CHAPTER ONE

SOME MINIMISATION TECHNIQUES FOR BINARY AND TERNARY VALUED SWITCHING CIRCUITS
INTRODUCTION

The switching function minimisation problem can be studied in very simple terms: given the true and false output values corresponding to all specified input combinations applicable to a switching function, determine a minimal (in some specified sense) Boolean form for the function. The basic tenet of this process when expressed in AND/OR terms is

\[ AB + A\overline{B} = A \quad (B+\overline{B}) = A \cdot 1 = A \]

identifying such combinable pairs becomes difficult when the number of switching variables increases. The map method of Karnaugh [1952] is an effective tool up to a maximum of six variables. Hall [1962] discussed binary sieve method based on the relations disclosed by a hypothetical sieve, whereas Nelson [1955] introduced the idea of an algebraic approach based on successive expansion utilising the maxterm type of expressions. Hwa [1974] introduced a faster method in which all the prime implicants are obtained in a single pass by the operation of subsuming on the set of binary numbers arranged in the order of increasing index, where the subsuming operation on two successive \(n\) cubes are given by

\[
0 \ 1 \ \ldots \ \dagger \ x_{i+1} \ \ldots \ x_n \ + \ 1 \ \ldots \ \dagger \ x_{i+1} \ \ldots \ x_n \\
\rightarrow \ \beta \ \dagger \ \ldots \ \dagger \ x_{i+1} \ \ldots \ x_n
\]

In the first section of this chapter, a technique has been developed to solve the above problem by arranging all \(k\)-classes in groups according to decreasing index number,
and the checking operation on two classes has been defined as

\[ 0 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \]

\[ 1 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \]

\[ \rightarrow [0 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n] \]

In this technique, all the prime factors are obtained in a single phase after the operation sequence is complete, and functions involving don't care combinations are also considered.

In the second section of this chapter, the same problem of minimisation of binary switching circuit is tackled through a different angle. The Quine McCluskey technique [1956] decomposes the problem in two steps, i.e., PI identification and PI selection for a minimal cover of the function. Schienmann [1962] developed a technique by operating on the set of decimal numbers using Shannon's expansion theorem as

\[ f (x_1, x_2, \ldots, x_n) = x_1 \cdot f(1, x_2, \ldots, x_n) \]

\[ + x_1 \cdot f(0, x_2, \ldots, x_n) \]

When discussing different conditions for residues, in this section the idea of quotient expansion proposition operating on the set of decimal numbers has been used towards solving the problem of PI identification, which is

\[ f (x_1, x_2, \ldots, x_n) = f (x_1, x_2, \ldots, 0) x_n \]

\[ + f (x_1, x_2, \ldots, 1) x_n \]
and then various conditions have been examined for the possibility of minimisation depending on the nature of the quotients. The problem of PI selection has been tackled in this section through set theoretical concept, by defining each prime Implicant in the form of equivalent set of decimal numbers and then forming redundant and selected covers. It has been established that all the selected covers for the switching function constitutes the minimal form, and also the method is programmable.

In the third section of the chapter, the problem of minimisation of ternary valued switching circuit has been solved, which is an extension of the method presented in section - II. Yoeli and Rosenfeld [1964] presented a device oriented ternary switching algebra, and solved the problem in that structure, as an extension of the technique of Schienmann [1962], which is based on Shannon’s expansion theorem as,

\[ g (x_1, x_2, \ldots, x_n) = x_1^0 \cdot g (0, x_2, \ldots, x_n) + x_1^1 \cdot g (1, x_2, \ldots, x_n) + x_1^2 \cdot g (2, x_2, \ldots, x_n) \]

and then discussing nature of residues for the subsuming operation. In this section, assuming the structure of the device oriented algebra of Yoeli et al [1964], the idea of quotient expansion proposition on the set of decimal numbers describing the ternary function has been applied. Thus the set of decimal numbers are partitioned to three disjoint congruent classes, namely \( \equiv 0 \pmod{3} \), \( \equiv 1 \pmod{3} \) and \( \equiv 2 \pmod{3} \) by the formula,
\[ g(x_1, \ldots, x_n) = g(x_1, x_2, \ldots, 0) x_n^0 + g(x_1, x_2, \ldots, 1) x_n^1 + g(x_1, x_2, \ldots, 2) x_n^2 \]

and then discussing on the nature of quotients for the operation of subsuming, which leads to the identification prime Implicants \((g - \text{type, } h - \text{type})\). The case for the involvement of don’t care parts have also been discussed here. Towards solving the problem for obtaining minimal cover for the function, i.e. PI selection, each prime Implicants \((g - \text{type and } h - \text{type})\) is represented in the form of PITD \((g - \text{type and } h - \text{type})\) of the set of decimal numbers, which is obtained as a carry forward operation on the residue classes. Next step consists of selection of redundant and selected ternary covers following algorithms. It has been shown in this section that the class of selected ternary covers give the minimal form for the ternary switching function. A computer programme for the algorithm has also been discussed, the source listing for which is attached at the end of this section.
SECTION - I

A METHOD FOR GENERATING PRIME FACTORS OF A BOOLEAN EXPRESSION IN A CONJUNCTIVE NORMAL FORM WITH THE POSSIBILITY OF INCLUSION OF DON'T CARE COMBINATIONS
The problem of developing techniques for minimisation of two-valued switching function has been of concern for last two decades. The concept of Single Pass Procedure to obtain minimal form were discussed by Sureshchander (1975), Hwa (1974) and others. Hwa used the technique of logic minimisation from a function-lattice to generate successive subcubes in a unique sequence. The method presented by us in this section, deals with generation of Prime factors in a single pass procedure. Starting from the maximum unions of \((n-1)\) cubes, a unique sequence of successive union of less number of \((n-1)\) cubes are generated, each appearing only once in the table. Here, we discuss definition and some of the adjoining properties of an n-cube.

**n-cube representation of Boolean Functions:**

An equivalent geometrical representation of an \(n\)-variable Boolean function, are considered at \(2^n\) discrete points. The different inputs are unique combinations of 1's and 0's, that define the \(2^n\) vertices of a unit n-cube in an \(n\)-dimensional space, as illustrated in figure - 1.1, for a 3-cube representation. Also, n-cube representation for 3-variables in a conjunctive Normal form can be shown of its advantage in the tree-structure as in figure - 1.2.

n-cube representation has been seldomly applied (because of its complicated geometrical representation when \(n > 4\)) to the problem of minimisation of switching circuits. But the intuitive approach, inspite of complicacy due to actual graphical concept after \(n > 4\), leads to several significant contribution towards the problem.
The vertices of an n-cube corresponds to the rows of an n-variable truth-table and also to the minimal surfaces of an n-variable Veinn diagram. In conjunctive Normal form for every term $M_j$, the vertex $2^n-1-j$ of the n-cube is marked by a circle (0).

**Subcube**

**Def 1.1** - Subcube of an n-cube is defined as the union of minimal subcubes and denoted by an appropriate set of vertices of the n-cube. Thus an i-cube is the union of $2^{n-1-i}$, (n-1) cubes.

**Vertex**

**Def 1.2** - A vertex referred to as i-cube can be defined by all the n-variables, thus

$$\left( x_1^{d_1} + x_2^{d_2} + \ldots + x_n^{d_n} \right)$$

defines one of the $2^n$ vertices of an n-cube.

**Adjacency of Vertices**

**Def 1.3** - Two vertices are said to be adjacent, whenever they are end points of the edge of an n-cube, i.e. they differ in one variable only.

**Mutual Adjacency of Vertices**

**Def 1.4** - A set of $2^i$ vertices of an i-cube, are said to be mutually adjacent, whenever every vertex in the set is adjacent to i other vertices in the set.
FIG: 1 - 1

3-CUBE  2-CUBE  1-CUBE  0-CUBE

\begin{array}{cccc}
\text{b+c} & 0 & 0 & 0 \\
\text{b+c}' & 0 & 0 & \text{'} \\
\text{a+b+c} & 0 & 1 & \text{'} \\
\text{a+b+c}' & 0 & 1 & \text{'} \\
\text{a'+b+c} & 1 & 0 & 0 \\
\text{a'+b+c}' & 1 & 0 & 1 \\
\text{a'+b'+c} & 1 & 1 & 0 \\
\text{a'+b'+c}' & 1 & 1 & \text{'} \\
\end{array}

FIG: 1 - 2
Simplification of Boolean Function with n-cube techniques

Since this technique is classical, we will simply mention it by means of an example only. A two level realisation of an expression

\[ f(x_1, x_2, \ldots, x_n) = \prod_{i=0}^{2^n-1} \left( \eta_i + \overline{m}_i \right) \]

using Multi-Input or - gates can be illustrated in fig-1.3. Applying the technique, we see that

\[ f(x_1, x_2, x_3) = \sum (2, 3, 4, 5, 6, 7) = \prod (0, 1) \]

has minimal form as \( x_1 \cdot x_2 \) and \( x_1 + x_2 \) respectively in disjunctive and conjunctive form, as illustrated in fig-1.4 and fig-1.5.

Thus the example clearly indicates the simplicity of conjunctive Normal form in some cases. Further any switching function can be conveniently represented either in a disjunctive Normal form or equivalent Conjunctive Normal form. In the following paras we will develop a method, which will determine Prime Factors in a single Pass Procedure, where the Boolean function is taken in conjunctive Normal form.
II A method for generating Prime factors of a Boolean
expression in a conjunctive normal form with the
possibility of inclusion of Don't care combination

If a Boolean expression $F$ of $n$ binary variables
$x_1, x_2, \ldots, x_n$ is converted into conjunctive
Normal form, then a maxterm of the Boolean expression
$F$ in $E^n$ space denoted by $c_j$ ($j = 0, 1, \ldots, n$)
is the union of $j$ ($j = 0, 1, \ldots, n$) number of
$(n-1)$ subcubes, where $j$ is the number of zeros occurring
in the maxterm, and a Prime factor of $(n-j)$
variables is defined as a minimum $j$ - class. A minimum
$j$ - class is not a subclass of any other class, i.e.
has a disjoint meet with other $(n-1)$ subcubes.

If the class of $j$ unions of $(n-1)$ subcubes are
classified into groups by the number of zeros, i.e.
number of $(n-1)$ subcubes in each union, the distribu­
tion of these unions may be shown in Table-14, where
the common set of variables are described as $x_{i+4} + \ldots + x_n$.
Throughout the section, we shall use group order of
decreasing index from above.

As in Table-14, where $G_\Pi$ denotes the group with
index $\Pi$ and $n_\Pi$ denotes the number of $\alpha$ - classes
present in the group, where $\Pi$ obviously denotes the
number of zeros in the maxterm and $K$ is the number of
zeros present in the common portion of the maxterm
$x_{i+4} + \ldots + x_n$.

* Published in Journal of Technology, Vol XXII, No.1,
June 1977, pp 1 - 11
The following properties of a set of $\alpha$-classes may be observed on a deep observation of Table 1.1.

i) For any $U$ set, the pattern of distribution of $\alpha$-classes is the same as long as the common term is ignored.

ii) There is only one $\alpha$-class in $G_{0+k}$, which will be called as the top class, and only one $\alpha$-class in $G_{i+k}$ which will be called the bottom class.
iii) Every \( \alpha \)-class in the \( U \) set can be the top class of a subclass of the \( U \) set, say \( j \)-class, with the original bottom class \( 0 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \) as its bottom class. Dual is the case if an \( \alpha \)-class in the \( U \) set is considered as the bottom class of a subclass of the \( U \) set.

Let us start to construct all possible \( \alpha_i \) classes, consisting of union of \( (n-1) \) no. of \( (n-1) \) subcubes. This can be performed by meeting all top classes in \( G_{(j-1)+k} \) with common bottom class \( 0 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \) in \( G_{i+k} \), for instance:

\[
\left[ 0 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \right]
\]

\[
\left[ 1 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \right] \quad \Rightarrow
\]

\[
\left[ \emptyset + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \right]
\]

Here \( \emptyset \) denotes a don't care bit with original value 1.

We call \( 1 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n \) as the checking class, and

\[
0 + 0 + \ldots + 0 + x_{i+1} + \ldots + x_n
\]

as the checked class. It is clear that \( \alpha_i \) class is generated by converting the 1 bit in the checking class into \( \emptyset \) bit. Thus all the \( \alpha_i \) classes so formed may be written as
The construction of the $\alpha_2$ - classes can be performed by checking two $\alpha_1$ - classes as top class in $G(i-2)+k$ into a top class in $G(i-1)+k$. For instance,

\[
\begin{align*}
\emptyset + 0 + \cdots + 0 + x_{i+1} + \cdots + x_n & \\
0 + \emptyset + 0 + \cdots + 0 + x_{i+1} + \cdots + x_n & \\
1 + 0 + 0 + \cdots + 0 + x_{i+1} + \cdots + x_n & \\
\emptyset + \emptyset + 0 + \cdots + 0 + x_{i+1} + \cdots + x_n & \rightarrow
\end{align*}
\]

It is important to note here that the distance between the $\alpha_2$ - class as top class and $\alpha_1$ - class as top class must be one, i.e. $\alpha_2$ - class must contain only one number of subsube less in the union than $\alpha_1$ - class for the checking operation to be performed. Also, the Don't care bit in the checked $\alpha_1$ - class has the same interpretation as that in the $\alpha_2$ - class.

Thus the $\alpha_2$ - classes as top class in $G(i-2)+k$ and the common bottom class

\[
\begin{align*}
0 + 0 + \cdots + 0 + x_{i+1} + \cdots + x_n & \\
\emptyset + \emptyset + 0 + \cdots + 0 + x_{i+1} + \cdots + x_n & \\
0 + 0 + \cdots + \emptyset + \emptyset + x_{i+1} + \cdots + x_n &
\end{align*}
\]
From these $\alpha_2$ classes, the $\alpha_3$ classes as top class in $G_{(i-3)+k}$ and the common bottom class as usual can be constructed by checking three $\alpha_2$ class into the top class in $G_{(i-3)+k}$ i.e.

$$[\emptyset + \emptyset + o + \ldots + o + x_{i+1} + \ldots + x_n] \rightarrow$$

$$[\emptyset + o + \emptyset + o + \ldots + o + x_{i+1} + \ldots + x_n] \rightarrow$$

$$[o + \emptyset + o + o + \ldots + o + x_{i+1} + \ldots + x_n] \rightarrow$$

$$[i+1+1+ o + \ldots + o + x_{i+1} + \ldots + x_n] \rightarrow$$

$$[\emptyset + \emptyset + \emptyset + o + \ldots + o + x_{i+1} + \ldots + x_n]$$

This process is continued until all the top classes in the $U$ set are operated, i.e. a $\alpha_\infty$ class of the form

$$\emptyset + \emptyset + \ldots + \emptyset + x_{i+1} + \ldots + x_n$$

is obtained. It is clear that an intermediate class is formed by changing some of '1' bits as bottom class into Don't care bits. Based on the above discussion, the following theorem may be established.

**Theorem - 1.1**

A set of $(j+1)$, $\alpha_j$ classes all as top class in $G_m$ with the common bottom class XB and a class $\alpha_{j+1}$ in $G_{m-1}$ can form a class of index $j+1$ as $\alpha_{j+1}$ the top class and the common bottom class XB, provided the following conditions are satisfied.
i) The number of \((n-1)\) dimensional subcubes in \(\mathcal{A}_{j+1}\) is one less than the number of \((n-1)\) dimensional subcubes in \(\mathcal{A}_j\), i.e. the difference is unity.

ii) If the bits in any \(\mathcal{A}_j\) class are Don't care bits, they must also be Don't care bits in \(\mathcal{A}_{j+1}\) class.

We shall consider now the following example

**Example - 1.1**

A set of 8 classes in \(\mathbb{F}^5\) is shown in Table - 1.2

Let \(F(x_1, x_2, x_3, x_4, x_5) = \prod(16, 17, 18, 19, 24, 25, 26, 27)\)

The table may be arranged as

<table>
<thead>
<tr>
<th>Decimal representation</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>1</td>
<td>1 + 0 + 0 + 0 + 0 + 0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>0</td>
<td>1 + 0 + 0 + 0 + 0 + 1</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>3</td>
<td>1 + 0 + 0 + 1 + 0 + 0</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>1</td>
<td>1 + 1 + 0 + 0 + 0 + 0</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>0</td>
<td>1 + 0 + 0 + 1 + 1 + 1</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>3</td>
<td>1 + 1 + 0 + 0 + 1 + 1</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>1</td>
<td>1 + 1 + 0 + 1 + 1 + 0</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>1</td>
<td>1 + 1 + 0 + 1 + 1 + 1</td>
</tr>
</tbody>
</table>

Table - 1.2
We start with $\mathfrak{c}$-classes in $G_3$. First consider 17 as the checking class of a $\mathfrak{c}_1$-class. The operation can be described as

\[
\begin{array}{c|c|c|c}
16 & 1 + 0 + 0 + 0 + 0 & 16 & 1 + 0 + 0 + 0 + 0 \\
17 & 1 + 0 + 0 + 0 + 1 & 17 & 1 + 0 + 0 + 0 + 0
\end{array}
\]

Here the cross sign ($\times$) indicates that the checked class is met in the checking class. Until all the classes in $G_3$ have been operated, Table 1.2 becomes Table 1.3.

<table>
<thead>
<tr>
<th>Decimal representation</th>
<th>$G_\Pi$</th>
<th>$n_\Pi$</th>
<th>$\mathfrak{c}$-Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>1</td>
<td>$1 + 0 + 0 + 0 + 0 + 0 \times$</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + 0 + 0 + 0$</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>3</td>
<td>$1 + 0 + 0 + 0 + 0 + 0$</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + 0 + 0 + 0$</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + 1 + 1$</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>3</td>
<td>$1 + 1 + 0 + 0 + 1$</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td>$1 + 1 + 0 + 1 + 0$</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>1</td>
<td>$1 + 1 + 0 + 1 + 1$</td>
</tr>
</tbody>
</table>

Table 1.3
Next the classes in $G_2$ will be dealt with. Let 19 be the first checking class:

<table>
<thead>
<tr>
<th>24</th>
<th>$1 + \phi + 0 + 0 + 0$</th>
<th>24</th>
<th>$1 + \phi + 0 + 0 + 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>$1 + 0 + 0 + 1 + 1$</td>
<td>19</td>
<td>$1 + 0 + 0 + 1 + 1$</td>
</tr>
</tbody>
</table>

Since the differences between 19 and 24 is greater that one, proceed to the next checked class in

<table>
<thead>
<tr>
<th>18</th>
<th>$1 + 0 + 0 + \phi + 0$</th>
<th>18</th>
<th>$1 + 0 + 0 + \phi + 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>$1 + 0 + 0 + 1 + 1$</td>
<td>19</td>
<td>$1 + 0 + 0 + 1 + \phi$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17</th>
<th>$1 + 0 + 0 + 0 + \phi$</th>
<th>17</th>
<th>$1 + 0 + 0 + 0 + \phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>$1 + 0 + 0 + 0 + \phi$</td>
<td>19</td>
<td>$1 + 0 + 0 + 0 + \phi$</td>
</tr>
</tbody>
</table>

It is important to note that 19 can be started with any checked class in $G_3$ without any effect on the result. The same applies to 24 and 18. Thus Table - 1.3 becomes Table - 1.4.
<table>
<thead>
<tr>
<th>Decimal representation</th>
<th>$6_π$</th>
<th>$9_π$</th>
<th>$\alpha$-Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>1</td>
<td>$1 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>3</td>
<td>$1 + 0 + 0 + 0 + \beta$ X</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + \beta + 0$ X</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>$1 + \beta + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + \beta + \beta$</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>3</td>
<td>$1 + \beta + 0 + 0 + \beta$</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td>$1 + \beta + 0 + \beta + 0$</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>1</td>
<td>$1 + 1 + 0 + 1 + 1$</td>
</tr>
</tbody>
</table>

Table - 1.4

Next on $6_π$

<table>
<thead>
<tr>
<th>26</th>
<th>$1 + \beta + 0 + \beta + 0$</th>
<th>26</th>
<th>$1 + \beta + 0 + \beta + 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>$1 + 1 + 0 + 1 + 1$</td>
<td>27</td>
<td>$1 + 1 + 0 + 1 + \beta$</td>
</tr>
</tbody>
</table>

Then compare

<table>
<thead>
<tr>
<th>25</th>
<th>$1 + \beta + 0 + 0 + \beta$</th>
<th>25</th>
<th>$1 + \beta + 0 + 0 + \beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>$1 + 1 + 0 + 4 + \beta$</td>
<td>27</td>
<td>$1 + 1 + 0 + \beta + \beta$</td>
</tr>
</tbody>
</table>

Again compare

<table>
<thead>
<tr>
<th>19</th>
<th>$1 + 0 + 0 + \beta + \beta$</th>
<th>19</th>
<th>$1 + 0 + 0 + \beta + \beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>$1 + 1 + 0 + \beta + \beta$</td>
<td>27</td>
<td>$1 + \beta + 0 + \beta + \beta$</td>
</tr>
</tbody>
</table>
Thus Table - 1.4 becomes Table - 1.5

<table>
<thead>
<tr>
<th>Decimal representation</th>
<th>$g_\pi$</th>
<th>$n_\pi$</th>
<th>$\alpha$-Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>1</td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>3</td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>3</td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>1</td>
<td>$1 + 0 + 0 + 0 + 0 + 0$ X</td>
</tr>
</tbody>
</table>

Table - 1.5

From the above tables, it can be seen that after an operation the operated checking class is used for the comparison with the next checked $f$ class. It is clear that all the above operations may be performed in a single table like Table - 1.5. In this case the Prime factor is $x_1 + x_3$.

III ALGORITHM OF CONSTRUCTING MINIMAL CLASS OF A BOOLEAN FUNCTION IN A CONJUNCTIVE NORMAL FORM

During the construction of minimum class from $U$-set, the following problems may arise:

1) It is not necessary that all $\alpha$-classes may have the same bottom and same class as top-class.
ii) A class may be the bottom class of several minimum classes.

iii) A class may also be the top class of several minimum classes.

The first and second problems may be solved automatically. If two or more distinct classes with identical top class are placed under the name of the same top class, the third problem will be solved naturally.

e.g.

13 0 + 0 + 1 + \emptyset + 0 + \emptyset

45 1 + 0 + \emptyset + 1 + 0 + \emptyset

The distance of 13 and 45 is 1, but Theorem - 1.1 cannot be applied here. Thus the theorem can only be applied if both the \emptyset's which are not Don't care in both are replaced by 1. i.e.

13 0 + 0 + 1 + \emptyset + 0 + \emptyset → 13 0 + 0 + 1 + 1 + 0 + \emptyset

45 1 + 0 + \emptyset + 1 + 0 + \emptyset → 45 1 + 0 + 1 + 1 + 0 + \emptyset

13 0 + 0 + 1 + 1 + 0 + \emptyset

45 \emptyset + 0 + 1 + 1 + 0 + \emptyset

The class \emptyset + 0 + 1 + 1 + 0 + \emptyset is the minimum and one generated from 1 + 0 + \emptyset + 1 + 0 + \emptyset and 0 + 0 + 1 + \emptyset + 0 + \emptyset with top class as 1 + 0 + 1 + 1 + 0 + 1.
It is obvious that $0 + 0 + 1 + 1 + 0 + \emptyset$ is a subclass of $0 + 0 + 1 + \emptyset + 0 + \emptyset$ and $1 + 0 + 1 + 0 + \emptyset$ is a subclass of $1 + 0 + \emptyset + 1 + 0 + \emptyset$, but $\emptyset + 0 + 1 + 1 + 0 + \emptyset$ is not a subclass of $1 + 0 + \emptyset + 1 + 0 + \emptyset$ or vice versa. The above conversion may be written as:

\[
\begin{array}{c|c|c}
13 & 0 + 0 + 1 + \emptyset + 0 + \emptyset & 13 \\
45 & 1 + 0 + \emptyset + 1 + 0 + \emptyset & 45 \\
\end{array}
\]

The above process is explained by an example.

**Example - 1.2**

Consider the function given by

\[ f(x_1, x_2, x_3) = \prod (1, 2, 3, 4, 5, 6) \]

The complete table is shown in Table - 1.6

<table>
<thead>
<tr>
<th>$G \pi$</th>
<th>Decimal representation</th>
<th>$\alpha$ - Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0 + 0 + 1 + \emptyset$</td>
<td>$X$</td>
</tr>
<tr>
<td>2</td>
<td>$0 + 1 + 0 + \emptyset$</td>
<td>$X$</td>
</tr>
<tr>
<td>4</td>
<td>$1 + 0 + 0 + \emptyset$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$0 + \emptyset + 1 + 0 + 4 + \emptyset$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$\emptyset + 0 + 1 + 1 + 0 + \emptyset$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$\emptyset + 1 + 0 + 1 + \emptyset + 0$</td>
<td></td>
</tr>
</tbody>
</table>

*Table - 1.6*
The set of prime factors are
\[(x_1 + x_3')(x_2 + x_3')(x_2' + x_3')(x_1' + x_2')(x_1' + x_3')(x_1' + x_2')\]

It is easily verified that equivalent disjunctive Normal form for the same Boolean function is
\[F = \sum (0, 7) = x_1 x_2 x_3 + x_1' x_2 x_3\]
and both the forms are identical.

IV DON'T CARE COMBINATIONS

If a Boolean function with Don't care combinations is to be simplified, they are arranged according to increasing number of 1 in the vertex and then subsuming operation is performed as before.

Example 1.3
Consider the function given by
\[F = \sum (0, 2, 4, 6, 8) + \sum (10, 11, 12, 13, 14, 15),\]
whereas the subsuming operation is carried in Table 1.7 as follows.

<table>
<thead>
<tr>
<th>G</th>
<th>Decimal representation</th>
<th>Vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0 0 1 0 X</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0 1 0 0 X</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1 0 0 0 X</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0 1 1 0 X</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1 0 1 0 X</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1 1 0 0 X</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>1 0 1 1 X</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>1 1 0 1 X</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>1 1 1 0 X</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1 1 1 1 X</td>
</tr>
</tbody>
</table>

Table 1.7
So, the prime implicant is $z'$.

Here it may be mentioned that, in the method given by Quine-McCluskey [1956], $wy$, $wx$, and $z'$ are the prime implicants for the function. To find the essential Prime Implicants, following selection is to be performed from Prime Implicant chart, where only the terms in the part is taken into account.

<table>
<thead>
<tr>
<th></th>
<th>$w'x'y'z'$</th>
<th>$wx'y'z'$</th>
<th>$w'x'y'z$</th>
<th>$wx'y'z$</th>
<th>$w'x'y'z$</th>
<th>$wx'y'z$</th>
<th>$w'x'y'z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z'$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>$wy$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$wx$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table - 1.8

and thus obviously $z'$ is the essential Prime Implicant.

In Table - 1.7, the prime Implicant due to the Don't care part $\Sigma (10, 11, 12, 13, 14, 15)$ is $wy + wx$ which is further simplified while operated in Table - 1.7.

Thus in the method by Hes [1974], any addition to the original function is a don't care combination. Also, a Boolean function with a don't care combination can be simplified in a single Pass procedure.

We propose to introduce the similar concept of simplification of Boolean function with Don't care combinations in a conjunctive Normal form. This may be explained by taking a short example.
Example - 1.4

\[ F(x, y, z) = \prod (1, 3, 5) \cdot \prod (7) \]

This may be arranged as in Table - 1.9 according to decreasing order of zeros, and similarly check operation may be performed as discussed earlier.

<table>
<thead>
<tr>
<th>( G \ )</th>
<th>( \kappa )</th>
<th>Decimal representation</th>
<th>( \kappa ) - Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0 + 0 + 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0 + 1 + 1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1 + 0 + 1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>1 + 1 + 1</td>
<td></td>
</tr>
</tbody>
</table>

Table - 1.9

After the checking operation is performed Table - 1.9 becomes Table - 1.10 with uncrossed minimum \( \kappa \) - class to be \( z' \) as its prime factor.

It can be easily verified that the equivalent disjunctive Normal form of the same Boolean function, \( F = \sum (0, 2, 4) + \sum (6) \) when arranged in a Table cf. Xi17 gives the uncrossed maximum cube \( z' \) as its prime Implicant.

<table>
<thead>
<tr>
<th>( G )</th>
<th>( \Pi )</th>
<th>Decimal representation</th>
<th>( \kappa ) - Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0 + 0 + 1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0 + \emptyset + 1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>\emptyset + 0 + 1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>\emptyset + \emptyset + 1</td>
<td></td>
</tr>
</tbody>
</table>

Table - 1.10
V Conclusion

In this method, every operation is recorded by modifying the checking \(\alpha\)-class, or generating a new \(\alpha\)-class, and the past sequence of operations may affect the result of the present operation. The operations are limited in the scope of only two consecutive classes belonging to two adjacent groups \(G_{i-1}\) and \(G_i\) where \(i\) starts from \(n\) to \(1\) in the decreasing order from above for an \(n\)-variable Boolean function.

In the classical method due to Quine-McCluskey, each operation is performed individually so far as the individual table is concerned. In a faster method introduced by Hwa [1974], the same number of comparisons may be performed in a single table, probably with the possibility of performing additional operations in the table itself.

We have introduced here a similar method for obtaining minimal product form of a Boolean function in terms of products of prime factors. If the switching function of \(n\) variables is in disjunctive Normal form with minterms greater than \(2^{n-1}\), it will be convenient to write the same function in conjunctive Normal form and obtain the equivalent prime implicant in terms of product of minimal factors.

Further don't care combinations play a convenient role in the simplification of switching functions. In the classical approach introduced by Quine-McCluskey [1956], don't care operations are operated in the original table, and while finding essential Prime Implicants, don't care terms are omitted. Also, in the method introduced by Hwa [1974], and a similar method introduced in this section, don't care combinations may be arranged in the same table and usual operation will lead to the result. In fact any addition (multiplication) of literals not present in the original function implies addition (multiplication) of don't care combinations.
SECTION - II

ON SIMPLIFICATION OF BOOLEAN FUNCTIONS
In this section, the minimisation process is introduced in two steps. First step consists of reduction of the given Boolean function after breaking it into two classes either congruent to 0 (mod 2) or to 1 (mod 2). These classes named as quotient class determine the original function in terms of quotient expansion proposition. As a matter of fact, the function is expanded from the least weighted variable to the maximum weighted variable. Several cases have been discussed to obtain the set of all prime Implicants. Second step consists of a set theoretic approach to find a minimal cover of the given switching function from the family of all prime Implicants each represented as a set of decimal number avoiding any chart. We give here the following definitions, which will be used throughout this section.

**Quotient Class**

**Def - 1.5** Quotient class of a Boolean function $f(x_1, x_2, \ldots, x_n)$ is defined to be a function $f(x'_1, x'_2, \ldots, x'_n)$, where each $x'_i$ is obtained after dividing $x_i$ by 2 and adding 1 if $x_i$ is 1, or adding zero if $x_i$ is zero.

**S - set**

**Def - 1.6** $S$ - set denoted by $S_A$ is the union of all the sets in the family $F$ other than $A$, where $F$ is the family of sets in decimal numbers.

* The paper is published in the Journal of the Institution of Electronics & Telecommunication Engineers (1977) Vol.23, No.9, pp. 549-553
Prime Implicant Decimal Sets

Def - 1.7
Prime Implicant Decimal Sets are the set of decimal numbers corresponding to Prime Implicants of the given switching function initially written in terms of decimal numbers.

Obviously, the decimal numbers of the set on simple Boolean operations imply the corresponding Prime Implicant. The set in short will be denoted by PID sets.

As an example, consider the function
\[ f(v, w, x, y, z) = \sum(1, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 18, 19, 20, 21, 22, 23, 25, 26, 27) \]
which has the prime Implicant \( w'x \) from one of its PID set \( A = \{4, 5, 6, 7, 20, 21, 22, 23\} \).

Redundant Cover

Def - 1.8
A prime Implicant Decimal Set \( Y \) is said to be a redundant cover of the Boolean function \( f \) if
\[ Y \subseteq S_Y^* = S_Y \cup W \cup Z \ldots \ldots \]
where \( W, Z, \ldots \ldots \) etc are \( \{Y\} \) PID sets with the property that
\[ W \subseteq S_W^*, Z \subseteq S_Z^*, \ldots \ldots \text{etc} \]

Selected Cover

Def - 1.9
Let \( F \) be the family of PID sets of a given Boolean function \( f \). Then a cover \( X \in F \) said to be selected if
\[ X \notin S_X^* \]
where \( S_X^* \) has its usual meaning as explained.
Any Boolean function, say \( f(x_1, x_2, \ldots, x_n) \) for example, may be expanded into the following form:

\[
f(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_{n-1}, 1) x_n + f(x_1, x_2, \ldots, x_{n-1}, 0) x_n'
\]

The above proposition may be named as quotient expansion of the given function. As shown above, the given function is expanded about the variable \( x_n \). Two new functions are thus formed, the first one on division by 2 with remainder 1 and the second one on division by 2 with remainder 0. Thus the above proposition enables one to expand any \( n \)-variable switching function about its \( i \)th variable \( x_i \) also. Assume further that the variables are assigned binary weights in the order as \( x_1 \) being assigned the highest weight and \( x_n \) the lowest. When the variable weights are assigned to these literals, the expansion becomes very simple. In particular, if a specific series of decimal numbers characterizes the function, the quotients may be formed as

\[
f(x_1, x_2, \ldots, x_n) = \left[ 2 Q_1 + x_n \right] + \left[ 2 Q_2 + x_n' \right]
\]

\( Q_1 \) is a sum consisting of decimal numbers on division by 2 of the odd numbers in the switching function with remainder 1 and \( Q_2 \) is the corresponding quotient sum with remainder 0.
Noting that the quotients $Q_1$ and $Q_2$ are now each specified summations, and the new sets of decimal numbers are interpreted as being functions of all proceeding variables, generally we may write

$$Q = f(x_1, x_2, \ldots, x_{n-1})$$

Each of the quotients can now be expanded about $x_{n-1}$ which is the lowest weighted variables of the quotients. Thus the expansion can be carried out in the manner using nothing more complicated than the division by 2 and its integral power.

Example - 1.5

Consider the function $f$ given by

$$f(A, B, C, D) = (1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14)$$

$$= Q_1 \circ D + Q_2 \circ D!$$

where if $D$ is the least weighted variable, and the operation $\circ$ means multiplying $Q_1$ by 2 and adding 1. Similarly $\circ$ means multiplying by 2 and adding zero. In further operations $\circ$, $\circ$ will be replaced by $\cdot$ in its first and second position respectively.

i.e.

$$f(A, B, C, D) = Q_1 \cdot D + Q_2 \cdot D!$$
DIFERENT STEPS OF SIMPLIFICATION DEPENDING ON THE NATURE OF QUOTIENTS

We shall discuss the nature of quotients here. There are five possibilities.

(a) \( Q_1 = Q_2 = Q \) so that
\[
f(A, B, C, D) = Q_1 D + Q_2 D' = Q
\]
indicating that \( D \) and \( D' \) are redundant.

(b) \( Q_1 > Q_2 \), let \( Q_1 = Q_2 + R_b \), and
\[
f(A, B, C, D) = Q_2 + R_b \cdot D
\]

(c) \( Q_2 > Q_1 \), let \( Q_2 = Q_1 + R_a \), and
\[
f(A, B, C, D) = Q_1 + R_a \cdot D'
\]

(d) The quotients do not have any number in common
i.e. \( Q_1 \cap Q_2 = \emptyset \), so that
\[
f(A, B, C, D) = Q_1 D + Q_2 D'
\]

(e) Some of the numbers in each quotient are same,
i.e. \( Q_1 \cap Q_2 \neq \emptyset \) with \( Q_1 \neq Q_2 \) or \( Q_2 \neq Q_1 \)
Then let, \( Q_1 = Q_c + R_d \), \( Q_2 = Q_c + R_e \)
So in this case, the function may be written as
\[
f(A, B, C, D) = Q_c + R_d \cdot D + R_e \cdot D'
\]
i.e. the summation \( Q_c \) may be included as a redundancy.
The following method employs all the above ideas and will be shown by means of an example first.

Assume

\[ f(A, B, C, D) = \sum (1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14) \]

Quotient expansion about the variable D, the least weighted variable yields.

\[ f(A, B, C, D) = \left[ (1, 2, 3, 4, 6, 7) \right] \cdot D' + \left[ (1, 3, 4, 5, 6) \right] \cdot D \]

We shall denote it by

\[ 1, 2, 3, 4, 6, 7 \] \cdot D' + 0, 1, 3, 4, 5, 6 \cdot D \]

where the operation \' \cdot \' has its usual meaning as explained earlier. Thus the function may be written as

\[ = \left[ 1, 3, 4, 6 \right] \cdot D' + \left[ 1, 2, 3, 4, 6, 7 \right] \cdot D' + 0, 1, 3, 4, 5, 6 \cdot D \]

where \[ 1, 3, 4, 6 \] is that part of the D quotients which are common.

Clearly numbers 1, 3, 4, 6 are now redundant in the summation multiplying D and D'. The summation 1, 3, 4, 6 actually represents those corresponding terms of the standard sum which differ only in the D variable.
There are now three summations each of which is a function of $A$, $B$, $C$ only. Each of these functions must now be quotient expanded about the variable $C$. These operations are repeated exactly until the expansion about all variables is complete.

At this point, the above illustration will be repeated, showing a mechanical technique to organize and simplify the procedure.

Step 1 - Arrange the decimal numbers representing the given function.

Step 2 - Divide the decimal numbers into two columnar groups headed with $D$ and $D'$.

Step 3 - Include a third column headed by a dash to indicate the redundancy of $D'$ and $D$ consisting of the numbers which are common to column $D'$ and $D$. Check the corresponding numbers in column $D'$ and $D$ to record the fact that they are redundant. If any of the numbers in the dashed column have been previously checked in both the $D'$ and $D$ 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.

Each of the columns are now expanded about $C$ by repeating the above steps according to the process described earlier. The numbers below the column $C'$ and $C$ corresponding to $D'$ column are thus $0, 1, 3$ and $1, 2, 3$ respectively.
The numbers 2, 3 in the C column must be checked, for corresponding numbers 4, 6 in D column have already been checked. Similarly with 1, 3 in C' column. A dashed column consisting of the numbers 1, 3 is now included. Check the numbers 1, 3 in both C' and C columns. Thus all the numbers in C' and C columns are now checked. Hence both columns may be eliminated as shown in Fig. 1.6 in the next page.

Example 1.6

\[ f(A, B, C, D) = \sum (1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14) \]

(a) Quotient expansion about D
(b) Quotient expansion about C

Fig. 1.7 illustrates the complete development. When the function is expanded about the final variable, note that the quotient expansion must be zero. At this point, the prime implicant may be determined by simply tracing a path back to the start and reading the appropriate column or headings.

Not all the prime implicants obtained may be required to describe the function.

Similarly the function

\[ f(A, B, C, D) = \sum (0, 1, 2, 3, 4, 6, 7, 8, 9, 11, 15) \]

on simplification yields prime implicants as

A', D', C D', B' D', B' C', A' C' and A' B'.
(a)

(b)

FIG: 1 - 6
FIG 1 - 7
APPLICATION OF THE METHOD FOR A FUNCTION
WITH DON'T CARE COMBINATIONS

Let us consider the function given by
\[ f(A, B, C, D) = \sum (2, 3, 10, 11, 12, 13, 14, 15) + \sum (1, 6, 7) \]

Fig. 1.8 in the next page illustrates the complete development. On simplification, the following terms are obtained. Thus \( C, AB, A'B'D, \) and \( ABD \), are the prime Implicants of the function, where Don't care terms are taken into account during operations, and check is carried forward during operations of don't care terms. As an example \( 1 \) is carried forward under the head \( D \) as \( \sqrt{D} (1) \), and thus the total column is cancelled. If this factor is not considered we have, prime Implicant

\[ C, AB, A'B'D, ABD \]

whereas essential prime Implicants will be \( C, AB \) as can be seen in the next section. Thus if \( 1 \) is considered as a don't care element, the minimal form directly yields to \( C + AB \).

METHOD FOR SELECTION OF ESSENTIAL
PRIME IMPlicants

Method discussed above gives all the prime Implicants of a given Boolean function not necessarily all of which are essential. So the second part consists of constructing a prime-Implicant chart, and searching the redundancy of prime Implicants by inspection. If the function involves larger number of variables, then the chart becomes very often complicated. Sometimes depending on the nature of all prime Implicants (cyclic), decomposition of charts become necessary by several methods like branching methods etc. as discussed by Marcus [1969], and Kohavi [1970].
FIG. 1-8
We discuss here a simple set-theoretic approach to find a minimal cover of the given Boolean function from the set of all prime Implicants expressed in terms of decimal numbers.

Method

Write all the prime Implicant Decimal sets. Let them be $A_1, A_2, \ldots , A_n$. Start from $A_1$ and construct $S_{A_1}$ as defined. If $A_1 \subset S_{A_1}$ then $A_1$ is redundant and exclude from the following $S^*$ sets. If $A_1 \not\subset S_{A_1}$ then $A_1$ is selected for inclusion in constructing further $S^*$ sets.

For any general P.I.D. set $A_i$, examine if $A_i \subset S_{A_i}$ or $A_i \not\subset S_{A_i}$. Now $S_{A_i} = \bigcup A_j$ such that $j \neq i, m, k, \ldots$ etc with $1 \leq m, k, \ldots \leq i$ and if $A_m, A_k, \ldots$ have already been declared to be redundant as defined earlier. Repeat the process till all are examined. The family of selected covers will give a set of minimal covers for the function.

Theorem - 1.2

The family of all selected covers from the PID sets of a given switching function forms a minimal cover of the function.

Proof

Let $f$ be a given switching function and $f^*$ be the family of selected covers from the PID sets $F$ of the given function. Clearly every term of $f$ is covered by $f^*$ for rejection of redundant sets actually omits only those sets whose every element is covered by at least one set of the family $F$ which is a cover of the given function. Next step consists of showing that $f^*$ is
minimal i.e. there does not exist any other \( f^* \) family \( \in F \) with fewer number of literals, which also covers \( f \).

If \( f^* \subseteq f^* \) then the case is obvious, for \( \exists \) a member \( A \) in \( f^* \) which is not in \( f^* \) with the property that \( A \notin S_{A^*} \) i.e. \( A \) is a PID set which is not in \( f^* \) and at least one element of \( A \) is not covered by the rest member sets of the family excluding redundant sets already rejected upto the selection of \( A \).

Finally, we shall conclude the theorem after discussing the case when \( f^* \nsubseteq f^* \). Obviously if \( f^* \) covers \( f \), then \( f^* = \{ A \mid A \subseteq F, \text{ s.t. } A \notin S_{A^*} \} \). Hence \( f^* \) is the family of selected covers (by def - 1.9). Thus \( f^* \) must either be identical or another family (intersecting or disjoint) with same number of literals as in \( f^* \). The above proof depends on elementary set theoretical concepts which are available in Text as discussed by Kohavi (1970). We now discuss different cases depending on the nature of prime Implicants as well as the nature of function, and study the implication of the method at different levels.

**Case I**

When all the prime Implicants of the given switching function are selected. Consider the function given by

\[
f_1 (w, x, y, z) = (1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14)
\]

After the reduction procedure, as discussed earlier, the prime Implicant decimal sets are

\[
\begin{align*}
A &= w'y = (2, 3, 6, 7) \\
B &= wy' = (8, 9, 12, 13) \\
C &= x'z = (1, 3, 9, 11) \\
D &= xz' = (4, 6, 12, 14)
\end{align*}
\]
It can be seen that neither of the following set inclusions are satisfied and thus

\[ A \notin B \cup C \cup D \]
\[ B \notin A \cup C \cup D \]
\[ C \notin A \cup B \cup D \]
\[ D \notin A \cup B \cup C \]

Thus, A, B, C, D all are essential i.e.

\[ f_1 = w'y + wy' + x'z + xz' \]

forms a minimal cover.

Case II

Let all the prime Implicants of the given switching function be cyclic, i.e. none is essential. Finding of E.P.I can be performed in two steps. First step consists of testing all the prime Implicants to be redundant by satisfying the set theoretic inclusions, that each prime Implicant Decimal set is a proper subset of the rest decimal sets. Second step consists of choosing redundant and selected prime Implicant covers by the method as described. Thus consider the function given by

\[ f_2(w, x, y, z) = \sum (0, 1, 5, 7, 8, 10, 14, 15) \]

After the reduction process the prime Implicant decimal sets are

\[ A = w' x' y' = (0,1) \]
\[ B = x' y' z' = (0,8) \]
\[ C = w' y' z = (1,5) \]
\[ D = w x' z' = (8,10) \]
\[ E = w' x z = (5,7) \]
\[ F = w y z' = (10,14) \]
\[ G = x y z = (7,15) \]
\[ H = w x y = (14,15) \]
Hess as discussed in the classical approach by McCluskey and Bartee (1968) to solve such problems, is known as branching method, since the prime implicant chart has no dominated rows or dominating columns. Thus branching method involves several complicated charts. Our approach is as follows:

Step 1
Start with A and check for the following inclusions.

\[ A \subseteq B \cup C \cup D \cup E \cup F \cup G \cup H \Rightarrow A \text{ is not essential.} \]

Similar argument follows for other sets, and thus none of the sets in the class is essential.

Step 2
This step consists of reduction of this family, which we proceed in the similar manner.

\[ A \subseteq S_A \Rightarrow A \text{ is redundant, so } A \text{ is to be omitted in further unions. Next} \]

\[ B \notin S_B \cup A \Rightarrow B \text{ is selected, and } B \text{ is to be included in further unions. The process is continued. Thus it can be seen that, } B, C, F, G \text{ sets alone form the minimal cover. Thus} \]

\[ f_2 = x'y'z' + w'y'z + wyz' + xyz \]

Other family of minimal covers can be obtained by simply interchanging the order of starting, which may be identical or different to give the total family of minimal covers.
Case III

Here we discuss the implication when the switching function contains the Don't care parts. Then the reduction process is performed as usual by taking don't care part into account, while in finding the minimal cover, Don't care parts are omitted. Thus prime Implicant Decimal sets contain don't care part as well, which is not taken into account while testing for set inclusion relation.

As an example consider the function given by

\[ f_3(v, w, x, y, z) = \sum(13, 15, 17, 18, 19, 20, 21, 23, 25, 27, 29, 31) + \sum(1, 2, 12, 24) \]

After the reduction process the prime Implicant decimal sets are

- \( A = vz = (17, 19, 21, 23, 25, 27, 29, 31) \)
- \( B = wxx = (13, 15, 29, 31) \)
- \( C = wxx'y = (24, 25) \)
- \( D = w'xy = (20, 21) \)
- \( E = w'yxy = (18, 19) \)
- \( F = v'wxy = (12, 13) \)
- \( G = w'x'yz = (12, 18) \)
- \( H = w'x'y'z = (1, 17) \)

Similar procedures as discussed earlier imply that, \( f_3 \) has the minimal cover given by

\[ f_3 = vz + wxx + w'x'y + w'x'y' \]

That these families are minimal can be justified from the very definition of minimal cover.
A computer programme has been developed to implement the algorithm presented in the subsection XII of this section. This programme determines the essential prime implicants among a maximum of 26 sets, each having 32 as maximum number of elements. So, a switching function having 32 variables, with a maximum of 26 prime implicants can be solved by this programme for its minimal covering. Source listing in Fortran is attached at the end of the section. The first subroutine INTUNT determines union and intersection of sets. The second subroutine ISBSET determines set inclusion. This subroutine has been generalised in the third section of the chapter by accepting the don't care elements by putting tags as an identifier while directly accepting from the input. The third subroutine UNION deletes the elements in the carry forward operation and is not considered while forming the S-sets. In the main programme, checking for the property of being a set to be redundant or selected cover is performed following the algorithms. This programme has been generalised in the third section of this chapter for obtaining minimal covering of ternary switching function, where don't care states present in the function has also been taken care of.

XIV CONCLUSION

In this section, the minimisation of Boolean function has been performed in two steps. First step determines the prime implicants using the property of quotient expansion proposition which the second step determines the minimal covering from the class of prime implicant decimal sets using simple set theoretical operations.
C THIS PROGRAM IMPLEMENTS THE ALGORITHM TO DETERMINE THE
C PRIME IMPLICANTS AMONG A MAXIMUM OF 25 SETS WITH 32 AS
C MAXIMUM NUMBER OF ELEMENTS.
C INPUT: THE FIRST CARD CONTAINS THE NUMBER OF SETS, FORMAT=12
C SUBSEQUENT CARDS CONTAIN THE SETNAME(A1) FOLLOWED BY ELEMENTS(I2).
INTEGER UC(32),INT(32),ISKIP(32),ISET1(32),TEM(12),CHARS(26)
INTEGER TABLE(26,32)
READ 10,NMAX
IF(NMAX.GT.26) NMAX=26
PRINT 17
17 FORMAT (*15X,*GIVEN SETS*/14X,10(*-1))
IFLAG4 = 0
DO 15 I=1,26
DO 15 J=1,32
15 TABLE(I,J)=0
DO 20 I=NMAX
READ 25,ICHAR,TEMP
PRINT 10,ICHAR,TEMP
25 FORMAT(A1,32*I2)
10 FORMAT(I2)
CHARS(I)=ICHAR
DO 30 J=1,32
IF(TEMP(J).EQ.0 .AND. TEMP(J).LT.33) TABLE(I,TEMP(J))=1
30 CONTINUE.
20 CONTINUE
105 FORMAT(*15X,*PRIME IMPLICANTS*)
PRINT 7
7 FORMAT(*15X,*PRIME IMPLICANTS*)
DO 53 INIT=1,32
ISKIP(INIT)=0
U(INIT)=0
53 ISET1(INIT)=0
DO 100 IND=1,NMAX
ISKIP(IND)=1
DO 57 INIT=1,32
57 U(INIT)=0
CALL UNION(ISET1,U,INT, U,NMAX,ISKIP,TABLE)
DO 54 IT=1,32
54 ISET1(IT)=TABLE(IND,IT)
ICHECK=ISBSET(ISET1,U,NMAX)
IF(ICHECK.NE.0) 60 TO 100
IFLAG4 = 1
ISKIP(IND)=0
PRINT 60,CHARS(IND)
60 FORMAT(*15X,*NONE*)
CONTINUE
IF (IFLAG4.EQ.1) STOP
PRINT 27
27 FORMAT(*15X,*NONE*)
STOP
END
SUBROUTINE INTUNT PERFORMS THE TWO BASIC OPERATIONS:
VIZ: UNION AND INTERSECTION OF TWO SETS ISET1 AND ISET2.
INTSET IS THE INTERSECTION WHEREAS IUNSET IS THE UNION.

INTEGER ISET1(32), ISET2(32), INTSET(32), IUNSET(32)

DO 10 I=1,32
  IF (ISET1(I).EQ.0 .AND. ISET2(I).EQ.0 ) GO TO 10

C UNION
  IF (ISET1(I).NE.0) IUNSET(I)=ISET1(I)
  IF (ISET2(I).NE.0) IUNSET(I)=ISET2(I)

C INTERSECTION
  IF (ISET1(I).EQ.ISET2(I)) INTSET(I)=ISET1(I)

10 CONTINUE
RETURN
END
C FUNCTION ISUBSET(ISET1, ISET2, N)
C DETERMINES WHETHER ISET1 IS A SUBSET OF ISET2. IF YES
C ISBSET IS SET TO 1 ELSE ISBSET IS SET TO ZERO.
FUNCTION ISUBSET(ISET1, ISET2, N)
INTEGER ISET1(N), ISET2(N), ISUBSET, IFLAG
IFLAG=1
DO 10 I=1, N
  IF (ISET1(I).EQ.0) GO TO 10
  IF (ISET1(I).NE.ISET2(I)) IFLAG=0
  10 CONTINUE
ISUBSET=IFLAG
RETURN
END
SUBROUTINE UNION(ISET1, ISET2, INTSET, IUNSET, NMAX, ISKIP, TABLE)
  INTEGER ISET1(32), ISET2(32), INTSET(32), IUNSET(32), ISKIP(32),
  TABLE(26,32), INTSET(32)
  DO 60 ITER=1,NMAX
  IF(ISKIP(ITER).EQ.1) GO TO 60
  DO 50 IT=1,32
    ISET1(IT)=TABLE(ITER,IT)
  50 CONTINUE
  CALL INTUNI(ISET1, ISET2, INTSET, IUNSET, NMAX)
  CONTINUE
  RETURN
END
GIVEN SETS

\[ A = \{2, 4, 6, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
SECTION III

SIMPLIFICATION OF TERNARY VALUED SWITCHING CIRCUITS
This paper in its first part discusses a method for minimisation of Ternary switching circuits. In the second step, a set theoretic approach is presented in the form of an algorithm to find a minimal form for the Ternary switching function from the class of its prime Implicants and thus avoids use of prime Implicant chart as suggested by Yooie et al. (1964) as an extension of the method by McCluskey (1956). Each prime Implicant (g-type or h-type), and then the algorithms is implemented to get the minimal form for the Boolean function. The method discussed in this section is an adaptation of our binary minimisation technique discussed in section II. The notation used in this note for Ternary function is same as used by Yooie et al. (1964) and parallel definition as in section II (1977) is developed here to make the presentation self contained.

We define prime Implicants and minimal form for any Ternary switching function as follows.

Inclusion

Def. 1.9 The ternary n-variable function \( f_2 \) is included in \( f_1 \) symbolized by \( f_2 \leq f_1 \), if

\[
\forall \alpha \in L^n \text{ where } L = \{0, 1, 2\}, \text{ we have } f_2(\alpha) \leq f_1(\alpha).
\]

Two types of Implicants of any function \( f \) are defined as follows.

* The paper is communicated for publication to the Journal of the Australian Computer Society.
g - type Implicants

Def - 1.10  denoted by $\eta_g$ is a product of irredundant literals satisfying

$$\eta_g(\alpha) \leq f(\alpha) \quad \forall \alpha \in \mathbb{L}^n$$

h - type Implicants

Def - 1.11  denoted by $\eta_h$ is a product of irredundant literals satisfying

1. $\eta_h(\alpha) \leq f(\alpha) \quad \forall \alpha \in \mathbb{L}^n$

Thus prime g-type implicants denoted by $\gamma_g$ may be characterized as

$$\gamma_g \leq \eta_g \leq f \Rightarrow \gamma_g = \eta_g$$

and also prime h-type implicants denoted by $\gamma_h$ may be characterized as

$$\gamma_h \cdot f \leq \eta_h \cdot 1 \leq f \Rightarrow \gamma_h = \eta_h$$

obviously

$$f = \sum \eta_h + 1 \cdot \gamma_h$$

has a minimal cover given by

$$\sum \gamma_g + 1 \cdot \sum \gamma_h$$

Now we shall define certain more definitions which will be used frequently in this section.

Quotient T-class

Def - 1.12  of a Ternary switching function $f(x_1, \ldots, x_n)$ is defined to be a function $f(x'_1, \ldots, x'_n)$, where each $x'_i$ is obtained after dividing $x_i$ by 3. Thus $x'_i$ may attain
any value and will come under the class $x_i^0$, $x_i^1$ or $x_i^2$
depending on the nature of the remainder as 0, 1 or 2
respectively, i.e.

$$x_i = 3 \cdot x_i^3 + 0$$

or

$$x_i = 3 \cdot x_i^1 + 1$$

or

$$x_i = 3 \cdot x_i^2 + 2$$

depending on the remainder.

**S - Ternary set**

Def - 1.13 is the union of all the sets in the family
F of the sets of decimal numbers other than A, and will
be denoted by $S_A$.

Prime Implicant g-type Ternary Decimal sets

Def - 1.14 denoted by $\text{PITD} (g)$ are the set of decimal
numbers corresponding to the g-type prime Implicants of
the Ternary function in terms of decimal numbers.

Similar definitions will follow for h-type and will be
denoted by $\text{PITD} (h)$.

As an example, consider the Ternary function given in
fig - 1.9. The function has one of its g-type prime
Implicant as $x_1^1 \cdot x_2^1 \cdot x_3^0$, so that the corresponding
$\text{PITD} (g)$ sets are $A_3 = (3, 5, 12, 15)$. 
Redundant Ternary Cover

Def 1.15 Let F be the family of PITD (g) sets. Then Y ∈ F is redundant if Y ⊆ S _Y*, where Y* = Y U W U Z ... with the property that

W ⊆ S_w*, Z ⊆ S_z* ... etc.

otherwise Y will be termed as Selected Ternary Cover.

XVI

SYSTEMATIC REDUCTION TO
PRIME IMPLICANTS

The reduction process is based on the iterative use of Quotient expansion Theorem. The given function is separated into g-type and h-type Implicants, where f = g + 1. Each of these secondary functions g and h can now be expanded in the following way.

\[ g(x_1, x_2, \ldots, x_n) = g(x_1, \ldots, x_{n-1}, 0) \cdot x_n^0 + g(x_1, \ldots, x_{n-1}, 1) \cdot x_n^1 + g(x_1, \ldots, x_{n-1}, 2) \cdot x_n^2 \]

and similarly for h. We shall deal with g-type functions and similar explanation will hold for h-type as well.

Thus,

\[ g(x_1, x_2, \ldots, x_n) = [3 q_0 + x_n^0] + [3 q_1 + x_n^1] + [3 q_2 + x_n^2] \]

where \( q_0 \) is the quotient-T-class of decimal numbers with
remainder zero on division by 3, and similarly for others. Noting that $Q_0$, $Q_1$ and $Q_2$ are specified summations, we may write

$$Q = g(x_1, x_2, \ldots, x_{n-1})$$

Each of the quotient-T-class now be expanded about $x_{n-1}$ and so with division by 3 and its integral power. Thus in general

$$g(x_1, x_2, \ldots, x_n) = Q_0 \cdot x_n^0 + Q_1 \cdot x_n^1 + Q_2 \cdot x_n^2$$

We discuss now different possibilities.

**XVII**

**STEPS FOR MINIMISATION**

There are following possibilities depending on the nature of quotients.

(a) The quotient functions are equal, i.e.

$$Q_0 = Q_1 = Q_2 = Q, \text{ so}$$

$$g = Q_0 \cdot x_n^0 + Q_1 \cdot x_n^1 + Q_2 \cdot x_n^2$$

$$Q = Q(x_n^0 + x_n^1 + x_n^2) = Q$$

(b) There exist a quotient function $Q$ s.t.

$$Q \leq Q_0$$

$$Q \leq Q_1$$

$$Q \leq Q_2$$

So,

$$g = (Q_0 - Q) \cdot x_n^0 + (Q_1 - Q) \cdot x_n^1 + (Q_2 - Q) \cdot x_n^2 + Q \cdot x_n^{Q_1}$$

or

$$g = Q_0 \cdot x_n^0 + (Q_1 - Q) \cdot x_n^1 + (Q_2 - Q) \cdot x_n^2 + Q \cdot x_n^{Q_2}$$
(c) There exist a quotient function \( Q \), s.t.
\[
\begin{align*}
q &\leq q_0 \\
q &\leq q_1 \\
q &\leq q_2
\end{align*}
\]
so,
\[
g = (q_0 - q) \cdot x_n^0 + (q_1 - q) \cdot x_n^1 + (q_2 - q) \cdot x_n^2 + q
\]

(d) If none of the above conditions are satisfied, i.e.
\[
q_0 \cap q_1 \cap q_2 = \emptyset
\]
then \( g \) remains unchanged and thus,
\[
g = q_0 \cdot x_n^0 + q_1 \cdot x_n^1 + q_2 \cdot x_n^2
\]

**Algorithm - I**

The following algorithm employs all the above ideas and will be shown first by means of an example. For the sake of clarity, the example as discussed by Yoele et al (1964) is taken here also. Consider the three variable Ternary function given by
\[
f = x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3
\]

The brackets marked by \( d \) corresponds to don't care combinations.
After writing the function in its canonical sum form and then its corresponding decimal form, we get

\[ f = \sum_g (0, 1, 2, 3, 6, 12, 15, 24, 25) \]

\[ + 1 \cdot \sum_h (8, 22, 23, 26) \]

\[ + \sum (11, 14, 17) \cdot d \]

where summation over \( g \) and \( h \) indicates the secondary \( g \) and \( h \) part of the given function \( f \).

The decimal numbers will be written in two separate columns corresponding to \( g \)-type and \( h \)-type part of the function. Don't care elements are marked by \( \check{\cmark} \) as well as \( g \) elements are also included in \( h \)-type, and are marked by \( \check{\cmark} \). We now discuss the algorithm and its various steps as follows, whereas \( g \)-type implicits will be considered here and \( h \)-type will follow similarly.

1. The first expansion is about \( x_n \), i.e. the decimal numbers are divided by 3 and quotients are listed under the head \( x_n^0 \), \( x_n^1 \), \( x_n^2 \) with remainders 0, 1 and 2 respectively. With each quotient, the original decimal number is written in bracket, which is an element of the PITD (\( g \)) sets.

2. Entries derived from don't care are marked with \( \check{\cmark} \) and also in PITD (\( g \)) sets.

3. If the columns of \( x_n^0 \) and \( x_n^1 \) contain the same entry, it is listed under \( x_n^0 \) as an additional entry and originals are marked with \( \check{\cmark} \). Similarly for \( x_n^1 \) and \( x_n^2 \) entries appearing in common are listed in \( x_n^{12} \) and originals marked with \( \check{\cmark} \).
(4) If an entry appears in all the three columns, it is entered in \( x_n^{01} \) and \( x_n^{12} \) and an additional \( x_n^\Delta \), and all the entries are checked except \( x_n^\Delta \).

(5) If all the entries of a column are marked with \( \checkmark \), the column is cancelled.

(6) Now each of the derived column, which is not cancelled is expanded about \( x_{n-1} \) and the same steps (1) to (5) is repeated. In the expansion about the last, is \( \emptyset \) which denotes null set. At the end of the procedure, the resulting implicates are found by moving upwards from any lowest entry which is not cancelled. Similarly the above process is repeated for h-type implicates.

Now, applying the above algorithm as shown in fig - 1.9 and fig - 1.10, we get g-type prime Implicates as

\[
g = x_2^2 x_3^0 + x_1^6 x_3^0 + x_1^{01} x_2^{12} x_3^0 + x_1 x_2^2 x_3^0 + x_1^0 x_2^0
\]

and the respective PIMD (g) sets are given by

\[
A_1 = (6, 5, 24)
\]
\[
A_2 = (0, 3, 6)
\]
\[
A_3 = (3, 6, 12, 15)
\]
\[
A_4 = (24, 25)
\]
\[
A_5 = (0, 1, 2)
\]

Similarly h-type prime Implicates can be given by

\[
h = x_2^2 x_3^2 + x_1^{12} x_2^{12} x_3^2 + x_1 x_3^2 + x_1^2 x_2^{12} x_3^2 + x_1^2 x_3^2
\]
FIG: 1-9 g-TYPE IMPLICANTS
FIG: 1-10 h-TYPE IMPLICANTS
and the respective PITD (h) sets are:

- \( B_1 = (9, 17, 16) \)
- \( B_2 = (14, 17, 23, 26) \)
- \( B_3 = (11, 14, 17) \)
- \( B_4 = (22, 23, 25, 26) \)
- \( B_5 = (24, 25, 26) \)

Here nonappearance of the term \( x_1^0 x_2^0 x_3^1 \) from g-type prime Implicants as computed by Yoelie et al [1964] is because of don't care terms. Similarly appearance of the terms \( x_1^1 x_2^1 x_3 \) and \( x_1^2 x_2^2 \) in h-type prime Implicants compared with the result discussed in the same paper is again due to the same reason, which will be eliminated while searching for essential prime Implicants as can be seen in the following subsection.

### Algorithm - II

Method discussed above gives g-type and h-type prime Implicants of the ternary function, all of which are not necessarily essential. So the next part determines the essential prime Implicants which has been referred by Yoelie et al [1964] after using diagram due to McCluskey [1956]. If the function involves larger number of variables, then the chart becomes very often complicated.

We discuss here in this subsection, a simple set theoretic technique to find a minimal form for the ternary switching function. This is an extension of the similar method as discussed in section II [1977] of this chapter. As earlier, we will discuss about g-type and similar arguments will hold for h-type as well.

1. Write all the PITD (g) sets. Let them be \( A_1, A_2, \ldots, A_n \).

   Construct \( S_{A_1}^* \) as defined.

2. [Continues]
(2) If $A_1 \subseteq S_{A_1}^*$ then $A_1$ is redundant and to be excluded while forming the following S-Ternary-sets, otherwise $A_1$ is selected and to be included while forming S-Ternary-sets. While testing for the inclusion, any checked element arriving due to don't care nature etc are neglected.

(3) In general for any PITD (g) sets, examine if

$$A_1 \subseteq S_{A_1}^* \quad \text{or} \quad A_1 \notin S_{A_1}^*$$

where $S_{A_1}^* = \bigcup_{j \neq i} A_j \quad \text{s.t.} \quad j \neq i, m, k, \ldots \text{etc.}$

with $1 \leq m, k, \ldots < i$ and if $A_{m}, A_{k}, \ldots \text{etc.}$ have already been declared to be redundant.

**Theorem 1.3**

The family of all selected Ternary cover from the PITD (g) and PITD (h) sets of a given Ternary switching function form the minimal cover for the Ternary function.

The proof clearly parallels that of binary result discussed in Theorem 1.2, and thus the details can be verified similarly. We shall illustrate the above method in the following example, which is a continuation of earlier example considered in the section.

The PITD (g) sets are

- $A_1 = (6, 15, 24)$
- $A_2 = (0, 3, 6)$
- $A_3 = (3, 6, 12, 15)$
- $A_4 = (24, 25)$
- $A_5 = (0, 1, 2)$
To start with

\[ S_{A_1} = (0, 1, 2, 3, 6, 12, 15, 24, 25) \]

So, \( A_1 \subseteq S_{A_1} \), thus \( A_1 \) is redundant and to be excluded from further unions. Similarly \( A_2 \) is also redundant, while \( A_3, A_4 \) and \( A_5 \) are essential because e.g.

\[ A_3 \not\subseteq S_{A_3} = (0, 1, 2, 24, 25) \]

Similarly the PITD (h) sets are

\[
\begin{align*}
P_1 &= (8, 17, 26) \\
P_2 &= (14, 17, 23, 26) \\
P_3 &= (11, 14, 17) \\
P_4 &= (22, 23, 25, 26) \\
P_5 &= (24, 25, 26)
\end{align*}
\]

neglecting the checked element and applying the above procedure, clearly \( P_1 \) and \( P_5 \) are essential and others are nonessential. So the minimal form of the given function is

\[
f = x_1 x_2 x_3^0 + x_1 x_2^0 x_3 + x_1 x_2^1 x_3 x_3^0 + 1 \cdot \left[ x_2 x_3^2 + x_1 x_2 x_3 x_3^{12} \right]
\]

which resembles with the result discussed by Yoslie et al [1964].
C THIS PROGRAM IMPLEMENTS THE ALGORITHM TO DETERMINE
C THE MINIMAL FORM OF A TERNARY SWITCHING CIRCUITS.
C THE CLASS OF ESSENTIAL PRIME IMPLICANTS, BOTH G AND
C H-TYPE ARE DETERMINED FROM PITS G-TYPE AND H-TYPE
C SETS. THE ALGORITHM IS IMPLEMENTED AMONG A MAXIMUM
C OF 25 SETS, EACH HAVING 32 AS MAXIMUM NUMBER OF ELEMENTS.
C INPUT: THE FIRST CARD CONTAINS THE NUMBER
C OF SETS FOR THAT IT
C SUBSEQUENT CARD CONTAINS SETNAME(A1) FOLLOWED BY ELEMENTS.
C EACH DATA CARD IS FOLLOWED BY AN INDICATOR CONTAINING EITHER
C ZERO OR 1 INDICATING THE NATURE OF THE ELEMENT AS
C DON'T CARE OR NOT.
C
INTEGER UC32); INT(32); ISKIP(32); ISETI(32); TEMP(32); CHAR(26)
INT 'G TABLE(26,32); ITICK(26,32); TEMP(32); IDROP(32)
DATA IBLNK/* */*
DO.500 ITER=1,2
READ 10; NMAX
IF(NMAX.GT.25) NMAX=25
PRINT 17
17 FORMAT ('19,15X,'GIVEN SETS',/14X,10(''-'''))
IFLAG = 0
DO.15 I=1,26
DO.15 J=1,32
ITICK(I,J)=IBLNK
15 TABLE(I,J)=C
DO 20 I=1,NMAX
READ 25; ICHAR; TEMP
READ 19; ICHAR; TEMP2
20 FORMAT(12,32X)
PRINT 105; ICHAR; TEMP
PRINT 109; TEMP2
109 FORMAT(22X,32(A2,1X))
25 FORMAT(1,3212)
10 FORMAT(12)
CHARS(I)=CHAR
DO 30 J=1,32
IF(TEMP(J).LT.0) OR TEMP(J).GT.32) GO TO 1000
IF(TEMP(J).GT.0 AND TEMP(J).LT.33) TABLE(I,TEMP(J))=1
ITICK(I,TEMP(J))=TEMP2(J)
30 CONTINUE
28 CONTINUE
Iff ITER.EQ.1 GO TO 9
PRINT 8
PRINT 22
GO TO 11
PRINT 7
PRINT 22
CONTINUE
F
FORMAT(//,15X,*G-TYPE IMPLICANTS*,//)
22 FORMAT(//,15X,*H-TYPE IMPLICANTS*,//)
DO 53 INIT=1,32
ISKIP(INIT)=0
U(INIT)=0
CONTINUE
DO 100 IND=1,NMAX
DO 115 I JK=1,32
115 IDROP(I JK)=ITICKD(IND, I JK)
ISKIP(IND)=1
DO 57 INIT=1,32
U(INIT)=0
CALL UNION( ISET1, U, INIT, U, NMAX, ISKIP, TABLE)
DO 54 IT=1,32
54 ISET1 (IT) = TABLE (IND, IT)
ICHECK = ISBSET (ISET1, U, NMAX, IDROP)
IF (ICHECK .NE. 0) GO TO 100
IFLAG = 1
ISKIP(IND)=9
PRINT 60, CHARs (IND)
60 FORMAT(//,15X,*A1)
CONTINUE
IF (IFLAG .EQ. 1) GO TO 500
PRINT 27
27 FORMAT(//15X,*NONE*)
CONTINUE
STOP
500 FORMAT(32,TEMP(J))
STOP
END

15X,A1,2X,*= *«t32f12,1X5 »*>»/?.
C SUBROUTINE INTUNT PERFORMS THE TWO BASIC OPERATIONS:
C VIZ: UNION and INTERSECTION OF TWO SETS ISET1 AND ISET2.
C INTSET IS THE INTERSECTION WHEREAS IUNSET IS THE UNION.
SUBROUTINE INTUNT(ISET1,ISET2,INTSET,IUNSET,N)
C N REPRESENTS THE MAXIMUM DIMENSION.
INTEGER ISET1(I),ISET2(I),INTSET(I),IUNSET(I)
DO 10 I=1,N
  IF (ISET1(I) .EQ. 0 .AND. ISET2(I) .EQ. 0) GO TO 10
C UNION
  IF(ISET1(I) .NE. 0) IUNSET(I)=ISET1(I)
  IF(ISET2(I) .NE. 0) IUNSET(I)=ISET2(I)
C INTERSECTION
  IF(ISET1(I) .EQ. ISET2(I)) INTSET(I)=ISET1(I)
10 CONTINUE
RETURN
END
C FUNCTION SSBSET (ISET1, ISET2, N, ITICKED)
C DETERMINES WHETHER ISET1 IS A SUBSET OF ISET2
C WHERE THE DON'T CARE ELEMENTS IN ISET2 ARE NOT
C TAKEN INTO ACCOUNT. DEPENDING ON THIS FLAG IS
C SET TO ZERO OR ONE.

FUNCTION ISRSET (ISET1, ISET2, N, ITICKD)
INTEGER ISET1(1), ISET2(1), ISRSET, IFLAG, ITICKD(1)
DATA IBLNK/* */
IFLAG=1
DO 10 I=1, N
IF (ISET1(I).EQ.0) GO TO 10
IF (ISET1(I).EQ.ISET2(I)) GO TO 10
IF (ITICKD(I).NE.IBLNK) GO TO 10
IFLAG=0
10 CONTINUE
ISRSET=IFLAG
RETURN
END
### Given Sets

\[
\begin{align*}
A &= (6\ 15\ 24\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \\
B &= (0\ 0\ 6\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \\
C &= (3\ 6\ 12\ 15\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \\
D &= (24\ 25\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \\
E &= (0\ 1\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0) \\
\end{align*}
\]

### G-Type Implicants

\[
\begin{align*}
\text{G} \\
\text{G} \\
\text{G} \\
\text{G} \\
\text{G} \\
\end{align*}
\]
### GIVEN SETS

<table>
<thead>
<tr>
<th>Set</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>4 17 26</td>
</tr>
<tr>
<td>$j$</td>
<td>14 17 23 26</td>
</tr>
<tr>
<td>$k$</td>
<td>11 14 17 6</td>
</tr>
<tr>
<td>$l$</td>
<td>22 23 25 26</td>
</tr>
<tr>
<td>$m$</td>
<td>24 25 26</td>
</tr>
</tbody>
</table>

### K-TYPE IMPLICANTS

<table>
<thead>
<tr>
<th>Implicant</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I$</td>
<td></td>
</tr>
<tr>
<td>$L$</td>
<td></td>
</tr>
</tbody>
</table>