COMPUTER AIDED METHOD FOR
SYNCHRONOUS SEQUENTIAL MACHINE
STATE ASSIGNMENT

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

By
Miriam B. Hernandez Aji
June, 1987
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INTRODUCTION .................................. 1</td>
</tr>
<tr>
<td>2.</td>
<td>REDUCTION OF STATE TABLES .................... 10</td>
</tr>
<tr>
<td>2.1</td>
<td>State Equivalence ........................... 10</td>
</tr>
<tr>
<td>2.2</td>
<td>Algorithm for Reduction of State Tables ...... 15</td>
</tr>
<tr>
<td>3.</td>
<td>SYMBOLIC COVER AND POSITIONAL CUBE NOTATION .. 17</td>
</tr>
<tr>
<td>3.1</td>
<td>Symbolic Cover and Positional Cube Notation .. 17</td>
</tr>
<tr>
<td>3.2</td>
<td>Positional Cube Notation .................... 19</td>
</tr>
<tr>
<td>3.3</td>
<td>Constraint Relation .......................... 22</td>
</tr>
<tr>
<td>4.</td>
<td>ALGORITHMS USED ................................ 34</td>
</tr>
<tr>
<td>4.1</td>
<td>Positional Cube Representation: Translation from Symbolic Cover and Grouping of the States 35</td>
</tr>
<tr>
<td>4.2</td>
<td>State Assignment ............................. 36</td>
</tr>
<tr>
<td>5.</td>
<td>SOFTWARE IMPLEMENTATION AND RESULTS .......... 52</td>
</tr>
<tr>
<td>6.</td>
<td>CONCLUSIONS .................................... 75</td>
</tr>
</tbody>
</table>

APPENDIX

| A. | USER MANUAL .................................. 77 |
| B. | PROGRAM LISTINGS ............................ 80 |
| C. | PROOF OF THE THEOREMS ....................... 116 |
|    | BIBLIOGRAPHY ................................... 121 |
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>Title</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>General Model for Sequential Circuits</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>SM Chart</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Initial Implication Chart</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>First pass through the Implication Chart</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Final Implication Chart</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Block Diagram of State Table Reduction</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Geometrical Interpretation of Face Matrix</td>
<td>30</td>
</tr>
<tr>
<td>4.1</td>
<td>Block Diagram of the System</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Algorithm for symbolic cover conversion to positional cube notation</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>State Assignment Block Diagram</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Flow-Chart for Subroutine ORDER</td>
<td>41</td>
</tr>
<tr>
<td>4.5</td>
<td>Flow-Chart for Subroutine CSELEC</td>
<td>42</td>
</tr>
<tr>
<td>4.6</td>
<td>Flow-Chart for Subroutine VERTIC</td>
<td>43</td>
</tr>
<tr>
<td>4.7</td>
<td>Flow-Chart for Subroutine ADJOIN</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>Digital lock ASM Chart</td>
<td>54</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>PLA Table........................................ 5</td>
</tr>
<tr>
<td>1.2</td>
<td>State Table........................................ 4</td>
</tr>
<tr>
<td>1.3</td>
<td>Number of Non-equivalent state assignments... 6</td>
</tr>
<tr>
<td>2.1</td>
<td>State Table of a Synchronous Sequential Circuit for Example 2.1.................. 11</td>
</tr>
<tr>
<td>2.2</td>
<td>Reduced State Table for Example 2.1........ 14</td>
</tr>
<tr>
<td>2.3</td>
<td>Ordered Reduced State Table.................... 14</td>
</tr>
<tr>
<td>3.1</td>
<td>State Table of a Synchronous Sequential Machine for Example 3.1..................... 18</td>
</tr>
<tr>
<td>3.2</td>
<td>Symbolic Cover for State Table in Table 3.1.. 18</td>
</tr>
<tr>
<td>3.3</td>
<td>Multiple-valued Cover........................... 20</td>
</tr>
<tr>
<td>3.4</td>
<td>Reduced Multiple-valued Cover.................... 22</td>
</tr>
<tr>
<td>3.5</td>
<td>Definition of Conjunction Operation............. 25</td>
</tr>
<tr>
<td>3.6</td>
<td>Definition of Disjunction Operation............. 25</td>
</tr>
<tr>
<td>4.1</td>
<td>Symbolic Cover for Example 4.1.................. 44</td>
</tr>
<tr>
<td>4.2</td>
<td>Reduced State Table without ordering the State Numeration.......................... 45</td>
</tr>
<tr>
<td>4.3</td>
<td>Ordered Reduced Symbolic Cover.................. 46</td>
</tr>
<tr>
<td>4.4</td>
<td>Symbolic Cover in Positional Cube Representation........................................ 46</td>
</tr>
<tr>
<td>4.5</td>
<td>Symbolic Cover in Positional Cube Representation and with grouped States.......... 47</td>
</tr>
<tr>
<td>4.6</td>
<td>Constraint Matrix................................ 47</td>
</tr>
<tr>
<td>4.7</td>
<td>Constraint Matrix with Sorted Columns........... 48</td>
</tr>
<tr>
<td>4.8</td>
<td>Final Code Matrix................................ 50</td>
</tr>
<tr>
<td>4.9</td>
<td>Transition Table in implicant form............... 51</td>
</tr>
<tr>
<td>5.1</td>
<td>State Table for the Digital Combination........ 56</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.2</td>
<td>Initial minterms of next state variables obtained with SEQDES program</td>
</tr>
<tr>
<td>5.3</td>
<td>Simplified minterms of next state variables obtained with SEQDES program</td>
</tr>
<tr>
<td>5.4</td>
<td>Straight binary assignment</td>
</tr>
<tr>
<td>5.5</td>
<td>Parameters of some synchronous sequential machines encoded by SEQDES and compared with straight binary assignment</td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

The computer-aided synthesis of a synchronous sequential function can be divided into several tasks. Among them there are two fundamental blocks for the design procedure: the state table reduction and the state assignment problem. The work presented here deals with the treatment of these two problems.

Sequential networks are often used in the implementation of MSI, LSI and VLSI circuits. Special purpose digital systems and digital computers are very complex examples of sequential systems configured by an interaction of sequential networks.

A sequential network can be divided into two parts:
1. The memory elements that are capable of storing one bit of information each. For synchronous sequential circuits the memory elements are clocked flip-flops.
2. The combinational circuit which realizes the excitation functions and the output functions.

Figure 1.1 shows a general model of a sequential circuit.

Frequently, in the operation of these systems, clock pulses distributed throughout the networks are used to synchronize their operation avoiding instability problems and to allow a more systematic analysis and design because the timing of the circuits can be broken down easily into
Independent discrete steps, each of which can be considered separately. For these reasons the type of sequential circuits that we are going to consider is the synchronous one.

The behavior of a synchronous circuit can be described by a state table where the sequence of inputs, outputs, present states and next states are enumerated. The present state part of the table lists the states of the flip-flops before the occurrence of a clock pulse. The next state part indicates the states of the flip-flops under certain input values and after the occurrence of a clock pulse. The output section enumerates the values of the output variables during a present state and under a designated input combination.

![Diagram](image)

**Figure 1.1: General model for sequential circuits**

The next states can be expressed as functions of the inputs and the present states. According with the definitions the outputs can be defined as functions of the present states for Moore machines or as functions of the present states and inputs for Mealy machines.
\[ S^+ = d( S, X ) \] for Mealy and Moore machines, where:
\[ S^+ = \text{next state} \]
\[ d = \text{next state function} \]
\[ S = \text{present state} \]
\[ X = \text{present input} \]

The output for Mealy and Moore machines, respectively, is expressed as:
\[ Z = W( S, X) \]
\[ Z = W( S ) \]

where:
\[ Z = \text{output of the network} \]
\[ W = \text{output function} \]
\[ S = \text{present state} \]
\[ X = \text{present input} \]

Values of \( S^+ \) and \( Z \) can be determined from the state table. In this thesis we are going to refer to PLA tables as a variation of state tables or Symbolic cover which is defined in more detail in Chapter 3. In this representation, each set of input variables with its respective next state and outputs, is applied separately. This type of table is more appropriate for larger networks generally described by SM charts in which not all of the inputs are always tested before changing from one state to another.

Example 1.1:

As an example, let's consider in Figure 1.2, the SM
chart from page 490 of the book "Fundamentals of Logic Design", (2). The resulting PLA table is shown in Table 1.1 from page 493 of the same book. For this circuit, row 2 of the PLA table would be expanded in the way described in Table 1.2, to form a state table whose definition is larger and more difficult because each "..." in Table 1.1 has to be filled with the respective values.

Table 1.2: State Table

<table>
<thead>
<tr>
<th>Present Inputs</th>
<th>(RB RESET D7 D11 D2312 EQ) Outputs</th>
<th>State</th>
<th>00000 ... 01111 10000 ... 11111 WIN LOSE ROL SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A+B+C+ A+B+C+ A+B+C+ A+B+C+</td>
<td></td>
<td>-----------------------------------------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Present Inputs</th>
<th>(RB RESET D7 D11 D2312 EQ) Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0 ... 0 0 0 - - - - - - 0 0 0 0 0</td>
</tr>
</tbody>
</table>

We can consider the cost of a table as the number of gates, inputs and flip-flops that have to be used.

In minimizing the cost of the final circuit that results from a table, two most important factors are to be considered:

1. Reduce, whenever possible, the number of states, without altering the input-output requirements of the network.

2. Obtain an optimal state assignment.

State assignment procedures are concerned with methods for determining binary values for the states, values which allow a maximal simplification of the next state functions so that they will produce shorter and simpler combinational networks maintaining the number of the flip-flops at a
Figure 1.2: SM Chart

Table 1.1: PLA Table

<table>
<thead>
<tr>
<th>ABC RB RESET D7 D11 D2312 EQ</th>
<th>A'B'C' WIN LOSE Roll SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1    000 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>2    000 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>3    001 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>4    001 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>5    001 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>6    001 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>7    001 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>8    010 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>9    010 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>10   011 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>11   011 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>12   100 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>13   100 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>14   101 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>15   101 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>16   101 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>17   101 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
minimum. Because of their importance in producing minimal cost circuits these two problems have been extensively studied and several solutions have been proposed. However, for the state assignment problem, to date, there is no state assignment procedure that guarantees a minimal cost combinational circuit without testing all the possible networks completely, which is expensive in terms of computational time. Table 1.3 shows the number of non-equivalent assignments for each number of state variables used in a circuit.

Table 1.3: Number of non-equivalent state assignments

<table>
<thead>
<tr>
<th>Number of states</th>
<th>Number of state var.</th>
<th>Number of assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>140</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>420</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>840</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>840</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>10,810,800</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>~5.5 X 10^6</td>
</tr>
</tbody>
</table>

The state table which is formed by replacing the binary state assignment into the present and next state fields, is called a transition table.

In this thesis, we will study, implement and program algorithms for the solution of the two stated problems related to reduction of state tables and optimal state assignment. The final result that is going to be obtained is a variation of a transition table that leads to a good
realization of the network in terms of its cost.

The state assignment technique presented in this work is based, with some modifications, upon the paper: "Optimal State Assignment for Finite Machines" (1), by G. De Micheli, et al. This method was programmed initially for this thesis, with the exception of the type of input minimizer that is going to be discussed later. Then it was modified systematically in order to obtain a heuristic procedure that outputs an encoding which produces a shorter realization of the networks in most of the examples tested.

The mentioned paper introduces a new method for state encoding of synchronous sequential functions; it reduces the area of a PLA implementation with D flip-flops as memory elements. It is important to notice that the assignment obtained leads to a reasonably short combinational part of the network but not always to the best possible solution.

In this paper logic minimization is applied before state assignment over a positional cube representation of the state table. The state table reduction is performed by multiple-valued-input minimization and yields a minimal sum-of-products implementation of the next state transition function. As a first step, the present states with same output, same next state under a specific input are grouped together. If it is possible, further simplifications of the next state function are made using multiple-valued logic
algebra.

As a result of this process, a present state grouping is obtained. The encoding is then based on the group information to obtain the desired adjacencies for the state binary codes. It is interesting to notice that state grouping is generally used as a basis for most state assignment techniques to allow further simplification of the state table that leads to a shorter minterm representation of the combinational part of the network.

In the present work we are going to use other criteria to group the present states. An identical symbolic representation of the state table is going to be used.

Once the state groups are obtained, the paper defines them as constraints of the encoding problem whose solution leads to an optimal implementation of the circuit. The problem is solved with matrix notation. The state group matrix with an iterative algorithm and some defined operators and theorems are used to obtain a state code matrix which satisfies the wanted adjacencies of the states. The defined operators and the foundations of the algorithm are also utilized in this thesis for the state assignment part of the program.

Chapter 2 is devoted to the study of equivalent states and reduction of the state table by the implication chart method. The respective algorithm is presented.

Chapter 3 defines concepts, theorems and lemmas
essential for the understanding of the encoding technique used.

Chapter 4 presents a general block diagram of the system and explains the algorithms to be implemented for state assignment.

Chapter 5 shows features of the software produced and analyzes the processes inside the system that culminate with the determination of the state encoding and the respective transition table. Several examples are presented.

Chapter 6 presents conclusions and makes some recommendations related to new programs, that could be developed in the future. Such a software could be linked to our system to obtain different state assignment or to get the final excitation functions.

In addition, three appendices are included. Appendix B shows the program listings. Appendix A is a user's manual that explains the utilization of the system. Appendix C consists in proofs of theorems enunciated in Chapters 2 and 3.
CHAPTER 2
REDUCTION OF STATE TABLES

Before state assignment is carried out, it is desirable to reduce the state table to a minimal number of states, because this will produce a reduction in the cost of the network.

2.1 State Equivalence

State tables can be reduced by eliminating equivalent states. Let's define formally state equivalence.

Definition 2.1:

Let N1 and N2 be sequential networks. Let X represent a sequence of inputs of arbitrary length. Let Z1 and Z2 be the output sequences from each sequential network, where:

\[ Z_1 = W_1( S_1, X ) \]
\[ Z_2 = W_2( S_2, X ) \]

Then state S1 in N1 is equivalent to state S2 in N2 iff

\[ W_1( S_1, X ) = W_2( S_2, X ) \]

for every possible sequence of X.

Theorem 2.1:

Two states S1 and S2 of a sequential network are equivalent iff for every single input x, the outputs are the same and the next states are equivalent, i.e.,

\[ W_1( S_1, x ) = W_2( S_2, x ), \]
\[ d( S_1, x ) = d( S_2, x ) \]

Where \( W( S_1, x ) \) is the output given the present state S1 and input x and \( d( S_1, x ) \) is the next state
given the present state S1 and input x.

The proof of this theorem is shown in Appendix C.

2.1.2 Implication table

This is a procedure to find and replace all the equivalent states present in a state table. Then, the redundant entries in the state table can be eliminated to reduce it to a minimal number of states.

The implication chart method for reduction of state tables is described by example 2.1.

Example 2.1:

Let's consider the state table shown in table 2.1 that is not incompletely specified. Find all the equivalent states by the implication chart method and obtain the reduced state table.

Table 2.1: State table of a synchronous sequential circuit

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State X=0</th>
<th>Next State X=1</th>
<th>Present Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S4</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S6</td>
<td>S8</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>S5</td>
<td>S4</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>S1</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S5</td>
<td>S3</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>S6</td>
<td>S6</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>S7</td>
<td>S2</td>
<td>S8</td>
<td>0</td>
</tr>
<tr>
<td>S8</td>
<td>S3</td>
<td>S7</td>
<td>1</td>
</tr>
</tbody>
</table>

First we construct a chart which contains a square for
each pair of states. Then we compare each pair of rows in the state table. If the outputs associated with states \( i \) and \( j \) are different, we place an x in square \( i-j \) to indicate that these states are not equivalent. If the outputs are the same, we place in square \( i-j \) the pairs of states that need to be equivalent (implied pairs) for the states \( i \) and \( j \) to be equivalent. If the outputs and next state are the same, place a check in square \( i-j \) to indicate that state \( i \) is equivalent to state \( j \). Figure 2.1 shows this initial implication chart. Then we go through the chart checking each implied pair. If one of the implied pairs in square \( i-j \) is not equivalent then state \( i \) is not equivalent to state \( j \). If we found one or more non-equivalent state pairs we must go through the chart again until no more non-equivalent states are added. The remaining states are equivalent. Figures 2.2 and 2.3 illustrate this process.

In this example, we have obtained the following pairs of equivalent states: \((S_1, S_4)\) and \((S_3, S_5)\). The reduced state table is shown in Table 2.2. By reordering the state numeration we will obtain Table 2.3. The same procedure can be used to reduce a Mealey machine.

This method has been implemented by a computer program to reduce tables before the state assignment is done, whenever this is allowed. The input is a variation of the state table where each line corresponds to the next state
Figure 2.1: Initial Implication chart

Figure 2.2: First pass through the implication chart

Figure 2.3: Final implication chart
Table 2.2: Reduced state table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State X=0</th>
<th>Next State I</th>
<th>Present Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S1</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S6</td>
<td>S8</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>S3</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>S6</td>
<td>S6</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>S7</td>
<td>S2</td>
<td>S8</td>
<td>0</td>
</tr>
<tr>
<td>S8</td>
<td>S3</td>
<td>S7</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.3: Ordered reduced state table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State X=0</th>
<th>Next State I</th>
<th>Present Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S1</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S4</td>
<td>S6</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>S3</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>S4</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>S2</td>
<td>S6</td>
<td>0</td>
</tr>
<tr>
<td>S6</td>
<td>S3</td>
<td>S5</td>
<td>1</td>
</tr>
</tbody>
</table>

and outputs under a specific set of input variables.

The initial data input continues unchanged to the next part of the program if one of the following conditions are present:

1. There are not equivalent states present.
2. The table was incompletely specified.

Otherwise the input data is reduced.
2.2 Algorithm for Reduction of State Table

The algorithm shown in Figure 2.4 is applied in case the conditions 1. and 2., just mentioned, are not present.

START

FORM INITIAL IMPLICATION CHART
COMPARE OUTPUTS TO OBTAIN IMPLIED PAIRS

CHECK EQUIVALENCE OF IMPLIED PAIRS

HAVE NEW NON-EQUIVALENT STATES BEEN ADDED

yes

no

ELIMINATE EQUIVALENT STATES

ORDER THE REDUCED STATE TABLE

RETURN

Figure 2.4: Block Diagram of State Table Reduction

The program which implements this algorithm is called IMCHAR. It uses only one subroutine called COMPF.

Subroutine COMPF compares the output fields to form and update in each pass the implication chart.

Next, in page 16, the computer listing for Example 2.1 is shown. Since this Chapter is only about reduction of state tables, the part related to state assignment is not presented.
Computer Output for Example 2.1

<table>
<thead>
<tr>
<th>SYMBOLIC COVER</th>
<th>INPUT PS NS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 1 4 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000002 2 6 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000003 3 5 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000004 4 1 00000002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000005 5 3 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000006 6 6 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000007 7 2 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000008 8 3 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000009 9 3 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000010 1 3 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000011 2 4 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000012 3 4 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000013 4 5 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000014 5 1 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000015 6 2 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000016 7 8 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000017 8 7 00000001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REDUCED SYMBOLIC COVER</th>
<th>INPUT PS NS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 1 1 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000002 2 4 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000003 3 3 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000004 4 1 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000005 5 2 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000006 6 3 00000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000007 1 3 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000008 2 6 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000009 3 1 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000010 4 2 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000011 5 6 00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000012 6 5 00000001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 3

SYMBOLIC COVER AND POSITIONAL CUBE NOTATION

The concepts of symbolic cover and positional cube notation are going to be used. First, let us define these concepts that will later be used to find the state assignment of minimal code length that minimizes the amount of logic required to realized a circuit.

3.1 Symbolic Cover and Symbolic Implicants

Symbolic cover is a set of primitive elements called symbolic implicants. A symbolic implicant, for our purposes, has four fields corresponding to the primary inputs \(i\), present state \(s\), next state \(s^+\), and primary outputs \(o\).

A symbolic implicant represents a state transition if:

\[ s^+ = d(i, s), \text{ and} \]
\[ o = W(i, s) \]

where:

\(d\) = next state function
\(W\) = output function

A symbolic cover consisting of symbolic implicants that represent all state transitions is equivalent to a representation of the machine by a state table.

Example 3.1:

Consider the state table described in Table 3.1, and do the respective conversions.
Table 3.1: State table of a synchronous sequential machine

<table>
<thead>
<tr>
<th>Present State</th>
<th>X=0</th>
<th>1</th>
<th>Next State</th>
<th>Present Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S4</td>
<td>S5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>S5</td>
<td>S5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>S6</td>
<td>S6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>S6</td>
<td>S7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S6</td>
<td>S1</td>
<td>S1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S7</td>
<td>S1</td>
<td>S1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.2: Symbolic cover for state table in Table 3.1

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S1</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>S2</td>
<td>S4</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>S3</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S4</td>
<td>S6</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S5</td>
<td>S6</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>S6</td>
<td>S1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S7</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>S1</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S2</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S3</td>
<td>S5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>S4</td>
<td>S6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>S5</td>
<td>S7</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S6</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>S7</td>
<td>S1</td>
<td>0</td>
</tr>
</tbody>
</table>
The following is a symbolic implicant:

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

The symbolic cover can be obtained translating one by one all the symbolic implicants. Table 3.2 illustrates the symbolic cover for Table 3.1.

### 3.2 Positional Cube Notation

Positional cube notation consists of the representation of a p-valued logical variable as a string of p binary symbols. Value \( r \) is represented by a "1" in the \( r \)th position, all others being "0". The logical OR of logical variables is represented by "1"s in the corresponding positions. The don't care value is represented by a string of "1"s and the empty value as a string of "0"s.

Using the last definition, we can transform straightforwardly a symbolic cover into a multiple-valued cover with positional cube notation. Let us consider example 3.2.

**Example 3.2:**

Transform the symbolic cover of example 3.1 to a multiple-valued cover with positional cube notation. Let us consider example 3.2.

Each state in the symbolic implicant is translated into positional cube notation. Thus, \( S_1 \) is represented by 1000000, \( S_2 \) by 0100000, etc.

The multiple-valued symbolic cover obtained is illustrated in Table 3.3.
Table 3.3: Multiple-valued cover

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000000</td>
<td>0100000</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0100000</td>
<td>0001000</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0010000</td>
<td>0000100</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0001000</td>
<td>0000010</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0000100</td>
<td>0000010</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0000010</td>
<td>1000000</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0000001</td>
<td>1000000</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1000000</td>
<td>0010000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0100000</td>
<td>0000100</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0010000</td>
<td>0000100</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0001000</td>
<td>0000010</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0000100</td>
<td>0000001</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0000010</td>
<td>1000000</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0000001</td>
<td>1000000</td>
<td>0</td>
</tr>
</tbody>
</table>

In general, most states assignment techniques produce encodings which utilize the adjacencies due to grouping of the states to simplify the realization of the combinational part (excitation functions) of the networks.

In the present work, the following guideline will be used in grouping the states to obtain a state assignment that leads with good approximation to an optimal machine:

"States which have the same next state for a given input should be given adjacent assignments."
Based in this guideline we can reduce the multiple-valued cover, packing together the states with same next state for a given input. This can be done because a symbolic implicant can represent a transition from one or more states to a next state under some input conditions.

Example 3.3:
Reduce the multiple-valued cover of Table 3.4 according to the guideline stated before.

Table 3.4 enumerates the symbolic implicants of the reduced symbolic cover. For instance the fourth symbolic implicant means that under an input of "0" the present states S4 and S5 have S6 as a next state and an output that can be either "0" or "1".

The states that are grouped together are called a state group.

Given a state assignment and a state group, the corresponding group face or face is the minimal dimension subspace containing the encodings of the states assigned to that group.

The state assignment presented here uses the guideline to form state groups and then encodes the states according to the following consideration:

Given a set of state groups find an encoding that satisfies as much as possible the desired state adjacencies, i.e., each group face contains all or most of the encodings assigned to that group.
Table 3.4: Reduced multiple-valued cover

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000000</td>
<td>0100000</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0100000</td>
<td>0001000</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0010000</td>
<td>0000100</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0001100</td>
<td>0000010</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0000011</td>
<td>1000000</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>1000000</td>
<td>0010000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0110000</td>
<td>0000100</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>0001000</td>
<td>0000010</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0000100</td>
<td>0000001</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0000011</td>
<td>1000000</td>
<td>-</td>
</tr>
</tbody>
</table>

Whenever possible, and without incrementing the code length, try to avoid intersection between a group face and the code assigned to any state not contained in the corresponding group.

3.3 Constraint Relation

The constraint relation can be stated as follows:

Given a set of state groups, find an encoding such that each group face does not intersect the code assigned to any state not contained in the corresponding group.

We are going to construct an encoding of the states that satisfies the constraint relation whenever possible without incrementing the code length and therefore maintaining the number of flip-flops used in the
implementation, to a minimum.

With the guideline mentioned in Section 3.2, a grouping of the states can be obtained that allows one to find a state assignment that will place the "1"s on the flip-flop input Karnaugh maps in adjacent squares or that produces a similar combination of terms in whichever simplification method is used, so that the corresponding terms can be combined.

The constraint matrix in the paper by De Micheli, et al. is formed with the grouped rows of states that appeared in the minimal symbolic cover which is an output of a minimal symbolic multiple-valued-input minimizer.

The approach presented in this thesis reduces first the state table by the implication chart method, and then groups the states according to the guideline stated in Section 3.2.

As it is known, the efficiency of a state assignment is dependent on the kind of memory elements to be used. This method works for a good number of cases to get a state assignment that leads to a minimum two level sum-of-products solution for D and J-K flip-flops.

The geometric interpretation of the constrained encoding problem in the way treated here is:

Find the minimal dimension Boolean space in which each group face contains as many as possible of the
encodings assigned to that group.

To solve the constrained encoding problem we are going to use matrix notation and pseudo-boolean variables with values that can be 0, 1, * (don't care: either 0 or 1), and \( \emptyset \) (empty: neither 0 nor 1).

Let's define some operations and concepts that are going to be utilized in the definition of the problem.

3.3.1 Conjunction

This operation has as symbol \( \land \), and can be defined according to the contents of Table 3.5.

3.3.2 Disjunction

The symbol of this operation is \( \lor \), and its description is illustrated in Table 3.6.

3.3.3 Selection

Let: \( a \in \{ 0, 1 \} \)
\( b \in \{ 0, 1, *, \emptyset \} \)

The selection of \( b \) according to \( a \) is:

\[
\begin{align*}
    a \cdot b &= b, & \text{if } a = 1 \\
    &\quad \emptyset, & \text{if } a = 0
\end{align*}
\]

Selection can be extended to two-dimensional arrays and is similar to matrix multiplication.

The replacements of operations that have to be made as a substitute for the multiplication operations are the following:

1. Disjunction is used instead of addition.
2. Selection replaces multiplication.
Table 3.5: Definition of conjunction operation

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
<th>( I \wedge I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 *</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 ( \phi )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 *</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 ( \phi )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>* 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>* 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>* *</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* ( \phi )</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( \phi ) 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \phi ) 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \phi ) *</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( \phi ) ( \phi )</td>
<td>( \phi )</td>
<td>( \phi )</td>
</tr>
</tbody>
</table>

Table 3.6: Definition of disjunction operation

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
<th>( I \vee I )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0 *</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0 ( \phi )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 *</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1 ( \phi )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>* 0</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* 1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* *</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>* ( \phi )</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( \phi ) 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \phi ) 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \phi ) *</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>( \phi ) ( \phi )</td>
<td>( \phi )</td>
<td>( \phi )</td>
</tr>
</tbody>
</table>
Example 3.4:

Obtain the selection of the matrix $B$ according to matrix $A$, where:

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

and:

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

$$
A \cdot B = \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0
\end{bmatrix}
= \begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 0 \\
* & 1 & *
\end{bmatrix}
$$

3.3.4 Constraint Matrix $A$

It is the matrix formed by the present state field of the multiple-valued cover without considering neither the rows corresponding to one state only nor the repeated groupings that have already been described.

A row of the constrained matrix is called a meet if it represents the conjunction of two or more state groups, otherwise the row is called prime. For instance if we have two prime rows representing the groups formed by (state 3 and 4), and (state 3 and 6) that will be equal to 001100 and 001001 respectively, then another row equal to 001000 would be a meet because it represents state 3 that has been already considered in the first two rows.
In the state assignment problem it is necessary to consider only the prime rows of $A$ because all the information of state grouping according to the used guideline is contained in these rows.

Let us consider an example.

Example 3.5:

From the reduced multiple-valued cover of Example 3.3, obtain the respective constraint relation matrix.

The constraint relation matrix produced is shown in Table 3.7.

Table 3.7: Constraint matrix for Example 3.3

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

3.3.5 State Code Matrix

The matrix whose rows are binary assignments for each state is called State Code Matrix.

$$S \in \{0, 1\}^{ns \times nb}$$

$$S = \begin{bmatrix}
S_1 \\
S_2 \\
S_3 \\
\vdots \\
S_{ns}
\end{bmatrix}$$

where:

$ns = \text{number of states}$
nb = code length
Sl = state code

3.3.6 Face Matrix

It is the matrix whose rows are the group faces defined in Section 3.2.

\[ F \in \{ 0, 1, *, \emptyset \}^{n_1 \times nb} \]

\[
\begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
\vdots \\
f_{n_1}
\end{bmatrix}
\]

where:

\( n_1 = \) number of groups
\( nb = \) code length
\( f_1 = \) group faces

Selection operation is going to be useful to determine the group faces corresponding to a group set according to a constraint matrix and a given state assignment.

In other words, the face matrix can be obtained as the selection of the state code matrix \( S \) according to the constraint matrix \( A \):

\[ F = A \cdot S \]

An encoding that has all the desired adjacencies in the group faces and in which any group face does not intersect the code assigned to any other state not contained in the corresponding group, satisfies the
following relation:

\[
\begin{bmatrix}
  f_1^i \\
  f_2^i \\
  f_3^i \\
  \vdots \\
  f_{nl}^i
\end{bmatrix} \land 
\begin{bmatrix}
  f_1 \\
  f_2 \\
  f_3 \\
  \vdots \\
  f_{nl}
\end{bmatrix}
\]

For all:

\[i = 1, 2, 3, \ldots, ns, \text{ and}\]

\[
\overline{F}^i = a.i \land si
\]

where:

\(f_i\) = minimal subspace containing state encodings for the \(i\) group, and

\(a.i\) = the complement of column \(i\) of the constraint matrix,

\(si\) = encoding of the state corresponding to column \(i\) of the constraint matrix

\(nl\) = number of groups

\(ns\) = number of states

Example 3.6:

Obtain the face matrix for the constraint matrix of Example 3.3.

\[
A = \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
Let the state matrix \( S \) be:

\[
S = \begin{bmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
0 & 0 & 1 \\
1 & 1 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

\[
F = A \cdot S = \begin{bmatrix}
* & 1 & 0 \\
* & 0 & 0 \\
* & 0 & 1 \\
\end{bmatrix}
\]

The geometrical interpretation of the faces is shown in Figure 3.1. Each face in this case is a line because it is formed by two state codes.

![Geometrical Interpretation of the Face Matrix](image)

**Fig 3.1: Geometrical interpretation of the face matrix**

### 3.3.7 Some Useful Theorems and Lemmas

Let us define some theorems and lemmas which deal with a set of transformations that are going to be applied in constructing an algorithm for optimal state assignment. The
corresponding proofs of these theorems and lemmas can be found in Appendix C.

**Lemma 3.1**

If \( S \) satisfies the constraint relation for a given \( A \), then \( S' = [ S \ T ] \) satisfies the constraint relation, where \( T \) is any \( \{ 0, 1 \} \) matrix with \( ns \) rows.

**Lemma 3.2**

If \( S \) satisfies the constraint relation for a given \( A \), then \( S \) satisfies the constraint relation for \( A' = \begin{bmatrix} A \\ am \end{bmatrix} \), where \( am \) is a meet of \( A \).

Lemma 3.1 and 3.2 are utilized to demonstrate the following theorems that are going to be useful in the state assignment algorithm.

**Theorem 3.1**

\( S \) satisfies the constraint relation for \( A \) if and only if \( S \) satisfies the constraint relation for \( Ap \), where \( Ap \) are the prime rows of \( A \).

Theorem 3.1 allows one to reduce the size of the problem considering only the prime rows of \( A \) to obtain an optimal state assignment. This justifies the assumption that was made before, that for the state assignment problem it is enough to consider the prime rows of the constraint matrix because they already have all the information about the present state groupings.

**Theorem 3.2**

Let \( S \) satisfy the constraint relation for a given \( A \).
Then, there is a matrix $S'$ such that:

$$S' = \begin{bmatrix} S & T \end{bmatrix}, \ T \in \{0\}$$

where:

$ns$ = number of states

$\sigma$ = encoding for the state.

$S'$ satisfies the constraint relation for:

$$A' = \begin{bmatrix} A \\ \alpha \end{bmatrix}$$

such that either:

i) $\alpha$ is a column of "0"s. Or,

ii) $\tilde{A}$ has a column of "1"s where $\tilde{A}$ is the set of nonzero rows of $A$ corresponding to the nonzero entries of $\alpha$.

This theorem allows one to obtain a state assignment through the use of an iterative procedure in the following way. Assume we have an encoding being represented by $S$ satisfying the constraint relation for a given constraint matrix $A$, and we want to assign a new state that corresponds to an expanded $A' = \begin{bmatrix} A & \alpha \end{bmatrix}$, where $\alpha$ is a new column of the matrix, i.e., a new state to be coded. The best solution will be to find a new $S' = \begin{bmatrix} S \\ \sigma \end{bmatrix}$, where $\sigma$ is the code for state $\alpha$, without incrementing the binary length of $S$. This situation is not always present because the Boolean dimension could be totally used and thus could there be no more possible codes to be chosen. In this case it is possible to increment the code matrix with a column of zeros, i.e., a $T$ vector, in the rightmost position and then obtain $S' = \begin{bmatrix} S & T \end{bmatrix}$, that will satisfy the constraint
relation if the conditions i and ii are met.

In the other cases where conditions i and ii are not present, \( S' = [ S T ] \) will satisfy the required adjacencies for the state to be coded but it will not produce an encoding of that state which does not intersect with other group faces.

In the paper by De Micheli, et.al., if one state cannot be encoded such that it satisfies the constraint relation, then the code length is incremented even though the Boolean space would have not been totally used. This results in an encoding that is not always of minimal length. Understanding as minimal code, one that needs for \( m \) states \( n \) number of flip-flops \( ( n-1 < \log m \leq n ) \).

In the program presented here the adjacencies determined by the constraint matrix and the constraint relation are satisfied as much as possible without incrementing the code length. This modification was made because, in most of the examples tested, increasing the Boolean dimension of the state code in order to satisfy the constraint relation did not yield a shorter realization of the design (i.e. one with lower cost) as compared to the design obtained by maintaining the code length at a minimum. This makes our method a general one that can be used not only for PLA-based networks (as De Micheli, et. al. algorithm is), but also for any kind of synchronous networks.
CHAPTER 4
ALGORITHMS USED

The software developed in the present work accepts as data, a set of present states, next states and outputs, i.e. a boolean cover. The program reduces this table, if the conditions stated in Chapter 2 are met, and encodes the states with a state assignment that produces a reasonably small two-level combinational implementation and a minimal number of D or J-K flip-flops.

The outline of the system is presented as a block diagram in Figure 4.1.

![Block Diagram of the System](image)

Figure 4.1 Block Diagram of the System
4.1 Positional Cube Representation: Translation from Symbolic Cover and Grouping of the States

The next part of the system is fed with the reduced table, if there is any, or with the initial one, if the conditions for state reduction were not satisfied. This part of the program converts this table into positional cube notation.

The algorithm for this conversion is detailed in the block diagram of Figure 4.2.

![Block Diagram](image)

**Figure 4.2: Algorithm for symbolic cover conversion to positional cube notation**

As was explained before in Chapter 1 and 2, the symbolic cover is entered as a set of implicants, i.e., PLA table. The translation of symbols from the symbolic cover to positional cube notation is made by subroutine TRANS.

Subroutine MINIM reduces the positional cube representation of the symbolic cover by merging states with the same input and next state according to the guideline mentioned in Section 3.2.

In order to perform these tasks this program uses the
subroutines ONENP, COMPF and MERGE.

Subroutine ONENP detects the position of each "1" in the present state field of the symbolic cover.

Subroutine COMPF compares the input and next state fields and determines whether they are identical or not.

Subroutine MERGE combines the "1"s in the present state field of the rows having the same input and next state field. In this way, this subroutine applies the Guideline stated in Section 3.2.

Then, MINIM forms the constraint matrix with the grouped symbolic implicants, drops the meet rows of the symbolic cover and keeps only the prime rows that already have all the necessary group information according to theorem 3.1.

4.2 State Assignment

As it was stated before, the amount of logic required to implement a sequential network is strongly dependent on the state assignment and, to date, it is not known whether an optimal solution can be computed by a procedure which does not try all possible non-equivalent state assignments.

For type D and J-K flip-flops generally the guideline stated in Chapter 3 works well, in most cases, to obtain a state encoding that leads to a network of minimal cost.

A heuristic algorithm that obtains adjacencies according to this guideline is presented here.

If it is desired, the procedure permits pre-assigning
one or more state binary codes. This is a new option implemented here that has not been included by De Micheli's program that allows the user to have control over one or more of the codes to be obtained. The encoding algorithm constructs the state code matrix $S$ of the remaining states by the following iterative procedure:

1. Initial code length set to 1. Assign $S(1,1) = 0$, where $S(1,1)$ is the first bit of the first state to be assigned.
2. Select an uncoded state.
3. Determine the encodings for that state which satisfy the partial constraint relation that takes into account only the set of states formed by the considered state and the ones already coded.
4. If no encoding satisfies the constraint relation and the boolean space has not been totally used, go to step 5, otherwise increment the code length and go to step 3.
5. Assign a code to the state.
6. If there is a state(s) that has not been coded yet, go to step 2, else go to step 7.
7. If it is the first pass and the constraint relation was not satisfied one or more times, go to step 1 again, but this time select $S(1,1) = 1$.

If this is the second pass, by comparing the assignment that started with $S(1,1)=0$ and the one that started with $S(1,1)=1$, determine which assignment satisfied better the constraint relation and choose that one.
If it is the first pass and the constraint relation was always satisfied, stop.

As it can be observed, the state assignment procedure implemented here checks two different kinds of encoding, one initiating the process with "0" and the other with "1", then chooses the encoding that satisfies better the constraint relation and probably leads to a realization of smaller cost. The De Micheli, et al. method starts assigning "0" to the first bit corresponding to the first state coded and it does not try any other encoding.

A more detailed description of the algorithm is presented in Figure 4.3.

Subroutine ORDER performs the following tasks:
1) Interchanges the columns of A according to their one count such that the uncoded state belonging to the largest number of prime groups (highest column count of "1" count in A) is selected first. The fewer the states that have been encoded, the higher the probability of finding a code that satisfies the constraint relation. This is the reason why the state that is the most critical to code, i.e., belonging to more groups is the state that is encoded first.
2) Keeps a record of the order in which the states are being coded and of the states that have been pre-assigned.

A flow-chart of this subroutine is presented in Figure 4.4.
Subroutine MOVE takes the present state field of the constraint matrix, that is $A(Ng,Ns)$, where $Ng$ is the number of the state groups, and $Ns$ is the number total the states.

Subroutine COUNT counts the number of "1"s in each column of the constraint matrix and creates a $2 \times Ns$ array called JP, where $JP(1,J)$ is the number of "1"s and $JP(2,J)$ is the ordinal of the state. J goes from 1 to the number of columns of the constraint matrix, i.e., to the number of the states in the constraint matrix.

Subroutine SORT sorts the first row of JP array and moves the second row accordingly.

Subroutine MOVEC permutes the columns of the constraint matrix according to the order given in the second row of the JP array.

Subroutine CSELEC assigns a code to a selected state if the candidate set is not empty. The encoding is made according to the following criteria:

1) If the state is not considered in the constraint matrix choose any code not covered by any face.

2) If the state appears in the constraint matrix choose a candidate among others that better satisfies as many as possible of the following conditions:
   a) The constraint relation.
   b) The desired adjacencies.
   c) It is related with the less covered face.

The flow-chart is described in Figure 4.5.
Figure 4.3: State Assignment Block diagram
Figure 4.4: Flow-chart for Subroutine Order

Subroutine SELECT implements the selection operation according to its definition in Section 3.3.3. This program calls subroutine DISJ which implements the disjunction operation defined in Section 3.3.2.

Subroutine VERTIC orders candidates in conformity with less used number of vertices in each face. The corresponding flow-chart is presented in Figure 4.6.

Subroutine ADJOIN is invoked when the code length has to be increased because all of the boolean space has been used and the candidate set is empty. ADJOIN receives the state code matrix S and adds a column of "0"s to the rightmost position of each of its rows that has not being pre-assigned, i.e. returns \[ S \oplus T \], where \( T = \{ 0 \} \) and \( n_s \times 1 \) ns is the number of non-pre-assigned states. This subroutine uses theorem 3.2 to increment the length of the state code matrix when there are no more codes available in
SUBROUTINE SELECT APPLIED TO S AND A TO OBTAIN THE FACE MATRIX FOR LAST CODE SET

STATE NOT IN CONST. MATRIX?

YES

CHOOSE ANY CODE NOT COVERED BY A FACE

RETURN

NO

SUBROUTINE VERTIC

TAKE FIRST CODE AFTER SUBROUTINE VERTIC

SUBROUTINE SELECT APPLIED TO S1 AND ai TO OBTAIN Fi AS DESCRIBED IN SECTION 3.3.6

SUBROUTINE CONJ

Fi∧ F

Fi∧ F is empty?

YES

SELECTED CODE IS VALID

RETURN

NO

IS THIS THE LAST AVAILABLE CODE?

YES

CHOOSE AS CODE CANDIDATE FROM VERTIC

RETURN

NO

CHOOSE NEXT AVAILABLE CODE

10

Figure 4.5: Flow-chart for Subroutine CSELEC
a Boolean space of a specific dimension.

Say, for a code length of 1, \( S = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \). Therefore all the possible codes are being used. The dimension has to be incremented to 2 and the codes are now: \( S = \begin{bmatrix} 00 \\ 10 \end{bmatrix} \) hence, the candidate codes are 01 and 11.

When subroutine ADJOIN appends T to S the size of a face not related to the selected state is not increased
and a state code that satisfies all of the three conditions mentioned before is always found after a call is made to this subroutine when the requirements of Theorem 3.2 are met. The respective flow-chart is illustrated in Figure 4.7.

Subroutine GCAND generates the boolean matrix each time the code length is increased.

Subroutine CAND compares the coded states with the boolean matrix and returns the candidate codes that have not been used yet.

Let us consider an example of the algorithm performance.

Example 4.1:

Consider the synchronous sequential Moore network described by Table 4.1.

Table 4.1: Symbolic Cover for example 4.1

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State X=0</th>
<th>Output</th>
<th>Present State X=1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S6</td>
<td>S4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>S4</td>
<td>S1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>S8</td>
<td>S2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>S2</td>
<td>S3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>S7</td>
<td>S2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>S1</td>
<td>S8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td>S5</td>
<td>S3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S8</td>
<td>S3</td>
<td>S6</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The conditions stated in Chapter 2 are satisfied. Thus, this symbolic cover can be reduced to the one presented in Table 4.2 using the implication chart method.

The equivalent states found after 2 passes through the implication chart are the sets: $(S_1, S_3), (S_2, S_4)$ and $(S_6, S_8)$.

Table 4.2: Reduced state table without ordering the state numeration

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>X=0</th>
<th>1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S6</td>
<td>S2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S2</td>
<td>S1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>S7</td>
<td>S2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S6</td>
<td>S1</td>
<td>S6</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S7</td>
<td>S5</td>
<td>S1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

The rightmost states are replaced by the states at the left of the equal sign. Then the redundant entries in the table are dropped. The result is described in the state table in Table 4.2. The ordered table in implicant cover form is presented in Table 4.3.

Then the symbolic cover is translated to positional cube representation as shown in Table 4.4.

The states are grouped according to the guideline and Table 4.5 is obtained.

The program drops the rows that are composed by only one state and the rows that are meets, if there are any, according to the definition in Section 3.3.4, and forms the
Table 4.3: Ordered reduced Symbolic Cover

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>S1</td>
<td>S4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S2</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>S3</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S4</td>
<td>S1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S5</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S1</td>
<td>S2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S2</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>S3</td>
<td>S2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S4</td>
<td>S4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>S5</td>
<td>S1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4: Symbolic Cover in positional cube representation

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000</td>
<td>00010</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>01000</td>
<td>01000</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>00100</td>
<td>00001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>00010</td>
<td>10000</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>00001</td>
<td>00100</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10000</td>
<td>01000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01000</td>
<td>10000</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>00100</td>
<td>01000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>00010</td>
<td>00010</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>00001</td>
<td>10000</td>
<td>0</td>
</tr>
</tbody>
</table>
constraint matrix as it is shown in Table 4.5.

Here the state assignment process starts. It is an iterative one.

The number of "1"s in each column is counted and the columns of the constraint matrix are ordered in such a way that the columns with more "1"s corresponding to the states that belong to more groups, can be encoded first.

Table 4.5: Symbolic Cover in positional cube representation and with grouped states

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000</td>
<td>00010</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>01000</td>
<td>01000</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>00100</td>
<td>00001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>00010</td>
<td>10000</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>00001</td>
<td>00100</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10100</td>
<td>01000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01001</td>
<td>10000</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>00010</td>
<td>00010</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.6: Constraint matrix

<table>
<thead>
<tr>
<th>Present State</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10100</td>
<td></td>
</tr>
<tr>
<td>01001</td>
<td></td>
</tr>
</tbody>
</table>

The constraint matrix is shown in Table 4.6. In this case all columns, except one, have one "1" and the SORT routine is then called. Because of the way in which this
subroutine is implemented: the order of the columns is always changed. The interchanged columns of the constraint matrix are illustrated in Table 4.7.

The new order of the states is S5, S1, S3, S2, S4.

Table 4.7: Constraint matrix with sorted columns

<table>
<thead>
<tr>
<th>Present State</th>
<th>01100</th>
<th>10010</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Several iterations take place in each pass. Let us see each one of them.

**First Iteration**

Since no pre-assignment code was defined, and according to the permuted constraint matrix, the first state to be coded is state 5.

\[ S(1,1) = 0 = S5 \]

The first bit of the encoding of state 5 is 0.

The dimension is 1 and the remaining code of the boolean space is the next candidate: "1".

**Second Iteration**

The candidate code is 1 and satisfies the constraint relation, therefore it is assigned to the next state in the constraint matrix that is S1. Then,

\[ S1 = 1 \]

The boolean space has been totally used and the candidate set is empty.
Third Iteration

The boolean space is increased to 2. A column of zeros is added to the rightmost position of the states 5 and 1, such that:

S5 = 00, and
S1 = 10

Now the candidates are 01, 11.

The candidates are ordered in such a way that the program will next take a bit of code that corresponds to the less covered face. In this case, the chosen code is 11, then:

S3 = 11

Fourth Iteration

The next candidate is assigned to state 2:

S2 = 01

Fifth Iteration

The candidate set is empty, so the length of the code is now incremented to 3.

The third bit of each coded state is assigned a 0:

S5 = 000
S1 = 100
S3 = 110, and
S2 = 010

The candidate set is now:
C = { 001, 101, 011, 111 }

Since S4 is not a part of the constraint matrix
because its respective column in the constraint matrix consists of "0"s, S4 can be any code not covered yet, for instance 001. Then:

\[ S4 = 001 \]

There are no more states to be encoded, now the program checks if the constraint relation was always satisfied. In fact this was the case in this example so the program stops. In case the constraint relation was not always satisfied, the assignment routine starts again with \( S(1,1) = 1 \), i.e., encoding as 1 the first bit of the first state to be considered. This initialization will produce a different assignment. Both encodings are compared and the one with more adjacencies satisfied is the one chosen.

The final code matrix is shown in Table 4.8, the states appear in the same order in which they were encoded. Subroutine ORDER stores the information about which code corresponds to each state. The transition table, in implicant form, is formed replacing the codes of the states in the symbolic cover as shown in Table 4.9.

**Table 4.8: Final code matrix**

\[
S = \begin{bmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

The codes correspond to states 5, 1, 3, 2 and 4
respectively.

Table 4.9: Transition table in implicant form

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>010</td>
<td>010</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>110</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>000</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>
CHAPTER 5
SOFTWARE IMPLEMENTATION AND RESULTS

The algorithms discussed in Chapter 4 have been implemented by a computer program called SEQDES (Sequential Design).

The system is coded in VS Fortran in the IBM 4381 of the Ohio University Computer Center and it consists of approximately 1200 lines.

The program accepts as input a symbolic cover, composed of a group of symbolic implicants. For the user convenience the software has been designed so that the input data is loaded into a disk file. This method offers the advantages of permanent storage, i.e., the user will not have to enter the same table every time the program is run and he/she will be able to make further modifications.

SEQDES generates an output file that, by request of the user can either be, only the input symbolic cover and the resulting transition table in implicant form, or these and the intermediate results. The intermediate steps show the following: reduced symbolic cover, if any, positional cube representation, grouped states, constraint matrix and state assignment.

The maximal size of the input symbolic cover is limited only by the memory size available on the computer. The program presented here is able to input a table with a maximum of 99 boolean implicants and with an input and
output field formed by a top of 8 binary values each having up to 15 variables.

The entered table is allowed to have "dash" inputs that can be either "0" or "1". These are applied as "9"s in the output field. For incompletely specified tables in which there are unspecified next states or inputs, the reduction by implication chart is not performed. If only the output part has "don't cares" the state assignment part of the program works without problem. However, tables with unspecified next states are not permitted. In the output field of the symbolic cover outputs are also entered as "9"s.

Let us see some examples.

Example 5.1:

This example has been taken from the book "The Art of Digital Design" (6). Figure 5.1 shows an ASM chart from page 216 of the referred book. This represents an electronic combination lock.

The lock combination is entered in binary, one digit at a time, a wrong digit aborts the procedure and take us to an error state that can again be initialized by a reset signal.

For our purposes we are going to consider the following nomenclature for the input variables, the position after the arrow shows the initial name, the letter before it corresponds to the new name.
Asynchronous
RESET

Figure 5.1: Digital lock ASM chart
D $\rightarrow$ TRY
E $\rightarrow$ READ
F $\rightarrow$ BIT=REF(CNT)
G $\rightarrow$ CNT=M
H $\rightarrow$ RESET

The new names of the states are described in the same way:

START $\rightarrow$ S1
LOOP $\rightarrow$ S2
TEST $\rightarrow$ S3
INCR $\rightarrow$ S4
OK $\rightarrow$ S5
ERROR $\rightarrow$ S6

Table 5.1 presents the symbolic cover corresponding to the ASM chart of Figure 5.1.

The computer listings are in the next pages and they describe the following: symbolic cover in positional cube representation, table with grouped states, constraint matrix, state assignment and transition table.

Let us compute the cost and the number of gates required for realizing a given table as a measure of the efficiency of the state assignment obtained.

The following assumptions will be made:

1. D flip-flops will be used.
2. The minimal number of flip-flops will be used.
3. Output equations will not be considered.
Table 5.1: State table for the digital combination

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>00----</td>
<td>S2</td>
<td>S2</td>
</tr>
<tr>
<td>0111-</td>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>0110-</td>
<td>S2</td>
<td>S4</td>
</tr>
<tr>
<td>1----</td>
<td>S2</td>
<td>S6</td>
</tr>
<tr>
<td>-10--</td>
<td>S2</td>
<td>S6</td>
</tr>
<tr>
<td>00----</td>
<td>S3</td>
<td>S3</td>
</tr>
<tr>
<td>10----</td>
<td>S3</td>
<td>S5</td>
</tr>
<tr>
<td>-1----</td>
<td>S3</td>
<td>S6</td>
</tr>
<tr>
<td>-----</td>
<td>S4</td>
<td>S2</td>
</tr>
<tr>
<td>-----</td>
<td>S5</td>
<td>S5</td>
</tr>
<tr>
<td>-----</td>
<td>S6</td>
<td>S6</td>
</tr>
<tr>
<td>----1</td>
<td>S2</td>
<td>S1</td>
</tr>
<tr>
<td>----1</td>
<td>S3</td>
<td>S1</td>
</tr>
<tr>
<td>----1</td>
<td>S4</td>
<td>S1</td>
</tr>
<tr>
<td>----1</td>
<td>S5</td>
<td>S1</td>
</tr>
<tr>
<td>----1</td>
<td>S6</td>
<td>S1</td>
</tr>
</tbody>
</table>

4. The cost of realizing each excitation function is defined as the total number of gate inputs required for a two-level AND-OR gate network. All variables and their complements are assumed to be available as network inputs.

5. Multiple-output realizations with common gates will not be considered, so each cost of the D input equations
will be counted separately.

To compute the cost and number of gates that will be used, the steps to follow will be:
- Find the minterms of each variable in the transition table in implicant form.
- Use as a method of function simplification either Karnaugh maps or Quine-McKluskey according to the number of variables involved.
- Count the number of inputs and gates required for the network implementation.

The cost and number of gates necessary for the realization of the sequential network of this example will be computed before the output listings.

For this example the Quine-McKluskey method is going to be used in order to simplify the combinational part of the circuits. The initial minterms of the next state variables are shown in Table 5.2.

The resultant minimized three next state variables are formed by the minterms presented in Table 5.3.

Therefore, the cost of this realization is 53 inputs and 14 gates.

We can compare this result to the cost obtained making a straight binary assignment in which the codes are as shown in Table 5.4. The cost of the implementation of the network using this encoding is 80 inputs and 18 gates.

As it can be seen, the state assignment obtained
with our program uses 27 less inputs and 4 less gates and consequently our encoding has smaller cost than the straight binary one.

Table 5.2: Initial minterms of next state variables obtained with SEQDES program

<table>
<thead>
<tr>
<th>A+ Minterms</th>
<th>C+ Minterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E F G H</td>
<td>A B C D E F G H</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>0 0 1 0 1 1 1 -</td>
<td>1 0 1 - - - -</td>
</tr>
<tr>
<td>0 0 1 0 1 1 0 -</td>
<td>0 0 1 0 0 - - -</td>
</tr>
<tr>
<td>0 0 0 - - - - 1</td>
<td>1 0 0 - - - -</td>
</tr>
<tr>
<td>1 1 0 0 0 - - -</td>
<td>0 0 1 - - - - 1</td>
</tr>
<tr>
<td>0 0 1 - - - - 1</td>
<td>1 1 0 - - - - 1</td>
</tr>
<tr>
<td>1 1 0 - - - - 1</td>
<td>1 0 0 - - - - 1</td>
</tr>
<tr>
<td>1 0 0 - - - - 1</td>
<td>0 1 0 - - - - 1</td>
</tr>
<tr>
<td>0 1 0 - - - - 1</td>
<td>0 0 0 - - - - 1</td>
</tr>
</tbody>
</table>

B+ MINTERMS

<table>
<thead>
<tr>
<th>A B C D E F G H</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>0 0 1 0 1 1 1 -</td>
</tr>
<tr>
<td>1 1 0 0 0 - - -</td>
</tr>
<tr>
<td>1 1 0 1 0 - - -</td>
</tr>
<tr>
<td>0 1 0 - - - -</td>
</tr>
</tbody>
</table>

Table 5.5 illustrates parameters of some synchronous sequential machines encoded by SEQDES and compares them with those obtained using a straight binary assignment.

The output listings and the input state tables obtained by the computer are shown in pages 61 - 70.
Table 5.3: Simplified minterms of next state variables obtained with SEQDES program

<table>
<thead>
<tr>
<th>A+ Minterms</th>
<th>C+ Minterms</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E F G H</td>
<td>A B C D E F G H</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Table 5.4: Straight binary assignment

<table>
<thead>
<tr>
<th>State</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>000</td>
</tr>
<tr>
<td>S2</td>
<td>001</td>
</tr>
<tr>
<td>S3</td>
<td>010</td>
</tr>
<tr>
<td>S4</td>
<td>011</td>
</tr>
<tr>
<td>S5</td>
<td>100</td>
</tr>
<tr>
<td>S6</td>
<td>101</td>
</tr>
</tbody>
</table>

It is interesting to notice that machine number 1 has been also solved by the Micheli, et. al. method and presented in the mentioned paper. Their implementation after the excitation functions are simplified, gives a cost
of 12 gates and 31 inputs and SEQDES gives a cost of 10 gates and 24 inputs.

Table 5.5: Parameters of some synchronous sequential machines encoded by SEQDES and compared with straight binary assignment

<table>
<thead>
<tr>
<th>No.</th>
<th>Inputs</th>
<th>Outputs</th>
<th>States</th>
<th>Gates</th>
<th>Inputs</th>
<th>Gates</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>10</td>
<td>24</td>
<td>13</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>17</td>
<td>9</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>16</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>17</td>
<td>75</td>
<td>18</td>
<td>88</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-</td>
<td>6</td>
<td>14</td>
<td>53</td>
<td>18</td>
<td>80</td>
</tr>
</tbody>
</table>

In Table 5.5, machine number 4 corresponds to the one described by the SM chart in Figure 1.2, machine number 5 is the one that was considered in Example 5.1. The symbolic cover of the others are shown only in the computer listings.

As we can observe the program developed here also compares favorably with the encodings obtained with straight binary assignment.

Also it can be noticed that the program is more useful for larger circuits where it is difficult to find a good assignment by a trial and error process.
Machine 1 from Table 5.5

SYMBOLIC COVER
-------------

---INPUT PS NS ---OUTPUT
00000000 1 6 00000000
00000000 2 5 00000000
00000000 3 5 00000000
00000000 4 6 00000000
00000000 5 1 00000000
00000000 6 1 00000000
00000000 7 5 00000000
00000001 1 4 00000000
00000001 2 3 00000000
00000001 3 7 00000000
00000001 4 5 00000000
00000001 5 2 00000000
00000001 6 2 00000000
00000001 7 6 00000000

SYMBOLIC COVER IN POSITIONAL CUBE REPRESENTATION
-----------------------------------------------

---INPUT PRESENT STATE NEXT STATE ---OUTPUT
12345678 123456789012345 123456789012345 12345678
00000000 10000000000000 00000000000000 00000000
00000000 01000000000000 00000000000000 00000000
00000000 00100000000000 00000000000000 00000000
00000000 00010000000000 00000000000000 00000000
00000000 00001000000000 00000000000000 00000000
00000000 00000100000000 00000000000000 00000000
00000000 00000010000000 00000000000000 00000000
00000000 00000001000000 00000000000000 00000000
00000000 00000000100000 00000000000000 00000000
00000000 00000000010000 00000000000000 00000000
00000000 00000000001000 00000000000000 00000000
00000000 00000000000100 00000000000000 00000000
00000000 00000000000010 00000000000000 00000000
00000000 00000000000001 00000000000000 00000000
### FILE: MACHINE DATA

AI DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING

```
00000001 00000100000000 01000000000000 00000000
00000001 00000100000000 00000100000000 00000000
```

### TABLE WITH GROUPED STATES

<table>
<thead>
<tr>
<th>--- INPUT</th>
<th>PRESENT STATE</th>
<th>NEXT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345679</td>
<td>123456789012345</td>
<td>123456789012345</td>
</tr>
<tr>
<td>00000000 10010000000000 00001000000000</td>
<td>00000000 00000000000000 00000000000000</td>
<td></td>
</tr>
<tr>
<td>00000000 01100100000000 00000000000000</td>
<td>00000000 00000000000000 00000000000000</td>
<td></td>
</tr>
<tr>
<td>00000000 00001100000000 00000000000000</td>
<td>00000000 00000000000000 00000000000000</td>
<td></td>
</tr>
<tr>
<td>00000000 00000000000000 00000000000000</td>
<td>00000000 00000000000000 00000000000000</td>
<td></td>
</tr>
</tbody>
</table>

### CONSTRAINT MATRIX

<table>
<thead>
<tr>
<th>PRESENT STATE</th>
<th>123456789012345</th>
</tr>
</thead>
<tbody>
<tr>
<td>10010000000000</td>
<td>00001000000000</td>
</tr>
<tr>
<td>01100100000000</td>
<td>00000000000000</td>
</tr>
<tr>
<td>00001100000000</td>
<td>00000000000000</td>
</tr>
<tr>
<td>00010001000000</td>
<td>00000000000000</td>
</tr>
</tbody>
</table>

### STATE ASSIGNMENT

<table>
<thead>
<tr>
<th>STATE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>002</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
</tr>
<tr>
<td>6</td>
<td>012</td>
</tr>
<tr>
<td>5</td>
<td>115</td>
</tr>
<tr>
<td>3</td>
<td>091</td>
</tr>
<tr>
<td>2</td>
<td>191</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
</tr>
</tbody>
</table>

### TRANSITION TABLE

<table>
<thead>
<tr>
<th>--- INPUT</th>
<th>PS</th>
<th>NS</th>
<th>--OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>111</td>
<td>010</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>101</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>001</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>100</td>
<td>010</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>110</td>
<td>111</td>
<td>00000010</td>
</tr>
<tr>
<td>00000000</td>
<td>010</td>
<td>111</td>
<td>00000081</td>
</tr>
<tr>
<td>00000000</td>
<td>000</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>FILE: MACHINE1 DATA</td>
<td>41 DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 111 100 00000001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 101 031 00000010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 001 000 00000010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 100 010 00000010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 110 101 00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 010 101 00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001 000 010 00000000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FILE: MACHINE2 DATA
AI DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING
Machine 2 from Table 5.5

SYMBOLOCIC COVER

---INPUT PS NS ---OUTPUT

00000000 1 1 00000000
00000000 2 3 00000000
00000000 3 1 00000000
00000000 4 3 00000000
00000000 5 6 00000000
00000000 6 1 00000000
00000001 1 5 00000000
00000001 2 2 00000001
00000001 3 5 00000000
00000001 4 2 00000001
00000001 5 5 00000000
00000001 6 5 00000000

SYMBOLOCIC COVER IN POSITIONAL CUBE REPRESENTATION

---INPUT PRESENT STATE NEXT STATE ---OUTPUT

12345678 123456789012345 123456789012345 12345678

00000000 10000000 10000000 04 0000 0 00000000 04 0000 0
00000000 01000000 04 0000000 000000000000 00000000
00000000 01000000 04 000000000000 00000000 00000000
00000000 03010000 04 000000000000 00000000 00000000
00000000 03010000 04 000000000000 00000000 00000000
00000000 10000000 04 00000000 00000000 00000000
00000000 10000000 04 00000000 00000000 00000000
00000000 01000000 04 00000000 00000000 00000000
00000000 01000000 04 00000000 00000000 00000000
00000000 01000000 04 00000000 00000000 00000000
00000000 01000000 04 00000000 00000000 00000000
00000000 01000000 04 00000000 00000000 00000000
### TABLE WITH GROUPED STATES

<table>
<thead>
<tr>
<th>---INPUT</th>
<th>PRESENT STATE</th>
<th>NEXT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>123456789012345</td>
<td>123456789012345</td>
</tr>
<tr>
<td>00000000</td>
<td>101001000000000</td>
<td>100000000000000</td>
</tr>
<tr>
<td>00000000</td>
<td>010100000000000</td>
<td>010000000000000</td>
</tr>
<tr>
<td>00000000</td>
<td>000100000000000</td>
<td>000001000000000</td>
</tr>
<tr>
<td>00000001</td>
<td>100010000000000</td>
<td>000010000000000</td>
</tr>
<tr>
<td>00000001</td>
<td>010100000000000</td>
<td>010000000000000</td>
</tr>
<tr>
<td>00000001</td>
<td>010010000000000</td>
<td>000001000000000</td>
</tr>
</tbody>
</table>

### CONSTRAINT MATRIX

<table>
<thead>
<tr>
<th>PRESENT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789012345</td>
</tr>
<tr>
<td>101001000000000</td>
</tr>
<tr>
<td>010100000000000</td>
</tr>
<tr>
<td>100010000000000</td>
</tr>
<tr>
<td>001001000000000</td>
</tr>
</tbody>
</table>

### STATE ASSIGNMENT

<table>
<thead>
<tr>
<th>STATE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>001</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
</tbody>
</table>

### TRANSITION TABLE

<table>
<thead>
<tr>
<th>---INPUT</th>
<th>PS</th>
<th>NS</th>
<th>---OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>010</td>
<td>010</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>101</td>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>000</td>
<td>010</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>010</td>
<td>100</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>100</td>
<td>010</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>010</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>101</td>
<td>101</td>
<td>00000001</td>
</tr>
<tr>
<td>00000001</td>
<td>000</td>
<td>100</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>001</td>
<td>101</td>
<td>00000001</td>
</tr>
<tr>
<td>00000001</td>
<td>110</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>100</td>
<td>100</td>
<td>00000000</td>
</tr>
</tbody>
</table>
Machine 3 from Table 5.5

---INPUT PS NS --OUTPUT

```
    000000  1 2 000000
    000000  2 4 000000
    000000  3 5 000000
    000000  4 5 000000
    000000  5 6 000000
    000000  6 1 000000
    000000  7 1 000000
    000000  1 3 000000
    000000  2 5 000000
    000000  3 5 000000
    000000  4 6 000000
    000000  5 7 000000
    000000  6 1 000000
    000000  7 1 000000
```

---INPUT PRESENT STATE NEXT STATE --OUTPUT

```
12345678 123456789012345 123456789012345 12345678

    000000  11001001000000 11001001000000 000000
    000000  11001001000000 11001001000000 000000
    000000  01001000000000 01001000000000 000000
    000000  01001000000000 01001000000000 000000
    000000  00000000000000 00000000000000 000000
    000000  00000000000000 00000000000000 000000
    000000  11001001000000 11001001000000 000000
    000000  11001001000000 11001001000000 000000
    000000  10001000000000 10001000000000 000000
    000000  10001000000000 10001000000000 000000
    000000  00000000000000 00000000000000 000000
    000000  00000000000000 00000000000000 000000
    000000  11001001000000 11001001000000 000000
    000000  11001001000000 11001001000000 000000
    000000  10001000000000 10001000000000 000000
    000000  10001000000000 10001000000000 000000
```

SYMBOLIC COVER

---INPUT PS NS --OUTPUT

---INPUT PRESENT STATE NEXT STATE --OUTPUT
TABLE WITH GROUPED STATES

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>PRESENT STATE</td>
<td>NEXT STATE</td>
</tr>
<tr>
<td>12345678</td>
<td>123456789</td>
<td>123456789</td>
</tr>
<tr>
<td>00000000</td>
<td>01000000</td>
<td>01000000</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

CONSTRAINT MATRIX

PRESENT STATE

123456789012345

STATE ASSIGNMENT

<table>
<thead>
<tr>
<th>STATE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>000</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
</tr>
</tbody>
</table>

TRANSITION TABLE

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>PS</td>
<td>NS</td>
</tr>
<tr>
<td>00000000</td>
<td>011</td>
<td>111</td>
</tr>
<tr>
<td>00000000</td>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>00000000</td>
<td>001</td>
<td>011</td>
</tr>
<tr>
<td>00000000</td>
<td>110</td>
<td>101</td>
</tr>
<tr>
<td>00000000</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>00000000</td>
<td>100</td>
<td>111</td>
</tr>
<tr>
<td>00000000</td>
<td>020 011</td>
<td>00000011</td>
</tr>
<tr>
<td>00000001</td>
<td>011 001</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>101 010</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>001 010</td>
<td>00000011</td>
</tr>
<tr>
<td>00000001</td>
<td>110 100</td>
<td>00000001</td>
</tr>
<tr>
<td>00000001</td>
<td>010 000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000001</td>
<td>100 011</td>
<td>00000021</td>
</tr>
<tr>
<td>00000001</td>
<td>000 011</td>
<td>00000000</td>
</tr>
</tbody>
</table>
Machine 4 from Table 5.5

--- INPUT PS NS -- OUTPUT

09999900 1 1 00000000
19999900 1 2 00000000
19999900 2 2 03100000
29199900 2 3 00000000
39019903 2 3 00000000
79031903 2 4 00000000
79000900 2 5 00010000
90999900 3 3 10000000
91999900 3 1 10000000
90999900 4 4 01000000
91999903 4 1 01000000
99999900 5 5 00000000
19799900 5 6 00000000
19999900 5 6 00100000
99999100 6 6 00000000
99009900 6 5 00000000
39199900 6 4 00000000

SYMBOLIC COVER IN POSITIONAL CUBE REPRESENTATION

--- INPUT PRESENT STATE NEXT STATE -- OUTPUT

12345673 123456789012345 123456789012345 12345678

09999900 1000000000000000 1000000000000000 00000000
19999900 1000000000000000 0100000000000000 00000000
19999903 0100000000000000 0100000000000000 00100000
09199900 0100000000000000 0010000000000000 00000000
39019903 0100000000000000 0010000000000000 00000000
39000900 0100000000000000 0001000000000000 00910000
39029900 0311000000000000 0310000000000000 10000000
91999903 0311000000000000 1000000000000000 10000000
FILE: MACHINE DATA

<table>
<thead>
<tr>
<th>INPUT</th>
<th>PRESENT STATE</th>
<th>NEXT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>123456789012345</td>
<td>123456789012345</td>
</tr>
<tr>
<td>90999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>91999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>99999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>19999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>19999300</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09999100</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09099000</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09199000</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
</tbody>
</table>

TABLE WITH GROUPED STATES

<table>
<thead>
<tr>
<th>INPUT</th>
<th>PRESENT STATE</th>
<th>NEXT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>123456789012345</td>
<td>123456789012345</td>
</tr>
<tr>
<td>90999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>91999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>99999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>19999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>19999300</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09999100</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09099000</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09199000</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
</tbody>
</table>

CONSTRAINT MATRIX

<table>
<thead>
<tr>
<th>INPUT</th>
<th>PRESENT STATE</th>
<th>NEXT STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>123456789012345</td>
<td>123456789012345</td>
</tr>
<tr>
<td>90999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>91999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>99999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>19999900</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>19999300</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09999100</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09099000</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
<tr>
<td>09199000</td>
<td>000100000000000</td>
<td>000100000000000</td>
</tr>
</tbody>
</table>

STATE ASSIGNMENT

<table>
<thead>
<tr>
<th>STATE CODE</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>110</td>
<td>3</td>
</tr>
<tr>
<td>001</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
</tbody>
</table>

TRANSITION TABLE
<table>
<thead>
<tr>
<th>---INPUT</th>
<th>PS</th>
<th>NS</th>
<th>---OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>09999900</td>
<td>101</td>
<td>101</td>
<td>00000000</td>
</tr>
<tr>
<td>19999900</td>
<td>101</td>
<td>001</td>
<td>00000000</td>
</tr>
<tr>
<td>09199900</td>
<td>001</td>
<td>001</td>
<td>00100000</td>
</tr>
<tr>
<td>09019900</td>
<td>001</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>09001900</td>
<td>001</td>
<td>010</td>
<td>00000000</td>
</tr>
<tr>
<td>09000900</td>
<td>001</td>
<td>100</td>
<td>00010000</td>
</tr>
<tr>
<td>90999900</td>
<td>110</td>
<td>110</td>
<td>10000000</td>
</tr>
<tr>
<td>91999900</td>
<td>110</td>
<td>101</td>
<td>10000000</td>
</tr>
<tr>
<td>90999900</td>
<td>010</td>
<td>010</td>
<td>01000000</td>
</tr>
<tr>
<td>91999900</td>
<td>011</td>
<td>101</td>
<td>01000000</td>
</tr>
<tr>
<td>09999900</td>
<td>100</td>
<td>100</td>
<td>00000000</td>
</tr>
<tr>
<td>19999903</td>
<td>100</td>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>09999900</td>
<td>000</td>
<td>000</td>
<td>00100000</td>
</tr>
<tr>
<td>09999100</td>
<td>000</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>09097900</td>
<td>000</td>
<td>100</td>
<td>00000000</td>
</tr>
<tr>
<td>09197000</td>
<td>000</td>
<td>010</td>
<td>00000000</td>
</tr>
</tbody>
</table>
**FILE: MACHINE5 DATA**

**Dept. of Electrical and Computer Engineering**

Machine 5 from Table 5.5

---

**SYMBOLIC COVER**

--- Input PS NS -- Output ---

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0109999</td>
<td>1 2 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0000999</td>
<td>2 2 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0001119</td>
<td>2 3 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0001129</td>
<td>2 4 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0001999</td>
<td>2 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0091099</td>
<td>2 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0100999</td>
<td>3 3 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0010999</td>
<td>3 5 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0021999</td>
<td>3 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>4 2 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0049999</td>
<td>5 5 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0069999</td>
<td>6 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>7 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>3 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>4 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>5 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>6 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**SYMBOLIC COVER IN POSITIONAL CUBE REPRESENTATION**

--- Input Present State Next State -- Output ---

<table>
<thead>
<tr>
<th>Input</th>
<th>Present State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0109999</td>
<td>1 2 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0000999</td>
<td>2 2 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0001119</td>
<td>2 3 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0001129</td>
<td>2 4 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0001999</td>
<td>2 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0091099</td>
<td>2 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0100999</td>
<td>3 3 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0010999</td>
<td>3 5 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0021999</td>
<td>3 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>4 2 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0049999</td>
<td>5 5 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0069999</td>
<td>6 6 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>7 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>3 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>4 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>5 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
<tr>
<td>0029999</td>
<td>6 1 J0000000</td>
<td>1 2 J0000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>
FILE: MACHINES DATA  

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>PRESENT STATE</td>
<td>NEXT STATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
<td>----------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>12345678</td>
<td>123456789012345</td>
<td>123456789012345</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>00000000000000</td>
<td>00000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE WITH GROUPED STATES

CONVERSION MATRIX

PRESENT STATE

STATE ASSIGNMENT

<table>
<thead>
<tr>
<th>STATE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>000</td>
</tr>
<tr>
<td>5</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>001</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
</tr>
</tbody>
</table>

TRANSITION TABLE

--- INPUT | PS | NS | OUTPUT ---
FILE: MACHINES DATA  AL Dept. of Electrical and Computer Engineering

<table>
<thead>
<tr>
<th>Code</th>
<th>101</th>
<th>001</th>
<th>00000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>02099999</td>
<td>101</td>
<td>001</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>001</td>
<td>001</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>001</td>
<td>112</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>001</td>
<td>102</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>001</td>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>001</td>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>110</td>
<td>110</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>110</td>
<td>010</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>110</td>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>100</td>
<td>001</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>010</td>
<td>012</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>000</td>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>001</td>
<td>101</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>112</td>
<td>101</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>100</td>
<td>101</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>010</td>
<td>121</td>
<td>00000000</td>
</tr>
<tr>
<td>02099999</td>
<td>005</td>
<td>101</td>
<td>00000000</td>
</tr>
</tbody>
</table>
CHAPTER 6

CONCLUSIONS

In this work we presented a program that, when given a symbolic cover representation of a synchronous sequential machine (either Moore or Mealey), performs the following tasks:

1) Using the implication chart method, if the input symbolic cover is not incompletely specified, the program reduces it, eliminating equivalent states, if there are any.

The simplified table produces a realization which requires less amount of logic than the original one and may also need fewer flip-flops.

2) The program obtains a state assignment which leads to a good two-level AND-OR implementation of the network maintaining the number of memory elements to a minimum. This state assignment based in the guideline works well for D and J-K flip-flops. Then, the software replaces the encodings in the reduced symbolic cover and the final result is represented by a transition table, in implicant form, of the circuit.

In the future, it could be of interest to develop a logic multiple-valued input minimizer to be connected to this program in such way that the symbolic cover could be reduced with this method before the state assignment and after the translation to positional cube notation. The
obtained results could then be compared to those obtained in the present work. Another application of this kind of minimizer could be the manipulation and combination of the term adjacencies in the output, i.e., transition table in implicant form, to reduce the amount of logic needed for the network realization obtaining the simplified excitation functions to be implemented.

The average computation time is 3 or 4 seconds. A more precise timing could not be obtained in each example because the program is executed by a time shared system where the duration of the processes may vary according to the number of computer users at a certain moment.

The circuit implementation of the examples analyzed in this thesis gives a lower cost than the ones obtained with straight binary assignment and the one presented in the paper by De Micheli, et. al., therefore, we can say that the results obtained with program SEQDES compare favorably to other techniques.

In conclusion, the software developed in this thesis achieves satisfactorily the proposed goals related to simplification of the symbolic covers and optimum state assignment and it can be a useful tool in the process of designing synchronous sequential networks.
APPENDIX A

USER MANUAL

SEQDES is a program which accepts as input a symbolic cover. If this table is not incompletely specified and if equivalent states are present, then the program reduces the input data. Next, it produces an optimal state assignment and finally outputs a transition table in implicant form.

To run this program the following steps are required:

1. Create an input file of DATA type and load the state table with the editor. The commands to be used are:

   Xedit Program.name DATA (to access the editor)
   Input (to set the input mode)

   Then load the symbolic cover with this format:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>PRESENT STATE</th>
<th>NEXT STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 binary values</td>
<td>2 integer positions</td>
<td>2 integer positions</td>
<td>8 binary values</td>
</tr>
</tbody>
</table>

   End the load with a string of twenty "9"s.

Example A.1:

Enter a symbolic cover consisting of two implicants.

It would be done the following way (the answers of the computer are after => sign).

X TDATA DATA

=> XEDIT

I

=> Input

00000001 1 20000001
2. Prepare the environment to execute the program and define the location of your files with the following instructions:

GLOBAL TXTLIB VFORLIB
Fi 6 Output.File.name DATA DISK
Fi 5 Output.File.name DATA DISK
Fi 4 Terminal

This step always has to be done before running the program. For convenience it can be stored in a EXEC type file that can be run to execute a set of CMS commands. This file can be created using the Editor.

3. Execute the program using the command:

LOAD SEQDES (START

The computer response will be:

=> EXECUTION BEGINS

Then, a screen of information pertaining this work is displayed (Master Thesis, Author's name, etc.)

DO YOU WANT PARTIAL RESULTS?

PLEASE ENTER YES OR NO
Using this option the user has the opportunity to choose whether he/she wants the intermediate results together with the input and transition tables or just the input and transition tables.

**DO YOU WANT TO PRE-ASSIGN THE CODE?**

**PLEASE ENTER YES OR NO**

If the user enters yes, he or she is to follow the corresponding instructions provided by the program in order to load the codes, otherwise the following message is displayed on the screen:

**THANK YOU, IT HAS BEEN A PLEASURE WORKING WITH YOU.**

=>R;

4. To print the results use:

`ROUTE Printer.Destination A`

For instance:

If in Alden Library use ALDLIB for Printer.Destination; if in Stocker building use STOCKER for Printer.Destination.

Then write:

`PRINT Output.File.name DATA`

To type on the screen the output file, use:

`TYPE Output.File.name DATA`
LEVEL 1.3.1 (FEB 1984)

VS FORTRAN

DATE: MAY 10, 1987

REQUESTED OPTIONS (EXECUTE): VJSDEBUG

OPTIONS IN EFFECT:  MULIST  NOMAP NOXREF NOXSUBST NODECK SOURCE TERM OBJECT FIXED

OPT(0) LANGLEY(77) NODIPS FLAG(I) NAME(MAIN ) LINECOUNT(60)

*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...*...

L C ** PROGRAM: MAIN PROGRAM FOR THE SIMPLIFICATION OF **
L C ** SYMBOLLIC COVERS AND FOR STATE ASSIGNMENT OF **
L C ** SEQUENTIAL MACHINES **
L C ** AUTHOR: MIRIAM HERMANZEL **
L C ** DATE: 2/10/87 **

L DIMENSION ISTA2(120,18),IPCU1(120,40),IPCU2(100,40)
L INTEGER S(100,15),JP(2,100),ISTA1(100,24)
L CHARACTER*3 CHAR
L COMMON /BL1/IPCU1/BL2/S
L DATA IEND/9/

L FILE 4 IS TERMINAL, FILE 5 IS INPUT,
C FILE 6 IS OUTPUT
C
L Li=4,
L LR=5,
L LW=6
L I4=0
L WRITE(LI+450)
L WRITE(LM+50)

L 450 FJFORMAT(/'2X*57(*')+/2X,***,53(*')**,+/2X,**,9)
L 5X,**,MASTER'S THESIS**,9
L 35(*')**,+/2X,**,5X,**,IPT,JF ELECETRICAL AND COMPUTER **
L 5**ENGINEERING**(*')**,+/2X,**,14X,**,OHIO UNIVERSITY**,+/9,
L 5**,+/2X,**,5X,**,**/*
L 2X,**,**,5X,**,THESS DIRECTION: JR. HAROLD KLOCK*,16(*'),**/*
L 52X,**,**,5X,**,STUDENT: MIRIAM HERMANZEL**,21(*'),**/*.+/9
L 52X,**,**,5X,**,---- SPRING 1987 --------14Y(''),**/*
L 52X,**,**,55(*')**,+/2X*57('**')//

L 13 WRITE(LT+195)
L 14 195 FJFORMAT(/'2X,*DO YOU WANT PARTIAL RESULTS ? */,
L 5 2X,PLEASE ENTER YES OR NO */) /
L 15 READ(LT+195)CHAR
L 16 195 FJFORMAT(A3)
L 17 CALL YESNO(CHAP*IC)
L 18 READ(LR+200)JSTAT
L 19 200 FJFORMAT(L2)
L 20 DO 10 I=1,120
L 21 READ(L9+140)(ISTA2(I,J),J=1,18)
L 22 140 FJFORMAT(B11,L2(12)*911)
C C CHECK IF EVO OF LMAT
L 23 IF(ISTA2(I,1) .NE.1E+0)GO TO 60
L 24 IF (ISTA2(I,2) .EQ.1E+0)GO TO 70
L 25 60 IY=IY+1
L 26 10 CONTINUE
L 27 70 CONTINUE
L 28 WRITE(LW,160)
LEVEL 1.3.1 (FEB 1984) VS FJTRAN  DATE: MAY 10, 1987

29 160  FORMAT(/82X,'SYMBOLIC COVER*+/,*2X,14(*--*)+/2X,*--INPUT*+/2X,*--OUTPUT*+/)
30  WRITE(L4*100)((ISTA2(I+J),J=1,18),I=1,IN)
31  100  FORMAT(2X,*9Il*2X,*I2*2X,*12*2X,*811)
32  C  C  FIND ALL THE EQUIVALENT STATES IN THE STATE
33  C  TABLE, THEN REDUCE TABLE ACCORDINGLY
34  C
35   NSTL=4STAT
36   CALL INCHAR(ISTA2,NSTAT,IV)
37   INTRAN=IV
38   IF(NSTAT.EQ.NSTL)GO TO 302
39   WRITE(L4*170)
40  170  FORMAT(/82X,'REDUCED SYMBOLIC COVER*+/,*2X,22(*--*)+/2X,*--INPUT*+/2X,*--OUTPUT*+/)
41   WRITE(L4*100)((ISTA2(I+J),J=1,18),I=1,IN)
42  C  C  SYMBOLIC COVER TRANSLATION INTO A MULTIPLC-VALUED
43  C  POSITIONAL CUBE REPRESENTATION
44  CALL TRANS(ISTA2,IPCU1,IN)
45   IF(1C.EQ.0)GO TO 300
46   WRITE(L4*111)
47  111  FORMAT(/82X,'SYMBOLIC COVER IN POSITIONAL CUBE REPRESENTATION*+/2X,*468(*--*)+/)
48   WRITE(L4*110)
49  110  FORMAT(/82X,'INPUT*+/2X,*PRESENT STATE*+/4(I*,*),'*NEXT STATE*+/2X,*OUTPUT*+/2X,*12345678*+/2X,*12345678+/)
50   WRITE(L4*130)((IPCU1(I+J),J=1,14),I=1,14)
51  C  C  MINIMIZE POSITIONAL CUBE REPRESENTATION AND OBTAIN
52  C  CONSTRAINT MATRIX
53  CALL MINIM(IPCU1,IV,INC)
54   IF(1C.EQ.0)GO TO 301
55   WRITE(L4*115)
56  115  FORMAT(/82X,'TABLE WITH GROUPED STATES*+/2X,25(*--*)+/)
57   WRITE(L4*129)
58  129  FORMAT(/82X,'INPUT*+/2X,*PRESENT STATE*+/4(*),'*NEXT STATE*+/2X,*12345678*+/2X,*12345678+/)
59   WRITE(L4*139)((IPCU1(I+J),J=1,39),I=1,39)
60  139  FORMAT(2X,*9Il*2X,*I2*2X,*12*2X,*811)
61   WRITE(L4*116)
62  116  FORMAT(/82X,'CONRAIN'T MATRIX*+/2X/*17(*--*)+/)
63   WRITE(L4*119)
64  119  FORMAT(2X,*PRESENT STATE*+/2X,*123456781345+/)
65   WRITE(L4*131)((IPCU2(I+J),J=9,23),I=1,INC)
66  131  FORMAT(2X,*1511)
67  C  C  OBTAIN THE STATE ASSIGNMENT MATRIX S
68  C
69   NR=INC
LEVEL 1.3.1 (FEB 1984)  

VS FORTRAN  
DATE: MAY 10, 1987

62 301 WRITE(LT,400)  
63 400 FORMAT(///,2X,'DO YOU WANT TO PRE-ASSIGN THE CODE?/')  
5 2X* OF ANY STATE(5) ? */2X*  
6 'PLEASE ENTER YES OR NO*/)  
64 READ(LT,195)CHAR  
65 CALL YESNO(CHAR,IPREC)  
66 IF(IPREC.EQ.1)GO TO 401  
67 L=1  
68 NS=1  
69 NPRES=3  
70 GO TO 331  
71 401 WRITE(LT,402)  
72 402 FORMAT(///,2X,'HOW MAY STATE(5) DO YOU WANT TO*/)  
5 2X*PRE-ASSIGN THE CODE ? */2X*  
6 'PLEASE ENTER THE NUMBER WITH FORMAT *//*  
6 2X*12 AND UNDER XX*/*/XX*)  
73 READ(LT,453)NPRES  
74 403 FORMAT(12)  
75 IF(12.GT.15)CALL ERROR(3)  
76 WRITE(LT,404)  
78 404 FORMAT(///,2X,'PLEASE ENTER THE NUMBER OF THE STATE*/)  
5 2X* WITH FORMAT 12 UNDER # AND THE */2X*  
6 'RESPECTIVE CODE UNDER THE X'S FROM */2X*  
5 'LEFT TO RIGHT, FILL THE REMAINING SPACES*/2X*  
5 '*'WITH U'S, ENTER ONE STATE AND ITS ASSIGNMENT*/2X*,  
5 '*'LEFT TO RIGHT, ONE EACH TIME*/2X*#*/2X*  
6 15('X*'))  
79 DJ 405 I=1,NPRES  
80 READ(LT,425)JP(I,J)+((S(I,J)+J=1,15))  
81 405 FORMAT(12+1X,15:I)  
82 490 CONTINUE  
83 407 IF(STAT.GE.NPRES)GO TO 408  
84 LL=LL+1  
85 408 IF(IPRES.EQ.0)GO TO 409  
86 LL=LL+1  
87 409 IF(NSTAT.EQ.1)GO TO 31C  
88 L=LL+1  
89 410 IF(NSTAT.LT.0)CALL ERR(222)  
90 220 FORMAT(///,2X,'STATE ASSIGNMENT */2X*10(('*-)'),/*  
5 3X* STATE CODE*)  
91 DJ 290 J=1,NSTAT  
92 WRITE(LT,210)JP(I,J)+((S(I,J)+J=1,L))  
93 210 FORMAT(1X,1Z,12+7X,31I1)  
94 290 CONTINUE  
95 C FILL THE IMPLICITNS OF THE TRANSITION TABLE  
C BEGIN WITH THE INPUT AND OUTPUT FIELDS  
C  
96 310 DJ 410 I=1,NTRAN  
97 DJ 411 J=1,8  
98 411 ISTAT(I,J)=ISTAT(I,J)  
99 412 JJ=11,18  
100 413 JJ=12+1,J+8  
101 DJ 414 I=1,18  
102 DJ 415 J=1,8  
103
LEVEL 1.3.1 (FEB 1984) VS FORTRAN  DATE: MAY 10, 1987

**...**

103 412  ISTA1(I,J1)=ISTA2(I,J1)
104 413  CONTINUE
105  L
106  CONTINUE
107  IF(JP(2,1).LE.ISTA2(I1,9))GO TO 414
108  DJ 415  J=1+L
109  JJ=8+J
110  IS1A1(I1, JJ)=S(I,J)
111  IF(JP(2,1).LE.ISTA2(I1,10))GO TO 413
112  DJ 416  J=1+L
113  JJ=12+J
114  IS1A1(I1, JJ)=S(I,J)
115 413  CONTINUE
116  WRITE(Ld+,417)
117  417  FORMAT(/,2X, 'TRANSITION TABLE'//,2X, ' /* ')//
118  WRITE(Ld+,418)
119  418  FORMAT(2X, '/* INPUT', 2X, '/* OUTPUT', 2X, '/*')//
120  GJ 10(425,426,427,428)
121  425  DJ 429  J=1, INTRAN
122  429  WRITE(Ld+,419) (ISTA1(I,J), J=1,8) + (ISTA1(I,J1), J1=9,8+L) +
5  (ISTA1(I,J2), J2=13,12+L) + (ISTA1(I,J3), J3=17,24) +
123  GJ 10430
124  430  DJ 431  J=1, INTRAN
125  431  WRITE(Ld+,420) (ISTA1(I,J), J=1,8) + (ISTA1(I,J1), J1=9,9+L) +
5  (ISTA1(I,J2), J2=13,12+L) + (ISTA1(I,J3), J3=17,24) +
126  GJ 10430
127  430  DJ 432  J=1, INTRAN
128  432  WRITE(Ld+,421) (ISTA1(I,J), J=1,8) + (ISTA1(I,J1), J1=9,9+L) +
5  (ISTA1(I,J2), J2=13,12+L) + (ISTA1(I,J3), J3=17,24) +
129  GJ 10430
130  430  WRITE(Ld+,422) (ISTA1(I,J), J=1,24) + (ISTA1(I,J1), J1=9,9+L) +
131  429  419  FJRMAT(2X,911,4X,11,4X,11,4X,811)
132  425  420  FJRMAT(2X,811,3X,211,4X,211,3X,311)
133  421  424  FJRMAT(2X,311,3X,311,2X,311,3X,011)
134  422  424  FJRMAT(2X,911,2X,411,2X,411,2X,411)
135  430  430  WRITE(Ld+,23)
136  430  423  FJRMAT(/,2X, 'THANK YOU, IT HAS BEEN A PLEASUE */ ,2X, ')
5  'WORKING WITH YOU, */ //)
137  STOP
138  END

**STATISTICAL SOURCE STATEMENTS = 137, PROGRAM SIZE = 43096 BYTES, PROGRAM NAME = MAIN
**STATISTIC** NO DIAGNOSTICS GENERATED.
**MAINEND OF COMILATION 1**
SUBROUTINE YESNO(CHAR, IC)
CHARACTER CHAR1, CHAR2, CHAR3
DATA CHAR1/'YES'/, CHAR2/'NO'/, CHAR3/'YES'/
IC=0
IF(CHAR.EQ.CHAR1) IC=1
RETURN
END

*STATISTICS*  SOURCE STATEMENTS = 7, PROGRAM SIZE = 350 BYTES, PROGRAM NAME = YESNO
*STATISTICS*  NO DIAGNOSTICS GENERATED.
*YESNO* END OF COMPILATION 1  * * * *
** SUBROUTINE: FIND ALL THE EQUIVALENT STATES **
* IN A SYMBOLIC COVER, REDUCE IT *
* BY IMPLICATION CHART METHOD. *
**
** AUTHOR: MIRIAM HERNANDEZ **
**
** DATE: 3/19/87 **

PARAMETER DESCRIPTION:
- ISTA2 : (I/O) INPUT STATE TABLE AND
  REDUCED STATE TABLE
- IN : (I) NUMBER OF ENTRIES OF STATE
  TABLE
- VSTAT : (I/O) NUMBER OF STATES

C
C FORM INITIAL IMPLICANT CHART
C
I2=2
NS=5*9(VSTAT-1)
NS1=2*VSTAT

C
C INITIALIZE VARIABLES
C
DJ 30J J=1,28
DJ 30J J=1,10
30J
DJ 30J I=1,29
DJ 30 J I=1,2
DJ 30 J JJ=1,2
13 30J IeD(I=I1, JJ=J)
14 DJ J I=1,NS
15 DJ 30 J J=1,18
16 16 L(J)=ISTA2(I,J)
17 DJ 20 II=12, VS1
18 DJ 15 J=1,18
19 15 L(J)=ISTA2(I,J)

C
C COMPARE OUTPUT FIELDS TO OBTAIN IMPLIED PAIRS
C
CA.L COMPF(L,L1=I1, I1=1,18,18,18,DJ)
IC(1,11)=IC1
IF IC(1,11) NE 1 GO TO 20

C
C STORE IMPLIED PAIRS
C
IEQ(1,11)=ISTA2(1,10)
IEQ(1,11,2)=ISTA2(11,10)

CONTINUE
26  I2=I2+1
27  10  CJNTINUE
   C
   MAKE THE NECESSARY PASSES
   C
28  50  I2=2
   C
   INITIALIZE NUMBER OF STATES ADDED
   C
29  I3=2
30  DJ 40  I=1,(NSTAT-1)
31  DJ 30  I=12,NSTAT
   C
   CHECK IF THE STATE PAIR IS NOT EQUIVALENT
   C
32  IF(IC(I,i1),EQ,1)GO TO 30
   C
   CHECK IF THE IMPLIED PAIR OF STATES ARE EQUIVALENT
   C
33  I3=IEd(I,i1,i1)
34  I4=IEd(I,i1,i2)
35  IF(IC(I3,i4),EQ,1)GO TO 31
36  IF(IC(I3,i4),NE,9)GO TO 32
37  HOLD=13
38  I3=I4
39  I4=HOLD
40  IF(IC(I3,i4),EQ,1)GO TO 31
41  SJ TO 32
42  31  IL=1*VSTAT
43  I=1*VSTAT
44  I3=IEd(IL,iK+1)
45  I4=IEd(IL,iK+2)
46  IF(IC(I3,i4),EQ,1)GO TO 30
47  IF(IC(I3,i4),NE,9)GO TO 32
48  HOLD=13
49  I3=I4
50  I4=HOLD
51  IF(IC(I3,i4),EQ,1)GO TO 33
   C
   ADD ONE X
   C
52  32  IC(I,i1)=J
53  I3=I5+1
54  30  CJNTINUE
55  I2=I2+1
56  40  CJNTINUE
57  IF(I5*NE,1)GO TO 30
   C
   NO MORE PASSES ARE NECESSARY IF IS IS 0
   C
58  I2=2
59  VST=VSTAT
60  DJ 40  I=1*VSTAT-1
61  DJ 70  I=12*VSTAT
62  IF(IC(I,i1),EQ,1)GO TO 70
63  VST=VSTAT-1
LEVEL 1.3.1 (FEB 1984)  VS FORTRAN  DATE: MAY 10, 1987

```
64 I3=IEQ(I+11,1)
65 I4=IEQ(I+1+12)
   C
   C  THE STATES ARE EQUIVALENT, REPLACE ONE
   C  FOR THE OTHER
   C
66 DJ II II=1+IN
67 IF(ISTA2(II,9)<=I.4) GO TO 75
68 ISTA2(II,9)=I3
69 75 IF(ISTA2(II,10)<=I.4) GO TO 71
70 ISTA2(II,10)=I3
71 71 CONTINUE
72 70 CONTINUE
73 I2=I2+1
74 60 CONTINUE
   C
   C  ELIMINATE REDUNDANT ENTRIES IN THE STATE TABLE
   C
75 M=M+NST
76 DJ 100 J=1+18
77 100 ISTA(1,J)=ISTA2(1,J)
78 N=1
79 DJ 103 I=2+1N
80 DJ 113 J=1+18
81 113 (J)=ISTA2(I,J)
82 DJ 104 II=1+N
83 DJ 112 J=1+18
84 112 LI(J)=ISTA(11+J)
85 CALL COMP(L+L+1C2,1+B+1D+1)
86 IF(LC2+2+D) GO TO 1N4
87 IF(L=ISTA2(1+V) I=EISTA(11,9)) GO TO 73
88 104 CONTINUE
89 N=4+1
90 DJ 107 J=1+18
91 107 ISTA(V,J)=ISTA2(V,J)
92 103 CONTINUE
93 DJ 100 I=1+N
94 DJ 106 J=1+18
95 108 ISTA2(I,J)=ISTA(1,J)
96 IF(I4+E+N) RETURN
   C
   C  ORDER THE REDUCED STATE TABLE
   C
98 DJ 101 I=1+N
99 NIJ=ISTA2(I,9)
100 IF(I+GT.+M) GO TO 109
101 N=1
102 DJ TO 110
103 109 NI=I-V4
104 110 ISTA2(I,9)=NI
105 DJ 102 II=1+N
106 IF(ISTA2(II,9)<=I.4) GO TO 111
107 ISTA2(II,9)=ISTA2(I,9)
108 111 IF(ISTA2(I,10)<=I.4) GO TO 102
109 ISTA2(I,10)=ISTA2(I,9)
110 102 CONTINUE
```
LEVEL 1.3.1 (FEB 1984)     VS FORTRAN     DATE: MAY 10, 1987

*..\*..*1..*..*2..*..*3..*..*4..*..*5..*..*6..*..*

111 101 CONTINUE
112  INV
113  NSTAT=44
114  RETURN
115  END

*STATISTICS: SOURCE STATEMENTS = 114, PROGRAM SIZE = 21310 BYTES, PROGRAM NAME = IMCHAR
*STATISTICS: NO DIAGNOSTICS GENERATED.
**IMCHAR** END OF COMPI LATION 1.****
SUBROUTINE MINIMIZE POSITIONAL CUBE REPRESENTATION

DATE: 2/12/87

PARAMETER DESCRIPTION:
- IPCU1 : [1] INPUT ARRAY
- IPCU2 : [0] MINIMIZED ARRAY
- 4 : [1]0 IPCU1 ROW NUMBER
- NC : [0] IPCU2 ROW NUMBER

INITIALIZE VARIABLES
- NN: TOTAL NUMBER OF MERGED LINES
- NJ: MINIMIZED MATRIX ROW NUMBER

DO 10 I=1,15
10 JP1(I)=0
JP2(I)=0
LIN=1

DO 70 J=1,A
70 L(J)=IPCU1(I,J)
IF(LL(15,LJ)*LJ=1
DO 40 L=1,15
40 IF(1.E7,JP2(J))GO TO 65
GO TO 67
65 I=I+1
67 GOTO 10

DU 20 I=1,N
LI(J)=IPCU1(I,J)
JUMPARE INPUT FIELDS
CALL COMPFLLJ,LI,IC1,1,8,46,1)  
IF(1C1.EQ.0)GO TO 20  
COMPARE NEXT STATE FIELDS  
CALL COMPFLLJ,LI,IC2,24,18,46,1)  
IF(1C2.EQ.0)GO TO 20  
CALL ONEMP(L1,N1,JP1,ERR)  
IS=15+1  
JP2(IS)=12  
IF(1ERR.EQ.1)CALL ERR01(1)  
CONTINUE  
II=II+1  
IF(IN=LT.1)GO TO 66  
MERGE PRESENT STATES OF ENTRIES WITH SAME INPUT, AND  
NEXT STATE  
CALL MERGE(L1,L2,N1,JP1)  
NJ=NJ-N1  
INITIALIZE ARRAY THAT STORES POSITION OF ONE'S (JP1)  
DO 30 K=I,N1  
JP1(K)=0  
MOVE ONE-DIMENSIONAL ARRAYS INTO TWO-DIMENSIONAL ONES  
DO 52 K=1,13  
DO 53 J=1,46  
L(I,J)=IFCU2(IK,J)  
CALL COMPFLLJ,L2,IC4,9,23,45,0)  
IF(1IC4.EQ.0)GO TO 65  
CONTINUE  
INCREMENT ROW NUMBER OF CONSTRAINT MATRIX  
N3=1J+1  
DU 50 J=1,46  
IFCU2(I3,J)=L2(IJ)  
N4=N4+1  
DO 51 J=1,46  
IFCU(N4,J)=L1(IJ)  
CONTINUE  
N4=1J  
RETURN  
END

#STATISTICS#  
SOURCE STATEMENTS = 59, PROGRAM SIZE = 2468 BYTES, PROGRAM NAME = MINIM  
#STATISTICS#  
NO DIAGNOSTICS GENERATED.  
#MINIM# END OF COMPIlATION
**SUBROUTINE**: DETECT POSITION OF ONE'S IN A ARRAY

**AUTHOR**: HIRAM HERNANDEZ A.

**DATE**: 2/12/87

**PARAMETER DESCRIPTION:**
- \( L \): [1] INPUT ARRAY
- \( N \): (0) NUMBER OF ONE'S
- \( JP \): [0] ARRAY WITH THE POSITION OF ONE'S
- ERR: [0] ERROR CODE
  - ERR = 1 ERROR NO ONE'S DETECTED
  - ERR = 0 NO ERROR

```c
SUBROUTINE ONE5(L,N,JP,ERR)
DIMENSION L(46),JP(15)
ERR=1
DO 10 I=9,23
IF(L(I).NE.1)GO TO 10
N=N+1
10 JP(N)=I
CONTINUE
IF(N.EQ.0)RETURN
END
```

**STATISTICS**: SOURCE STATEMENTS = 12, PROGRAM SIZE = 440 BYTES, PROGRAM NAME = HEND

**LANGUAGE**: C

**FILE**: 92

**PROGRAM**: END OF COMPILATION
SUBROUTINE COMPARE ARAYS AND DETERMINE IF THEY ARE IDENTICAL
AUTHOR: MIRIAM HERNANDEZ A.
DATE: 2/12/87

PARAMETER DESCRIPTION:
- L : (I) ARRAY 1 TO BE COMPARED
- L1 : (I) ARRAY 2 TO BE COMPARED
- IC : (O) CODE
  IC = 1 ARRAYS ARE IDENTICAL
  IC = 0 OTHERWISE
- ID : (I) ARRAY DIMENSION
- IN : (I) CODE
  IN = 1 INPUT FIELD IS BEING COMPARED
  IN = 0 OTHERWISE
  NOTE: ? INSIDE INPUT FIELD CORRESPONDS TO AN INPUT THAT NEVER OCCURS
  * INSIDE OUTPUT FIELD CORRESPONDS TO A DON'T CARE

1 DIMENSION L(ID),L1(ID)
2 IC=0
3 DO 10 I=IN,NF
4 IF(L(I),EQ,9,AND,IN,EN,9)GO TO 10
5 IF(L(I),NE,L1(I))RETURN
6 10 CONTINUE
7 IC=1
8 RETURN
9 END

STATISTICS: SOURCE STATEMENTS = 10, PROGRAM SIZE = 542 BYTES, PROGRAM NAME = COMP
LEVEL 1.3.1 (FEB 1984) FORTRAN DATE: APR 27, 1987

REQUESTED OPTIONS (EXECUTE): NOSDUMP

OPTIONS IN EFFECT: NOLIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED OPT(0) LANGLEVEL(77) NOFIPS FLAG(1) NAME(MAIN) LINECOUNT(60)

*....9..................2..................3..................4..................5..................6............

C
C  ** SURROUTINE : MERGE THE PRESENT STATE FIELD ONE'S AND OR
C  ** PUT THEM IN THE CONSTRAINT FIELD.  **
C  ** AUTHOR     : MIRIAM HERNANDEZ A.  **
C  ** DATE       : 2/12/87  **
C
C  PARAMETER DESCRIPTION:
C  - L   : (1|J)  ARRAY 1 TO BE MERGED AND MERGED ARRAY
C  - NS  : (1)  ONE'S NUMBER IN ARRAY 2
C  - JPI : (1)  ONE'S POSITION ARRAY 2
C  - L2  : CONSTRAINT FIELD

C
1  SUBROUTINE MERGE(L,L2,NS,JPI)
2  DIMENSION JPI(NS),L(46),L2(46)
3  DO 20 I=1,46
4   20  L2(I)=L(I)
5  DO 10 I=1,15
6    J=JPI[I]
7    L(J)=I
8    L2(J)=1
9  10  CONTINUE
10  RETURN
11  END

*STATISTICS: SOURCE STATEMENTS = 11, PROGRAM SIZE = 560 BYTES, PROGRAM NAME = MERGE
*STATISTICS: NO DIAGNOSTICS GENERATED.

**MERGE** END OF COMPILATION
SUBROUTINE ERROR(I)
LW=5
GO TO (10,20,30)IC
10 WRITE(LW,100)
100 FORMAT(//'** ERROR 1: NO ONES IN FIELD **'/)
RETURN
20 WRITE(LW,120)
120 FORMAT(//'** ERROR 2: LESS THAN ONE STATE TO ','
       ' BE CODED**'/)
30 WRITE(LW,130)
130 FORMAT(//'** ERROR 3: NUMBER OF PRE-ASSIGNED STATES**'/
       ' EXCEEDS MAXIMUM NUMBER OF STATES**'/)
RETURN
END

*STATISTICS*  SOURCE STATEMENTS = 12, PROGRAM SIZE = 542 BYTES, PROGRAM NAME = ERROR
*STATISTICS*  NO DIAGNOSTICS GENERATED.
*ERROR* END OF COMPILE 1
REQUESTED OPTIONS (EXECUTE): VJSDUMP
OPTIONS IN EFFECT: NLIST VMAP NOREPL NUGGSTMT NODECK SOURCE TERM OBJECT FIXED OPT(0) LANGYUL(77) YOFIPS FLAG(I) NAME(MAIN ) LINECOUNT(60)

** SUBROUTINE : TRANSLATE THE SYMBOLIC COVER TO **
** POSITIONAL CUBE REPRESENTATION **
* AUTHOR : MIRIAM MÉRANDEZ A. *
** DATE : 2/13/87 **

PARAMETER DESCRIPTION:
- ISTA2 : (I) INITIAL STATE TABLE
- IPCUI : (O) STATE TABLE IN POSITIONAL CUBE REPRESENTATION
- IN : (I) STATE TABLE NUMBER OF ENTRIES

SUBROUTINE TRANS(ISTA2,IPCUI,I4)
DIMENSION ISTA2(100,18),IPCUI(100,40)

INITIALIZE POSITIONAL CUBE ARRAY

START TRANSLATION

*STATISTICS* SOURCE STATEMENTS = 18, PROGRAM SIZE = 8370 BYTES, PROGRAM NAME = TRANS
*STATISTICS* NO DIAGNOSTICS GENERATED.
*TRANS* END OF COMILATION 1  *******
SUBROUTINE FIND AN ASSIGNMENT OF MINIMUM CODE LENGTH AMONG THE ASSIGNMENTS THAT MINIMIZE THE LOGIC IMPLICATION OF THE NETWORK.

PARAMETER DESCRIPTION:
- IPCU2 : (I) CONSTRAINT MATRIX
- S : (O) CODED STATE MATRIX
- NR : (I) CONSTRAINT MATRIX NUMBER OF ROWS
- NSTAT : (I) NUMBER OF STATES TO BE CODED
- L : (I) CODE LENGTH
- NPRES : (I) NUMBER OF PREASSIGNED STATES
- VS : (I) NUMBER OF CODED STATES

INTEGER SI 100,151,CI 100,IS1100,15',AI 100,15',PI 100,15
DIMENSION IPCU2(100,46),JP(2,15)
INTEGER CI(100,15),59(100,15)
COMMON /311/IPCU2/3L2/5
COMMON /BL9/AP/3L4/A
COMMON /BL3/51
COMMON /3L6/C
I N R S = 0
MOSAI = 0
NSTAT = 0
IF(NINSTAT.GT.1) GO TO 100
CALL ERRMIN(2)
GO TO 99
SECIBE
IF(NPRES.EQ.0) I5(1,1) = 0
IF(NPRES.NE.0) I5(1,1) = 0
IF(NSTAT.EQ.1) GO TO 99
FORM A MATRIX WITH PRESENT STATE FIELD OF CONSTRAINT MATRIX AND ORDER THE COLUMNS ACCORDING TO COUNT
CALL ORDER(JP,NP,JSTAT,NPRES)
IC = 1
GO TO J=NS+1,JSTAT
K = J
KK = K-1
GENERATE BOOLEAN COMBINATIONS FOR L DIMENSION
IF(IC.EQ.1) CALL GCAND(L,NS1)
IC = 0
C COMPARE ENCODED SET WITH THE BOOLEAN SPACE AND
C RETURN STATES THAT HAVE NOT BEEN USED YET
C
29 CALL CAND(NS,N,S1,L)
30 L=1
C
C CHOOSE A CANDIDATE THAT SATISFY THE CONSTRAINT
C RELATION. IF THE STATE APPEAR IN THE
C CONSTRAINT MATRIX, CHOOSE A CANDIDATE RELATED
C WITH THE LESS COVERED FACE. IF THE STATE IS NOT
C IN THE CONSTRAINT MATRIX CHOOSE ANY CODE NOT
C COVERED BY ANY FACE
C
31 CALL CSELECT(K,N,NR,L,NOSAT)
32 IF(L1*NE.L1)IC=1
34 IF(L1+EQ.L)NS=NS+1
36 IF(L1*EQ.L10 TO 10
C
C ADJOIN A COLUMN TO S MATRIX BECAUSE CODE LENGTH
C HAS BEEN INCREMENTED
C
37 CALL ADJOIN(K,KK,L,NSTAT,NPRES)
38 GO TO 20
39 CONTINUE
40 IF(NOSTAT.EQ.0)OR.NOJAT.LF.NOJALS)GO TO 99
41 IF(NTRY.EQ.0)GO TO 31
42 DO J=1,NSTAT
43 DO J=1,L
44 32 S(I,J)=S(I,J)
45 GO TO 99
46 DO J=1,NSTAT
47 DO J=1,L
48 30 S(I,J)=S(I,J)
49 NOSAT=NOSAT
50 NOSAT=0
51 NTRY=NTRY+1
52 L=1
53 IF(NPRES.S.NF,0)GO TO 41
54 N=1
55 MS=1
56 S(I,J)=1
57 GO TO 10
58 41 NPRES
59 NS=IPRES
60 S(NPRES+1,I)=1
51 GO TO 40
62 99 RETURN
63 END

*STATISTICS* SOURCE STATEMENTS = 59, PROGRAM SIZE = 7634 BYTES, PROGRAM NAME = STASS

*STATISTICS* NO DIAGNOSTICS GENERATED.

**STASS** END OF COMPILATION I ****
SUBROUTINE MOVE
PRESENT STATE OF CONSTRAINT MATRIX
INTO ANOTHER MATRIX, PERMUTE COLUMNS
ACCORDING COLUMN COUNT OF ONES.

AUTHOR: MIRIAM HERNANDEZ A.
DATE: 2/27/87

PARAMETER DESCRIPTION:
- IPCU2 : (I) COMPLETE CONSTRAINT MATRIX
- A : (I) PRESENT STATE MATRIX
- AP : (I) PERMUTED PRESENT STATE MATRIX IN THE
GIVEN ORDER
- JP : (I) ARRAY WITH THE GIVEN ORDER
- N2 : (I) NUMBER OF ROWS OF THE MATRICES
- NSTAT : (I) NUMBER OF COLUMNS OF THE ARRAYS
- NPRES : (I) NUMBER OF PRE-ASSIGNED STATES
- NPRES: (I) NUMBER OF STATES PRE-ASSIGNED

INTEGER A(100, 15), AP(100, 15)
DIMENSION JP(2, NSTAT), IPCU2(100, 46), JP1(2, 15)
COMMON /NL/1/IPC2/Bl9/A/Bl/4/AP
NF=NI*NSTAT-1
IF(NPRES=EQ.0)GO TO 23
DO 10 I=1, NPRES
10 JP(I,2,1)=JP(I,2,1)
23 CALL MOVE(INP, INSTAT, NI, NF)
11 CALL Dollen(JP, NI, NSTAT)
12 CALL DIBIT(JP, NSTAT)
13 IF(NPRES.NE.0)GO TO 22
14 CALL MOVE(JP, NPRES, NSTAT)
15 RETURN
16 22 II=II+1
17 DO 20 II=1, NSTAT
18 DO 21 II=1, NPRES
19 IF(JP(I,2,1).NE.JP1(2,1))GO TO 20
20 CONTINUE
21 JP1(1,II)=JP1(1,II)
22 JP1(2,II)=JP1(2,II)
23 II=II+1
24 CONTINUE
25 CALL MOVE(JP1, NI, NSTAT)
26 DO 24 II=1, NSTAT
27 JP1(1,II)=JP1(1,II)
28 JP1(2,II)=JP1(2,II)
29 CONTINUE
30 RETURN
31 END
LEVEL 13.1 (FEB 1984) VS FORTRAN

STATISTIC: NO DIAGNOSTICS GENERATED.

ORDER: END OF COMPILATION 1
C ** SUBROUTINE : CREATE AN ARRAY OF ONE COUNT **
C ** OF A MATRIX COLUMNS. **
C ** AUTHOR : MIRIAM HERNANDEZ A. **
C ** DATE : 2/27/87 **

PARAMETER DESCRIPTION:
- A : (I) INPUT MATRIX
- JP : (I) ONE COUNT ARRAY
- NR : (I) INPUT MATRIX NUMBER OF ROWS
- NSTAT : (I) NUMBER OF COLUMNS

SUBROUTINE COUNT(JP,NR,NSTAT)
INTEGER A(100,15)
DIMENSION JP(2,NSTAT)
COMMON /BL9/A
DO 10 J=1,NSTAT
  K=0
  DO 20 I=1,NR
  10 JP(I,J)=K
  20 IF(A(I,J).EQ.1)K=K+1
JP(1,J)=J
CONTINUE
RETURN
END

*STATISTICS* SOURCE STATEMENTS = 13, PROGRAM SIZE = 618 BYTES, PROGRAM NAME = COUNT

*STATISTICS* NO DIAGNOSTICS GENERATED.

*COUNT** END OF COMPILATION 1 *COUNT**
**SUBROUTINE** GCAND(L,NSI)

INTEGER SI(100,15)
COMMON /BL3/SI
LL=1-1
NSI=2**0(LL)
L=2**0(LL)
IF(L.NE.1)GO TO 30
SI(1,1)=0
SI(2,1)=1
RETURN
30 DO 10 J=LL,1,-1
10 DO 20 I=1,L2
II=I+L2
SI(I,J)=SI(I,J)
SII(I,L)=0
SI(I,L)=1
20 CONTINUE
10 CONTINUE
RETURN
END

@STATISTICS@ SOURCE STATEMENTS = 20, PROGRAM SIZE = 720 BYTES, PROGRAM NAME = GCAND

@STATISTICS@ NO DIAGNOSTICS GENERATED.

**GCAND** END OF COMPILED 1
REQUESTED OPTIONS (EXECUTE): NOSDUMP

OPTIONS IN EFFECT: NOLIST NOMAP NOKREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED

```fortran
C
C ** SUBROUTINE : SORT THE FIRST ROW OF A TWO-ROW C
C ** ARRAY AND MOVE THE SECOND ROW ACCORDINGLY. C
C ** AUTHOR : MIRIAM HERNANDEZ A. C
C ** DATE : 2/27/87 C
C
C PARAMETER DESCRIPTION:
C - JP : (1) ARRAY TO BE SORTED
C - NS : (1) INPUT ARRAY NUMBER OF COLUMNS
C
SUBROUTINE SORT(JP,NS)
DIMENSION JP(2,NS)
NELE=NS
NPASS=NELE-1
DO 10 J=1,NS
IMAX=NELE-J
DO 10 I=1,IMAX
IF(JP(I,1)+GT.JP(I,1+1))GO TO 10
10 HOLD1=JP(I,1)
HOLD2=JP(2,1)
JP(1,1)=JP(1,1+1)
JP(2,1)=JP(2,1+1)
JP(1,1+1)=HOLD1
JP(2,1+1)=HOLD2
CONTINUE
10 RETURN
END
```

STATISTICS: SOURCE STATEMENTS = 17, PROGRAM SIZE = 838 BYTES, PROGRAM NAME: SORT

STATISTICS: NO DIAGNOSTICS GENERATED.

END OF COMPILATION
SUBROUTINE MOVE(NR,NSTAT,NI,NF)
INTEGER IPCU2(100,16)
INTEGER A(100,15)
COMMON /BL1/IPC2/BL2/A
DO 10 I=1,NR
DO 10 J=NI,NF
JJ=-J+I
A(I,JJ)=IPC2(I,J)
CONTINUE
RETURN
END

PARAMETER DESCRIPTION:
- IPCU2 : (1) SOURCE ARRAY
- A : (0) OBJECT ARRAY
- NR : (1) NUMBER OF ROWS OF THE ARRAYS
- NSTAT : (1) OBJECT ARRAY NUMBER OF ROWS
- NI : (1) INITIAL POSITION TO BE MOVED
- NF : (1) FINAL POSITION TO BE MOVED

SOURCE ARRAY
OBJECT ARRAY
NUMBER OF ROWS OF THE ARRAYS
OBJECT BOARD NUMBER OF ROWS
INITIAL POSITION TO BE MOVED
FINAL POSITION TO BE MOVED

MOY! PART OF AN ARRAY INTO ANOTHER.
MIRIAM HERNANDEZ A.
2/27/87
*STATISTICS*  SOURCE STATEMENTS = 13, PROGRAM SIZE = 606 BYTES, PROGRAM NAME = MOVECP

*C* *STATISTICS*  NJ DIAGNOSTICS GENERATED.

**MOVECP** END OF COMPILATION 1 00000
SUBROUTINE CSELECT(K,NR,L,NSAT)

INTEGER S(100,15),C(100,15),A(100,15),AP(100,15)
INTEGER F(100,15),SS(100,15),B(100,15)
INTEGER Cf(100,15),F1,D1
COMMON /SL2/SL4/C/SL4/A/SL4/C1
IF(NF.NE.0)GO TO 10

CANDIDATE SET IS EMPTY, INCREMENT CODE LENGTH

L=L+1
RETURN

DO 20 I=1,NSAT
IF(A(I,K).EQ.0)AP(I,1)=1
20 IF(A(I,K).EQ.1)AP(I,1)=0

CALL SSELECT(A,S,F,VR,KK,L)

FOR STATES NOT IN THE CONSTRAINT MATRIX, CHOOSE ANY CODE NOT COVERED BY ANY FACE

DO 30 I=1,NSAT
30 IF(C(I,J1).EQ.0)GO TO 100
DO 220 J1=1,N
DO 215 I=1,NSAT
DO 210 J1=1,L
210 IF(C(I,J1).EQ.2)GO TO 210
220 IF(C(I,J1).EQ.2)GO TO 210
GO TO 215
CONTINUE
GO TO 220
CONTINUE
DO 230 J=1,L
S(I,J)=C(I,I,J)
RETURN
CONTINUE
C THERE IS NOT A CANDIDATE NOT COVERED BY A FACE
C INCREMENT CODE LENGTH
L=L+1
RETURN
C OBTAIN CANDIDATES ORDERED ACCORDING MINIMUM
C NUMBER OF VERTICES COVERED
CONTINUE
CALL VERTIC(K,MR,L,N)
SELECT CODE
DO 30 I=1,N
DO 40 J=1,L
S(I,J)=C(I,I,J)
S(I,J)=C(I,I,J)
OBTAI A0.55 TO CHECK CONSTRAINT RELATION
IP=1
CALL SELECT(A0.55,B,MR,IP,L)
CHECK IF B+I IS AN EMPTY MATRIX, I.E., A MATRIX
WHOSE ROWS HAVE AT LEAST ONE EMPTY ENTRY
DO 60 II=1,MR
DO 70 JJ=1,L
BI=BI+1,II
FI=FII,II
CALL CONJ(BI,FI,1V)
IF(IV.E1,II)GO TO 60
CONTINUE
C IF B+I IS NOT AN EMPTY MATRIX CHOOSE ANOTHER
C CANDIDATE
GO TO 30
CONTINUE
C CANDIDATE SATISFY CONSTRAINT RELATION, IF IT IS VALID CODE
DO 99 J=1,L
S(I,J)=SS(I,J)
RETURN
LEVEL 1.3.1 (FE3 1984) VS FORTRAN DATE: APR 27, 1987

53  30  CONTINUE
54  C
      MOSAT=MOSAT+1
55  C
      NO CANDIDATE SATISFY CONSTRAINT RELATION,
56  C
      TAKE FIRST CANDIDATE AVAILABLE
57  C
      DO 05  J=1,L
58  05  S(K,J)=C1(I,J)
59  06  RETURN
60  END

*STATISTICS*  SOURCE STATEMENTS = 56, PROGRAM SIZE = 26214 BYTES, PROGRAM NAME = CSELF

*STATISTICS*  NO DIAGNOSTICS GENERATED.

**CSELF** END OF COMPILATION 1 **

---

In this FORTRAN code, the program is designed to handle selection criteria, as indicated by the comments and variables. The code uses a do loop to assign values to a matrix, `S`, where each element `S(K,J)` is assigned the value of `C1(I,J)` for `K` and `J` within specified ranges. The program appears to be part of a larger system, possibly for decision-making or optimization processes.
**SUBROUTINE**: IMPLEMENT SELECTION OPERATOR. **

**AUTHOR**: MIRIAM HERNANDEZ A. **

**DATE**: 3/02/87 **

**PARAMETER DESCRIPTION:**

- A : (I) INPUT MATRIX
- B : (I) INPUT MATRIX
- C : (O) RESULT MATRIX
- K : (I) A AND C NUMBER OF ROWS
- M : (I) A NUMBER OF COLUMNS, 9 NUMBER OF ROWS
- L : (I) 0 AND C NUMBER OF COLUMNS

1 SUBROUTINE SELECT(A,B,C,K,M,L)
2   INTEGER A(100,15),B(100,15),C(100,15)
3   INTEGER C1,C2,C3
4   DO 10 J=1,L
5   DO 10 I=1,K
6     C3=3
7   DO 10 J=1,M
8     IF(A(I,J).EQ.0)C1=3
9     IF(A(I,J).EQ.1)C1=8(J,J1)
10    C2=C3
11    CALL DISJ(C1,C2,C3)
12    C1(J,J1)=C1
13   10 CONTINUE
14   RETURN
15 END

**STATISTICS**: SOURCE STATEMENTS = 15, PROGRAM SIZE = 772 BYTES, PROGRAM NAME = SELECT

**STATISTICS**: NO DIAGNOSTICS GENERATED.

**SELECT** END OF COMPILATION
C ** SUBROUTINE : IMPLEMENT DISJUNCTION OPERATOR. **  
C ** AUTHOR : MIRIAM HERNANDEZ A. **  
C ** DATE : 3/02/87 **

PARAMETER DESCRIPTION:
- C1 : (1) INPUT NUMBER
- C2 : (1) INPUT NUMBER
- C3 : (0) RESULT

(0,1 BINARY VARIABLES,
 2 = DON'T CARE,
 3 = EMPTY VALUE)

SUBROUTINE DISJ(C1,C2,C3)
INTEGER C1,C2,C3
IF(C2.EQ.0)C2=4
IF(C1.EQ.0)C1=4
GO TO (10,20,30,40)C2
10 10 GO TO (50,70,90,70)C1
20 20 GO TO 70
GO TO (50,70,90,50)C1
10 40 GO TO (70,70,50,50)C1
20 50 C3=0
13 RETURN
14 60 C3=1
15 RETURN
16 70 C3=2
17 RETURN
18 80 C3=3
19 RETURN
20 FRA

STATISTICS: SOURCE STATEMENTS = 18, PROGRAM SIZE = 59A BYTES, PROGRAM NAME = DISJ

NO DIAGNOSTICS GENERATED.
SUBROUTINE ORDER CANDIDATES ACCORDING TO LESS USED

PARAMETER DESCRIPTION:
- \( A \) : (1) Present state field of constraint matrix
- \( S \) : (1) Coded state set
- \( C \) : (1) Candidate set
- \( Cl \) : (0) Candidate set ordered according to less used number of vertices per face
- \( X \) : (1) Matrix A number of rows
- \( C \) : (1) Number of state to be coded
- \( L \) : (1) Code length (dimension)
- \( N \) : (1) Number of candidates

SUBROUTINE VERT(k, M, N, L, N)
INTEGER F(100, 15), C(100, 15), A(100, 15), S(100, 15)
INTEGER C(100, 15)
DIMENSION M(2, 100), S(2, 100)
COMMON /SL6/C/BL6/C1/BL4/A/BL2/S
DO 50 J = 1, N
50 \( M[2, J] = 0 \)
DO 10 I = 1, N
10 \( M[2, I] = 1 \)
DO 50 J = 1, L
50 \( S[C(I, J)] = C(I, J) \)
DO 10 I = 1, N
10 CALL SELECT(A, S, F, M, C, L)
COUNT NUMBER OF VERTICES COVERED BY EACH FACE
DO 20 II = 1, NR
20 \( M[I, II] = 1 \)
M[I, II] = 0
DO 20 J = 1, L
IF(F[II, J] .NE. 2) GO TO 20
\( M[I, II] = M[I, II] + 1 \)
CONTINUE
TAKE THE FACE WITH MAXIMUM NUMBER OF VERTICES COVERED
CALL SORT(M, NR)
21 M2(I,1)=M1(I,1)
22 10 CONTINUE
   C TAKE CANDIDATE WITH THE FACE WITH MINIMUM NUMBER
   C OF VERTESES COVERED
   C
23 CALL SORT(M2,N)
24 I2=0
25 DO 30 I=M,1,-1
26 I2=I2+1
27 I1=M2(I2,1)
28 DO 30 J=1,L
29 C(I2,J)=C(I1,J)
30 30 CONTINUE
31 RETURN
32 END

*STATISTICS* SOURCE STATEMENTS = 32, PROGRAM SIZE = 8854 BYTES, PROGRAM NAME = VERTIC

*STATISTICS* NO DIAGNOSTICS GENERATED.

*VERTIC* END OF COMPILATION 1 000000
SUBROUTINE CONJ(C1,C2,C3)
INTEGER C1,C2,C3
IF(C1.EQ.0)C1=4
IF(C2.EQ.0)C2=4
GO TO (10,20,30,40)C2
GO TO (60,70,80,90)C1
GO TO (60,70,80,50)C1
GO TO A0
GO TO (60,50,80,50)C1
C3=0
RETURN
C3=1
RETURN
C3=2
RETURN
C3=3
RETURN
END)

*STATISTICS*  SOURCE STATEMENTS = 18, PROGRAM SIZE = 59B BYTES, PROGRAM NAME = CONJ

*STATISTICS*  NO DIAGNOSTICS GENERATED.

**CONJ** END OF COMPIRATION **End**
**SUBROUTINE ADJOIN(NR,K,L,NSTAT,NPRES)**

**PARAMETER DESCRIPTION:**
- **S** : (I) CODE SET
- **A** : (I) PRESENT STATE FIELD OF CONSTRAINT MATRIX
- **K** : (I) CONSTRAINT MATRIX POSITION OF LAST STATE THAT WAS CODED
- **L** : (I) CODE LENGTH
- **NSTAT** : (I) TOTAL NUMBER OF STATES TO BE CODED
- **NPRES** : (I) NUMBER OF PRE-ASSIGNED CODES

**SOURCE STATEMENTS = 12, PROGRAM SIZE = 530 Bytes, PROGRAM NAME = ADJOIN**

**NO DIAGNOSTIC) GENERATED.**

**ADJOINING END OF COMPILATION**
SUBROUTINE CANDIN(NS,NS1,L)
INTEGER C(100,15),SI(100,15),S(100,15)
N=0
DO 10 II=1,NS1
DO 20 JJ=1,L
10    IF(SI(II,J).NE.S(I,J))GO TO 20
20    GO TO 10
10    CONTINUE
N=N+I
DO 40 JJ=1,L
40    C(N,J2)=SI(II,J2)
10    CONTINUE
RETURN
END
STATISTIC: SOURCE STATEMENTS = 164, PROGRAM SIZE = 724 BYT.S, PROGRAM NAME = CAND
STATISTIC: NO DIAGNOSTICS GENERATED.
APPENDIX C

PROOF OF THE THEOREMS

The proof of the theorems presented in this thesis is detailed here as quick reference for the reader. Section C.1 has been taken from the book "Fundamentals of Logic Design" (2), pages 625 and 626. Section C.2 has been duplicated from the paper "Optimal State Assignment for Finite State Machines" (1), pages 277 and 278.

C.1 State Equivalence Theorem

Two states $S_1$ and $S_2$ of a sequential network are equivalent if and only if for every single input $x$, the outputs are the same and the next states are equivalent.

Proof:

a. (if) Assume that $W(S_1, x) = W(S_2, x)$ and $d(S_1, x) = d(S_2, x)$ for every input $x$. Then from Definition 2.1 for every input sequence $X$,

$W[d(S_1, X)] = W[d(S_2, X)]$

For the input sequence $Y=x$ followed by $X$, we have

$W(S_1, Y) = W(S_1, x)$ followed by $W[d(S_1, x), X]$

$W(S_2, Y) = W(S_2, x)$ followed by $W[d(S_2, x), X]$

Hence $W(S_1, Y) = W(S_2, Y)$ for every input sequence $Y$ and $S_1 = S_2$ by Definition 2.1.

b. (only if) Assume that $S_1 = S_2$. Then by Definition 2.1,

$W(S_1, Y) = W(S_2, Y)$ for every input sequence $Y$. Let $Y=x$ followed by $X$. Then:

$W(S_1, x) = W(S_2, x)$ and $W[(S_1, x), X] = W[(S_2, x), X]$
for every sequence \( X \). Hence from Definition 2.1, 
\[ W(S_1,x) = W(S_2,x). \]

C.2 STATE ASSIGNMENT THEOREMS AND LEMMAS

1. If \( S \) satisfies the constraint relation for a given \( A \), then \( S' = [S \ T] \) satisfies the constraint relation, where \( T \) is any \( (0,1) \) matrix with \( n_s \) rows.

Proof:

Let \( s_i.' = [s_i. \ t_i.] \ \forall i = 1,2,\ldots,n_s \) and suppose by contradiction that \( \exists k \) such that:
\[ [a.k \ .sk.'] \wedge [A.S'] \neq \emptyset \]
Then,
\[ [a.k \ .sk. \ a.k.tk.] \wedge [A.S \ A.T] \neq \emptyset \]
\[ \Rightarrow [a.k \ .sk.] \wedge [A.S] \neq \emptyset \]
and we have a contradiction.

2. If \( S \) satisfies the constraint relation for a given \( A \), then \( S \) satisfies the constraint relation for \( A' = \begin{bmatrix} A \\ am \end{bmatrix} \)
where \( am \) is a meet of \( A \).

Proof:

Let \( F = A.S \), and
\[ F = A'.S = \begin{bmatrix} A \\ am \end{bmatrix} \cdot S = \begin{bmatrix} A.S \\ am.S \end{bmatrix} = \begin{bmatrix} F \\ fm \end{bmatrix} \]

For the sake of contradiction, suppose that there exists a state, say \( k \), such that
\[ [a.k \ .sk.] \wedge [A'.S] \neq \emptyset \]
Since \[ [a.k \ .sk.] \wedge [A.S] = \emptyset, \] then \[ [amk. \ sk.] \wedge fm' \neq \emptyset. \]
Therefore, \( am_k = 0 \). Since \( am \) is a meet, there exists \( a_i \) such that \( a_{ij} \geq a_{mj}, \forall j = 1, 2, \ldots, ns \) and \( a_{ik} = 0 \). Since \( fm' \). is a subspace of \( fi' \), then \( [a_{ik} \cdot sk.] \cap fi' \neq \emptyset \), which implies \( [a_{ik} \cdot sk.] \cap A \cdot S \neq \emptyset \) and we have a contradiction.

3. \( S \) satisfies the constraint relation for \( A \) if and only if \( S \) satisfies the constraint relation for \( A_p \).

Proof:

a. (if) By lemma 3.2.

b. (only if) If \( S \) satisfies the constraint relation for \( A \), then \( S \) satisfies the constraint relation for any matrix obtained from \( A \) by dropping any number of rows.

4. Let \( S \) satisfy the constraint relation for a given \( A \). Then there exists a matrix:

\[
S' = \begin{bmatrix} S & T \end{bmatrix}, \quad T \subseteq \{0\}
\]

that satisfies the constraint relation for \( A' = [A \ \alpha] \) is either:

i) \( \alpha \) is a column of "0"s;

ii) \( \alpha \) is a column of "1"s, where \( \alpha \) is the set of non-zero rows of \( A \) corresponding to the non-zero entries of \( \alpha \).

Proof:

The new face matrix is:

\[
F = A' \cdot S' = [A \ \alpha] \cdot \begin{bmatrix} S & T \end{bmatrix} = [A \cdot S \ A \cdot T] \cap [\alpha \cdot \sigma]
\]

Let us consider the following cases:

i) \( \alpha \) is a column of "0"s.

The new state does not belong to any group represented
by A, and can be any encoding not covered by the existing faces.

Let $\sigma = [c \ 1]$, where $c$ is any binary vector of length $nb$. Note that $\sigma$ is disjoint from the other encodings represented by $[S \ T]$, because the trailing bit is different. Since $\alpha \cdot \sigma$ is a matrix of empty values, then $F' = [A \cdot S \ A \cdot T]$. Moreover, since $S$ satisfies the constraint relation for $A$, by lemma 3.1, $[S \ T]$ satisfies the constraint relation for $A$ as well.

Therefore, $[a_1 \ 1 \ s_1] \land F' = \emptyset$ for $i = 1, 2, \ldots, ns$. We need then only verify that: $[\bar{\alpha} \cdot \sigma] \land F' = \emptyset$. By construction the rightmost column of $\bar{\alpha} \cdot \sigma$ has all entries equal to "1" while the rightmost column of $F'$, i.e. $[A \cdot T]$, has all entries equal to "0" or $\emptyset$. Since $1 \land 0 = \emptyset$, the last column of $[\bar{\alpha} \cdot \sigma]$ consists of all $\emptyset$ and the constraint relation is satisfied.

ii) $A$ has a column of "1"s, then the intersection of the groups containing the new state contain another state as well. The new encoding can be constructed by appending a "1" to any state encoding in that intersection.

Let $c$ be the encoding of any state corresponding to a column of "1"s in $A$ and let $\sigma = [c \ 1]$.

For this choice of $\sigma$, the new face matrix is: $F' = [F \ \beta] = [A \cdot S \ A \cdot T] \lor [\alpha \cdot \sigma]$, where:

$$
\beta_k = \begin{cases} 
1 \text{ V}k \text{ such that } \alpha_k = 1 \text{ and } \alpha_k \text{ is a row of "0"s} \\
0 \text{ V}k \text{ such that } \alpha_k = 0 \\
\end{cases}
$$
Since $S$ already satisfies the constraints given by $A$, i.e.,
\[ [a \cdot i \cdot s] \land [A \cdot S] = \emptyset \ \forall i = 1, 2, ..., ns \]
then,
\[ [a' \cdot i \cdot s'] \land F' = \emptyset \ \forall i = 1, 2, ..., ns. \]

Therefore we need only consider the new state and in particular the rightmost column, $\psi$, of $\bar{\alpha} \cdot \sigma$. Since $\psi_k = 0 \ \forall k$ such that $\alpha_k = 1$ and $\psi_k = 1 \ \forall k$ such that $\alpha_k = 0$, then $\psi \land \beta = \emptyset$ and
\[ [\bar{\alpha} \cdot \sigma] \land F' = \emptyset. \]
BIBLIOGRAPHY


