobayashi (1967) gives the following necessary condition, whose proof is given in the Appendix.

**Theorem 7.1.** Let \( C(n) \) denote the number of \( n \)-variable functions that can be realized in an \( n \)-input network using the primitives \( F = \{f_1, \ldots, f_p\} \), each of which can be used a potentially infinite number of times, if necessary. If for any nonnegative integers \( a \) and \( b \)

\[
\lim_{n \to \infty} \frac{[C(n + a)n]^b}{2^{2^n}} = 0
\]

then the set \( F = \{f_1, \ldots, f_p\} \) is not almost complete.

Based on this theorem, it is shown that the characterization problem of an almost complete set is equivalent to deciding whether the set (AND, OR) is almost complete or not. We need to prove a number of theorems for this purpose.

**Theorem 7.2.** Let \( H \) be the set of all the reduced functions of not more than two variables that are obtained from \( f(x_1, \ldots, x_n) \) by identification of some inputs \( x_1, \ldots, x_n \) to constants 0 or 1. Then \( f(x_1, \ldots, x_n) \) is nonlinear monotonic if and only if \( n \geq 2 \), each of the reduced functions in \( H \) is monotonic, and at least one reduced function in \( H \) is the AND or OR function.

**Proof:** Suppose \( f \) is nonlinear and monotonic. Since \( f \) is nonlinear, \( n \geq 2 \). Also, since 0 and 1 functions are monotonic, all the reduced functions must also be monotonic (by Theorem 2.1). The theorem is obviously true for \( n = 2 \), since \( x_1 + x_2 \) and \( x_1x_2 \) are the only nonlinear monotonic functions. To prove the theorem for \( n \geq 2 \), we shall utilize a unique normal form representation
for a monotonic or positive unate function (Semon *et al.*, 1955)

\[ f = \sum_{i=1}^{k} P_i \]  

(16)

where \( P_i \) is an essential prime implicant\(^3\) (Quine, 1952) of \( f \) expressed as a product of uncomplemented variables. The prime implicants of \( f \) also possess the following property: for any arbitrary pair of prime implicants \( P_i \) and \( P_j \), \( i \neq j \), there must be at least one variable which does not occur in both \( P_i \) and \( P_j \). We have to consider two cases: (i) all the prime implicants of \( f \) are single variable, that is, \( f \) has the form \( x_1 + x_2 + \cdots + x_n \). In this case we set \( x_3 = x_4 = \cdots = x_n = 0 \) and we get an OR function \( x_1 + x_2 \); (ii) at least one of the prime implicants of \( f \) is a product of two or more variables. Let this prime implicant be, without loss of generality, \( x_1 x_2 \cdots x_k (2 \leq k \leq n) \). Now, set \( x_{k+1} = x_{k+2} = \cdots = x_n = 0 \); this will reduce the function to a \( k \)-input AND function \( x_1 x_2 \cdots x_k \), since at least one of the variables from \( (x_{k+1}, x_{k+2}, \ldots, x_n) \) must occur in all other prime implicants. Then we set \( x_3 = x_4 = \cdots = x_k = 1 \) and the function is reduced to AND function \( x_1 x_2 \). Thus the reduced set of functions includes at least one of the AND or OR functions. In most of the cases, both AND and OR functions can be realized as reduced functions (see Theorem 7.3).

To prove the converse, we note that since one of the reduced functions is nonlinear and a nonlinear function cannot be derived from a linear function by assigning constants 0 or 1 to some of the inputs, \( f \) must be nonlinear (Theorem 2.1). The function \( f \) must also be monotonic, otherwise we could have derived the negation function \( \overline{x}_1 \) from \( f \) (by Theorem 3.1) which contradicts the assumption that all the reduced functions are monotonic.

A nonlinear monotonic function \( f \) will be called an AND-type function, an OR-type function, or an AND-OR-type function if the reduced functions contain the AND but not the OR operation, the OR but not the AND operation, both the AND and the OR operations, respectively. Theorem 7.2 classifies all nonlinear monotonic functions into three disjoint classes: the AND-type functions, the OR-type functions, and the AND-OR-type functions. Furthermore, it is true that:

\(^3\) A prime implicant of a function \( f \) is a product term such that all the fundamental products contained in the term are true fundamental products of \( f \) and no variable can be deleted from the term without violating this condition. It is essential if it contains at least one fundamental product not contained in any other prime implicant. See Chapter V for further details.
THEOREM 7.3. A function $f(x_1, \ldots, x_n)$ is an AND-type (OR-type) function if and only if $f$ is an $m$-input AND(OR) function for $2 \leq m \leq n$.

The proof of this theorem will be left as an exercise.

THEOREM 7.4. An almost complete set of functions $F = \{f_1, \ldots, f_p\}$ includes a nonlinear function. If all nonlinear functions in $F$ are AND-type functions or OR-type functions, then $F$ must contain a linear function other than 0, 1, or $x_i$ ($1 \leq i \leq n$).

Proof: Suppose all the functions in $F$ are linear. Then all $n$-input functions realized from $f_1, \ldots, f_p$ are linear. The number of all $n$-variable linear functions is $C_1(n) = 2^n + 1$.

$$\lim_{n \to \infty} \frac{(C_1(n + a)n)^{ab}}{2^{2n}} \leq \lim_{n \to \infty} \frac{(2^{n+a+1})^{ab}}{2^{2n}} = 0$$

holds for any nonnegative integers $a$ and $b$. Thus, $F$ is not almost complete—a contradiction. Hence, $F$ must include a nonlinear function. Then suppose all nonlinear functions in $F$ are AND-type (OR-type) functions and all linear functions (if any) in $F$ are 0, 1, or $x_i$. Therefore, the functions that can be realized from $F$ are the 0, 1, $x_i$, and the $m$-input AND functions (OR functions), $2 \leq m \leq n$. The total number of such functions is obviously $C_2(n) = 2^n + 1$. Hence,

$$\lim_{n \to \infty} \frac{(C_2(n + a)n)^{ab}}{2^{2n}} \leq \lim_{n \to \infty} \frac{(2^{n+a+1})^{ab}}{2^{2n}} = 0$$

holds for any nonnegative integer $a$ and $b$. Thus, $F$ is not almost complete which is a contradiction. This proves the theorem.

An almost complete set $F$ is called trivially almost complete if

$$\{f_1, \ldots, f_p, 0, 1\}$$

is complete; otherwise it is nontrivially almost complete. A minimal nontrivially almost complete set is a nontrivially almost complete set such that none of its proper subsets is a nontrivially almost complete set.

THEOREM 7.5. Each function in a nontrivially almost complete set is one of 0, 1, $x_i$, or a nonlinear monotonic function. The set must include either an AND-OR-type function or both an AND-type and an OR-type function.
Proof: Suppose $F = \{f_1, \ldots, f_p\}$ is nontrivially almost complete. By Theorem 7.4, it contains a nonlinear function. All functions in the set must be monotonic since if one of the functions is nonmonotonic by using constants and the nonlinear function, $F$ becomes complete (Theorem 3.3); but this contradicts the assumption that $F$ is nontrivially almost complete. Hence, all nonlinear functions in the set are monotonic. Now, suppose that all nonlinear functions in the set are either AND-type or OR-type. Then, by Theorem 7.4, the set must contain a linear function other than 0, 1, $x_i$ which must necessarily be a nonmonotonic function. Applying Theorem 3.3, we again come to a contradiction. Hence the theorem is proved.

Theorem 7.6. A minimal nontrivially almost complete set of functions consists of either an AND-OR-type function or both AND- and OR-type functions.

Proof: By Theorem 7.5, the minimal nontrivially almost complete set is the union of four disjoint sets: (i) a set of AND-OR-type functions ($g_1, g_2, \ldots, g_d$), (ii) a set of AND-type functions ($h_1, h_2, \ldots, h_b$); (iii) a set of OR-type functions ($u_1, u_2, \ldots, u_c$), and (iv) a set of functions ($v_1, v_2, \ldots, v_d$) of the form 0, 1, $x_i$. By Theorem 7.5, either $a \geq 1$ or $b \geq 1$ and $c \geq 1$.

Suppose $a \geq 1$. With a single element, say $g$, from set (i) and 0 and 1, all functions in (i), (ii), and (iii), (iv) can be synthesized. Hence $g$ itself forms a nontrivially almost complete set and hence a minimal set. Suppose that $b \geq 1$ and $c \geq 1$. With two elements $h_1$ and $u_1$, say, and constant functions 0 and 1, we can synthesize $x_1x_2$ and $x_1 + x_2$, and hence all functions in (i), (ii), (iii), and (iv). Hence, $(h_1, u_1)$ form a nontrivially almost complete set and hence a minimal set.

It follows from Theorem 7.6 that if (AND, OR) form an almost complete set then there are two types of minimal nontrivial almost complete set: a set consisting of an AND-OR-type function or a set consisting of an AND-type function and an OR-type function. If (AND, OR) is not almost complete, then there is no nontrivially almost complete set.

There are no 0-, 1-, 2-variable AND-OR-type functions. There are three essentially different 3-variable AND-OR-type functions: $x_1(x_2 + x_3)$, $x_1 + x_2x_3$ and $x_1x_2 + x_1x_3 + x_2x_3$. The only AND- and OR-type set (AND, OR) is mentioned in Muller (1954) and Markov (1958).
APPENDIX. PROOF OF THEOREM 7.1

Let \( S \) denote an arbitrary logic circuit with inputs \( x_1, \ldots, x_n \) and consisting of the primitives from \( F = \{ f_1, \ldots, f_p \} \), called \( f \)-type primitives, each occurring unbounded number of times if necessary, and primitives from \( G = \{ g_1, \ldots, g_q \} \), called \( g \)-type primitives, occurring only bounded number of times such that each element \( g_t (1 \leq t \leq q) \) is used \( r_t \) times where for some positive integer \( k_t, r_t \leq k_t \) and \( \sum r_t \leq \sum k_t = K \). We attach a label \( \langle t, u \rangle \) with the \( u \)th element in these \( r_t \) elements \( g_t \).

According to our interconnection rules, \( S \) cannot have a closed feedback loop. But \( S \) might have a closed path which is nondirected, which occurs when the output of a certain primitive \( P \) is connected with inputs \( I_1, \ldots, I_s \) of other primitives. We wish to transform \( S \) into an equivalent circuit free from such closed paths, thus essentially obtaining a treelike circuit equivalent to \( S \). We start with one of the primitives \( P \) at the lowest level of the circuit. (If all the inputs to \( P \) are from the primary inputs \( x_1, \ldots, x_n \), then its level is 1. An element belongs to the \( g + 1 \) level if there is one input to it from another primitive whose level number is \( g \).) We construct \( s \) copies of the subcircuit whose output is \( P \) and connect the output of the \( j \)th subcircuit with the input \( I_j (1 \leq j \leq s) \). This process is repeated in the resulting circuit iteratively until the output of each primitive in all the levels is connected with only one input. The circuit \( S \) has thus been transformed to a treelike circuit producing the same output function.

To be able to calculate the total number of functions realizable by such a circuit, it is convenient to group some of the \( f \)-type primitives into a single \( f \)-type primitive \( f_{m,j} \) that is defined later. Qualitatively, the grouping consists of replacing the maximal subcircuit containing only \( f \)-type primitives by a single \( f \)-type primitive \( f_{m,j} \). More precisely, in the set of all \( f \)-type primitives an equivalence relation of connectedness is defined. Two \( f \)-type primitives \( P \) and \( P' \) are connected if either (i) \( P \) and \( P' \) are the same element, or (ii) there exists a sequence of \( f \)-type primitives \( P = P_1, P_2, \ldots, P_n = P' \) such that for each \( i, 1 \leq i \leq n \), the output of one of \( P_i \), \( P_{i+1} \) is connected with an input of the other. The primitives of one equivalence class form a maximal subcircuit \( S_m \) and each \( f \)-type primitive falls in exactly one subcircuit \( S_m \). Each subcircuit has only one output which is the output of the circuit or is connected to a \( g \)-type primitive and the inputs of the subcircuit are either \( x_1, \ldots, x_n \) or the outputs of \( g \)-type primitives. Consider now each \( S_m \) to be a basic block and let \( I_1 \) and \( I_2 \) be two inputs to some \( S_m \), which are connected to two \( g \)-type elements having the same label \( \langle t, u \rangle \). The circuit can be further reduced by identifying \( I_1 \) and \( I_2 \) within the subcircuit \( S_m \), deleting \( I_1 \) and also deleting the part of the circuit which produces \( I_2 \). This is repeated until no two inputs
of an $S_m$ are connected to outputs of two $g$-type elements having the same label number $\langle i, u \rangle$. Further, if there are two inputs $I_1$ and $I_2$ of an $S_m$ connected to the same primary input $x_i$ ($1 \leq i \leq n$), then these two inputs are identified inside the subcircuit and the input $I_2$ is deleted. Each subcircuit is now an $m$-input ($0 \leq m \leq n + K$) circuit consisting of $f$-type elements which produces at the output a function $f_{m,j}$ where $f_{m,j}$ is the $j$th function in the list of all $C(m)$ functions that can be produced by an $m$-input circuit using primitive $F = \{f_1, \ldots, f_K\}$ unbounded number of times. Replace the subcircuit $S_m$ by a single primitive $f_{m,j}$.

Thus the original circuit $S$ has been transformed to an equivalent circuit having the following properties:

(i) The circuit consists of $g$-type primitives (each primitive $g_i$ may have $m_i$ number of inputs $0 \leq m_i$) and $f_{m,j}$-type primitives ($1 \leq j \leq C(m)$, $0 \leq m \leq n + K$), and has a tree-like structure because no output of a primitive is connected to more than one input of other primitives.

(ii) The maximum level of the circuit is $2K + 1$. This is because input of one $f_{m,j}$-type primitive cannot be derived from the output of another $f_{m,j}$-type primitive; hence maximum level number occurs when $f_{m,j}$-type primitives and $g$-type primitives alternate from the output level to the first level, giving a total level number of $2K + 1$ since there are at most $K$ distinct $g$-type primitives in the transformed circuit.

Let $N(n)$ denote the number of different circuits satisfying (i) and (ii). Then we will prove\(^4\) that for sufficiently large $n$

\[ N(n) \leq (C(n + K)n)\, n^{K+1} \]

To prove this, let $B(n, \lambda)$ denote the set of different $n$-input circuits satisfying condition (i) and the condition that the level number of the circuit does not exceed $\lambda$. Let $N(n, \lambda)$ denote the number of circuits in $B(n, \lambda)$.

The output element of a circuit in $B(n, \lambda)$ could be either a $g$-type primitive or an $f_{m,j}$-type primitive. The input to this element could be either one of the $n$ inputs $x_i$, or the output of a circuit in $B(n, \lambda - 1)$. Hence for $\lambda \geq 2$ the following inequality holds:

\[ N(n, \lambda) \leq \sum_{i=1}^{q_n} (n + N(n, \lambda - 1))^m + \sum_{m=0}^{n+K} C(m)(n + N(n, \lambda - 1))^m \quad (A.1) \]

Also, since $N(n, 0) = 0$, we have

\[ N(n, 1) \leq \sum_{i=1}^{q_n} n^m + \sum_{m=0}^{n+K} C(m)n^m \quad (A.2) \]

\(^4\) The detailed proof given here is from Kobayashi (1970). Reddy and Mukhopadhyay (1971) obtained a simplified proof of Kobayashi's theorem after this chapter was written.
Let $M$ denote $\max \{m_1, \ldots, m_q\}$ and $M(n, \lambda)$ denote $n + N(n, \lambda)$. For $n \geq 0$ and $\lambda \geq 1$, we have

$$n \leq M(n, \lambda) \quad \text{(A.3)}$$

Also, from definition of $M$, for any $t$

$$m_t \leq M \quad \text{(A.4)}$$

Since any $n$-variable function may be regarded as an $(n + 1)$-variable function that does not depend on the $(n + 1)$th variable, we have

$$C(0) \leq C(1) \leq C(2) \leq \cdots \quad \text{(A.5)}$$

We assume $F = \{f_1, \ldots, f_p\}$ to be nonempty, otherwise the theorem is trivially true. Let $f_1$ be an $n_1$-variable function. Then the $n_1$-input circuit realizing $f_1$ is one of the members of $C(n_1)$. Hence $C(n_1) \geq 1$. Using (A.5) we have for any $n \geq n_1$

$$1 \leq C(n + K) \quad \text{(A.6)}$$

Let $n_2 = \max \{M - K, 1 + q, 3, K + 2, 2K + 1\}$. Then for any $n \geq n_2$ we have the following inequalities.

$$M \leq n + K \quad \text{(A.7)}$$

$$1 + q \leq n \quad \text{(A.8)}$$

$$3n \leq n^2 \quad \text{(A.9)}$$

$$2n \leq n^2 \quad \text{(A.10)}$$

$$1 \leq n \quad \text{(A.11)}$$

$$1 \leq 2n \quad \text{(A.12)}$$

$$1 \leq n + K \quad \text{(A.13)}$$

$$1 \leq M(n, \lambda) \quad \text{(by A.3 and A.11)} \quad \text{(A.14)}$$

$$n + K + 2 \leq 2n \quad \text{(A.15)}$$

$$n + K + 1 \leq 2n \quad \text{(A.16)}$$

$$2K + 1 \leq n \quad \text{(A.17)}$$
Let $n_0$ denote max\{$n_1, n_2$\}. If $n \geq n_0$ and $\lambda \geq 2$ we have the following inequalities:

$$M(n, \lambda) = n + N(n, \lambda)$$

$$\leq n + \sum_{t=1}^{q} (n + N(n, \lambda - 1))^m + \sum_{m=0}^{n+K} C(m)(n + N(n, \lambda - 1))^m \quad \text{(by A.1)}$$

$$= n + \sum_{t=1}^{q} M(n, \lambda - 1)^{m_t} + \sum_{m=0}^{n+K} C(m)M(n, \lambda - 1)^m$$

$$\leq M(n, \lambda - 1) + qM(n, \lambda - 1)^{M} + (n + K + 1)C(n + K)$$

$$M(n, \lambda - 1)^{n+K}$$

(by A.3, A.14, A.4, and A.5)

$$\leq M(n, \lambda - 1)^{n+K} + qM(n, \lambda - 1)^{n+K} + (n + K + 1)C(n + K)$$

$$M(n, \lambda - 1)^{n+K}$$

(by A.14, A.13, and A.7)

$$= (1 + q + (n + K + 1)C(n + K))M(n, \lambda - 1)^{n+K}$$

$$\leq (n + 2nC(n + K))M(n, \lambda - 1)^{n+K} \quad \text{(by A.8 and A.16)}$$

$$\leq (nC(n + K) + 2nC(n + K))M(n, \lambda - 1)^{n+K} \quad \text{(by A.6)}$$

$$= 3nC(n + K)M(n, \lambda - 1)^{n+K}$$

$$\leq n^2C(n + K)M(n, \lambda - 1)^{n+K} \quad \text{(by A.9)}$$

$$\leq M(n, \lambda - 1)^2C(n + K)M(n, \lambda - 1)^{n+K} \quad \text{(by A.3)}$$

$$= C(n + K)M(n, \lambda - 1)^{n+K+2}$$

$$\leq C(n + K)M(n, \lambda - 1)^{2^n} \quad \text{(by A.14 and A.15)} \quad \text{(A.18)}$$

If $n \geq n_0$ we have also the following inequalities:

$$M(n, 1) = n + N(n, 1)$$

$$\leq n + \sum_{t=1}^{q} n^m + \sum_{m=0}^{n+K} C(m)n^m \quad \text{(by A.2)}$$

$$\leq n + qn^M + (n + K + 1)C(n + K)n^{n+K} \quad \text{(by A.11, A.4, and A.5)}$$

$$\leq n^{n+K} + qn^{n+K} + (n + K + 1)C(n + K)n^{n+K} \quad \text{(by A.11, A.13, and A.7)}$$

$$= (1 + q + (n + K + 1)C(n + K))^n$$

$$\leq (n + 2nC(n + K))n^{n+K} \quad \text{(by A.8 and A.16)}$$

$$\leq (nC(n + K) + 2nC(n + K))n^{n+K} \quad \text{(by A.6)}$$

$$= 3nC(n + K)n^{n+K}$$

$$\leq n^2C(n + K)n^{n+K} \quad \text{(by A.9)}$$

$$= C(n + K)n^{n+K+2}$$

$$\leq C(n + K)n^{2^n} \quad \text{(by A.11 and A.15)} \quad \text{(A.19)}$$
If \( n \geq n_0 \) and \( \lambda \leq 2K + 1 \) we have the following inequalities:

\[
M(n, \lambda) \leq C(n + K)M(n, \lambda - 1)^{2n} \\
\leq C(n + K)^{1+2n}M(n, \lambda - 2)^{(2n)^3} \\
\leq C(n + K)^{1+2n+(2n)^3}M(n, \lambda - 3)^{(2n)^3} \\
\vdots \\
\leq C(n + K)^{1+2n+(2n)^3+\cdots+(2n)^{\lambda-1}}n^{(2n)^\lambda} \\
\leq C(n + K)(2n)^{\lambda-1+(2n)^{\lambda-1}+\cdots+(2n)^{\lambda-1}}n^{(2n)^\lambda}
\tag{by A.6 and A.12}
\]

\[
= C(n + K)^{\lambda(2n)^{\lambda-1}}n^{(2n)^\lambda}
\]

\[
\leq C(n + K)^{n(2n)^{\lambda-1}}n^{(2n)^\lambda}
\tag{by A.6 the condition \( \lambda \leq 2K + 1 \), and A.17}
\]

\[
\leq C(n + K)^{n(2n)^{\lambda-1}}n^{(2n)^\lambda}
\tag{by A.6, A.10, and A.11}
\]

\[
= C(n + K)^{n^{2\lambda-1}}n^{n^{2\lambda}}
\tag{A.20}
\]

Obviously \( N(n) = N(n, 2K + 1) \). Hence for any \( n \geq n_0 \)

\[
N(n) = N(n, 2K + 1) \\
\leq n + N(n, 2K + 1) \\
= M(n, 2K + 1) \\
\leq C(n + K)^{n^{2\lambda+1}}n^{n^{2\lambda+2}}
\tag{by A.20}
\]

\[
\leq C(n + K)^{n^{2\lambda+2}}n^{n^{2\lambda+2}}
\tag{by A.6 and A.11}
\]

\[
= (C(n + K)n)^{n^{2\lambda+2}}
\tag{A.21}
\]

which proves the required inequality.

Let \( L(n) \) denote the number of \( n \)-variable functions realized by \( n \)-input circuits using elements of \( F \) unbounded number of times and the elements of \( G \) bounded number of times. Then for large \( n \),

\[
L(n) \leq N(n) \leq (C(n + K)n)^{n^{2\lambda+2}}
\tag{A.22}
\]

Since \( K \) does not grow with \( n \), we have

\[
\lim_{n \to \infty} \frac{L(n)}{2^{2n}} \leq \lim_{n \to \infty} \frac{(C(n + K)n)^{n^{2\lambda+2}}}{2^{2n}} = 0
\tag{A.23}
\]

where \( 2^{2n} \) is the total number of \( n \)-variable functions. Therefore, Eq. (A.23) shows that there exist \( n \)-variable functions that cannot be realized by using the elements of \( F \) unbounded number of times and the elements of \( G \) bounded number of times. Hence \( F = \{f_1, \ldots, f_p\} \) is not almost complete.
REFERENCES