COMPUTER ORIENTED ALGORITHMS FOR MINIMIZING THE LOGIC FUNCTIONS

by

Narindar Singh Khabra

Submitted in the partial fulfillment of
the requirement for the degree of

Master of Applied Science

Department of Electrical Engineering,
Faculty of Science and Engineering,
University of Ottawa,
Ottawa, Ontario.

May, 1972

ABSTRACT

This thesis considers the problem of developing computer oriented algorithms for the minimization of logic functions. A brief review of some of the existing techniques for the generation of prime implicants, which is the first phase of the minimization procedure, is first presented starting from the sum-of-products or product-of-sums expressions of the functions. Next a new iterative procedure for generating the prime implicants by utilizing a new tabular mode of functional representation called clause-column table is described. This procedure generates all the prime implicants, and can be equally applied to functions given in the sum-of-products or in the product-of-sums form, both cononical and noncanonical. The procedure can also be adapted to finding the prime implicants of functions having many unspecified or don't care terms. The algorithm is programmable on a computer. The total amount of computational time required is reduced and memory requirement is also effectively less than the previously existing techniques. A review of the techniques for finding the minimal or nonminimal irredundant solutions of the functions starting from the set of all the prime implicants is also presented. A modification of a recently proposed method for finding the minimal solutions of logic functions is finally discussed.

ACKNOWLEDGEMENTS

The author wishes to express his deep and sincere appreciation to Dr. S.R. Das, of the Electrical Engineering Department, for his advice, criticism and discussions throughout the course of this work. His assistance is most gratefully acknowledged in writing, and providing ideas for the joint paper [48], the results of which appear in Chapter 3.

Special thanks are due to Professor P.M. Thompson for providing financial support for carrying out this research.

Thanks are also due to Dr. Dilip K. Banerji of Computer Science Department, for many helpful discussions and suggestions in programming the algorithm by using APL.

The author is particularly grateful to Miss Mirelle Gélinas, who typed the final manuscript, for her patience and understanding.

The financial support of the National Research Council of Canada under Grant No. A-3379 is gratefully acknowledged.

CONTENTS

CONTENTS (Continued)

Chapter 1

INTRODUCTION

The study of switching theory has been acquiring increasing importance for the large class of scientists, from the pure logician to the practical programmer. From the point of view of electronic computation, switching theory has offered the means for studying the functions and the potentialities of phenomena in depth, from the crude piece of engineering, the computer becomes an organism whose organizational principles can be discovered and put to useful purposes. Switching circuits are commonly classified as combinational or sequential depending upon whether the value of the output is completely determined by the present inputs or it depends upon the past values of inputs as well. The research presented in this thesis is restricted to combinational circuits only. One problem which has continued to plague both the logic designer and the theorist is the problem of minimal realization of a switching function. The first step in the process of minimization of switching function is to turn this problem into an unambigious mathematical specification and the second step is to convert the mathematical specification into a hardware realization. The reduction of network complexity leads to lower cost and greater ease of construction. Because there exist an infinitely large number of distinct combinational network structures with the same behavior, the task comes down to that of selecting from all these structures the network of minimum complexity. It is rather difficult to define the minimum complexity for a combinational network. A network which is minimum in one sense may not be minimum in another sense. It may be desired to minimize the total number of elements and interconnections or to simplify the design, layout or wiring pattern. The prime aim of the present study is to minimize the total number of circuit elements.

The usual approach for the solution of this simplification problem of switching functions involves, in general, a consideration of two distinct phases. In the first phase all the prime implicants of the function are found, while in the second phase, from this set of all the prime implicants, a minimal subset of prime implicants is selected such that their disjunction is equivalent to the function and from which none of the prime implicants can be dropped without sacrificing equivalence. Many different algorithms have been formulated for solving both the first and second phase of this simplification problem [1] - [48].

In the present thesis attempt is made to develop a straightforward and efficient algorithm for generating all the prime implicants of switching functions and also to find thier minimal solutions.

In chapter 2 of the thesis a brief review of the existing methods of finding the prime implicants of switching functions is presented. In the first part of this chapter, the different techniques for the generation of prime implicants of switching functions given in the minterm form are discussed, whereas in the second part of the chapter, the relevant features of the techniques for the generation of prime implicants starting from the maxterm form are presented.

Chapter 3 presents an iterative procedure for generating all the prime implicants of switching functions by utilizing a new tabular mode of functional representation called clause-column table. The procedure generates all the prime implicants, and can be applied equally well to functions given in the sum-of-products or in the product-of-sums forms, both canonical and noncanonical. The procedure can also be readily adapted to determine the prime implicants of functions having a large number of unspecified or don't care terms.

In chapter 4 a brief review of the existing important techniques for finding the minimal or nonminimal irredundant solutions, given all prime implicants of switching functions is presented. Next an attempt is made to

systematize a procedure recently suggested by Das [22] for finding the minimal solutions of switching functions. Finally at the end of the thesis the computer program of the algorithm along with some worked out examples is attached.

Chapter 2

GENERATION OF PRIME IMPLICANTS

2.1 Introduction

In this chapter we make a brief review of some of the existing techniques for the generation of prime implicants of switching functions starting from their sum-of-products or product-of-sums expressions (canonical or non-canonical). Quine [2] was the first to propose a method for the generation of prime implicants of switching functions and one of the basic requirements of his method was that the functions must be expressed in the developed normal form (canonical sum-of-products form). Quine's later method [6,9] eliminated this restriction. McCluskey [7] adopted the principle of Quine's first method, systematized the procedure by using the binary notation instead of algebraic notation and introduced the classification of terms to reduce the number of comparisons required. McCluskey [21] later on suggested an ingenious method for determining the prime implicants of functions having large number of don't care states utilizing only the knowledge of the specified or S-terms (the input conditions for which the functions are equal to 1) and the prohibited or P-terms (the input conditions for which the functions are equal to 0). Hall [10] developed a technique for generating the prime implicants of functions expressed in the standard sum-of-products form through the action of a hypothetical binary sieve. Scheinman [11] described a method which enables to obtain prime implicants by a simple operation on a set of decimal numbers which describe the function. Nelson [4,5] was the first to suggest a method for generating the prime implicants of switching functions starting from their product-of-sums form represented in algebraic mode. Das and Choudhury [15] utilized Nelson's idea to develop a tabular technique for generating the prime implicants of switching

functions starting from maxterm type expression represented in decimal mode. Das and Choudhury also extended their tabular method for generating the prime implicants having many unspecified or don't care terms. Slagle et al [17] described an algorithm for generating the prime implicants of switching functions represented in the algebraic mode which works equally well with either conjunctive or disjunctive (both canonical and noncanonical) form of the function. The basic approach in this method consists in first finding the frequency ordering of different literals in the product or sum terms and next the expansion is carried out around different variables in one or more levels through a series of trees. The relevant features of the aforementioned techniques for generating the prime implicants of switching functions are discussed below.

2.2 Quine McCluskey Method

In this section we briefly discuss the basic features of the method first developed by Quine and later systematized by McCluskey. The given function is assumed to be expressed in the canonical sum-of-products form. The method can be described as follows.

Make a list of all the fundamental product terms of the given function and extend this list as follows. Whenever two entries are found in the list which are related as $f_1 x_i$ and $f_1 \bar{x}_i$, add their common part f_1 as a new entry in the list. Check those entries $f_1 x_i$ and $f_1 \bar{x}_i$ which thus generate new entries, but a check mark is not to be treated as disqualifying an entry from reuse. When the list has been extended as far as possible by the above process, the prime implicants of the given function F can be read off from the list thus : they are the entries which bear no check marks.

McCluskey [7] systematized the above method by simplifying the notation and making the procedure more algorithmic. The notation is simplified by discarding the expression involving literals and using only binary characters. The theorem being used to combine terms is stated in terms of binary characters as follows.

Theorem : If two binary characters are identical in all positions except one and if neither character has a dash in the position in which they differ, then the two characters can be replaced by a single character which has a dash in the position in which the original characters differ and which is identical with the original characters in all other positions.

A list is made in a column for the binary equivalents of the decimal numbers which specify the function. These binary numbers are then ordered so that any number which contain no 1's come first, followed by any number containing a single 1, and so on. Lines are drawn to divide the columns into groups of binary numbers which contain a given number of 1's. The theorem stated above is applied to these binary numbers by comparing each number of one group with all the numbers of the next lower group. Other pairs of numbers need not be considered since any two numbers which are not from adjacent groups must differ in more than one digit. For each number which has 1's wherever the number with which it is being compared has 1's, a new character is formed according to the theorem. A check mark is placed next to each number which is used in forming a new character. The new characters are placed in a separate column which is again divided into groups of characters which has the same number of 1's.

After each number in the first column has been considered, a similar process is carried out for the characters of column two. Two characters from adjacent groups can be combined if they both have their dashes in the same position and if the character from the lower group has 1's whereas the upper character has 1's. If two characters are combined, the resulting character is placed in the third column and the column two characters from which the new characters are formed are checked. The procedure is repeated and new columns are formed until no further combinations are possible. The unchecked characters which have not entered into any combination represent the prime implicants of the function.

Example : $F(x_1, x_2, x_3, x_4) = \Sigma\,(0,1,2,3,4,6,7,8,9,11,15)$

The different columns as formed from the above given function are as follows :

Column I

	x_4	x_3	x_2	x_1	
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
4	0	1	0	0	✓
8	1	0	0	0	✓
3	0	0	1	1	✓
6	0	1	1	0	✓
9	1	0	0	1	✓
7	0	1	1	1	✓
11	1	0	1	1	✓
15	1	1	1	1	✓

Column II

	x_4	x_3	x_2	x_1	
(0,1)	0	0	0	-	✓
(0,2)	0	0	-	0	✓
(0,4)	0	-	0	0	✓
(0,8)	-	0	0	0	✓
(1,3)	0	0	-	1	✓
(1,9)	-	0	0	1	✓
(2,3)	0	0	1	-	✓
(2,6)	0	-	1	0	✓
(4,6)	0	1	-	0	✓
(8,9)	1	0	0	-	✓
(3,7)	0	-	1	1	✓
(3,11)	-	0	1	1	✓
(6,7)	0	1	1	-	✓
(9,11)	1	0	-	1	✓
(7,15)	-	1	1	1	✓
(11,15)	1	-	1	1	✓

Column III

	x_4	x_3	x_2	x_1
(0,1,8,9)	-	0	0	-
(0,1,2,3)	0	0	-	-
(0,2,4,6)	0	-	-	0
(1,3,9,11)	-	0	-	1
(2,3,6,7)	0	-	1	-
(3,7,11,15)	-	-	1	1

The set of prime implicants P(F) of the switching function above is given by

$$P(F) = \{(0,1,8,9),\ (0,1,2,3),\ (0,2,4,6),\ (1,3,9,11),\ (2,3,6,7),\ (3,7,11,15)\}$$

or $P(F) = \{\bar{x}_2\bar{x}_3,\ \bar{x}_3\bar{x}_4,\ \bar{x}_1\bar{x}_4,\ x_1\bar{x}_3,\ x_2\bar{x}_4,\ x_1x_2\}$ which are read directly from the above column III.

2.3 Iterated Consensus Technique

Quine [9] explained a speedy and direct method known as iterated consensus method that converts a function F given in the sum-of-products form, canonical or noncanonical, into the disjunction or sum of all its prime implicants. The generation of prime implicants by iterated consensus method is particularly attractive if the function is given in the noncanonical form and if a conversion to a canonical form is impractical due to the large number of terms of the function. In this method, the prime implicants can be determined by systematically applying the consensus operation on the terms of the function, as explained below.

Definition : If f_1 and f_2 are two simple products such that there is exactly one literal which occurs unnegated in f_1 and negated in f_2 or vice versa, then the conjunction or product $f_1 f_2$ with opposing literals and any duplicate literal deleted is called the consensus of f_1 and f_2.

Example : $F(x_1, x_2, x_3, x_4) = x_1\bar{x}_3\bar{x}_4 + x_2\bar{x}_3x_4 + \bar{x}_1 + x_1x_2x_3$

In the above example, the consensus of $x_1\bar{x}_3x_4$ and $x_2\bar{x}_3x_4$ is $x_1x_2\bar{x}_3$, consensus of $\bar{x}_1$ and $x_1x_2x_3$ is x_2x_3, consensus of $x_1\bar{x}_3\bar{x}_4$ and $x_1x_2x_3$ is $x_1x_2\bar{x}_4$, consensus of $x_2x_3x_4$ and $x_1x_2x_3$ is $x_1x_2x_4$ and the consensus of $x_1\bar{x}_3x_4$ and $\bar{x}_1$ is $\bar{x}_3\bar{x}_4$.

If the switching function F is given in the sum-of-products form, then the following two rules can be applied as long as possible and when neither is applicable further, then the disjunctive form of all and only the prime implicants are found.

Rule I : If a product term f_1 in a given function F subsumed* another product term in a given function F, delete the subsumed product term.

Rule II : Adjoin, as an additional product term, the consensus of two product terms f_1 and f_2.

---

* Foot Note : A product (sum) term f_1 is said to subsume a product (sum) term f_2 iff the set of all the literals in f_1 are among the set of all the literals in f_2.

Example : $F(x_1,x_2,x_3,x_4,x_5) = x_1x_2x_4x_5 + x_1x_2\bar{x}_5 + x_1\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_4 + x_2x_3x_4 + \bar{x}_2x_3x_4$

Apply rule II : $F = x_1x_2x_4x_5 + x_1x_2x_4 + x_1x_2\bar{x}_3x_5 + x_1x_2x_3x_5 + x_1x_2\bar{x}_5 + x_2\bar{x}_4\bar{x}_5 + x_1x_3x_4x_5 + x_1x_3x_4 + \bar{x}_3x_4 + x_1x_2x_4 + x_1x_4 + x_2x_3x_4 + \bar{x}_2x_3x_4$

Apply rule I : $F = x_1x_2x_4 + x_1x_2\bar{x}_3x_5 + x_1x_2x_3x_5 + x_1x_2\bar{x}_5 + x_2\bar{x}_4\bar{x}_5 + \bar{x}_3\bar{x}_4 + x_1x_2x_4 + x_1x_4 + x_2x_3x_4$

Apply rule II : $F = x_1x_2x_4 + x_1x_2\bar{x}_5 + x_1x_2\bar{x}_3 + x_1x_2 + x_1x_2x_3 + x_1x_2\bar{x}_3x_5 + x_1x_2x_5 + x_1x_2x_3 + x_1x_2x_3x_4 + x_2x_3x_4x_5 + x_1x_2x_4x_5 + x_1x_2x_3x_5 + x_1x_2x_3 + x_1x_2x_3x_4 + x_1x_2x_4x_5 + x_2x_3x_4x_5 + x_1x_2x_5 + x_2x_4x_5 + x_2x_4x_5 + x_3x_4 + x_2x_4 + x_1x_2x_4 + x_2x_4 + \bar{x}_1x_4 + x_2x_3x_4$

Apply rule I : $F = x_1x_2 + \bar{x}_1\bar{x}_4 + x_2\bar{x}_4 + \bar{x}_3\bar{x}_4$

Apply rule II : $F = x_1x_2 + x_2\bar{x}_4 + \bar{x}_1\bar{x}_4 + x_2\bar{x}_4 + \bar{x}_3\bar{x}_4$

$\therefore F = x_1x_2 + x_2\bar{x}_4 + \bar{x}_1\bar{x}_4 + \bar{x}_3\bar{x}_4$

The rules I and II cannot be applied any further, thus the set of all the prime implicants P(F) of the given function is :

$P(F) = \{x_1x_2,\ x_2\bar{x}_4,\ \bar{x}_1\bar{x}_4,\ \bar{x}_3\bar{x}_4\}$

2.4 Binary Sieve Method

Hall [10] purposed a method in which he imagined that a sieve is capable of shifting out the 1's in any given column of binary numbers and leaving 0's at their places. If the sieve is to shift out the 1's in the k-th column and the number of terms in the binary form are dropped into the sieve, then the terms with a 1 in the k-th column will lose it and change, but the terms that has a 0 in the k-th will be unchanged. If a multiple shifting is used, that is, if the terms are shifted for one column and the results immediately shifted again for one or more other columns, then the general rule for redundant variables can be applied directly.

For a function given in the canonical sum-of-products form, its terms are written in a row designated as "T-row" followed by the 1,2, 4,8,...,2^n rows. The entries in the 1-row result when the T-row terms are shifted to remove the 1's in the first column. The entries in the 2-row result when the T-row terms are shifted to remove the 1's in the second column. This is continued until there is a row corresponding to each column of the variables of the function. In each case an entry is made only if the terms in the T-row have changed in passing through the sieve. If shifted terms also appear in the T-row, then these terms are encircled and are designated as "essential shifted terms".

Example :

T-row :	0000	0001	0010	0011	0100	0110	0111	1000	1001	1011	1111
1-row :	—	(0000)	—	(0010)	—	—	(0110)	—	(1000)	1010	1110
2-row :	—	—	(0000)	(0001)	—	(0100)	0101	—	—	(1001)	1101
4-row :	—	—	—	—	(0000)	(0010)	(0011)	—	—	—	(1011)
8-row :	—	—	—	—	—	—	—	(0000)	(0001)	(0011)	(0111)

In the decimal form the table is constructed in the same way as in the above example but the terms are used in the decimal form rather than in the binary form.

Example :

T-row :	0	1	2	3	4	6	7	8	9	11	15
1-row :	-	(0)	-	(2)	-	-	(6)	-	(8)	10	14
2-row :	-	-	(0)	(1)	-	(4)	5	-	-	(9)	13
4-row :	-	-	-	-	(0)	(2)	(3)	-	-	-	(11)
8-row :	-	-	-	-	-	-	-	(0)	(1)	(3)	(7)

Theorem 1 : In the shifted term array, if 2^j different terms identical in all but j columns can be found in the k-th row, where k-th column is not one of the j columns, then with the corresponding terms in the T-row they form 2^{j+1} different terms identical in all but j+1 columns.

Theorem 2 : In the shifted term array, if 2^j different terms in the T-row, identical in all but j columns, have 2^j shifted terms in the k-th row, where the k-th column is not one of the j columns, then with these shifted terms they form 2^{j+1} different terms identical in all but j+1 terms.

If the above theorems are applied only to the essential shifted terms and the ordinary shifted terms are ignored, then no term can enter a combination unless it is a term of the function. Each time one of these theorems is applied, the effect is to eliminate another variable and to double the number of terms in a combination. Any encircled term and its associated T-row term form a pair. For the previous example, each essential shifted term and its associated T-row term as a pair and the grouping of them by the row number are given below.

Shiftings

(1)	0-1	(2)	0-2	(4)	0-4	(8)	0-8
	2-3		1-3		2-6		1-9
	6-7		4-6		3-7		3-11
	8-9		9-11		11-15		7-15

The left hand members of the above shiftings are essential shifted terms from various rows, whereas the right hand members are the associated terms from the T-row.

Beginning with a first pair in the 1-shiftings, each combination of terms in that shifting is compared with the left hand members of the 2, 4, 8, ..., 2^n shiftings. Similarly, the combinations in the 2-shiftings are compared with the left hand members of the 4, 8, ..., 2^n shiftings and so on. The combinations in the last shifting i.e. 2^n shifting are examined only to see whether or not they have appeared in one of the large combinations generated by the previous comparisons. Each time the terms of combination are found in the left hand member of another shifting, theorem 1 is applicable. The associated right hand members of that shifting also become members of

another shifting, theorem 1 is applicable. The associated right hand members of that shifting also become members of the combination. Every possible combination starting with each pair in the shifting must be tested in this way with theorem 1. If a combination is generated, all of whose terms are already members of some other combination, the smaller combination is deleted. When the above procedure is carried out completely, it generates the following combinations for the above example.

$(1,2) \longrightarrow (0,1,2,3)$

$(1,8) \longrightarrow (0,1,8,9)$

$(1,4) \longrightarrow (2,3,6,7)$

$(2,4) \longrightarrow (0,2,4,6)$

$(2,8) \longrightarrow (1,3,9,11)$

$(4,8) \longrightarrow (3,7,11,15)$

A conversion of these combinations to algebraic form is done as follows :

(a) Take the smallest number of combination.

(b) Write it in binary form.

(c) Substitute dashes for the zeros in the columns indicated by the shifting numbers.

(d) Write the algebraic form from the remaining zeros and ones.

Example : Take a combination of (0,1,8,9)

(a) 0 (b) 0000 (c) -00- (d) $\bar{x}_2\bar{x}_3$

The combinations after conversion to algebraic form represent prime implicants. Thus the set of prime implicants P(F) for the above example is given by

$$P(F) = \{ \bar{x}_3\bar{x}_4, \bar{x}_2\bar{x}_3, x_2\bar{x}_4, \bar{x}_1\bar{x}_4, x_1\bar{x}_3, x_1x_2 \}$$

2.5 Method of Scheinman

Scheinman [11] presented a method for simplifying switching functions given in the decimal form in which all the essential prime implicants are found and some or all of the nonessential prime implicants not required in the function of minimal covers are automatically eliminated.

Any switching function $F(x_n, x_{n-1}, \ldots\ldots, x_1)$ given in the sum-of-products form can be written in the following form :

$$F(x_n, x_{n-1}, \ldots\ldots, x_1) = x_n f_1 (x_{n-1}, x_{n-2}, \ldots\ldots, x_1) + \bar{x}_n f_2 (x_{n-1}, x_{n-2}, \ldots\ldots, x_1)$$

It is assumed that the variables are assigned binary weights in the order shown, with x_n being assigned the highest weight and x_1 the lowest. Also it is assumed that the given function F is specified in the canonical sum-of-products form and in the decimal representation, the given function F can be written as

$F(x_n, x_{n-1}, \ldots\ldots, x_1) = \Sigma j = x_n R_1 + \bar{x}_n R_2$, where j is the decimal equivalent of the associated product term, R_2 is a sum consisting of the decimal numbers which are smaller that the weight of x_n and R_1 is a sum consisting of the numbers which are greater than the weight of x_n, from each of which the weight of x_n is subtracted. Each of the residues can then be expanded about x_{n-1}, which is the highest weighted variable of the residues. There are five different possibilities for R_1 and R_2 as follows :

(a) $R_1 = R_2 = R$, therefore, $F(x_n, x_{n-1}, \ldots\ldots, x_1) = x_n R + \bar{x}_n R = R$

(b) $R_1 > R_2$, indicating that the decimal numbers representing R_2 form a subset of those representing R_1, then $R_1 = R_2 + R_b$ and

$$F(x_n, x_{n-1}, \ldots\ldots, x_1) = x_n (R_2 + R_b) + \bar{x}_n R_2 = R_2 + x_n R_b$$

(c) $R_2 > R_1$, where $R_2 = R_1 + R_a$ and

$$F(x_n, x_{n-1}, \ldots\ldots, x_1) = x_n R_1 + \bar{x}_n (R_1 + R_a) = R_1 + \bar{x}_n R_a$$

(d) The residues have no number in common, in which case

$$F(x_n, x_{n-1}, \ldots\ldots, x_1) = x_n R_1 + \bar{x}_n R_2$$

(e) Some of the numbers in each residue are the same, in which case $R_1 = R_c + R_d$ and $R_2 = R_c + R_e$

Thus $F(x_n, x_{n-1}, \ldots\ldots, x_1) = x_n(R_c + R_d) + \bar{x}_n(R_c + R_e)$

$$= R_c + x_n (R_c + R_d) + \bar{x}_n(R_c + R_e)$$

The following algorithm is given to organize and to simplify the procedure for generating the prime implicants of a switching function.

Algorithm

Step 1 : Arrange the decimal numbers representing the given function in a column.

Step 2 : Divide the decimal numbers into two columnar groups, one headed with $\bar{x}_n$ and the other with x_n, the column $\bar{x}_n$ contains the numbers of the original function which are smaller than the binary weight of x_n and the column x_n contains the numbers which are equal to or greater than the weight of x_n, first subtracting the weight of x_n from each.

Step 3 : Include a third column, headed by a dash to indicate the redundancy of x_n and $\bar{x}_n$, containing the numbers which are common to columns x_n and $\bar{x}_n$. Check the corresponding numbers in column x_n and $\bar{x}_n$ to record the fact that they are redundant. If any of the numbers in the dashed column has been previously checked in both x_n and $\bar{x}_n$ columns, they should also be checked in the dashed column.

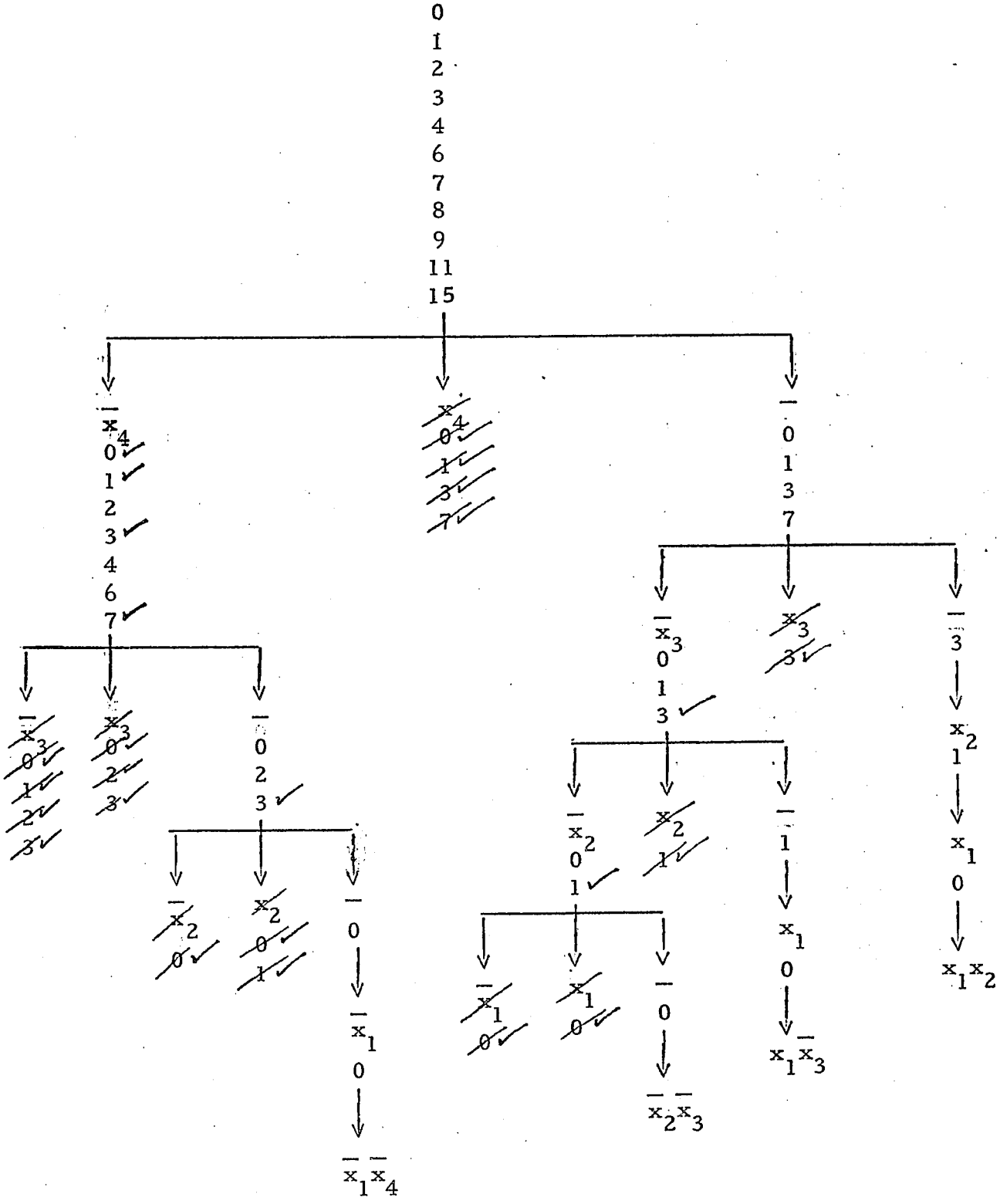
Step 4 : Examine each column. If any column consists of only checked numbers, eliminate the column entirely.

Step 5 : Repeat steps 2, 3, and 4 for each of the variable of the function for each column according to the weight ordering till the lowest weight of the variable is obtained.

The prime implicants can be found by tracing each successful path of the figure.

Example : $F(x_4, x_3, x_2, x_1) = \Sigma\ (0,1,2,3,4,6,7,8,9,11,15)$

The algorithm is applied to the given function F and the prime implicants are found as follows.



The prime implicants P(F) of the function $F(x_4, x_3, x_2, x_1)$ are :

$P(F) = \{ \bar{x}_1\bar{x}_4, \bar{x}_2\bar{x}_3, x_1\bar{x}_3, x_1x_2 \}$

2.6 Method due to Nelson

Nelson [4,5] discussed a method in which the prime implicants of a switching function given in the sum-of-products or product-of-sums form, canonical or noncanonical can be found. In the present section, this method is presented briefly.

Given a switching function F in the sum-of-products form (product-of-sums form), the following rules can be applied first to F to reduce it to a simpler form.

Rule 1 : If the function F is given in the sum-of-products form (product-of-sums form), drop the product terms (sum terms) for which the function F is equal to 0 (1).

Rule 2 : Delete all but one of the product terms (sum terms) with identical set of literals in the function.

Rule 3 : Delete the subsumed product (sum) terms from the function.

The procedure for transforming the disjunctive normal form of a function to a conjunctive normal form or vice versa is now given below.

Procedure

Step 1 : Using the distributive laws, transform a switching function F_1 given in the sum-of-products form into a switching function F_2 in the product-of-sums form or vice versa.

Step 2 : Apply rules 1, 2 and 3 in this order to the result of step 1.

It can be noticed that if the function is given in the product-of-sums form, the procedure given above is applied once to get the prime implicants of the given function whereas if the function is given in the sum-of-products form,

first the above procedure is applied to get all the prime implicates and then it is applied to get all the prime implicants of the function.

Theorem 1 : If the switching function F_1 given in the sum-of-products form is equivalent to another function F_2 which is also given in the sum-of-products form and F_1 is transformed by the procedure given above into another function F_3 given in the product-of-sums form, and also F_2 is transformed into F_4 given in the product-of-sums form, then F_3 is equal to F_4.

Theorem 2 : If the switching function G_1 given in the product-of-sums form is equivalent to another function G_2 which is also given in the product-of-sums form and G_1 is transformed by the procedure given above into another function G_3 given in the sum-of-products form, and also G_2 is transformed into G_4 given in the sum-of-products form, then G_3 is equal to G_4.

The following example illustrates the procedure.

Example : $F(x_1, x_2, x_3, x_4, x_5) = x_1x_2x_4x_5 + x_1x_2\bar{x}_5 + x_1\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_4 + x_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_3\bar{x}_4$

By applying the distributive laws to the given function F, we find that the function F is equivalent to F_1 where

$$F_1 = (\bar{x}_1 + x_2 + \bar{x}_3)(\bar{x}_1 + x_2 + \bar{x}_4)(\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_5)(\bar{x}_1 + x_2 + \bar{x}_4 + \bar{x}_5)$$
$$(\bar{x}_1 + x_2 + \bar{x}_3 + x_4 + \bar{x}_5)(x_1 + \bar{x}_4)(x_2 + \bar{x}_4)(x_1 + \bar{x}_4 + x_5)(x_2 + \bar{x}_4 + x_5)$$

After applying Step 2 of the procedure to F_1, we see that F_1 is equivalent to F_2, where

$$F_2 = (\bar{x}_1 + x_2 + \bar{x}_3)(x_1 + \bar{x}_4)(x_2 + \bar{x}_4), \text{ or}$$
$$F_2 = \bar{x}_1x_2\bar{x}_4 + \bar{x}_1\bar{x}_4 + x_1x_2 + x_1x_2\bar{x}_4 + x_2\bar{x}_4 + x_2\bar{x}_4 + x_1x_2\bar{x}_3 + x_1\bar{x}_3\bar{x}_4$$
$$+ x_2\bar{x}_3\bar{x}_4 + \bar{x}_3\bar{x}_4$$

After applying Step 2 again we see that F_2 is equivalent to F_3, where

$$F_3 = \bar{x}_1\bar{x}_4 + x_1x_2 + x_2\bar{x}_4 + \bar{x}_3\bar{x}_4$$

Thus the set of prime implicants P(F) of the given function F is given by :

$$P(F) = \{ x_1x_2, \bar{x}_1\bar{x}_4, x_2\bar{x}_4, \bar{x}_3\bar{x}_4 \}$$

2.7 Method of Das and Choudhury

Das and Choudhury [15] proposed a method for generating the prime implicants of switching functions starting from the maxterm form represented in decimal mode. In general, any function $F(x_n, x_{n-1}, \ldots\ldots, x_1)$ specified in the maxterm form can be written as

$$F(x_n, x_{n-1}, \ldots\ldots, x_1) = \pi S_i = (x_n + F_{x_n})(\bar{x}_n + F_{\bar{x}_n}) \quad \text{——————} \quad (1)$$

by expanding the function about the highest weighted variable x_n, F_{x_n} and $F_{\bar{x}_n}$ being each a residual function in maxterm form of the variables $(x_{n-1}, x_{n-2}, \ldots\ldots, x_1)$. Expression (1) can also be written as

$$F(x_n, x_{n-1}, \ldots\ldots, x_1) = x_nF_{\bar{x}_n} + \bar{x}_nF_{x_n} + F_{x_n}F_{\bar{x}_n} \quad \text{——————} \quad (2)$$

On examination of Expression (2) we see that if $F_{x_n} = F_{\bar{x}_n}$, then both the product terms $x_nF_{\bar{x}_n}$ and $\bar{x}_nF_{x_n}$ become redundant. Thus depending upon the nature of the residual functions, some of the terms in (2) may be spotted as redundant or null. By deleting the redundant and null terms, if any, the remaining product terms can again be expanded about the highest weighted variable of the remainder function and so on. In general, after the k-th expansion, the function F is in the form of a sum of partially expanded products as

$$F(x_n, x_{n-1}, \ldots\ldots, x_1) = p_1f_1 + p_2f_2 + \ldots\ldots + p_kf_k \quad \text{——————} \quad (3)$$

where each p_i is a product of literals of the variables $(x_n, x_{n-1}, \ldots\ldots, x_{n-k+1})$

and each f_i is a function in the maxterm form over the variables (x_{n-k}, x_{n-k-1}, ..., x_1). Some of these partially expanded products can be deleted as being redundant in the following way : if for some (i,j), $f_i = f_j$ and $p_i \subseteq p_j$ then $p_i f_i$ become redundant and can be removed from the sum. Also if for some i, f_i turns out to be zero, then $p_i f_i$ become null and can be removed from the sum.

The redundant and null terms are deleted at each stage of the expansion and the expansion of the residual functions is carried out until all the variables of the function are exhausted. This expansion and the elimination of the redundant and the null terms at each stage of expansion ultimately transforms the given function specified in the canonical product-of-sums form into a simple sum-of-products form. Each of the product terms in this sum-of-products expression represents a prime implicant of the given switching function.

The theory as given above can be very well adapted for determining the prime implicants of a switching function which is given in the decimal equivalent of the maxterm form. The algorithm given below explains the procedure for generating the prime implicants when the function is given in the decimal equivalent of the maxterms.

Algorithm

Step 1 : If the switching function F is given in the decimal equivalent of the minterm form, take the complement of the given function in the equivalent decimal mode.

Step 2 : Subtract each of the decimal numbers present in the complementary function from (2^n-1), where n represents the number of variables in the function F. These decimal numbers after subtraction represents the decimal equivalent of the maxterms if the original function was specified in the maxterm form.

Step 3 : Arrange the decimal numbers representing the given function

in a row.

Step 4 : Divide the decimal numbers into two groups, one headed with $\bar{x}_n$ and the other with x_n. The group headed with $\bar{x}_n$ contains the decimal numbers representing the function in the maxterm form which are smaller than the weight of x_n and the group headed with x_n contains the decimal numbers which are equal to or greater than the weight of x_n, first subtracting the weight of x_n from each.

Step 5 : Include a third group headed by a dash consisting of the decimal numbers occuring jointly in the groups headed by x_n and $\bar{x}_n$. Check the decimal numbers occuring in each of the three groups to spot the redundancy and the null terms. Delete the redundant and null terms, if any.

Step 6 : Repeat steps 4 and 5 for each of the variables of the function upto the least weighted one.

The prime implicants can be found by tracing the different paths in the same way as in the method of Scheinman.

Example : $F(x_4, x_3, x_2, x_1) = \Sigma\ (0,1,2,3,4,6,7,8,9,11,15)$

The complement of the above function is given by

$\bar{F}(x_4, x_3, x_2, x_1) = \Sigma\ (5,10,12,13,14)$

The equivalent decimal representation of the maxterm type expression of F is obtained by subtracting each term of $\bar{F}$ from $2^4-1 = 15$, as shown by

$F(x_4, x_3, x_2, x_1) = \pi\ (10,5,3,2,1)$

Figure 1 given below shows the worked out example given above. The redundant and the null terms produced in the expansion at each stage are deleted and the prime implicants of the function are determined by simply tracing a path back from the end to the start and noting the group headings. The set of prime implicants P(F) of the function worked out in Fig. 1 is given by

$$P(F) = \{\ \bar{x}_3\bar{x}_4,\ x_2\bar{x}_4,\ \bar{x}_1\bar{x}_4,\ \bar{x}_2\bar{x}_3,\ x_1\bar{x}_3,\ x_1x_2\ \}$$

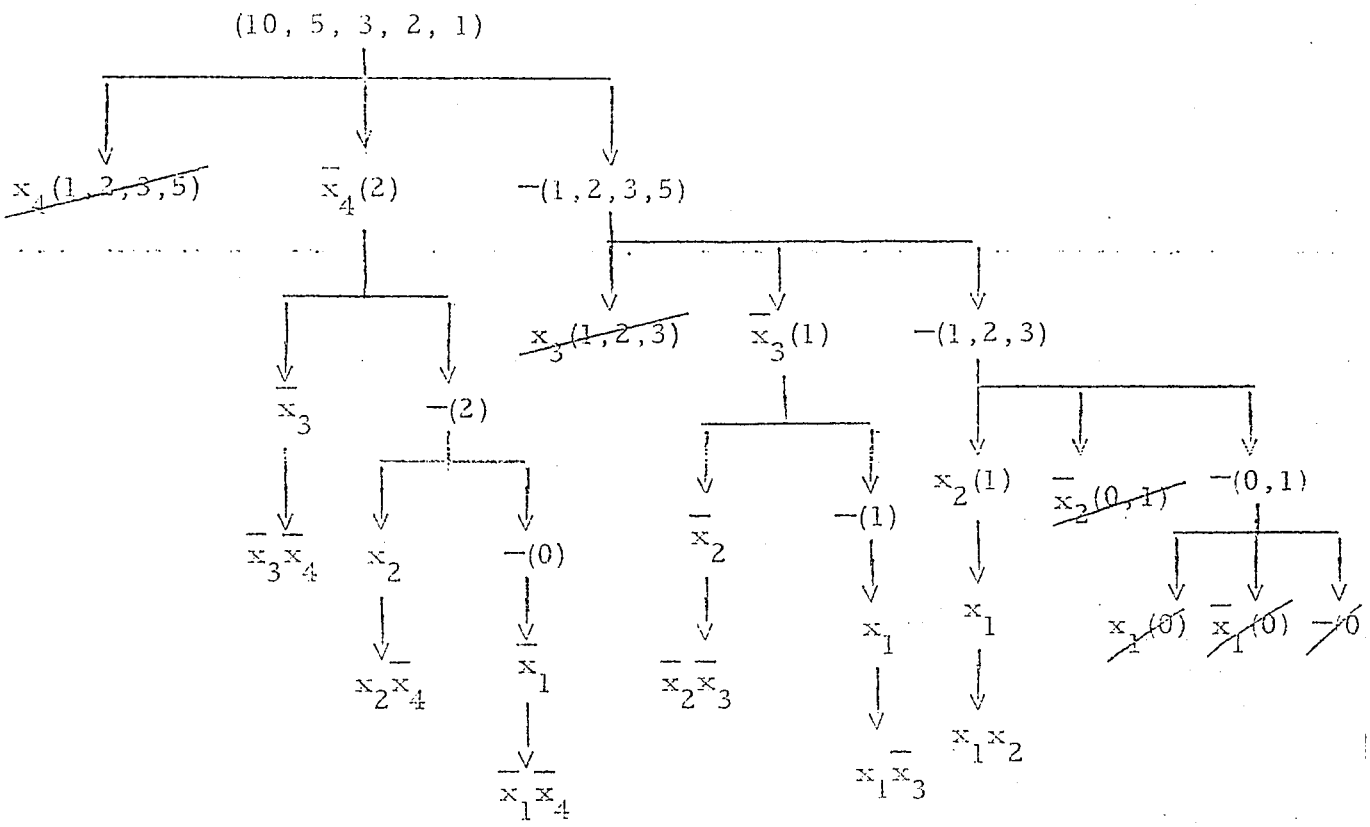


Fig 1 : Example : $F(x_4, x_3, x_2, x_1) = \Sigma\,(0,1,2,3,4,6,7,8,9,11,15)$

2.8 Method of Slagle, Chang and Lee

In this section an algorithm described by Slagle et al [17] for generating all the prime implicants of switching functions is discussed. If the switching function is given algebraically in the sum-of-products form or in the product-of-sums form, first the frequency ordering of the literals in the product or sum terms is found and then a process of expansion around the different variables in one or more levels is carried out through a series of trees called the sementic trees. If the function is given in the product-of-sums form, the algorithm is applied once to get all the prime implicants of the function, while if the function is specified in the sum-of-products form, the algorithm is applied first to get all the prime implicates

and then the algorithm is again applied to the set of all the prime implicates to get all the prime implicants of the given function.

Algorithm

Step 1 : Form a set S_F of all the clauses of the switching function given in the sum-of-products or product-of-sums form.

Step 2 : Delete the clauses containing a complementary pair from S_F, if any, and form a set S of the remaining clauses. If S is empty, Z_1 the product of no literals, is the only prime implicant of the function F and terminate the algorithm. If S is not empty, go to step 3.

Step 3 : Let S be the initial node and find the frequency ordering of the literals of S. Apply the sprouting from S with frequency ordering of the literals of S and let the nodes sprouted from S be $S_1, S_2, S_3, \ldots\ldots, S_r$. If there is any nonterminating node S_i, go to step 4.

Step 4 : Find the frequency ordering of the literals of S_i and apply the sprouting from S_i with the frequency ordering of the literals of S_i. Let $S_{i1}, S_{i2}, \ldots\ldots, S_{ir_i}$ be the nodes sprouted from S_i. Repeat step 4 until no more nonterminating node is left for sprouting, i.e. until each path in the tree leads to a terminating node. Get the final sementic tree by the application of above steps.

Step 5 : If there is no success node in the final sementic tree, Z_2 the sum of no literals, is the only prime implicant of function F and the set of prime implicants P(F) is given by $P(F) = \{Z_2\}$; terminate the algorithm. Otherwise, for each success node in the final sementic tree, collect the product of all the literals at the branches on the path from top down to the node of the final sementic tree and terminate the algorithm.

Example : $F(x_1, x_2, x_3, x_4) = (x_1 + x_2 + x_3)(x_1 + \bar{x}_2 + \bar{x}_4)$
$(x_1 + x_2 + \bar{x}_3 + x_4)(\bar{x}_1 + \bar{x}_2 + x_3 + \bar{x}_4)$

Step 1 : We form a set S_F of all the clauses of F as shown below :

$$S_F = \left\{ \begin{array}{l} (x_1 + x_2 + x_3) \\ (x_1 + \bar{x}_2 + \bar{x}_4) \\ (x_1 + x_2 + \bar{x}_3 + x_4) \\ (\bar{x}_1 + \bar{x}_2 + x_3 + \bar{x}_4) \end{array} \right\}$$

Step 2 : From the set S_F, we see that there is no complementary pair; hence $S = S_F$. Since S is not empty, thus Z_1 is not the only prime implicant of F.

Step 3 : The frequency ordering of the literals in S is given below.

$$x_1 \geq x_2 \geq \bar{x}_2 \geq x_3 \geq \bar{x}_4 \geq \bar{x}_1 \geq \bar{x}_3 \geq x_4$$

By applying the sprouting from S, the sementic tree is given in Fig. 2. In this sementic tree, S_1, S_2, S_3, S_4 are nonterminating nodes whereas S_5, S_6, S_7, S_8 are all terminating nodes.

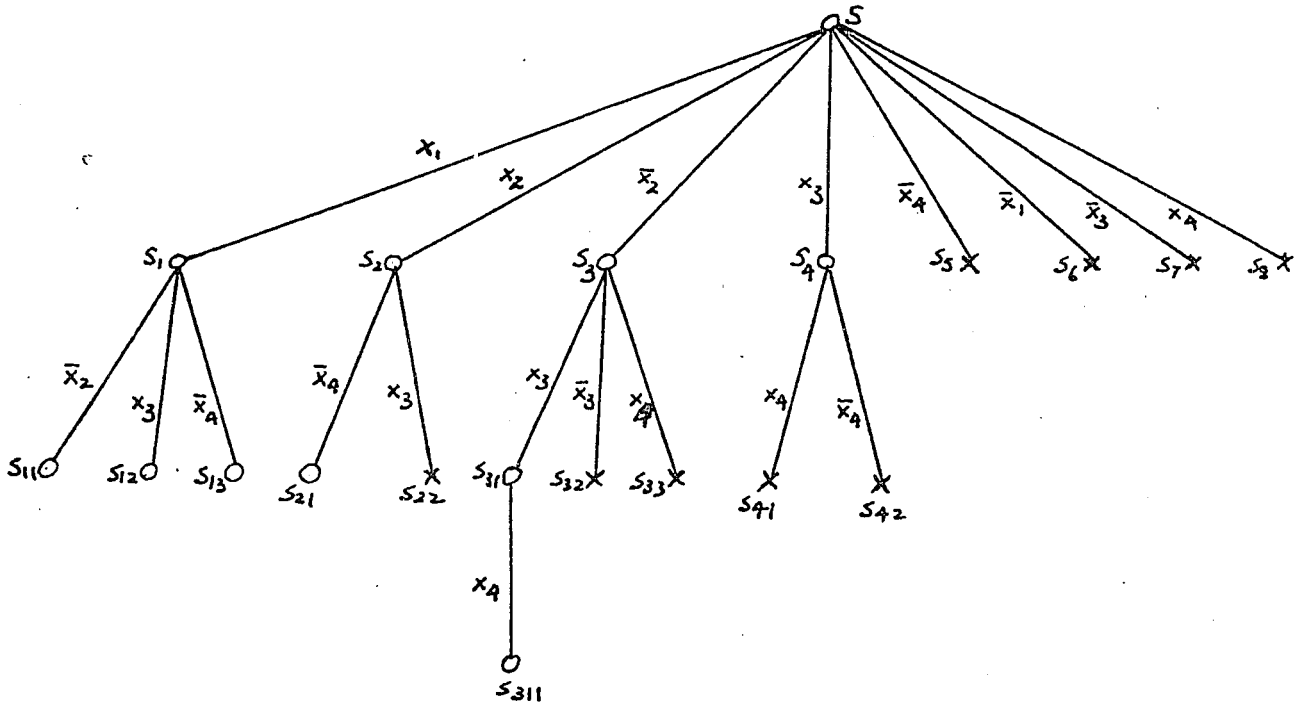


Fig. 2 : The final sementic tree for $F(x_1, x_2, x_3, x_4) = (x_1 + x_2 + x_3)$ $(x_1 + \bar{x}_2 + \bar{x}_4)$ $(x_1 + x_2 + \bar{x}_3 + x_4)$ $(\bar{x}_1 + \bar{x}_2 + x_3 + \bar{x}_4)$.

Step 4 : By applying step 4 to S_1, S_2, S_3 and S_4 nodes in the sementic tree, the final sementic tree is found as shown in Fig. 2. It can be noticed that S_{11}, S_{12}, S_{13}, S_{21} and S_{311} are the success nodes in the final sementic tree.

Step 5 : The set of prime implicants P(F) for the given function from the final sementic tree is given by

$$P(F) = \{ x_1\bar{x}_2,\ x_1x_3,\ x_1\bar{x}_4,\ x_2\bar{x}_4,\ \bar{x}_2x_3x_4 \}$$

Chapter 3

CLAUSE-COLUMN TABLE APPROACH FOR GENERATING ALL THE PRIME IMPLICANTS OF SWITCHING FUNCTIONS

3.1 Introduction

One of the important problems of combinational switching theory research is concerned with obtaining suitable algorithms for the simplification of switching functions in connection with the general problem of their economic realization. The usual approach for the solution of this simplification problem involves, in general, consideration of two distinct phases. In the first phase all the prime implicants of the function are found, while in the second phase, from this set of all the prime implicants, a minimal subset (according to some criterion of minimality) of prime implicants is selected such that their disjunction is equivalent to the function and from which none of the prime implicants can be dropped without sacrificing equivalence. Many different algorithms have been formulated for solving both the first and the second phase of this simplification problem [1] - [48]. In this chapter only this first phase of the simplification problem, that is, the problem of finding all the prime implicants of switching functions is considered.

Although there exist a good number of methods for determining the prime implicants of switching functions, the map methods of Veitch [1] and Karnaugh [2] and the tabular method of McCluskey [7], [21] which use the minterm form of switching functions, are the ones most popular. Recently developed methods of prime implicant determination include those of Scheinman [11], Hall [10], Das and Choudhury [15], and Slagle et al [17]. Scheinman developed a tabular technique for the generation of prime implicants of switching functions starting from the minterm type expression. In the method of Scheinman all the prime implicants of the function are not

generated sometime, though the prime implicants necessary for minimal solutions are always obtained. Further, one of the advantage of this method is that it is directly applicable for finding the minimal forms of the function as well. The binary sieve method as developed by Hall generates all the prime implicants starting from the minterm form of the function. Though the method is somewhat similar to the method of McCluskey, it differs considerably in procedure being based on relations disclosed by the action of a hypothetical sieve capable of shifting out 1's from any given binary column of a set of numbers. The method is quite suitable for manual computation with the decimal form of notation. An algebraic approach based primarily on successive expansion to generate all the prime implicants of a switching function utilizing the maxterm type expression was first proposed by Nelson [4]. One disadvantage of this method is that at every stage in the process of expansion, in general, a large number of redundant terms are generated which have to be eliminated at each stage by comparison with all the other terms. This basic idea of Nelson was subsequently utilized by Das and Choudhury in developing a tabular method for a more efficient generation of all the prime implicants of switching functions starting from the maxterm type expression represented in decimal mode. Das and Choudhury also extended their tabular method for generating prime implicants of functions having many unspecified or don't care terms. Slagle et al describe an algorithm for the generation of prime implicants of switching functions, which does not generate the same prime implicant more than once, though the algorithm sometimes generates some nonprime implicants, does not need large capacity of memory for implementation on a digital computer, and works equally well with either the conjunctive or the disjunctive (both canonical and noncanonical) form of the function. The function being specified algebraically in sum-of-products or product-of-sums forms, the basic approach of the authors consists in first finding the frequency ordering of the different literals in the product or sum terms and next carrying out a process of expansion around the different variables in one or more levels through a series of trees called semantic trees.

In this chapter a simple iterative procedure for the generation of prime implicants is developed which utilizes a new tabular mode of functional representation called clause-column table. In a clause-column table the set of literals or clauses defining a product or sum term in the sum-of-products or product-of-sums form of a function appears in a column, the total number of columns in the table being equal to the number of such product or sum terms. This form of representation not only reveals some interesting properties of switching functions, but also substantially aids in the process of generation of the prime implicants. Though the basic principle utilized in the procedure is that of successive expansion around the literals as used in some of the other existing methods, the way the expansion is carried out is entirely different and reduces the generation of redundant terms to a great extent. For a switching function specified in the product-of-sums form the procedure can be applied once to get all the prime implicants, while for a function specified in the sum-of-products form, the procedure is applied first to get all the prime implicates and is next applied to this set of prime implicates to get the prime implicants of the function.

3.2 Some Basic Definitions and Notations

In the present section we include, in brief, some of the basic definitions as well as the system of notations that will be used in this chapter.

We assume that a switching function F is defined in terms of n independent binary variables $(x_1, x_2, \ldots\ldots, x_n)$.

Definition : A variable, either complemented or uncomplemented, as it occurs in a switching function is called a literal. A simple product (simple sum) is a product or conjunction (sum or disjunction) of a subset of m variables, $1 \leq m \leq n$, each variable being either complemented or uncomplemented. A fundamental product or minterm (fundamental sum or maxterm) is a simple product (simple sum) of all the n distinct variables, each variable being either complemented or uncomplemented.

Example : $F(x_1, x_2, x_3) = (x_1 + x_2 + x_3)(\bar{x}_1 + x_2 + \bar{x}_3)(\bar{x}_2 + x_3)$

$= x_1\bar{x}_2\bar{x}_3 + x_2x_3 + \bar{x}_1x_3$

In the example, $(\bar{x}_2 + x_3)$ is a simple sum, whereas $(x_1+x_2+x_3)$ is a fundamental sum. Similarly, x_2x_3 is a simple product, while $x_1\bar{x}_2\bar{x}_3$ is a fundamental product.

Definition : An implicant p (implicate s) of a switching function F is a simple product (simple sum) that can be found in any sum-of-products (product-of-sums) form of F. Obviously then, p implies F (F implies s). An implicant p_i (implicate s_i) is a prime implicant (prime implicate) of a switching function F if and only if p_i implies F (F implies s_i), and no other implicant p_j (implicate s_j) of F exists such that p_i implies p_j (s_j implies s_i).

Example : $F(x_1,x_2,x_3) = \bar{x}_1 + x_1x_2x_3 = (\bar{x}_1+\bar{x}_2+x_3)(\bar{x}_1 + x_2)$.

In the above example, $x_1x_2x_3$ is an implicant of F, while $\bar{x}_1$ is a prime implicant of F. Likewise, $(\bar{x}_1+\bar{x}_2+x_3)$ is an implicate of F, whereas $(\bar{x}_1 + x_2)$ is a prime implicate of F.

Definition : A clause is a set of literals that occur in a simple product (simple sum) in any sum-of-products (product-of-sums) form of a switching function.

3.3 Switching Functions and Clause-Column Table Representations

For a switching function specified algebraically in the sum-of-products or product-of-sums form, canonical or noncanonical, the basic information can be conveniently portrayed in an alternative form of representation called clause-column table, as defined below.

Definition : Given a switching function F of n independent variables $(x_1,x_2,\ldots,x_n)$ in the sum-of-products (product-of-sums) form, canonical or noncanonical, a table can be readily made in the following manner. Designate the different simple product (simple sum) terms in the sum-of-products (product-of-sums) form of F as $a_1, a_2,\ldots,a_m$, where m represents the total number of simple product (simple sum) terms in F. Write all the simple product (simple sum) terms $a_1,a_2,\ldots,a_m$ in a row, and below each of the

product (sum) terms make an entry of the set of literals or clause it consists of, in a columnar form. Each of the columns headed by a specific product (sum) term then gives a collection of literals included in that product (sum) term. Such a table is called a clause-column table. This table contains the complete information as contained in the original sum-of-products (product-of-sums) form of the function, in a readily recognizable manner.

Example : $F(x_1,x_2,x_3,x_4) = (x_1 + x_2+x_3)\,(\bar{x}_1+x_2+x_4)$
$(x_1+\bar{x}_3+\bar{x}_4)\,(\bar{x}_1+\bar{x}_2+x_3+x_4)$.

The clause - column table representation of the function is given in Table I.

Table I. Clause-column table

a_1	a_2	a_3	a_4
x_1	$\bar{x}_1$	x_1	$\bar{x}_1$
x_2	x_2	$\bar{x}_3$	$\bar{x}_2$
x_3	x_4	$\bar{x}_4$	x_3
			x_4

Theorem 1 : In the clause-column table representation of a switching function there cannot exist a column with repeated occurrences of the same literal.

Proof : The theorem readily follows, since the literals that occur in a column of the clause-column table correspond to those that appear either in a simple product (when the function is given in the sum-of-products form) or in a simple sum (when the function is given in the product-of-sums form) term of the function that have all their literals distinct.

Theorem 2 : In the clause-column table representation of a switching function there cannot exist a column that contains the same variable in both complemented and uncomplemented forms.

Proof : The set of literals that occur in a simple product or in a simple sum term cannot contain, by definition, the same variable in both complemented and uncomplemented forms, and accordingly no variable can appear in a column of the clause-column table in both of these forms.

When a switching function is not specified in the canonical sum-of-products or product-of-sums form, then in the clause-column table representation of the function we may have, in general, 1) some of the columns consisting of only single literals; 2) some of the columns with identical sets of literals; 3) some of the columns with literals included in the sets of (i.e. forming proper subsets of) literals appearing in some other columns; 4) columns having some or none of the literals common between them.

Theorem 3: If in the clause-column table representation of a switching function specified in the sum-of- products (product-of-sums) form there appears a single literal in any column, then that literal is an essential literal and must appear in all the prime implicates (prime implicants) of the function.

Proof : A literal that appears alone in a column of the clause-column table can be considered as a product (sum) term consisting of a single literal in the given sum-of-products (product-of-sums) form of the function and thus must necessarily appear in all the prime implicates (prime implicants) of the function.

Definition : If some of the columns in a clause-column table have literals that are included in the sets of (i.e. forming proper subsets of) literals appearing in some other columns then the phenomenon of column dominance occurs, where the columns with lesser numbers of literals dominate the columns with larger numbers of literals [7] .

Theorem 4 : If in the clause-column table representation of a switching function there are columns with essential literals, columns with identical sets of literals, and columns with literals included in the sets of literals appearing in some other columns giving rise to column dominance, then the size of the clause-column table can be reduced by selecting the essential literals and deleting the columns in which these essential literals

are present, all but one of the columns having identical sets of literals, and columns with larger numbers of literals (dominated columns) in the case of column dominance.

Proof : Once the essential literals are selected, all the columns in which the essential literals occur can evidently be deleted. Similarly, all but one of the columns with identical sets of literals and the dominated columns can be deleted from the clause-column table because these columns contain redundant information.

It may be noted that in the clause-column table representation of a switching function specified in the canonical sum-of-products or product-of-sums form, obviously there cannot be columns with essential literals, or columns with identical sets of literals, or columns with literals included in the sets of literals appearing in some other columns giving rise to column dominance.

Definition : Given a clause-column table, if some of its columns are deleted, the resulting table is called a reduced clause-column table.

Example : $F(x_1, x_2, x_3, x_4) = x_1(x_1 + x_2 + x_4)(x_1 + \bar{x}_2 + \bar{x}_4)$
$(\bar{x}_1 + \bar{x}_3 + x_4)(\bar{x}_1 + x_2 + \bar{x}_3 + x_4)(x_3 + x_4)$.

The clause-column table representation of the function is shown in Table II.

Table II. Clause-column table

a_1	a_2	a_3	a_4	a_5	a_6
x_1	x_1	x_1	$\bar{x}_1$	$\bar{x}_1$	x_3
	x_2	$\bar{x}_2$	$\bar{x}_3$	x_2	x_4
	x_4	$\bar{x}_4$	x_4	$\bar{x}_3$	
				x_4	

An observation of the clause-column table shows that the literal x_1

is an essential literal, and the column a_4 dominates the column a_5 in the table. By deleting the columns of the clause-column table in which the essential literal x_1 occurs and also the dominated column, the reduced clause-column table is obtained, as shown in Table III.

Table III. Reduced clause-column table

a_4	a_6
$\bar{x}_1$	x_3
$\bar{x}_3$	x_4
x_4	

Definition : For any two distinct literals x_i^* and x_j^* (where the asterisk * indicates either complemented or uncomplemented forms of the variables) appearing in a clause-column table, if the literal x_i^* appears in more columns in the table than the literal x_j^*, then an ordering of the literals can be made as $x_i^* \geq x_j^*$. If both the literals x_i^* and x_j^* appear in the same number of columns in the clause-column table, the ordering can be made either as $x_i^* \geq x_j^*$ or as $x_j^* \geq x_i^*$. This kind of ordering ($\geq$) that can be established amongst different literals in a clause-column table is called the frequency ordering of the literals [17].

Example : Consider the clause-column table as given in Table IV.

Table IV. Clause-column table

a_1	a_2	a_3	a_4
x_1	x_1	$\bar{x}_1$	x_1
x_2	$\bar{x}_3$	$\bar{x}_2$	x_2
x_4	x_4	x_3	x_3
		$\bar{x}_4$	

To find the frequency ordering of the literals x_1, x_2, $\bar{x}_3$, $\bar{x}_4$ of the clause-column table, we note that the literal x_1 appears in three columns a_1, a_2, and a_4, the literal x_2 appears in two columns a_1 and a_4, whereas the literal $\bar{x}_3$ appears in only one column a_2, and the literal $\bar{x}_4$ also appears in only one column a_3. Thus the frequency ordering of the literals x_1, x_2, $\bar{x}_3$, $\bar{x}_4$ can be made as

$$x_1 \geq x_2 \geq \bar{x}_3 \geq \bar{x}_4 \text{ , or } x_1 \geq x_2 \geq \bar{x}_4 \geq \bar{x}_3 .$$

3.4 Clause-Column Table Representations and Prime Implicates (Prime Implicants) of Switching Functions

Given a switching function in the sum-of-products (product-of-sums) form, canonical or noncanonical, all its prime implicates (prime implicants) can be found readily by forming the clause-column table representation of the function and carrying out a process of successive expansion around the different literals according to the laws of Boolean algebra. But this procedure of direct expansion has the disadvantage of generating many redundant as well as null terms at every phase of the process. The generation of redundant terms may be reduced to some extent by frequency ordering of the literals and starting the initial expansions around literals which appear in maximum number of columns of the table.

Example : Consider the clause-column table as given in Table I. From an inspection of this table we see that each of the literals x_1, $\bar{x}_1$, x_2, x_3, x_4 appears in two different columns of the table, whereas the remaining literals $\bar{x}_2$, $\bar{x}_3$, $\bar{x}_4$ appear each in only one column. So the frequency ordering of the different literals in the table can be obtained as

$$x_1 \geq \bar{x}_1 \geq x_2 \geq x_3 \geq x_4 \geq \bar{x}_2 \geq \bar{x}_3 \geq \bar{x}_4 .$$

On the basis of this frequency ordering of the literals we may next search for a column of the table that contains a maximum number of those literals that occur in maximum number of columns each. We see that both the

columns a_1 and a_2 contain literals all of which appear in two different columns of the table each. We choose column a_1 and expand the table around literals x_1, x_2, x_3, which gives

1) x_1
$$\begin{bmatrix} a_2 & a_4 \\ \bar{x}_1 & \bar{x}_1 \\ x_2 & \bar{x}_2 \\ x_4 & x_3 \\ & x_4 \end{bmatrix},$$

2) x_2
$$\begin{bmatrix} a_3 & a_4 \\ x_1 & \bar{x}_1 \\ \bar{x}_3 & \bar{x}_2 \\ \bar{x}_4 & x_3 \\ & x_4 \end{bmatrix},$$

3) x_3
$$\begin{bmatrix} a_2 & a_3 \\ \bar{x}_1 & x_1 \\ x_2 & \bar{x}_3 \\ x_4 & \bar{x}_4 \end{bmatrix}.$$

Each of the three reduced tables in the above can again be similarly expanded in turn. Considering the first one of the three tables, for example, we see that the different literals in the table have a frequency ordering given by

$$\bar{x}_1 \geq x_4 \geq x_2 \geq \bar{x}_2 \geq x_3 .$$

On the basis of the frequency ordering, selecting the column a_2 (though both the columns a_2 and a_4 have equal preference as far as the frequency ordering of literals is concerned, it is desirable to select the column a_2 since it contains lesser number of literals) and expanding the table around literals $\bar{x}_1$, x_2, x_4, we get

a) $x_1 \cdot \bar{x}_1$,

b) $x_1 \cdot x_2 \begin{bmatrix} a_4 \\ \bar{x}_1 \\ \bar{x}_2 \\ x_3 \\ x_4 \end{bmatrix}$,

c) $x_1 \cdot x_4$.

Of these total of six terms, three terms are null and one term is redundant as can be seen readily.

The above example serves to illustrate how the frequency ordering of literals can be used to somewhat reduce the number of redundancies generated in the process of expansion of a clause-column table, though null terms may still be generated. An altogether different scheme of expansion in conjunction with frequency ordering of literals can drastically reduce the number of redundant terms generated and can completely avoid generation of null terms. The following theorems discuss this procedure of expansion.

Theorem 5 : Let x_i^* be a literal appearing in a column a_k in the clause-column table representation of a switching function F given in the product-of-sums form. Let f_1 be the function representing the reduced clause-column table obtained by deleting all the columns containing the literal x_i^* and by deleting the literal $\bar{x}_i^*$, if any, from all the remaining columns, and

let f_2 be the function representing the reduced clause-column table obtained from the original table by deleting the literal x_i^* from all the columns in which it appears. The function F can then be expressed as

$$F = x_i^* f_1 + f_2 .$$

Proof: Assume that the literal x_i^* occurs in r simple sums in the given product-of-sums form of F as

$$(x_i^* + s_1)(x_i^* + s_2) \ldots (x_i^* + s_r) ,$$

where each s_i, $i = 1, 2, \ldots, r$ represents a sum term of $(n-1)$ or fewer literals. If we denote the product of these r simple sum terms as f_r, then f_r can be written as

$$\begin{aligned} f_r &= (x_i^* + s_1)(x_i^* + s_2) \ldots (x_i^* + s_r) \\ &= x_i^* + s_1 s_2 \ldots s_r \\ &= x_i^* + f_s , \end{aligned}$$

where $f_s = s_1 s_2 \ldots s_r$. Since $F = (x_i^* + s_1)(x_i^* + s_2) \ldots (x_i^* + s_r)\,[S]$ where S represents the product of the remaining sum terms of F, we have

$$\begin{aligned} F &= (x_i^* + f_s)\,[S] \\ &= x_i^* S + f_s S . \end{aligned}$$

Now in expanding $x_i^* S$, null terms of the form $x_i^* \bar{x}_i^*$ will be generated by the presence of $\bar{x}_i^*$ in the sum terms in S. To avoid generation of these null terms, we can delete $\bar{x}_i^*$, if any, from all the sum terms of S. Let the resultant product-of-sums expression be denoted as S'. Hence F can be written as

$$F = x_i^* S' + f_s S .$$

But we have, by definition, $S' = f_1$ and $f_s S = f_2$. Thus

$$F = x_i^* f_1 + f_2 .$$

Corollary 5.1 : Expand the clause-column table representation of the function F following the procedure of theorem 5 until all the columns of the table are deleted. In the final set of terms obtained, compare the different terms for redundancy and delete redundant terms, if any. The resultant set corresponds to the terms which are all irredundant products and represents all the prime implicants R_p of F.

Theorem 6 : Let a switching function F be given in the sum-of-products form. Consider the clause-column table representation of the function F as that of its dual function F^D which is in product-of-sums form, and expand the table according to the procedure of theorem 5 until all the columns of the table are deleted. In the final set of terms, the deletion of all the redundant terms results in a set of irredundant products whose duals give all the prime implicates R_s of F.

Proof : For a switching function F given in the sum-of-products form, the clause-column table representation of F is the same as that of its dual function F^D which is in product-of-sums form. So if the clause-column table representation of F is expanded according to the procedure of theorem 5 and in the resultant set of terms, all the redundant terms are deleted, we are left with a set of product terms which represent all the prime implicants R_p of the dual function F^D. Now the prime implicates of F are the complements of the prime implicants of its complementary function $\overline{F}$, and consequently are the duals of the prime implicants of its dual function F^D.

If in the above theorems the choice of the literals selected at each stage of expansion is arrived at on the basis of the frequency ordering of the literals, very few redundant terms are in effect generated.

This expansion scheme is a modification of a branching technique often used in deriving the irredundant solutions from a prime implicant table.

We give below an algorithm to generate all the prime implicants of switching functions.

Algorithm

Step 1 : Given a switching function in the sum-of-products or product-of-sums form, canonical or noncanonical, form the clause-column table representation of the function.

Step 2 : Read the clause-column table. Select the essential literals, if any, and delete all those columns of the table containing the essential literals. If x_j^* is an essential literal, also delete the literal $\bar{x}_j^*$, if any, from all the remaining columns[1], and delete all but one of the columns with identical sets of literals, and dominated columns, if any. Form a reduced clause-column table consisting of the remaining literals of those columns from which literals are deleted and the rest of the columns, if any. Repeat step 2 iteratively until either all the columns are deleted, or there are no columns with essential literals, columns with identical sets of literals, and dominated columns in the reduced table.

Step 3 : Find the frequency ordering of the literals in the reduced clause-column table.

Step 4 : Select the literal x_i^* that occurs in the maximum number of columns in the reduced clause-column table. If more than one literal occur in the maximum number of columns of the table, select any one of these literals, x_i^* .

(a) Delete all the columns of the table in which the literal x_i^* appears, and also delete the literal $\bar{x}_i^*$, if any, from all the remaining columns.

---

1. See page 53.

Form a reduced clause-column table consisting of the remaining literals of the columns from which $\bar{x}_i^*$ is deleted and the rest of the columns, if any. Go to step 2.

(b) Delete the literal x_i^* from all the columns of the reduced clause-column table in which it appears, and form a reduced clause-column table consisting of the remaining literals of these columns and the rest of the columns, if any, that do not contain x_i^*. Go to step 2.

Step 5 : Follow steps 2 through 4 iteratively until all the columns of the clause-column table are deleted. In the final set of terms obtained, compare the different terms for redundancy. If there exist terms as A and AB, where A and B are simple products, then the term AB is redundant. Delete the redundant terms, if any.

(a) If the function F is given in the product-of-sums form, the set of irredundant products that remain represents all the prime implicants R_p of F. Stop.

(b) If the function F is given in the sum-of-products form, the set of irredundant products that remain represents all the prime implicants R_p of its dual function F^D. Take duals of these prime implicants and obtain an expression for F in the product-of-sums form. Go to step 1.

The flow chart of the algorithm is shown in Fig. 1.

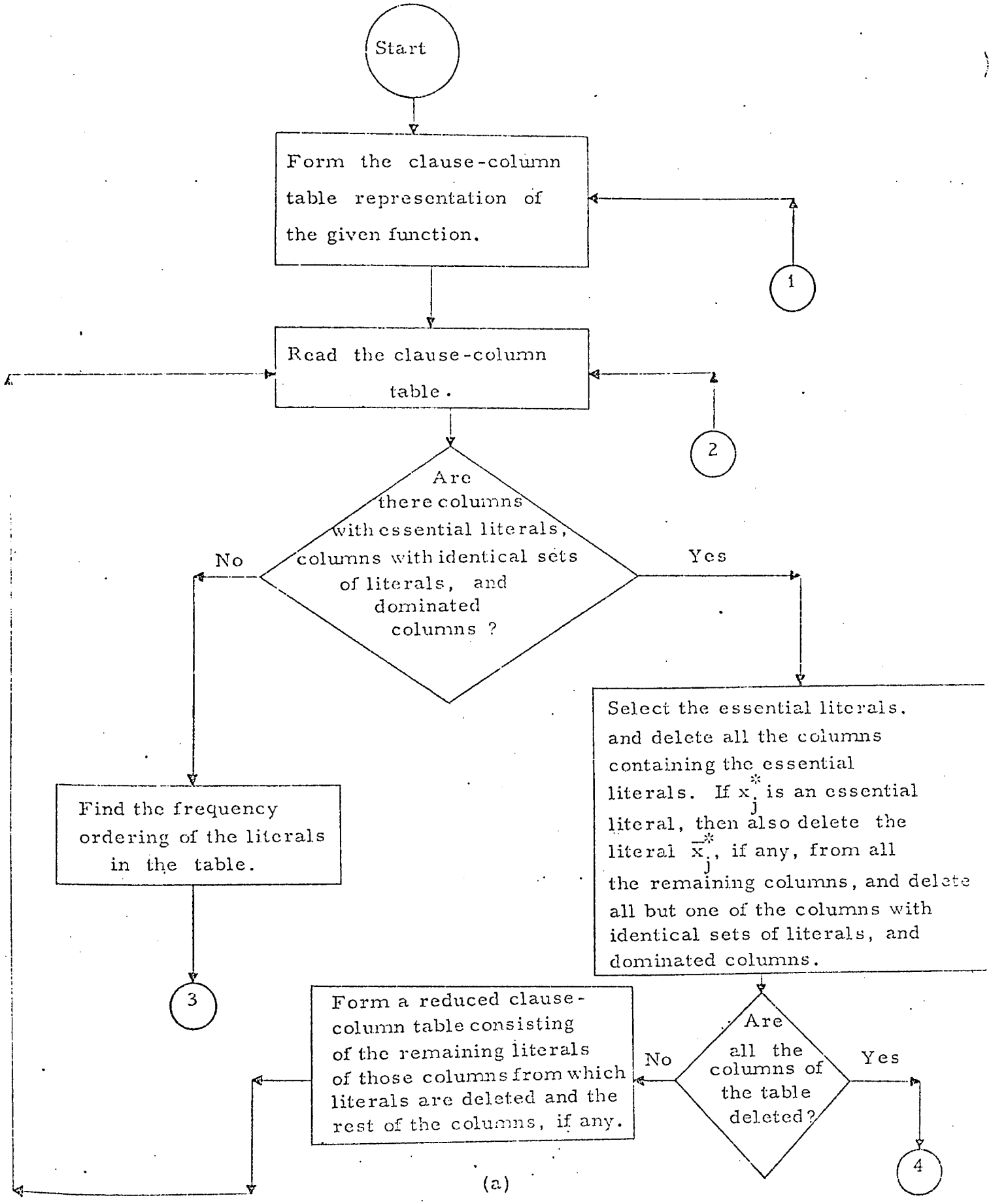
Example: $F(x_1,x_2,x_3,x_4) = (x_1+x_2+x_3)\,(x_1+\bar{x}_2+\bar{x}_4)\,(x_1+x_2+\bar{x}_3+x_4)$

$$(\bar{x}_1+\bar{x}_2+x_3+\bar{x}_4)\,.$$

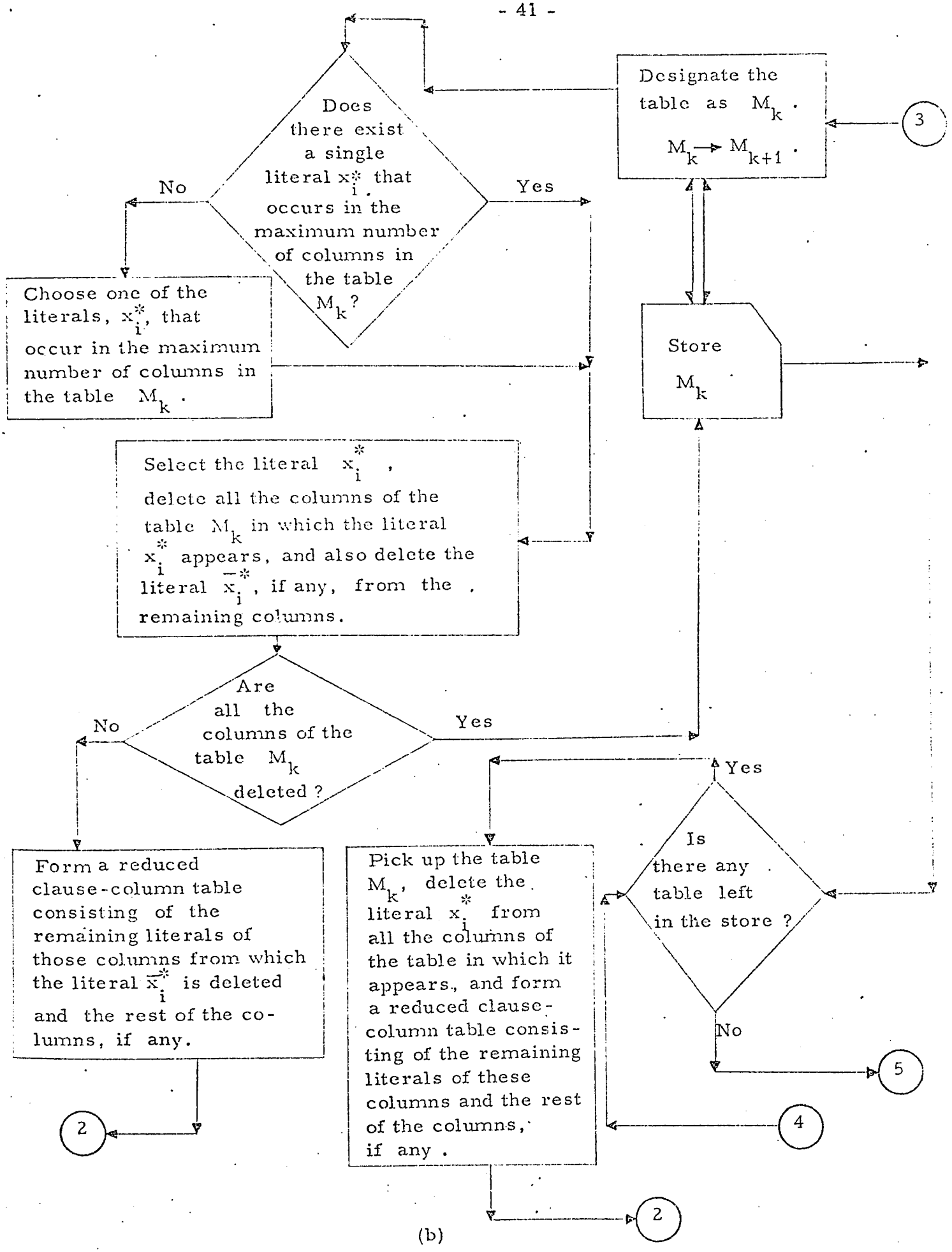
Step 1 : We form the clause-column table representation of the function as shown in Table V.

Table V. Clause-column table

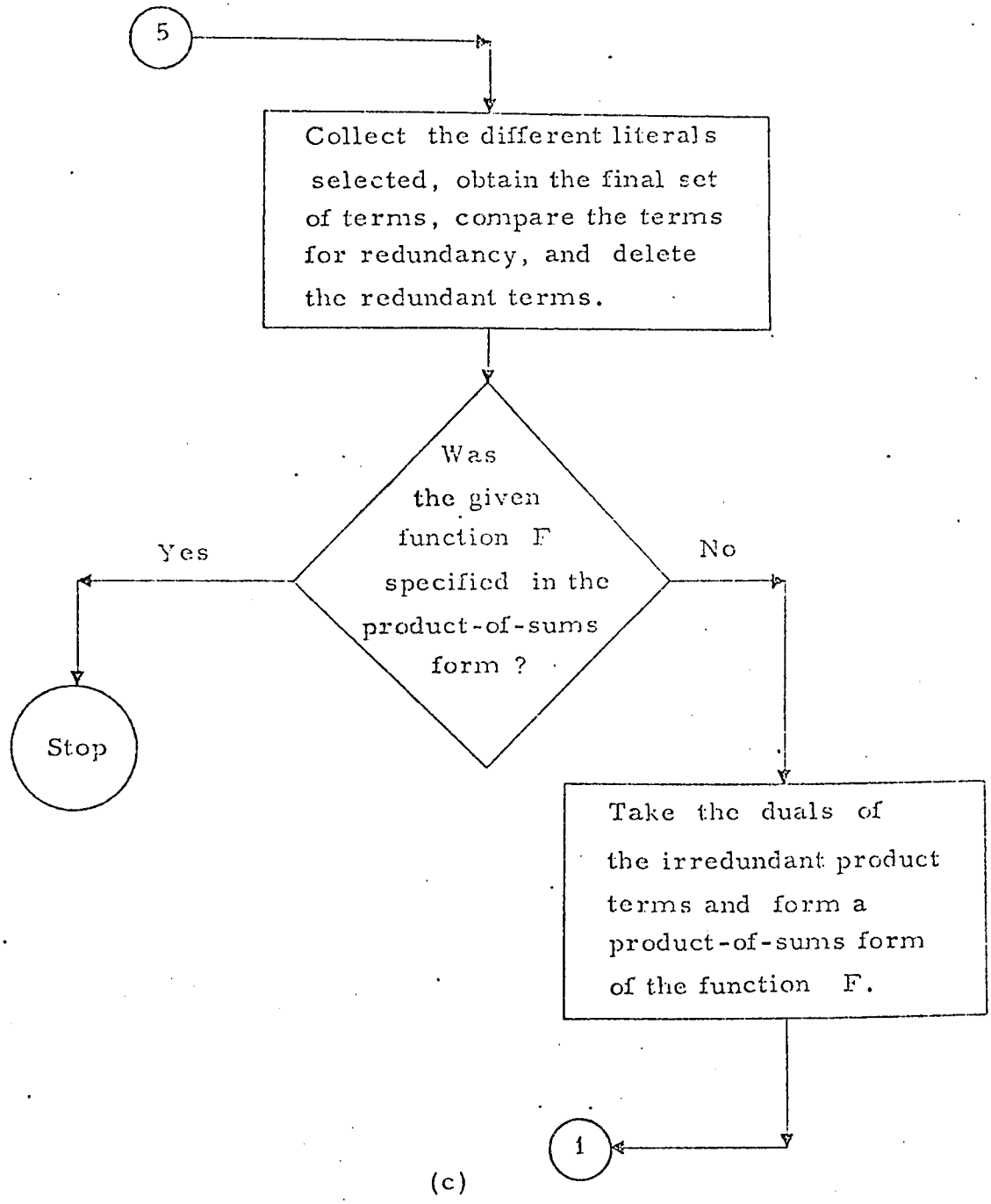
a_1	a_2	a_3	a_4
x_1	x_1	x_1	$\bar{x}_1$
x_2	$\bar{x}_2$	x_2	$\bar{x}_2$
x_3	$\bar{x}_4$	$\bar{x}_3$	x_3
		x_4	$\bar{x}_4$


Start
Form the clause-column table representation of the given function.
1
Read the clause-column table.
2
Are there columns with essential literals, columns with identical sets of literals, and dominated columns ?
No
Yes
Find the frequency ordering of the literals in the table.
3
Select the essential literals, and delete all the columns containing the essential literals. If x_j^* is an essential literal, then also delete the literal $\overline{x}_j^*$, if any, from all the remaining columns, and delete all but one of the columns with identical sets of literals, and dominated columns.
Form a reduced clause-column table consisting of the remaining literals of those columns from which literals are deleted and the rest of the columns, if any.
Are all the columns of the table deleted?
No
Yes
4

(a)


Designate the table as M_k. $M_k \rightarrow M_{k+1}$.
3
Does there exist a single literal x_i^* that occurs in the maximum number of columns in the table M_k?
No
Yes
Choose one of the literals, x_i^*, that occur in the maximum number of columns in the table M_k.
Store M_k
Select the literal x_i^*, delete all the columns of the table M_k in which the literal x_i^* appears, and also delete the literal $\bar{x}_i^*$, if any, from the remaining columns.
Are all the columns of the table M_k deleted?
No
Yes
Yes
Is there any table left in the store?
No
5
Form a reduced clause-column table consisting of the remaining literals of those columns from which the literal $\bar{x}_i^*$ is deleted and the rest of the columns, if any.
2
Pick up the table M_k, delete the literal x_i^* from all the columns of the table in which it appears, and form a reduced clause-column table consisting of the remaining literals of these columns and the rest of the columns, if any.
4
2

(b)



(c)

Fig. 1. The flow chart of the algorithm.

Step 2 : From an observation of the clause-column table in Table V we see that there do not exist columns with essential literals, columns with identical sets of literals, and dominated columns in the table.

Step 3 : We find the frequency ordering of the literals in Table V as given below.

$$x_1 \geq x_2 \geq \bar{x}_2 \geq x_3 \geq \bar{x}_4 \geq \bar{x}_1 \geq \bar{x}_3 \geq x_4 .$$

Step 4 : We select the literal x_1 that occurs in the maximum number of columns in the clause-column table.

(a) We delete all the three columns of Table V in which the literal x_1 appears, and also delete the literal $\bar{x}_1$ from the remaining column a_4. We are left with the reduced table shown below,

$$x_1 \begin{bmatrix} a_4 \\ \bar{x}_2 \\ x_3 \\ \bar{x}_4 \end{bmatrix}$$

which gives three product terms : (i) $x_1\bar{x}_2$, (ii) x_1x_3, (iii) $x_1\bar{x}_4$.

(b) Next we delete the literal x_1 from all the three columns of Table V in which it appears, and form a reduced clause-column table as shown below.

Table VI. Reduced clause-column table

a_1	a_2	a_3	a_4
x_2	$\bar{x}_2$	x_2	$\bar{x}_1$
x_3	$\bar{x}_4$	$\bar{x}_3$	$\bar{x}_2$
		x_4	x_3
			$\bar{x}_4$

At this stage we go back to step 2 again.

Step 2' : From an inspection of Table VI we see that the column a_4 is dominated by the column a_2 of the table. We delete the column a_4 and form a reduced clause-column table as shown in Table VII.

Table VII. Reduced clause-column table

a_1	a_2	a_3
x_2	$\bar{x}_2$	x_2
x_3	$\bar{x}_4$	$\bar{x}_3$
		x_4

Step 3' : We find the frequency ordering of the literals in Table VII as shown below.

$$x_2 \geq \bar{x}_2 \geq x_3 \geq \bar{x}_3 \geq x_4 \geq \bar{x}_4 .$$

Step 4' : We select the literal x_2 that occurs in the maximum number of columns in Table VII.

(a) We delete the two columns of the table in which the literal x_2 appears, and also delete the literal $\bar{x}_2$ from the remaining column a_2. We obtain the reduced table

$$x_2 \begin{bmatrix} a_2 \\ \bar{x}_4 \end{bmatrix}$$

which gives the product term : (iv) $x_2\ \bar{x}_4$.

(b) We next delete the literal x_2 from all the columns of Table VII in which it appears, and form a reduced clause-column table as presented in Table VIII .

Table VIII. Reduced clause-column table

a_1	a_2	a_3
x_3	$\bar{x}_2$	$\bar{x}_3$
	$\bar{x}_4$	x_4

We again go back to step 2.

Step 2" : We select the essential literal x_3 from Table VIII, delete the column a_1 containing the literal x_3, and also delete the literal $\bar{x}_3$ from the column a_3. We get the reduced table

$$x_3 \quad \begin{bmatrix} a_2 & a_3 \\ \bar{x}_2 & x_4 \\ \bar{x}_4 & \end{bmatrix}.$$

from which we get

$$x_3\ x_4 \quad \begin{bmatrix} a_2 \\ \bar{x}_2 \end{bmatrix}$$

which results in the product term : (v) $\bar{x}_2\, x_3\, x_4$.

Finally comparing all the five product terms that have been obtained we see that none of them is redundant. Thus all the five product terms represent the prime implicants of the given function.

3.5 Prime Implicants of Functions Having a Large Number of Unspecified or Don't Care Terms

The procedure as described above for generating the prime implicants of switching functions can be readily modified to generate the prime implicants of switching functions having a large number of unspecified fundamental products or don't care terms. The available methods for the generation of

prime implicants of switching functions involving don't care states usually use either the input conditions for which the functions are equal to 1 (called the specified or S-terms) and the input conditions for which the functions are unspecified (called the don't care or d-terms), or the input conditions for which the functions are equal to 0 (called the prohibited or P-terms) and the d-terms. When the functions involve a very large number of unspecified fundamental products or d-terms, these methods turn out to be too laborious, and generate many prime implicants which are formed solely of d-terms and are thus of no importance as far as finding of minimal sums or products is concerned. A significant contribution in this direction was first made by McCluskey who suggested a very novel approach for generating the prime implicants of functions involving a large number of unspecified fundamental products from a knowledge of only the specified and prohibited terms (S-terms and P-terms) of the functions [14]. His method not only avoids the generation of prime implicants formed solely of d-terms, but offers the additional advantage of readily spotting the essential prime implicants of the functions, if any. Subsequently this idea of McCluskey was utilized by Das and Choudhury [15] in developing a tabular technique for the generation of prime implicants of this class of functions starting from the maxterm type expression of the functions represented in decimal mode.

Let the S-terms and P-terms of the function be given in the form of matrices called the S-matrix and P-matrix respectively. Now in order that a prime implicant of the function $F_S + F_d$ comprised of S-terms and d-terms may not be formed solely of d-terms, it is essential that the prime implicant and the rows of the S-matrix must have some input combinations in common, which means that they should have no conflicting entries. Wherever there will be a 1 (0) in the S-matrix, there must be a 1 or '-' (0 or '-') in the prime implicant. This idea can be very well incorporated in the suggested algorithm to avoid the generation of or to spot the prime implicants formed entirely of d-terms. If we start with the function F_P formed of P-terms only,

then taking the complements of these P-terms gives the function $F_S + F_d$ in the canonical product-of-sums form. The function having a large number of d-terms, the number of P-terms is thus comparatively small, and accordingly the number of sum terms in the function $F_S + F_d$ is also small. In the process of generating the prime implicants by using the aforementioned algorithm starting from the function $F_S + F_d$ whenever a set of literals is selected along any branch that corresponds to a combination of variables conflicting with all the entries of the S-matrix, then further expansion of the table along that branch can be stopped, because further progress along that branch will result in the generation of product terms that will correspond to the prime implicants consisting solely of d-terms. The necessary check for the conflicts of the combinations of selected variables with the entries of the S-matrix can be made at the end of step 2 and step 4 (a) of the algorithm. It is to be noted that in using this procedure sometimes we may have to discontinue expansion along a branch only after complete expansion along that branch is done and the final product terms obtained.

Example : $F(x_1, x_2, x_3, x_4) = \Sigma(0, 1, 3) + \Sigma_{opt}(2, 4, 5, 6, 9, 10, 11, 13, 14, 15)$.

Table IX shows the S-matrix and P-matrix of the function.

Table IX. S-matrix and P-matrix

$$S = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

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

Taking the complements of the P-terms gives the function $F_S + F_d$ in the canonical product-of-sums form as

$$F_S + F_d = \overline{F}_P = (\overline{x}_1 + \overline{x}_2 + x_3 + x_4)(\overline{x}_1 + x_2 + x_3 + x_4)$$
$$(x_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4).$$

Step 1 : We form the clause-column table representation of the function as shown in Table X.

Table X. Clause-column table

a_1	a_2	a_3
$\overline{x}_1$	$\overline{x}_1$	x_1
$\overline{x}_2$	x_2	$\overline{x}_2$
x_3	x_3	$\overline{x}_3$
x_4	x_4	$\overline{x}_4$

Step 2 : From an inspection of Table X we see that there do not exist columns with essential literals, columns with identical sets of literals, and dominated columns in the table.

Step 3 : We find the frequency ordering of the literals in Table X as given below.

$$\overline{x}_1 \geq \overline{x}_2 \geq x_3 \geq x_4 \geq x_1 \geq x_2 \geq \overline{x}_3 \geq \overline{x}_4 .$$

Step 4 : We select one of the literals, $\overline{x}_1$, that occur in the maximum number of columns in the clause-column table.

(a) We delete the two columns of Table X in which the literal $\overline{x}_1$ appears, and also delete the literal x_1 from the remaining column a_3. We are left with the reduced table

$$\overline{x}_1 \quad \begin{bmatrix} a_3 \\ \overline{x}_2 \\ x_3 \\ \overline{x}_4 \end{bmatrix}$$

which gives three product terms : (i) $\bar{x}_1 \bar{x}_2$, (ii) $\bar{x}_1 \bar{x}_3$, (iii) $\bar{x}_1 \bar{x}_4$, of which none conflicts with the entries of the S-matrix of Table IX.

(b) Next we delete the literal $\bar{x}_1$ from all the columns of Table X in which it appears, and form a reduced clause-column table as shown in Table XI.

Table XI. Reduced clause-column table

a_1	a_2	a_3
x_2	$\bar{x}_2$	x_1
x_3	x_3	$\bar{x}_2$
x_4	x_4	$\bar{x}_3$
		$\bar{x}_4$

Now we go back to step 2.

Step 2' : From an observation of Table XI we see that there do not exist columns with essential literals, columns with identical sets of literals, and dominated columns in the table.

Step 3' : We find the frequency ordering of the literals of Table XI as shown below.

$$\bar{x}_2 \geq x_3 \geq x_4 \geq x_1 \geq x_2 \geq \bar{x}_3 \geq \bar{x}_4 .$$

Step 4' : We select one of the literals, $\bar{x}_2$, that occur in the maximum number of columns of Table XI.

(a) We delete the two columns of the table in which the literal $\bar{x}_2$ occurs, and also delete the literal x_2 from the remaining column a_1. We obtain the reduced table

$$\bar{x}_2 \quad \begin{bmatrix} a_1 \\ x_3 \\ x_4 \end{bmatrix}$$

which gives two product terms : (iv) $\bar{x}_2 x_3$, (v) $\bar{x}_2 x_4$, none of which again conflicts with the entries of the S-matrix in Table IX.

(b) We next delete the literal $\bar{x}_2$ from all the columns of Table XI in which it appears, and form a reduced clause-column table as shown below.

Table XII. Reduced clause-column table

a_1	a_2	a_3
x_2	x_3	x_1
x_3	x_4	$\bar{x}_3$
x_4		$\bar{x}_4$

We again go back to step 2.

Step 2" : From an observation of Table XII we see that the column a_1 of the table is dominated by the column a_2. We delete the column a_1 and the resulting reduced table is shown in Table XIII.

Table XIII. Reduced clause-column table

a_2	a_3
x_3	x_1
x_4	$\bar{x}_3$
	$\bar{x}_4$

Step 3": We find the frequency ordering of the literals of Table XIII as given below.

$$x_1 \geq x_3 \geq \bar{x}_3 \geq x_4 \geq \bar{x}_4 .$$

Step 4": We select one of the literals, x_3, that occur in the maximum number of columns of Table XIII.

(a) We delete the column a_2 of the table in which the literal x_3 appears, and also delete the literal $\bar{x}_3$ from the remaining column a_3. We get the reduced table

$$x_3 \quad \begin{bmatrix} a_3 \\ x_1 \\ \bar{x}_4 \end{bmatrix}$$

which gives two product terms : (vi) x_1x_3, (vii) $x_3\bar{x}_4$, both of which conflict with the entries of the S-matrix, and thus represent prime implicants formed solely of d-terms.

(b) Next by deleting the literal x_3 from the column a_2 of Table XIII in which it appears, we get the reduced table as shown in Table XIV,

Table XIV. Reduced clause-column table

a_2	a_3
x_4	x_1
	$\bar{x}_3$
	$\bar{x}_4$

from which we get

$$x_4 \quad \begin{bmatrix} a_3 \\ x_1 \\ \bar{x}_3 \end{bmatrix}$$

which gives two product terms : (viii) x_1x_4, (ix) $\bar{x}_3x_4$, of which the first one conflicts with the entries of the S-matrix of Table IX.

Thus the required prime implicants of the function $F_S + F_d$ are : 1) $\bar{x}_1\bar{x}_2$, 2) $\bar{x}_1\bar{x}_3$, 3) $\bar{x}_1\bar{x}_4$, 4) $\bar{x}_2x_3$, 5) $\bar{x}_2x_4$, 6) $\bar{x}_3x_4$.

3.6 Conclusion

The present chapter describes an iterative procedure for generating the prime implicants of switching functions by utilizing a new tabular mode of functional representation called clause-column table. The procedure can be equally applied to functions given in the sum-of-products or in the product-of-sums form, both canonical and noncanonical, and can be adapted to determine the prime implicants of functions having a large number of unspecified fundamental products or don't care terms. Though the procedure generates all the prime implicants of switching functions, sometimes product terms corresponding to nonprime implicants may as well be generated. But usually the number of nonprime implicants that are generated is very small, if they are generated at all. However, the generation of null as well as duplicate product terms is completely avoided in the procedure. The suggested algorithm becomes particularly advantageous when the switching functions are given in the minterm form with a large number of specified or S-terms, i.e. with a comparatively small number of prohibited or P-terms, or when the functions have a large number of don't care or d-terms, in which cases the algorithm can be directly applied on the maxterm form of the functions involving lesser number of sum terms and thus requiring much less computation. The algorithm has been programmed on an IBM 360 system by using APL. The computer experience with the algorithm demonstrates that the algorithm is not only straightforward and simple, but is also efficient and rapidly converging so that the final result can usually be obtained even in the case of large tables without too much iterations. In view of the fact that in the present algorithm each table is expanded by a selected column in contrast to what is done in some of the other existing methods, the number of branching becomes substantially reduced, which in effect reduces the total amount of required computation time. The total memory requirement is also effectively less because of the fact that once some of the litarals from a table are selected, the deleted

columns of the table no longer have to be preserved in the memory, since these columns will be of no use in subsequent computations.

---

[1] It is assumed that if x_j^* is an essential literal, then the literal $\bar{x}_j^*$ does not appear alone in any column, that is, $\bar{x}_j^*$ is not an essential literal as well. When both the literals x_j^* and $\bar{x}_j^*$ become essential at some point along a branch, then further expansion along that branch is to be discontinued to avoid the generation of null terms.

Chapter 4

FINDING MINIMAL OR NEAR MINIMAL SOLUTIONS FROM THE SET OF ALL THE PRIME IMPLICANTS

4.1 A Brief Review of Existing Techniques

In this section we make a brief review of some of the existing important techniques for finding the minimal or nonminimal irredundant solutions of switching functions, assuming the set of all the prime implicants being given. After all the prime implicants of switching functions are found, the next problem is usually to find a minimal subset (according to some criterion of minimality) of prime implicants such that their disjunction is equivalent to the given function and from which none of the prime implicants can be dropped without sacrificing equivalence. Of the available methods for the selection of prime implicants for occurrence in minimal solutions, the prime implicant table method suggested by McCluskey [7] and the map methods of Veitch [1] and Karnaugh [3] are widely used. Later methods of selection of prime implicants include those of Ghazala [32], Mott [33], Pyne and McCluskey [20], Choudhury and Das [28,42], Marcovitz and Shub [43], Das [22] and others. Quine [6] was the first to propose the idea of a dispensable prime implicant : A given prime implicant is dispensable or eliminable from the set of all the prime implicants of the function if it implies the alternation of some of the other prime implicants of the set. This concept of eliminable prime implicant was subsequently used by Ghazala, Mott, Choudhury and Das, Chang and Mott [34] and finally by Das in finding the irredundant normal forms of switching functions. The approach of Ghazala consists in first constructing a ratio chart as follows : Write all the prime implicants of the function in a row and also in a column and below each of the prime implicants make an entry of the

ratios of the corresponding prime implicant to the set of all the prime implicants in a columnar form. This kind of table is called a ratio chart. Nothing should be written along the diagonal entries. The eliminability conditions of a prime implicant can be found by taking the disjunction of the entries in a row in the ratio chart, i.e. the corresponding prime implicant is eliminable if the disjunction of some or all of the entries in that row is equal to 1. Next the idea of a presence factor was introduced. A presence factor g_i is a binary coefficient, attached to a prime implicant p_i such that the presence of p_i in a given irredundant normal form of switching function corresponds to $g_i = 1$, and its absence to $g_i = 0$. From these presence factors, all the irredundant solutions of a switching function can be found as follows : If m is the number of all the prime implicants of a function, the chart yields m equivalences of the form $g_i + G_i = 1$, where G_i is a function of the presence factors g_j for all values of j except $j = i$, then all the irredundant solutions can be found by

$$G = \prod_{i=1}^{m} (g_i + G_i)$$

Mott described an algebraic method for determining the irredundant normal forms of switching functions in which the eliminability conditions of prime implicants are expressed by means of certain irredundant implication relations, where the antecedent of the relation is a single prime implicant and the consequent a disjunction of prime implicants. These irredundant implication relations specify the eliminability conditions of prime implicants. The basic approach of Mott is as follows.

Given the set of all the prime implicants of a switching function, each prime implicant in the set is tested for consensus with all the other prime implicants. If there results a consensus which is not a prime implicant in the set, a new term is added to the end of the set. Testing is continued in this manner until every term in the set, including added ones, has been tested for consensus with all the succeeding terms in the set and none remains to be tested. The procedure can be represented in a tabular form and the

implication relations (eliminability conditions) of prime implicants can be read directly from the table. By letting a conjunction of prime implicants represent each subset of prime implicants whose disjunction is implied by a prime implicant, we can write an expression in conjunctive form, one conjunct for each prime implicant. Application of the distributive law will then yield an expression in disjunctive form such that every disjunct that does not subsume another represents an irredundant solution.

Choudhury and Das developed methods for obtaining all the irredundant or minimal covers of switching functions by utilizing the Connected Cover Term Matrices*. The basic approach of the authors consists in arranging the columns of Cover table [41] of a switching function into a number of matrices. The generation of the redundant solutions are reduced to a great extent in this method, and the generation of duplicate solutions are completely avoided.

4.2 An Approach for Finding Minimal or Near Minimal Solutions

All the irredundant solutions of a switching function can obviously be found by modifying the algorithm as given in Chapter 3. Since the prime implicants do not have their complements, the steps to delete the complement of a literal have to be removed from the algorithm in order to apply it for finding all the irredundant covers of switching functions. After forming the Cover table of the switching function, the modified algorithm can be applied to find all its minimal or nonminimal irredundant solutions.

---

* Foot Note : Connected Cover Term Matrix is an array of only two rows and a number of columns obtained by arranging the prime implicants in the sum terms of the minimizing function such that the element (a single prime implicant or a group of prime implicants) in its (1,i)th position is identical with the element in its [2,(i-1)]th position only for $i \geq 2$, that is, the element in its (1,i)th position does not occur in any other position except in the [2,(i-1)]th position.

Recently Marcovitz and Shub proposed an improved algorithm for the simplification of switching functions by utilizing the Karnaugh map, which was later modified by Das. Das also extended the basic idea of Marcovitz and Shub in finding minimal or nonminimal irredundant solutions by using the cover table representation of switching functions instead of the map. Das discussed how the cover table representation of the function in conjunction with the eliminability conditions of its prime implicants can be used with advantage to find whether such simple replacement of the prime implicants are actually possible, leading to a minimal solution.

Definition : The eliminability conditions of a nonessential prime implicant p_i in a function F are the combinations of other prime implicants of F that jointly cover all the minterms covered by the prime implicant p_i.

The eliminability conditions of any prime implicant p_i can be readily found from the cover table as follows :

1) Remove all the columns that do not contain p_i from the cover table and form a reduced cover table.

2) Delete the prime implicant p_i from all the columns of the reduced cover table.

3) Apply the algorithm given in Chapter 3 with modifications as explained previously. This will lead to the eliminability conditions of p_i.

Example : Find the eliminability conditions of the prime implicant, E from Table 1, as given below.

Table 1 : Cover Table Representation of a given function F

1	2	4	5	6	7	8	9	10	11	12	15	16	17	18
A	B	C	C	C	C	E	A	B	E	D	K	F	A	B
T	S	D	T	S	K	R	E	E	G	R	L	Q	F	F
							G	H	H				I	J
									L					

Table 1 : Cover Table Representation of a given function F

19	20	23	25	26	27	28	29	30	31
F	D	K	A	B	G	D	N	O	K
I	Q	M	G	H	H	P	P	P	N
J			I	J	I				M
M			N	O	J				L
					L				O
					M				P
					N				
					O				

1) Reduced cover table is given by

Table 2 : Reduced cover table after removing the minterms that are not covered by E

8	9	10	11
E	A	B	E
R	E	E	G
	G	H	H
			L

2) After deleting E from Table 2, we get Table 3 as given below.

Table 3 : After removing E from Table 2

8	9	10	11
R	A	B	G
	G	H	H
			L

3) Apply the algorithm given in Chapter 3 with modification to Table 3. We have the following eliminability conditions of E.

$E \longrightarrow RGB + RGH + RAH + RABL$

Theorem [22] : If in a nonminimal irredundant cover C_i of a function F there occurs a combination of prime implicants $q_1 q_2 \ldots . q_m$ that represents the eliminability condition of a group of prime implicants $p_1 p_2 \ldots . p_i$, $i < m$, not in the cover, then the combination of prime

implicants $q_1 q_2 \cdots\cdots q_m$ can be replaced by the group of prime implicants $p_1 p_2 \cdots\cdots p_i$ in C_i with a subsequent reduction in the number of prime implicants, if and only if the set of minterms being uncovered due to the withdrawal of $q_1 q_2 \cdots\cdots q_m$ from C_i is covered uniquely by the group of replaced prime implicants $p_1 p_2 \cdots\cdots p_i$.

Given any nonminimal irredundant cover C_i of a function, we can readily find a minimal or near minimal solution by using the following procedure.

Find the eliminability conditions of those prime implicants that do not appear in C_i. Next check the given nonminimal irredundant cover C_i to see if it contains combinations of prime implicants that represent the eliminability conditions of the prime implicants not contained in it. If the eliminability condition of some prime implicant is found to occur in C_i, check if that prime implicant can be substituted for its eliminability condition, satisfying the conditions of the above theorem. Proceed this way as long as any substitution is possible and the process ultimately leads to either a minimal or near minimal solution of the function.

To check if the minterms of the function being uncovered due to the withdrawal of the prime implicants $q_1 q_2 \cdots\cdots q_m$ from the cover C_i are uniquely covered by the introduction of the prime implicants $p_1 p_2 \cdots\cdots p_i$, the following procedure can be used.

If the eliminability condition of some prime implicant, say p_k not in the cover C_i is a set of prime implicants, say $q_1 q_2 \cdots\cdots q_r$, delete the prime implicants $q_1 q_2 \cdots\cdots q_r$ from the cover C_i. Let the remaining prime implicants of the cover be designated as C_j. Next put check marks on the minterms in the cover table which are covered by C_j. Note now the minterms in the cover table that are unchecked and see if these minterms can be covered by the prime implicant p_k. Under this condition we can replace $q_1 q_2 \cdots\cdots q_r$ by the prime implicant p_k in the cover C_i. This process is continued until either the prime implicants $q_1 q_2 \cdots\cdots q_r$ in the cover C_i can be replaced by the prime implicant p_k or all the eliminability conditions are checked by the above mentioned method.

Example : Consider the function whose cover table is given in Table 4.

Table 4 : Cover Table of a given function

3	5	7	11	13	27	29	31
A	C	A	B	D	E	F	G
B	D	C	E	F	G	H	H

All the nonminimal or minimal irredundant solutions of the above given function can be found from the cover table by applying the modified algorithm as explained before. One of the nonminimal irredundant solution C_i is given by

C_i = ABDFG

The eliminability conditions of C, E and H are given by

C $\longrightarrow$ AD

E $\longrightarrow$ BG

H $\longrightarrow$ FG

It can be noticed that all the eliminability conditions have prime implicants that are in the cover C_i, thus none of the product terms of the prime implicants can be deleted.

To check if BG can be replaced by E in the cover C_i by the above procedure, we have,

C_{j1} = ADF

The cover C_{j1} covers minterms 3,5,7,13,29 and check marks are made in the cover table as shown in Table 5. The unchecked minterms in Table 5 are 11,27,31 and the prime implicant E only covers the minterms 11,27. The minterm 31 is not covered by either C_{j1} or by the prime implicant E. Thus the prime implicants BG in the cover C_i cannot be replaced by the prime implicant E.

Similarly we can show that the prime implicants FG cannot be replaced by the prime implicant H.

Table 5 : Cover Table of a given function for checking the substitution of E in C_i

✓	✓	✓		✓		✓	
3	5	7	11	13	27	29	31
A	C	A	B	D	E	F	G
B	D	C	E	F	G	H	H

To check if the prime implicants AD can be replaced by the prime implicant C in cover C_i, we have

C_{j2} = BFG

The cover C_{j2} covers minterms 3,11,13,27,29,31 and check marks are made in the cover table as shown in Table 6. The unchecked minterms in Table 6 are 5,7 and these minterms are covered by the prime implicant C. Thus the prime implicants AD can be replaced in the cover C_i by the prime implicant C, leading to a minimal solution M_i as

M_i = BCFG

Table 6 : Cover Table of a given function for checking the substitution of C in the cover C_i

✓			✓	✓	✓	✓	✓
3	5	7	11	13	27	29	31
A	C	A	B	D	E	F	G
B	D	C	E	F	G	H	H

REFERENCES

[1] E.W. Veitch, "A chart method for simplifying truth functions", Proc. Assoc. Comput. Mach. (Pittsburgh, Pennsylvania Meeting), pp. 127-133, May 1952.

[2] W.V. Quine, "The problem of simplifying truth functions", Amer. Math. Mon., vol. 59, pp. 521-531, October 1952.

[3] M. Karnaugh, "The map method for synthesis of combinational logic circuits", AIEE Trans. (Commun. Electronics), vol. 72, pp 593-599, November 1953.

[4] R.J. Nelson, "Simplest normal truth functions", J. Symbolic Logic, vol. 20, pp. 105-108, June 1955.

[5] R.J. Nelson, "Weak simplest normal truth functions", J. Symbolic Logic, vol. 20, pp. 232-234, September 1955.

[6] W.V.Quine, "A way to simplify truth functions", Amer. Math. Mon., vol. 62, pp. 627-631, November 1955.

[7] E.J. McCluskey, Jr., "Minimization of Boolean functions", Bell Sys. Tech. J., vol. 35, pp. 1417-1444, November 1956.

[8] A.Svoboda, "Some applications of contact grids", Proc. Int. Symp. Theory of Switching (Part I), pp. 293-305, April 1957.

[9] W.V. Quine, "On cores and prime implicants of truth functions", Amer. Math. Mon., vol. 66, pp. 755-760, November 1959.

[10] F.B. Hall, "Boolean prime implicants by the binary sieve method", AIEE Trans. (Commun. Electronics), vol. 80, pp. 709-713, January 1962.

[11] A.H. Scheinman, "A method for simplifying Boolean functions", Bell Sys. Tech. J., vol. 41, pp. 1337-1346, July 1962.

[12] A.K. Choudhury and M.S. Basu, "A mechanized chart for simplification of switching functions", IRE Trans. Electronic Computers, vol. EC-11, pp 713-714, October 1962.

[13] A.K. Choudhury, M.S. Basu and S.R. Das, "On adaptation of the grouping chart and simplification of multiple output switching functions", Indian J. Phys., vol. 36, pp. 521-532, October 1962.

[14] E.J. McCluskey, Jr., "Minimal sums for Boolean functions having many unspecified fundamental products", AIEE Trans. (Commun. Electronics), vol. 81, pp. 387-392, November 1962.

[15] S.R. Das and A.K. Choudhury, "Maxterm type expressions of switching functions and their prime implicants", IEEE Trans. Electronic Computers, vol. EC-14, pp. 920-923, December 1965.

[16] K.K. Roy, A.K. Choudhury and S.R. Das, "Simplification of switching functions involving a very large number of 'don't care' states", Int. J. Control, vol. 3, pp. 17-28, January 1966.

[17] J.R. Slagle, C.L. Chang and R.C.T. Lee, "A new algorithm for generating prime implicants", IEEE Trans. Computers, vol. C-19, pp. 304-310, April 1970.

[18] S.R. Das, "Comments on 'A new algorithm for generating prime implicants' ", IEEE Trans. Computers, vol. C-20, pp. 1614-1615, December 1971.

[19] A. Mukhopadhyay, "A method of determination of all the minimal forms of Boolean functions", IEE (London) Monograph No. 487 E, December 1961.

[20] I.B. Pyne and E.J. McCluskey, Jr., "The reduction of redundancy in solving prime implicant tables", IRE Trans. Electronic Computers, vol. EC-11, pp. 473-482, August 1962.

[21] E.J.McCluskey, Jr., Introduction to the Theory of Switching Circuits. New York: McGraw-Hill, 1965.

[22] S.R. Das, "An approach for simplifying switching functions by utilizing the cover table representation", IEEE Trans. Computers, vol. C-20, pp. 355-359, March 1971.

[23] E. Morreale, "Partitioned list algorithms for prime implicant determination from canonical forms", IEEE Trans. Electronic Computers, vol. EC-16, pp. 611-620, October 1967.

[24] A. Svoboda, "Ordering of implicants", IEEE Trans. Electronic Computers, vol. EC-16, pp. 100-105, February 1967.

[25] N.N. Necula, "A numerical procedure for determining the prime implicants of a Boolean function", IEEE Trans. Electronic Computers, vol. EC-16, pp. 687-689, October 1967.

[26] R.H. Urbano and R.K. Mueller, "A topological method for the determination of the minimal forms of a Boolean function", IRE Trans. Electronic Computers, vol. EC-5, pp. 126-132, September 1956.

[27] J.P. Roth, "Algebraic topological methods in synthesis", Proc. of an International Symp. on the Theory of Switching", pt.1, pp. 57-73, April 1957.

[28] A.K. Choudhury and S.R. Das, "Direct determination of all the minimal prime implicant covers of switching functions", J. Electronics and Control, vol. 17, No. 5, pp. 553-576, November 1964.

[29] P. Tison, "Generation of consensus theory and application to the minimization of Boolean functions", IEEE Trans. Electronic Computers, vol. EC-16, pp. 446-456, August 1967.

[30] N.N. Necula, "An algorithm for the automatic approximate minimization of Boolean functions", IEEE Trans. Computers, vol. C-17, pp. 770-782, August 1968.

[31] S.R. Patrick, "A direct determination of the irredundant forms of a Boolean function from the set of prime implicants", Air Force Cambridge Res. Ctr., Bedford, Mass., Tech. Rept. AFCRC-TR-56-110, April 1956.

[32] M.J. Ghazala, "Irredundant disjunctive and conjunctive forms of a Boolean function", IBM J. Res. Develop., vol. 1, pp. 171-176, April 1957.

[33] T.H. Mott, Jr., "Determination of the irredundant normal form of a truth function by iterated consensus of prime implicants", IRE Trans. Electronic Computers, vol. EC-9, pp. 245-252, June 1960.

[34] D.M.Y. Chang and T.H. Mott, Jr., "Computing irredundant normal forms from abbreviated presence functions", IEEE Trans. Electronic Computers, vol. EC-14, pp. 335-342, June 1965.

[35] H. Mott and C.C.Carroll, "Numerical procedures for Boolean function minimization", IEEE Trans. Electronic Computers. vol. EC-13, pp. 470, August 1964.

[36] M.A. Breuer, "Heuristic switching expression simplification", Proc. ACM 23rd Natl. Conf., ACM Publ. P-68, pp. 241-250, August 1968.

[37] J.F. Gimpel, "A reduction technique for prime implicant tables", IEEE Trans. Electronic Computers, vol. EC-14, pp. 535-541, August 1965.

[38] F. Luccio, "A method for the selection of prime implicants", IEEE Trans. Electronic Computers, vol. EC-15, pp. 205-212, April 1966.

[39] B. Dunham and R. Fridshal, "The problem of simplifying logical expressions", J. Symbolic Logic, vol. 24, pp. 17-19, March 1959.

[40] R.E.Miller, Switching Theory : Combinational Circuits, vol. 1, New York : Wiley, 1965.

[41] A.K. Choudhury and S.R. Das, "Some studies on connected cover term matrices of switching functions", Int. J. Control, vol. 2, pp 441-501, November 1965

[42] A.K. Choudhury and S. R. Das, "On a method of determination of one of the minimal solutions of switching functions by utilizing their connected cover term matrices", Int. J. Control, vol. 1, pp. 565-583, June 1965.

[43] A.B. Marcovitz and C.M. Shub, "An improved algorithm for the simplification of switching functions using unique identifiers on a Karnaugh map", IEEE Trans. Computers, vol. C-18. pp. 376-378, April 1969.

[44] A.K. Choudhury and S.R. Das, "Computing irredundant normal normal forms from abbriviated presence funcrions" IEEE Trans. Electronic Computers, vol. EC-15, pp. 387, June 1966.

[45] S.H. Caldwell, Switching Circuits and Logical Design, New York : Wiley, 1958.

[46] P.E. Wood, Jr., Switching Theory, McGraw Hill Book Company, 1968.

[47] R.S. Gaines, "Implication techniques for Boolean functions", Switching Circuit Theory and Logical Design, Proc. Fifth Ann. Symposium, Princeton, N.J., pp. 174-182, October 1964. Published by the IEEE, New York, Spec. Publ. S-164.

[48] S.R. Das and N.S. Khabra, "Clause-column table approach for generating all the prime implicants of switching functions", Fifteenth Midwest Symposium on Circuit Theory, University of Missouri-Rolla, Rolla, Missouri, U.S.A., May 4-5, 1972, Section 4.2, 10 pages. To appear in IEEE Trans. Computers.

```
      )FNS GPS
∇ GPS
 [1.1] FNC1
 [2] STORE←⍳0
 [2.5] C←0
 [3] C1←0
 [3.5] PR←⍳0
 [4] C2←0
 [5.1] NUM←⍳0
 [6] PICKUP←⍳0
 [9] PRIME1←⍳0
 [10] PRIME2←⍳0
 [10.1] VL←⍳0
 [11.1] CRT
 [12] I←1
 [13] CCTD←⍴CCT
 [14] L1:→(1=+/CCT[I; ]=T)/L2
 [15] I←I+1
 [16] →(I≤CCTD[1])/L1
 [17] →L3
 [18] L2:SELL
 [19] PRIME1←PRIME1,EL
 [20] CCTD←⍴CCT
 [21] →(0 0=CCTD)/PRIME
 [22] →12
 [23] L3:FRE
 [24] C1←C1+1
 [25] CCTD←⍴CCT
 [26] CD←CCTD[1]×CCTD[2]
 [27] CUM←CD⍴CCT
 [27.1] VLD←⍴VL
 [27.2] CD1←CD+VLD
 [28] NUM←NUM,CD1
 [29] STORE←STORE,CUM,VL
 [29.1] STORE
 [30] SELL1
 [31.1] PICKUP←PICKUP,FREORD[1]
 [32] PRIME2←PRIME2,FREORD[1]
 [33] CCTD←⍴CCT
 [34] →(0 0=CCTD)/PRIME
 [34.1] FRE1←T∈FREORD[1]
 [34.2] FRE2←FRE1/T
 [34.3] FRE3←FRE1\FRE2
 [34.4] VL←VL,FRE3
 [35] →12
 [36] PRIME:PRIM←PRIME1,PRIME2
 [36.2] DPRIM←⍴PRIM
 [36.3] →(0=DPRIM)/38
 [37.1] PRI1←T∈PRIM
 [37.2] PRI2←PRI1/T
 [37.3] PRI←PRI1\PRI2
 [37.4] PR←PR,PRI
 [38] C2←C2+1
```

GPS (CONTINUED)

```
∇ GPS
 [39] →(C2>C1)/52
 [40] GUM1←⍳NUM[C2]
 [41] STORE1←STORE[GUM1]
 [41.1] STD←⍴STORE
 [41.2] STORE←STORE[NUM[C2]+⍳(STD-NUM[C2])]
 [44] T1←⍴T
 [44.16000306] R←NUM[C2]÷T1
 [44.3] G←R,T1
 [45] CCT←G⍴STORE1
 [46] FT2←~(T∈PICKUP[C2])
 [47] CCT5←FT2/CCT
 [49] CCT6←FT2\CCT5
 [50] CCT←CCT6
 [51] →9
 [52] FN←(C2-C),T1
 [53] CCT←FN⍴PR
 [55] GF2←'P-O-S'
 [56] →(1=∧/GF1=GF2)/59
 [57] GF1←GF2
 [58.1] →2
 [59] CR
 [60] CCT
∇
```

```
      )ΨNS CRT
∇ CRΨ
 [4] M←1+L←1
 [5] LOOP3:CCΨD←⍴CCΨ
 [5.1] W1←CCΨD[2]⍴2
 [5.5] →(M>CCΨD[1])/T2
 [6.1] →((+/CCΨ[L; ]=Ψ)=+/W1=(CCΨ[L; ]=Ψ)+CCΨ[M; ]=Ψ)/SΨL1
 [7] →((+/CCΨ[M; ]=Ψ)=+/W1=(CCΨ[L; ]=Ψ)+CCΨ[M; ]=Ψ)/SΨL2
 [8] M←M+1
 [9] →(M≤CCΨD[1])/LOOP3
 [10] T2:M←1+L←L+1
 [11] →(L≤CCΨD[1]-1)/LOOP3
 [12.1] →25
 [13] SΨL1:U1←⍳CCΨD[1]
 [14] U2←U1[⍳M-1]
 [15] U3←U1[M+⍳CCΨD[1]-M]
 [16] LOOP4:U←U2,U3
 [17] UTT←U1∈U
 [18] CCΨ1←UTT/[1]CCΨ
 [19] CCΨ←CCΨ1
 [20] →LOOP3
 [21] SΨL2:U1←⍳CCΨD[1]
 [22] U2←U1[⍳L-1]
 [23] U3←U1[L+⍳CCΨD[1]-L]
 [23.5] M←L+1
 [24] →LOOP4
∇
```

```
      )FNS FRE
∇ FRE
 [1] CCTD←⍴CCT
 [2] T←1
 [3] MN1←⍳0
 [4] I3LOOP:M1←CCT[T; ]=T
 [5] MN1←MN1,M1
 [6] I←I+1
 [7] →(I≤CCTD[1])/I3LOOP
 [8] MN←CCTD⍴MN1
 [9] MAX←⍳0
 [10] J←1
 [11] I4LOOP:MA1←+/MN[ ;J]
 [12] MAX←MAX,MA1
 [13] J←J+1
 [14] →(J≤CCTD[2])/I4LOOP
 [15] MA←⍒MAX
 [16] FREORD←T[MA]
∇
```

```
      )FNS SELL1
∇ SELL1
 [1] SUM1←⍳0
 [1.5] CCTD←⍴CCT
 [2] J←1
 [3] LII2:VEC1←+/CCT[J; ]∈FREORD[1]
 [4] SUM1←SUM1,VEC1
 [5] J←J+1
 [6] →(J≤CCTD[1])/LII2
 [7] CCT2←(~SUM1)/[1]CCT
 [8] →(1=2|T⍳FREORD[1])/M3
 [9] MLL←T[(T⍳FREORD[1])-1]
 [10] M4:FT1←~(T∈MLL)
 [11] →M5
 [12] M3:MLL←T[1+T⍳FREORD[1]]
 [13] →M4
 [15] M5:CCT3←FT1/CCT2
 [17] CCT4←FT1\CCT3
 [18] CCT←CCT4
∇
```

```
      )PUS SELL
∇ SELL
 [1] EL←(CCT[I; ]=T)/T
 [2] SUM←⍳0
 [3] J←1
 [3.1] K←1
 [4] LI2:VEC←+/CCT[J; ]∈EL
 [5] SUM←SUM,VEC
 [6] J←J+1
 [7] →(J≤⍴CCT[1])/LI2
 [8] CCT2←(~SUM)/[1]CCT
 [9] →(1=2|T⍳EL)/L3
 [10] ELL←T[(T⍳EL)-1]
 [11] L4:⍎T←~(T∈ELL)
 [12] →L5
 [13] L3:ELL←T[1+T⍳EL]
 [14] →L4
 [14.1] L5:DCCT←⍴CCT2
 [14.20500057] →(0 0=DCCT)/16.1
 [14.21999943] ELL1←T∈ELL
 [14.22999962] ELL2←ELL1/T
 [14.23999981] ELL3←ELL1\ELL2
 [14.3] L6:→(1=∧/CCT2[K; ]=ELL3)/14.8
 [14.5] K←K+1
 [14.6] →(K≤DCCT[1])/L6
 [14.7] →16.1
 [14.8] CCT2←0/[1]CCT2
 [14.9] C←C+1
 [15.4] PRIME1←PRIME2←0/T
 [15.5] EL←⍳0
 [15.6] →16.5
 [16.1] EL1←T∈EL
 [16.2] EL2←EL1/T
 [16.3] EL3←EL1\EL2
 [16.4] VL←VL,EL3
 [16.5] CCT3←EL/CCT2
 [17] CCT4←EL\CCT3
 [18] CCT←CCT4
∇
```

```
      )PNS FNC
∇ FNC
 [1] T←'A1B2C3D4'
 [2] X1←'ABC'
 [3] X2←'A24'
 [4] X3←'AB3D'
 [5] X4←'12C4'
 [6] Y1←T∈X1
 [7] Y2←T∈X2
 [8] Y3←T∈X3
 [9] Y4←T∈X4
 [10] Z1←Y1\X1
 [11] Z2←Y2\X2
 [12] Z3←Y3\X3
 [13] Z4←Y4\X4
 [14] CC←Z1,Z2,Z3,Z4
 [15] CCT←4 8⍴CC
 [16] GF1←'P-OF-S'
∇
```

```
A   C
  B     4
A  2
   2C D
A       4
```

THE PRIME IMPLICANTS ARE GIVEN BY

AC

B4

A2

2CD

A4

```
      )FNS FNC2
∇ FNC2
 [1] T←'A1B2C3D4E5'
 [2] X1←'BC4E'
 [3] X2←'1BCE'
 [4] X3←'1BCD'
 [5] X4←'ACD'
 [6] X5←'A2CE'
 [7] X6←'23DE'
 [8] X7←'123'
 [9] X8←'2345'
 [10] X9←'B3D5'
 [11] X10←'AB45'
 [12] X11←'12C45'
 [13] Y1←T∈X1
 [14] Y2←T∈X2
 [15] Y3←T∈X3
 [16] Y4←T∈X4
 [17] Y5←T∈X5
 [18] Y6←T∈X6
 [19] Y7←T∈X7
 [20] Y8←T∈X8
 [21.1] Y9←T∈X9
 [22] Y10←T∈X10
 [23] Y11←T∈X11
 [24] Z1←Y1\X1
 [25] Z2←Y2\X2
 [26] Z3←Y3\X3
 [27] Z4←Y4\X4
 [28] Z5←Y5\X5
 [29] Z6←Y6\X6
 [30] Z7←Y7\X7
 [31] Z8←Y8\X8
 [32] Z9←Y9\X9
 [33] Z10←Y10\X10
 [34] Z11←Y11\X11
 [35] CC←Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8,Z9,Z10,Z11
 [36] CCT←11 10⍴CC
 [37] GF1←'S-OF-P'
∇

  B C D
  B C   E
A   C D
  B   D  5
A B      5
A   C   E
A B C
    2 3D E
 1 2 3
A  2  D E
 1 2   4 5
A     3 4 5
    2 3 4 5
 1    3D  5
```

VITA

Name : Narindar Singh Khabra

Born : December 1, 1946, Punjab, India.

Education :

Primary :	Government Primary School, Kaharpur, Punjab, India.
Secondary :	Government Higher Secondary School, Mahilpur, Punjab, India.
College :	Uxbridge Technical College, Uxbridge, Middlesex, England.
University :	Queen Mary College, University of London, London, England. B.Sc. E.E. (honors) (1970)