THE LOGIC OF DECISION TABLES AND THEIR APPLICATIONS

BY

Albert Pinhas

Submitted to the Department of Electrical Engineering in partial fulfilment of the requirements for the degree of

Master of Applied Science
Department of Electrical Engineering
Faculty of Science and Engineering
University of Ottawa

OTTAWA, ONTARIO.

JANUARY, 1971

© Albert Pinhas 1971
ABSTRACT

The purpose of this thesis is to present an in depth study of Decision Tables. The study includes both, the investigation of the logical structure and the nature of decision tables, as well as the practical application of conversion procedures for electronic data processing (EDP).

In the first part of this thesis a detailed mathematical description in terms of matrices, and the characterization of decision tables are introduced. Based upon these basic rules operations are defined and used for the precise investigation of ambiguity and completeness of decision tables. The problems associated with splitting and merging the decision tables are discussed with the aid of different types of links. Finally, different relations between decision tables are defined, and based upon them algorithms for minimization of different types of decision tables are included.

In the second part, a survey with detailed analysis of existing conversion procedures and algorithms is presented. This part also includes testing and diagnostic procedures with some powerful extensions.

The final part suggests an approach for implementing a decision table pre-compiler. The thesis is concluded by an attempt to design a comprehensive example that indicates the power and advantages of a good decision table pre-compiler.
ACKNOWLEDGEMENTS

I wish to thank Dr. C.D. Shepard for introducing me to the topic of decision tables and his encouragement to choose this topic for my Master's Thesis.

I am most deeply indebted to Prof. M. Kriger for guiding, encouraging and giving very generously of his time while serving as my advisor in the preparation of my Master's Thesis.

I am also very grateful to my wife for her encouragement, understanding and her great deal of patience with which she typed this work.
**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>i</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td></td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td></td>
<td>iii</td>
</tr>
<tr>
<td>CHAPTER I:</td>
<td>- INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER II:</td>
<td>- DEFINITION AND CHARACTERISTICS OF DECISION TABLES</td>
<td>6</td>
</tr>
<tr>
<td>II-1:</td>
<td>- DEFINITION AND CLASSIFICATION OF DECISION TABLES</td>
<td>6</td>
</tr>
<tr>
<td>II-2:</td>
<td>- CHARACTERISTICS OF DECISION TABLES</td>
<td>16</td>
</tr>
<tr>
<td>CHAPTER III:</td>
<td>- MANIPULATION OF AND RELATIONS BETWEEN DECISION TABLES</td>
<td>30</td>
</tr>
<tr>
<td>III-1:</td>
<td>- SPLITTING AND MERGING OF DECISION TABLES</td>
<td>30</td>
</tr>
<tr>
<td>III-2:</td>
<td>- RELATIONS BETWEEN DECISION TABLES</td>
<td>47</td>
</tr>
<tr>
<td>CHAPTER IV:</td>
<td>- CONVERSION METHODS</td>
<td>62</td>
</tr>
<tr>
<td>IV-1:</td>
<td>- THE TREE METHOD</td>
<td>62</td>
</tr>
<tr>
<td>IV-2:</td>
<td>- THE MASK METHOD</td>
<td>87</td>
</tr>
<tr>
<td>IV-3:</td>
<td>- THE COMPUTED GO TO METHOD</td>
<td>101</td>
</tr>
<tr>
<td>CHAPTER V:</td>
<td>- TESTING AND DIAGNOSTIC METHODS</td>
<td>109</td>
</tr>
<tr>
<td>V-1:</td>
<td>- TESTING AND DIAGNOSTIC METHODS IN CONVERSION TIME</td>
<td>110</td>
</tr>
<tr>
<td>V-2:</td>
<td>- TESTING AND DIAGNOSTIC METHODS IN RUN TIME</td>
<td>124</td>
</tr>
<tr>
<td>V-3:</td>
<td>- TESTING AND DIAGNOSTIC METHODS IN CONVERSION TIME AND IN RUN TIME</td>
<td>132</td>
</tr>
<tr>
<td>CHAPTER VI:</td>
<td>- PROPOSED CONVERSION METHOD</td>
<td>137</td>
</tr>
<tr>
<td>VI-1:</td>
<td>- REQUIREMENTS</td>
<td>137</td>
</tr>
<tr>
<td>VI-2:</td>
<td>- SUGGESTED DECISION TABLE COBOL PRE-COMPILER SYNTACTIC STRUCTURE</td>
<td>139</td>
</tr>
</tbody>
</table>
VI-3: DECISION TABLE PRE-COMPILED GENERAL BLOCK
DIAGRAM AND FILE CHART 147
VI-4: EXAMPLE 150
CHAPTER VII: CONCLUSIONS AND SUGGESTIONS 199
BIBLIOGRAPHY 203
VITAE 210
CHAPTER I: INTRODUCTION

The recent development of third generation hardware in the computers field and the increasing cost of the software necessary to support it has lead different scientists and mathematicians to design automated ways of software generation such as compiler generators. With the exception of report program generators (RPG'S) and sort program generators no real attempt was made to automate the generation of application programs, which could lead to the elimination (to some extent) of the costly job of the programmer. One possible approach for automation of user's program generation is decision tables.

The purpose of this thesis is to present an in-depth study of the logical structure of decision tables as well as the practical application of possible automated conversion procedures.

Decision tables, as such, represent an organized method for choosing a course of action upon the satisfaction of a set of specified conditions. The early use of decision tables was only as a systems analysis tool. With the advent of more advanced conversion techniques for converting decision tables into computer programs, today they are becoming an accepted tool for writing complex computer programs both in commercial and scientific applications. The main advantages of the use of decision tables are:

1. Direct communication language between the system analyst and the computer.
2. High level, computer independent, programming language.
3. An advanced systems analysis and design tool.
4. Enforces a clear problem statement and shows where information is ambiguous or missing.
5. Provides a precise means of communication among functional personnel.
6. Provides an effective means of documentation and eliminates much unnecessary verbal description which can be easily misinterpreted.
7. Encourages modularity of programs.
8. Enables visualization of complex situations.
9. An easily learned technique.
10. Decision tables are problem-oriented design methods rather than machine-oriented.
11. Reduces significantly the amount of technical knowledge of the computer required by the systems designer.
12. Provides easier maintenance of a system.

Presently, there are available a large number of conversion processors for the automatic conversion of decision tables into computer programs. With the acceptance of these automated conversion techniques, the precise definition and characterization of decision tables is of vital importance. This is required to avoid ambiguity between the intended logic in the decision table and the corresponding computer program.

This thesis is divided into four parts:
1. Definition and characteristics of decision tables.
2. Manipulation of, and relations between decision tables.
3. A survey of existing conversion methods and algorithms.
4. A suggested decision table pre-compiler.

In the first part a generalized definition of decision tables, that provides consistent nomenclature and classification, is introduced. This is done in a matrix format which is adaptable to mathematical treatment. Exact definitions for decision tables with unrelated conditions as well as with related conditions, in terms of ambiguity and completeness, are presented. These definitions are in terms of the above defined matrix, and thus lend themselves to mathematical treatment. Furthermore, this part includes the fundamentals required for the mathematical manipulation of decision tables.

The second part is devoted to the problems involved in the splitting, merging, relations and minimization of decision tables. The problem of splitting decision tables is considered from the point of view of splitting by rules, by conditions and by branching. This is done by defining different types of linkages between decision tables. The resulting decision tables are either in the form of concatenated tables, or in the form of a tree of decision tables. It was found that merging, the inverse of splitting, could not be generalized.

The problem of relations between decision tables was considered from the point of view of isomorphism and equivalence. In line with this a few algorithms, with the associated problems, are
presented. Finally, three procedures for the minimization of decision tables were developed. One, corresponding to rule minimization, utilizes the concept of rule partitioning. The second corresponding to condition minimization, combines a set of logically related conditions into a single extended condition. The third, corresponding to action minimization, combines a set of actions into a single extended action.

The third part contains a critical survey of existing conversion procedures and algorithms. The procedures corresponding to three different conversion methods (the tree method, the mask method and the computed GO TO method) are convered separately. The advantages and the disadvantages of each conversion method are analysed together with the characteristics of each algorithm. This part also includes a set of theorems proving necessary and sufficient conditions for testing and diagnostic procedures for conversion time and a set of testing procedures for run time.

The last part suggests a flexible decision table pre-compiler based upon practical usage requirements. This scheme is presented in terms of syntactic and semantic description together with suggested conversion procedures. Finally an attempt was made to design a comprehensive example that indicates the usefulness of a decision table pre-compiler (or compiler). The actual conversion of this example into COBOL program code, using the steps of the suggested pre-compiler was done manually due to lack of funds necessary
for an implementation of a desired pre-compiler. The resulting program was run on a Burroughs B-5500 computer.
CHAPTER II. DEFINITION AND CHARACTERISTICS OF DECISION TABLES

The purpose of this chapter is to introduce the general background required in the study of decision tables and to provide a standard nomenclature. The latter is important as there are many uses of decision tables and thus, a number of different nomenclatures are used in existing literature.

In the first part of this chapter a general definition of decision tables is given and a classification based upon this is presented. The classification introduced here subdivides decision tables into 18 different classes, while the existing literature subdivides them into 2, 4, or 6 classes. In the same section a formal notation is introduced to allow detailed mathematical presentation.

In the second part of this chapter, a set of basic rule operations is defined to allow the detailed characterization of decision tables. This characterization, which is based on redundancy, contradiction and completeness, is done separately for tables with unrelated conditions only as well as for tables with both related and unrelated conditions.

II-1 DEFINITION AND CLASSIFICATION OF DECISION TABLES.

In this section we shall introduce a general definition of decision tables, and based upon this we provide a classification of
decision tables. At the end of this section we also introduce a list of notations.

**II-1-1 DEFINITION OF DECISION TABLES:**

<table>
<thead>
<tr>
<th></th>
<th>RULES</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>...i...</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STUB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STUB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. II-1-1

A decision table as indicated in Fig. II-1-1 has p (or p+1) columns and q+r rows. The resulting table is subdivided into two sub-tables: A conditions sub-table and an actions sub-table.

The conditions sub-table is divided into two parts: the CONDITION STUB containing the q CONDITIONS (condition statements) and the CONDITION ENTRY MATRIX \( \{c_{ji}\}_{q \times p} \).

A condition is LIMITED, \( c_j \), whenever it is completely specified in the condition stub, or EXTENDED, \( C_j \), whenever the specification of the condition is extended into the appropriate condition row in the condition entry matrix.
provides a means of specifying the actions to be taken if none of the other p rules is satisfied. Further discussion of the else rule will be presented later.

The action sub-table is also divided into two parts; the ACTION STUB containing the r ACTIONS(action statements) and the ACTION ENTRY MATRIX \( \{a_{ki}\}_{r \times p} \). The action statement can be limited, \( a_k \), whenever the action is completely specified in the action stub, or extended, \( A_k \), whenever the specification of the action is extended into the appropriate action row in the action entry matrix.

With a limited action, \( a_k \), we associate the set \( W_k = \{x\} \) and with an extended action we associate the set \( W_k = \{W_{kl}, \ldots, W_{kn_k}\} \) with \( n_k > 1 \).

The \( i \)th column \( \delta^i \) of the action entry matrix represents the actions and the order in which they are to be taken whenever the \( i \)th rule is satisfied. The \( k \)th row of \( \delta^i \) denoted by \( \delta_k^i \) is "x" in the limited case, or an element of \( W_k \) in the extended case, wherever the \( k \)th action is to be executed; otherwise, it will be blank.

Accordingly, the \( ki \)th entry of the action entry matrix \( a_{ki} \) is given by:

\[
a_{ki} = \delta_k^i \quad \therefore = \text{blank} |x| \in W_k
\]

Note: The order in which the conditions of the table are arranged is insignificant in the above definition. This excludes conditions such as testing a quantity for zero before performing other tests which use that quantity as a divisor. This freedom in
With the \( j^{th} \) condition we associate the set \( V_j = \{V_{jl}, V_{j2}, \ldots, V_{jn_j}\} \) with \( n_j \geq 2 \). The \( j^{th} \) row of the condition entry matrix, \( \rho_j \), associated with \( j^{th} \) condition contains elements from the set \( V_j \) or a don't care (sometimes denoted by "d", or left blank, or by "-"). It should be noted that only for extended conditions can \( n \geq 2 \). However in the case of limited condition \( V_j = \{Y, N\}, \quad n_j = 2 \).

The \( i^{th} \) rule is represented by the \( i^{th} \) column of the condition entry matrix, \( \gamma_i \), whose \( j^{th} \) entry \( \gamma_{ij} \) is either a don't care entry or an element from the set \( V_j \). Such a column \( \gamma_i \) is called an INCOMPLETELY SPECIFIED COLUMN or INCOMPLETE RULE (providing at least one of the entries \( \gamma_{ij} \) is a don't care entry and not more than \( q-1 \) entries are don't care entries).

Thus the \( j^{th} \) entry of the condition entry matrix is given by;

\[
c_{ji} = \gamma_{ij} = \rho_{ij} \quad : = \text{don't care} \quad ; \in V_j
\]

Equivalently \( \gamma_i \) can be represented by a set of columns \( \omega_i = \{\omega_{i1}, \omega_{i2}, \ldots, \omega_{in_i}\} \). Such a column \( \omega_i \) is called a COMPLETELY SPECIFIED COLUMN (completely specified rule) and its \( j^{th} \) entry \( \omega_{ij} \) is an element of \( V_j \).

Note that:
\[
l_i \leq l_i \leq \max\{n_2 x_3 x \ldots n_p, n_1 x_3 x \ldots x n_p, \ldots, n_1 x n_2 \ldots n_p - 1\}
\]

A decision table may include the \( p+1^{st} \) column which is associated with the so called ELSE RULE. This is a special rule which
the order of conditions will allow us later to design efficient algorithms for treating decision tables.

To clarify the above definition of decision tables we introduce now an example of a decision table which includes the different types of conditions, actions and rules.

<table>
<thead>
<tr>
<th>CONDITION STUB</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>1</th>
<th>...</th>
<th>P</th>
<th>P+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A = B</td>
<td>Y</td>
<td>N</td>
<td>...</td>
<td>N</td>
<td>...</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>P =</td>
<td>3</td>
<td>-</td>
<td>...</td>
<td>15</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SIG Q</td>
<td>&quot;+&quot;</td>
<td>&quot;0&quot;</td>
<td>&quot;-&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C &gt; D</td>
<td>Y</td>
<td>N</td>
<td>...</td>
<td>Y</td>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>j</td>
<td>R =</td>
<td>S or P</td>
<td>-</td>
<td></td>
<td>S and P</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>j+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q-2</td>
<td>W =</td>
<td>3 or 4</td>
<td>5.5</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q-1 REL U, V</td>
<td>U &gt; V</td>
<td>U &lt; V</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>T &gt; A</td>
<td>&lt; A</td>
<td>...</td>
<td>&lt; A</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTION STUB</th>
<th>1</th>
<th>A ← B</th>
<th>x</th>
<th>...</th>
<th>x</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Q ← 10</td>
<td>x</td>
<td>...</td>
<td>-</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>K PERFORM</td>
<td>xl</td>
<td>x2</td>
<td>...</td>
<td>xl</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>K+1 Q ← E/Q</td>
<td>x</td>
<td>x</td>
<td>...</td>
<td>x</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>GO TO</td>
<td>P1</td>
<td>P2</td>
<td>...</td>
<td>P2</td>
<td>...</td>
</tr>
</tbody>
</table>

FIG. II-1-2
Example of IS-MCE-MAE decision table.
a. Completely specified rule (providing the entries not shown exclude don't care).
b. Incompletely specified rules.
c. The else rule.
d. Limited condition entries.
e. Extended condition entries.
f. Limited action entries.
g. Extended action entries.

Note: The extended condition entries and action entries do not necessarily represent all different possible conditions and actions. In existing precompilers, interpreters and compilers there are some limitations imposed on the user.

Many other condition can be used, for example, testing a field for numeric or alphabetic, etc. Some of the most commonly used conditions and actions are summarized in later sections.

II-1-2 CLASSIFICATION OF DECISION TABLES.

Decision tables are classified with respect to three major characteristics: The don't care condition, the number of elements in the sets \( V_j \) and the number of elements in the sets \( W_k \).

i) A decision table is called COMPLETELY SPECIFIED if it excludes don't care conditions in its condition entries \( c_{ji} \); otherwise it is called INCOMPLETELY SPECIFIED.

ii) A decision table is called LIMITED CONDITION ENTRY if all the
sets $V_j$, $j = 1, 2, \ldots , q$ are of the type $V_j = \{Y, N\}$ and it is called EXTENDED CONDITION ENTRY if all sets $V_j$ are of the type $V_j = \{V_{ji}, \ldots , V_{jn}\}$, $n \geq 2$ and $V_j \neq \{Y, N\}$, $j = 1, 2, \ldots , q$.

Otherwise it is a MIXED CONDITION ENTRY decision table.

iii) A decision table is called LIMITED ACTION ENTRY if the sets $W_k$, $k = 1, 2, \ldots , r$ are of the type $W_k = \{x\}$ and it is called EXTENDED ACTION ENTRY if all the sets $W_k$, $k = 1, 2, \ldots , r$ are of the type $W_k = \{w_{k1}, w_{k2}, \ldots , w_{kn_k}\}$, with $n_k \geq 1$. Otherwise it is called a MIXED ACTION ENTRY decision table.

Accordingly we can group the decision tables into the following classes:

1. CS-ICE-LAE :: = COMPLETELY SPECIFIED - LIMITED CONDITION ENTRY - LIMITED ACTION ENTRY.

2. CS-LCE-EAE :: = COMPLETELY SPECIFIED - LIMITED CONDITION ENTRY - EXTENDED ACTION ENTRY.

3. CS-LCE-MAE :: = COMPLETELY SPECIFIED - LIMITED CONDITION ENTRY - MIXED ACTION ENTRY.

4. IS-LCE-LAE :: = INCOMPLETELY SPECIFIED - LIMITED CONDITION ENTRY - LIMITED ACTION ENTRY.

5. IS-LCE-EAE :: = INCOMPLETELY SPECIFIED - LIMITED CONDITION ENTRY - EXTENDED ACTION ENTRY.

6. IS-LCE-MAE :: = INCOMPLETELY SPECIFIED - LIMITED CONDITION ENTRY - MIXED ACTION ENTRY.

7. CS-ECE-LAE :: = COMPLETELY SPECIFIED - EXTENDED CONDITION ENTRY - LIMITED ACTION ENTRY.

8. CS-ECE-EAE :: = COMPLETELY SPECIFIED - EXTENDED CONDITION ENTRY - EXTENDED ACTION ENTRY.

9. CS-ECE-MAE :: = COMPLETELY SPECIFIED - EXTENDED CONDITION ENTRY - MIXED ACTION ENTRY.

10. IS-ECE-LAE :: = INCOMPLETELY SPECIFIED - EXTENDED CONDITION ENTRY - LIMITED ACTION ENTRY.
matic translation from a formatted decision table into high level (or machine code) computer language.

II-1-3 NOTATION.

To simplify the future discussion on decision tables we will adopt the following notations:

$c_i \equiv \text{LIMITED CONDITION}$

$C_i \equiv \text{EXTENDED CONDITION}$

$d = " \" = "¬" \equiv \text{don't care condition|don't do action}$

$V_j \equiv \text{The set of values associated with the } j^{\text{th}} \text{ condition}$

$V_j = \{V_{ji}, \ldots, V_{jn_j}\}, \quad n_j \geq 2$

$Y,N \equiv \text{The two different values a limited entry can have}$

$V_j = \{Y,N\} (\text{with exception of the don't care})$

$p \equiv \text{The number of columns in a decision table (except the ELSE rule) equal to the number of rules.}$

$q \equiv \text{The number of rows in the condition sub-table equal to the number of conditions.}$

$r \equiv \text{The number of rows in the action sub-table equal to the number of actions.}$

$\gamma^i \equiv \text{The } i^{\text{th}} \text{ column in the condition entry matrix representing the } i^{\text{th}} \text{ rule.}$

$\gamma^i_j \equiv \text{The } j^{\text{th}} \text{ entry in the } \gamma^i \text{ column (can be a don't care or a value from the set } V_j\text{).}$

$\rho^j \equiv \text{The } j^{\text{th}} \text{ row in the condition entry matrix.}$

$\rho^i_j \equiv \text{The } i^{\text{th}} \text{ entry in the } j^{\text{th}} \text{ row of the condition entry matrix.}$
11. IS-ECE-EAE :: = INCOMPLETELY SPECIFIED - EXTENDED CONDITION
ENTRY - EXTENDED ACTION ENTRY.

12. IS-ECE-MAE :: = INCOMPLETELY SPECIFIED - EXTENDED CONDITION
ENTRY - MIXED ACTION ENTRY.

13. CS-MCE-LAE :: = COMPLETELY SPECIFIED - MIXED CONDITION ENTRY -
LIMITED ACTION ENTRY.

14. CS-MCE-EAE :: = COMPLETELY SPECIFIED - MIXED CONDITION ENTRY -
EXTENDED ACTION ENTRY.

15. CS-MCE-MAE :: = COMPLETELY SPECIFIED - MIXED CONDITION ENTRY -
MIXED ACTION ENTRY.

16. IS-MCE-LAE :: = INCOMPLETELY SPECIFIED - MIXED CONDITION
ENTRY - LIMITED ACTION ENTRY.

17. IS-MCE-EAE :: = INCOMPLETELY SPECIFIED - MIXED CONDITION
ENTRY - EXTENDED ACTION ENTRY.

18. IS-MCE-MAE :: = INCOMPLETELY SPECIFIED - MIXED CONDITION
ENTRY - MIXED ACTION ENTRY.

The hierarchy, then, in terms of complexity, can be represented by the following diagram:

![Diagram Image]

**Fig. II-1-3**

**Note:** This hierarchical diagram, as we shall see later, is quite important in developing or adapting algorithms for auto-
\[ c_{ji} = \gamma_i^j = \rho_i^j \] 
The \( ji \)th entry of the condition entry matrix.

\[ \omega^i \] 
The set of completely specified columns (or rules)
which represent the \( i \)th column (or rule) \( \gamma^i \).

\[ \omega^{i'} \] 
A completely specified rule from the set \( \omega^1 \).

\{c_{ji}\}_{qxp} \] = The condition entry matrix.

\{a_{ki}\}_{rxp} \] = the action entry matrix.

\[ a_k \] 
A limited action (action statement)

\[ A_k \] 
An extended action (action statement)

\[ W_k \] 
The set associated with the \( k \)th action.

\[ W_k = \{x\} \] for limited actions.

\[ W_k = \{w_{kl}, \ldots, w_{kn_k}\} \] for extended actions \( n_k > 2 \).

\[ \delta^i \] 
The \( i \)th column of the action entry matrix.

\[ \delta^i_k = a_{ki} \] 
The \( k \)th entry in the \( \delta^i \) column.

Finally, a general decision table can be expressed in terms of the matrix definition given above, as follows:

\[
\begin{bmatrix}
\begin{array}{c|cccc}
\sigma_1 & c_{11} & \cdots & c_{1i} & \cdots & c_{1p} \\
\vdots & & & & & \\
\sigma_j & c_{jl} & \cdots & c_{ji} & \cdots & c_{jp} \\
\vdots & & & & & \\
\sigma_q & c_{q1} & \cdots & c_{qi} & \cdots & c_{qp} \\
\end{array}
\end{bmatrix}
= \begin{bmatrix}
\begin{array}{c|cccc}
\Sigma & \gamma^1 & \gamma^2 & \cdots & \gamma^p \\
\hline
\delta^1 & \delta^2 & \cdots & \delta^p \\
\end{array}
\end{bmatrix}
\begin{bmatrix}
\begin{array}{c|cccc}
\alpha_1 & a_{11} & \cdots & a_{1i} & \cdots & a_{1p} \\
\vdots & & & & & \\
\alpha_k & a_{kl} & \cdots & a_{ki} & \cdots & a_{kp} \\
\vdots & & & & & \\
\alpha_r & a_{rl} & a_{ri} & a_{rp} \\
\end{array}
\end{bmatrix}
\]
\[
\begin{bmatrix}
\sum & \rho' \\
\rho^3 & A
\end{bmatrix} = \begin{bmatrix}
\sum & C \\
A & A
\end{bmatrix}
\]

Where:

\[\sigma_j = \frac{c_j}{c_j}\]
\[\alpha_k = \frac{a_k}{A_k}\]
\[\sum_T = \{\sigma_1, \ldots, \sigma_q\}\]
\[\alpha_T = \{\alpha_1, \ldots, \alpha_r\}\]

II-2 CHARACTERISTICS OF DECISION TABLES.

The major characteristics of decision tables are ambiguity and completeness. Ambiguity in decision tables deals with the problems of Redundancy, Contradiction and Inconsistency. Completeness, on the other hand, deals with the number of rules included in a decision table. In order to define the characteristics we introduce some basic "rule-operations" in the first part of this section. The actual characteristics are first considered for decision tables with unrelated conditions and then extended to include decision tables with related and unrelated conditions.

II-2-1 BASIC RULE OPERATIONS.

Here we shall only introduce those "rule-operations" which are required in the definition of ambiguity and completeness. Other rule-
operations are defined wherever necessary.

**Definition II-2-1;** "Simple Rule Splitting" - Let $\gamma^1$ be an incompletely specified rule with $c_{ji}$ as a don't care entry and the associated set $V_j = \{V_{j1}, V_{j2}, \ldots, V_{jn_j}\}$. Then a simple rule splitting of $c_{ji}$ (with respect to $V_j$), is given by:

$$
\gamma^1 = \begin{bmatrix}
\begin{bmatrix}
c_{11} \\
c_{21} \\
\vdots \\
c_{qi}
\end{bmatrix}
& & &
\begin{bmatrix}
\begin{bmatrix}
c_{11} \\
c_{21} \\
\vdots \\
c_{qi}
\end{bmatrix}
& & &
\begin{bmatrix}
\begin{bmatrix}
c_{11} \\
c_{21} \\
\vdots \\
c_{qi}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
= \begin{bmatrix}
\begin{bmatrix}
c_{11} \\
c_{21} \\
\vdots \\
c_{qi}
\end{bmatrix}
& & &
\begin{bmatrix}
\begin{bmatrix}
c_{11} \\
c_{21} \\
\vdots \\
c_{qi}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
V_{j1} \\
V_{j2} \\
\vdots \\
V_{jn_j}
\end{bmatrix}
= \begin{bmatrix}
\begin{bmatrix}
c_{11} \\
c_{21} \\
\vdots \\
c_{qi}
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
V_{jp} \\
\vdots \\
V_{j1}
\end{bmatrix}

**Note:** The remaining entries $c_{ji}$ in $\gamma^1$ may or may not be don't care entries.

**Definition II-2-2;** "Complete Rule Splitting" - A complete rule splitting is defined as a multiple simple rule splitting of all the don't care entries of an incompletely specified rule $\gamma^1$. That is, from the definition of the decision table in II-1-1, $\gamma^1$ can be expressed by the set of completely specified rules $\{\omega^1\}$. If, for example, $c_{ji}$ and $c_{j+k,i}$ are the only don't care entries in some rule $\gamma^1$ with the associated sets $V_j$ and $V_{j+k}$ ($j+k < q$) respectively, we denote this transformation as follows:

$$
\gamma^1 \xrightarrow{*} \{\omega^1\} = \{\omega^{i1}, \omega^{i2}, \ldots, \omega^{i\nu}, \ldots, \omega^{i\nu+\lambda}\}$$
Where $\omega^{1\nu}$ is a completely specified rule and $\omega^{1\nu} \neq \omega^{1\mu}$ if $\nu \neq \mu$. Furthermore, note that $l_i$ is the number of completely specified rules corresponding to $\gamma^i$.

Example:

Let $\gamma^1 = \begin{bmatrix} Y \\ 2 & - \\ 3 & - \\ 4 & \text{RED} \end{bmatrix}$

where $\{V_2\} = \{Y, N\}$
$\{V_3\} = \{100, 110, 120\}$

then simple rule splitting with respect of $c_{21}$ is given by:

$\gamma^1 \xrightarrow{V_2} \begin{bmatrix} Y \\ Y \\ \text{RED} \end{bmatrix} ; \begin{bmatrix} Y \\ N \\ \text{RED} \end{bmatrix}$

and a complete rule splitting is given by:

$\gamma^1 \xrightarrow{V_2, V_3} \begin{bmatrix} Y \\ Y \\ 100 \\ \text{RED} \\ 110 \\ \text{RED} \\ 120 \\ \text{RED} \end{bmatrix} ; \begin{bmatrix} Y \\ N \\ 100 \\ \text{RED} \\ 110 \\ \text{RED} \\ 120 \\ \text{RED} \end{bmatrix} ; \begin{bmatrix} Y \\ N \end{bmatrix}$

Note: In splitting a rule in an actual decision table, we have to assign the original action column to every one of the completely specified rules associated with the split rule. Thus:

$\gamma^1 \xrightarrow{\delta^1} \begin{bmatrix} \omega^{11} \\ \delta^1 \end{bmatrix} ; \begin{bmatrix} \omega^{12} \\ \delta^1 \end{bmatrix} ; \cdots ; \begin{bmatrix} \omega^{11} \\ \delta^1 \end{bmatrix}$
We shall consider now the decision table characteristics separately for independent and dependent conditions.

**II-2-2 CHARACTERISTICS OF DECISION TABLES WITH UNRELATED CONDITIONS**

*Definition II-2-4:* Two conditions (or condition statements) are called independent if and only if the variables in the respective condition stubs are logically independent, otherwise they are called dependent, or related conditions.

Using the definition II-2-4 we can define redundancy and contradiction in the case of limited condition entry decision tables (LCE) with independent conditions as follows:

*Definition II-2-5:* If two (or more) rules \( Y^i \) and \( Y^k \) have identical actions \( \delta^i = \delta^k \) respectively, and do not include at least one pair of \( Y, N \) in any of the \( \rho^P \) rows \( (p = 1, 2, \ldots, q) \), then redundancy exists in the LCE decision table.

*Definition II-2-6:* If two (or more) rules \( Y^i \) and \( Y^k \) have different actions \( \delta^i \neq \delta^k \) respectively, and do not include at least one pair of \( Y, N \) in any of the \( \rho^P \) rows \( (p = 1, 2, \ldots, q) \), then there is contradiction in the LCE decision table.

To include decision tables with mixed condition entries, these last two definitions can be generalized as follows:
**Definition II-2-3:** "Rule Intersection" - Let $\gamma^i$ and $\gamma^k$ be represented by the following sets of completely specified rules:

$$\gamma^i = \omega^i = \{\omega_1^i, \omega_2^i, \ldots, \omega_{1i}^i\}$$

$$\gamma^k = \omega^k = \{\omega_1^k, \omega_2^k, \ldots, \omega_{1k}^k\}$$

then the intersection of $\gamma^i$ and $\gamma^k$ is given by:

$$\gamma^i \cap \gamma^k = \omega^i \cap \omega^k = \{\Omega_{1k}^i, \Omega_{2k}^i, \ldots, \Omega_{1k}^k\} = \Omega_{ik}$$

where $\Omega_{ik} \subseteq \omega^i$ and $\Omega_{ik} \subseteq \omega^k$ (in the usual meaning of sub-sets).

Also note that the intersection is defined only between two rules of the same order (namely, with the same number of entries) and usually used for rules taken from the same table.

Furthermore, $\gamma^i \cap \gamma^k = \gamma^k \cap \gamma^i = \Omega_{ik} = \Omega_{ki}$ and $l_{ik} = l_{ki}$ represents the number of completely specified rules in the set $\{\Omega_{ik}\}$.

**Example:**

Let $\gamma^i = \begin{bmatrix} Y \\ - \\ - \end{bmatrix}$ and $\gamma^k = \begin{bmatrix} - \\ Y \\ - \end{bmatrix}$ where $\{V_3\} = \{\text{RED, BLUE, PINK}\}$

then $\gamma^i \cap \gamma^k = \left\{ \begin{bmatrix} Y \\ \text{RED} \end{bmatrix}; \begin{bmatrix} Y \\ \text{BLUE} \end{bmatrix}; \begin{bmatrix} Y \\ \text{PINK} \end{bmatrix} \right\} = \{\Omega_{ik}\}$
Definition II-2-7: - If two (or more) rules $\gamma^i$ and $\gamma^k$ have identical actions $\delta^i = \delta^k$ respectively, and do not include at least one pair of values $\gamma^i_p \neq \gamma^k_p$ in any of the rows $\rho^p$ ($p = 1,2,\ldots ,q$) there is redundancy in the MCE decision table.

Definition II-2-8: - If two (or more) rules $\gamma^i$ and $\gamma^k$ have different actions $\delta^i \neq \delta^k$ respectively, and do not include at least one pair of values $\gamma^i_p \neq \gamma^k_p$ in any of the rows $\rho^p$ ($p = 1,2,\ldots ,q$) then a contradiction exists in the MCE decision table.

Lemma II-2-1: Given any two rules $\gamma^i$ and $\gamma^k$ of a decision table, their intersection is empty ($\{\Omega_{ik}\} = \emptyset$) if and only if they contain at least one pair of different values $\gamma^i_p, \gamma^k_p \neq$ don't care in any of the rows $\rho^p$, ($p = 1,2,\ldots ,q$)

Proof: Let $\gamma^i \rightarrow \{\omega^i\} = \{\omega^{i1}, \omega^{i2},\ldots ,\omega^{i{l_i}}\}$
and $\gamma^k \rightarrow \{\omega^k\} = \{\omega^{k1}, \omega^{k2},\ldots ,\omega^{k{l_k}}\}$

a. First, let's prove that: If there is at least one pair $\gamma^i_p \neq \gamma^k_p$, ($p = 1,2,\ldots ,q$) then $\gamma^i \cap \gamma^k = \{\Omega_{ik}\} = \emptyset$.

Since, by assumption $\gamma^i_p \neq \gamma^k_p$ (and they are not don't care) then, $\omega^{iv} \neq \omega^{k\mu}$ ($v = 1,2,\ldots ,l_i$, $\mu = 1,2,\ldots ,l_k$) and therefore, $\{\Omega_{ik}\} = \emptyset$.

b. If $\{\Omega_{ik}\} = \emptyset$ then there is at least one $\gamma^i_p \neq \gamma^k_p$ ($p = 1,2,\ldots ,q$). Here, for simplicity, we shall start from completely specified rules:
i. For completely specified rules, $\omega^j = \omega^{j1} = \gamma^j$, $j = 1, \ldots, k$ thus if $\omega^i \cap \omega^k = \Omega_{ik} = \emptyset$ then $\omega^{i1} \neq \omega^{k1}$ and therefore, $\gamma^i \neq \gamma^k$ which means there is at least one pair of values $\gamma_p^i \neq \gamma_p^k$, $p = 1, 2, \ldots, q$.

ii. For incompletely specified rules, assuming $\{\Omega_{ik}\} = \emptyset$ we have this for every $p = 1, 2, \ldots, l_1, l_2 \ldots, l_i$:

$\omega^i \cap \{\omega^{kl}, \omega^{kl2}, \ldots, \omega^{klk}\} = \emptyset$ thus, $\omega^i \cap \gamma^k = \emptyset$ which means there is at least one pair of values $\omega^i_p \neq \omega^k_p$, $p = 1, 2, \ldots, q$ such that $\gamma^k_p \neq \gamma^i_p$ don't care. This implies the existence of at least one pair of values $\gamma^i_p \neq \gamma^k_p$ ($\gamma^i_p$, $\gamma^k_p$ don't care).

Theorem II-2-1: Given a decision table with unrelated conditions, the table is ambiguous if and only if there is at least one set $\{\Omega_{ik}\} \neq \emptyset$ for $i, k = 1, 2, \ldots, p$. Furthermore, if ambiguity exists then:

i. If $\delta^i = \delta^k$ the table includes redundancy.

ii. If $\delta^i \neq \delta^k$ the table includes contradiction.

Proof:

a. If there is at least one set $\{\Omega_{ik}\} \neq \emptyset$, $i, k = 1, 2, \ldots, p$, then by Lemma II-2-1, there can be no pair of values $\gamma^i_p \neq \gamma^k_p$ in any of the rows $\rho^p$, $p = 1, 2, \ldots, q$. Therefore:

i. By definition II-2-7 if $\delta^i = \delta^k$ the table includes redundancy.

ii. By definition II-2-7 if $\delta^i \neq \delta^k$ the table includes contradiction.
This concludes the first part of the proof.

b. If the table includes ambiguity, then, by definition II-2-7 and II-2-8, the table contains at least one pair of rules $\gamma^i$ and $\gamma^k$ for $i, k = 1, 2, \ldots, p$ such that they do not include any pairs of different values $\gamma^i_p \neq \gamma^k_p$ for $p = 1, 2, \ldots, q$. Thus by Lemma II-2-1 there is at least one set $\{\Omega_{ik}\} \neq \emptyset$ for $i, k = 1, 2, \ldots, p$.

This concludes the proof.

In a decision table where all the conditions are independent only data dependent inconsistency can exist.

**Definition II-2-9;** A rule $\gamma^i$ is said to be data dependent inconsistency if it includes $c_{ij} \notin V_j$ or if the combination of its entries cannot be satisfied by real data. This is a semantics problem and is beyond the scope of this document.

**Example:**

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE-OF-JACKET</td>
<td>42 42 42 82 ELSE</td>
</tr>
<tr>
<td>SIZE-OF-PANTS</td>
<td>40 42 48 46</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
</tr>
<tr>
<td>ORDER</td>
<td>x x x x</td>
</tr>
<tr>
<td>DE NOT ORDER</td>
<td></td>
</tr>
</tbody>
</table>

$$V_1 = \{40, 42, 44\}$$

**Fig. II-2-1**
In Fig. II-2-1 the third rule is data inconsistent of the second type because the combination of its entries cannot be satisfied by real data. Rule number 4 is also data inconsistent because \( \frac{82}{7} \notin V_1 \).

Note that this type of inconsistency cannot be identified by any mechanical procedure.

In decision tables with related conditions, logical inconsistency may also exist. This type of inconsistency is presented in section II-2-3.

**Definition II-2-10:** The Else Rule — Wherever included, represents all the possible rules not explicitly stated in the condition entry matrix.

**Definition II-2-11:** A decision table with independent conditions is said to be complete whenever:

1. It includes the else rule (in this case, the table may include ambiguities)

or

2. It does not include any ambiguity and,

\[
\sum_{i=1}^{p} l_i = \pi \sum_{j=1}^{q} n_j
\]

where \( n_j \) is the number of elements in the set \( V_j \).

Note that in the case of LCE decision tables, \( n_j = 2 \) and

\[
\sum_{i=1}^{p} l_i = 2^q
\]
II-2-3 CHARACTERISTICS OF DECISION TABLES WITH RELATED AND UNRELATED CONDITIONS.

The above definitions were developed for decision tables with unrelated conditions. Actually, they can also be applied to decision tables with related conditions, but, in this case, they become too restrictive for practical use. At this point, we shall introduce an example to indicate the restrictive nature of these definitions and then give a modified definition of ambiguity.

Example II-2-2;

In Electrical Engineering, it could happen that the following decision table with related conditions has to be used:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOLTAGE &lt; 110</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VOLTAGE = 110</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VOLTAGE &gt; 110</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>ACTIONS GO TO</td>
<td></td>
<td>LOW</td>
<td>LOW</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

Fig. II-2-2:

Decision table with related conditions.

Obviously, the conditions in the example are related and the rules are well designed and do not leave a place for ambiguity of
any kind. On the other hand, applying the rule for ambiguity and
completeness just defined, we can easily determine that:
1. Rule number 1 and rule number 2 are redundant.
2. Rule number 2 and rule number 3 are contradictory.
3. Splitting the rules completely we get $1_1 = 12 \neq 2^3 = 8$
4. Every rule includes inconsistency. For example:

$$
\begin{bmatrix}
Y \\
- \\
- \\
- \\
\end{bmatrix} 
\rightarrow 
\begin{bmatrix}
Y \\
Y \\
Y \\
Y \\
\end{bmatrix} ; 
\begin{bmatrix}
Y \\
Y \\
Y \\
N \\
\end{bmatrix} ; 
\begin{bmatrix}
Y \\
N \\
N \\
N \\
\end{bmatrix}
$$

The first three completely specified rules are inconsistent.

In this case the don't care entries have a different
meaning: they specify that the condition entry with a don't care
is irrelevant to the decision if the particular rule holds, and
therefore, in making the decision, that condition should not be
tested.

Actually, we could have
designed a decision table
that satisfies all the above
definitions performing the
same logical function as the
one in the given example,
Fig. II-2-3:

<table>
<thead>
<tr>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITIONS</td>
</tr>
<tr>
<td>VOLTAGE &lt; 110</td>
</tr>
<tr>
<td>VOLTAGE = 110</td>
</tr>
<tr>
<td>VOLTAGE &gt; 110</td>
</tr>
<tr>
<td>ACTIONS</td>
</tr>
<tr>
<td>GO TO</td>
</tr>
</tbody>
</table>

Fig. II-2-3
Decision table with related conditions
Note that, in this case, we must include the else rule for completeness (which will never take place in the example because it contains all the inconsistent rules which are logically impossible). Furthermore, in converting that table to a program, we have to make three tests in order to find the rule, which is an unnecessary increase in program running time. Actually to save program running time the designer may introduce in the decision tables with related conditions, irrelevant don't care entries, as in the above Example.

**Definition II-2-12:** - Given a mixed condition entry decision table with two (or more) rules $\gamma^i$ and $\gamma^k$ that have identical actions $\delta^i = \delta^k$, then:

i) Possible redundancy exists if $\gamma^i$ and $\gamma^k$ do not contain at least one pair $\gamma^i_p, \gamma^k_p$ of different values in any row $\rho^p$, $p = 1, 2, \ldots, q$ but have at least one pair $\gamma^i_p$, -- or --, $\gamma^k_p$.

ii) If they contain at least one pair $\gamma^i_p, \gamma^k_p$ of different values the rules are different and not redundant.

iii) If neither i) nor ii) holds, the rules are identical and therefore trivially redundant.

**Definition II-2-13:** - Given a mixed condition entry decision table with two or more rules $\gamma^i$ and $\gamma^k$ that have different actions $\delta^i \neq \delta^k$ then:

1. Possible contradiction exists if $\gamma^i$ and $\gamma^k$ do not contain at least one pair $\gamma^i_p, \gamma^k_p$ of different values in any row $\rho^p$, 
2. If the number of elements in the set \( \{ \omega \} = \bigcup_{i=1}^{p} \Omega_i \) is equal to \( q \)
\[ \prod_{j=1}^{q} n_j. \]

**Note:** If the table includes extended condition entries, inconsistent data may give unpredicted errors when running a program embodying such a table. (Depending on the way the table was translated into a computer program) therefore, the inclusion of the else rule is recommended whenever possible.
p = 1, 2, ..., q but have at least one pair $\gamma_i^p$ - or - $\gamma_k^p$.

ii) If the table contains at least one pair $\gamma_i^p$, $\gamma_k^p$ of different values the rules are different and do not contain contradictions.

iii) If neither i) nor ii) holds, the rules are identical and therefore trivially contradictory.

As stated before, in decision tables with related conditions outside data dependent inconsistency (see definition II-2-9), logical inconsistency may also exist.

Definition II-2-14: - In decision tables with related conditions, a rule $\gamma^i$ is said to include logical contradiction, if it can technically exist but implies logical contradiction.

Example II-2-3:

In the table shown in Fig. II-2-4 the first and last rules are technically possible, but obviously, logically impossible. In this case, the conditions are clearly logically related or dependent.

![Table](https://via.placeholder.com/150)

![Figure II-2-4](https://via.placeholder.com/150)

Note that even this trivial type of logical inconsistency cannot be, in general, identified by a mechanical procedure.

Definition II-2-15: - A decision table with related and unrelated conditions is said to be complete whenever:

1. It includes the Else Rule (even if it includes real ambiguities).
CHAPTER III. MANIPULATION OF AND RELATIONS BETWEEN DECISION TABLES

The purpose of this chapter is to consider some basic procedures for obtaining simpler decision tables from a given table.

In the first part of this chapter we consider the basic procedures for splitting decision tables. After defining the different linkages that can exist between decision tables, the decision table is split with respect to its rules, or conditions, or by branching. Furthermore, it is indicated that the merging of decision tables can be done only as each specific case dictates.

In the second part of this chapter, isomorphism and equivalence of decision tables is defined and then simplification of decision tables is done by minimization. The minimization techniques developed in this chapter are accomplished along three lines: rule minimization; condition stub minimization and action stub minimization.

III-1 SPLITTING AND MERGING OF DECISION TABLES.

In the previous section we presented the definition and the characteristics of a single decision table. In the present section we consider sets of a single decision table that are functionally interconnected. First we shall define a few procedures for splitting
a (large) decision table into a set of smaller decision tables such that the resulting set of interconnected decision tables will have the same function as the original one. Finally, we shall deal with the problem involved in merging sets of interconnected decision tables.

**III-1-1 SPLITTING OF DECISION TABLES.**

Splitting of decision tables can be obtained along two lines: splitting by rules and/or splitting by conditions. When splitting by rules the result is two or more decision tables with the same number of conditions, each containing else rules. On the other hand, splitting by conditions results in two or more decision tables with a reduced number of rules and conditions. It should be noted that in both cases, an extra action must be used to allow linkage between the resulting tables.

The linkage between decision tables can have one of the following three forms:

1. **Simple Linkage:** A decision table $T_1$ is said to be linked to a decision table $T_2$ by a "Simple Linkage" if it contains in its action stub a statement which transfers the control to $T_2$ upon satisfaction of one or more of its rules. A simple linkage may exist between a single decision table and a set of decision tables.
2. **Concatenated linkage**: - A decision table $T_1$ is said to be concatenated to decision table $T_2$ if it contains in its action stub a statement which transfers control to $T_2$ only if none of its explicit rules is satisfied. This implies that $T_1$ includes the Else rule.

3. **Reference linkage**: - A decision table $T_1$ is said to be linked by a "Reference Linkage" or simply, to refer to $T_2$ if it contains in its action stub a statement which transfers temporary control to $T_2$, such that upon completion of $T_2$ the control is transferred back to $T_1$ to the next action in sequence (if any). This implies that $T_2$ may not be linked to any other table by type 1 or type 2 linkes. That type of link allows for a single decision table to refer to a set of decision tables.

The third type of linkage is presented here only for the sake of completeness. It will be utilized only in later chapters. The usage of this type of linkage implies certain restriction which will be discussed later.

**Example III-1-1:**

**Simple Linkage** - In this example we shall use the statement "GO TO" to transfer control from one decision table to another. As shown below, a decision table can be linked by a "Simple Linkage" to one or more decision tables.
### Example III-1-3: Concatenation Linkage

**TABLE T₁**

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td></td>
<td></td>
<td></td>
<td>ELSE</td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_q )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Actions**

| \( a_1 \) | | | | |
| \( \vdots \) | | | | |
| \( a_r \) | | | | |

**GO TO**

T₂

**TABLE T₂**

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_q )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Actions**

| \( a_1 \) | | | | |
| \( \vdots \) | | | | |
| \( a_r \) | | | | |

**TABLE T₃**

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_q )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Actions**

| \( a_1 \) | | | | |
| \( \vdots \) | | | | |
| \( a_r \) | | | | |

**Fig III-1-2: Concatenation Linkage**

---

**TABLE T₁**

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td></td>
<td></td>
<td></td>
<td>ELSE</td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_q )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Actions**

| \( a_1 \) | | | | |
| \( \vdots \) | | | | |
| \( a_r \) | | | | |

**GO TO T₂**

- - - - - X

**TABLE T₂**

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_q )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Actions**

| \( a_1 \) | | | | |
| \( \vdots \) | | | | |
| \( a_r \) | | | | |

**TABLE T₃**

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sigma_q )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Actions**

| \( a_1 \) | | | | |
| \( \vdots \) | | | | |
| \( a_r \) | | | | |

**Fig III-1-1: Simple Linkage**
Example III-1-3:

Reference Linkage - In this example, we shall use the statement "PERFORM" for "REFERENCE LINKAGE".

<table>
<thead>
<tr>
<th>TABLE T₁</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITIONS</td>
<td>1</td>
</tr>
<tr>
<td>S₁</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Sₖ</td>
<td></td>
</tr>
<tr>
<td>ACTIONS</td>
<td>a₁</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>PERFORM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. III-1-3: Reference Linkage

The splitting of a decision table by its rules is equivalent to breaking a given decision table into a set of concatenated decision tables. The actual procedure, described below, breaks the set of rules \( \{ \gamma^1, \gamma^2, \ldots, \gamma^{p+1} \} \) into two or more concatenated sets
\{γ^1, γ^2, ..., γ^i, γ^{E_1} \} ... \{γ^{i+1}, ..., γ^{P+1}\} of rules, where the additional rules denoted by \(γ^{E_1}, γ^{E_2}, \ldots\) serve as concatenation links. A decision table is usually split by rules whenever the number of rules included in a single table are too large for practical usage.

**SPLITTING BY RULES:**

Given a decision table \(T_1\) with the set \(\{γ^1, γ^2, ..., γ^P\}\) of rules. To split this table into two decision tables \(T_{11}\) and \(T_{12}\) after the \(i^{th}\) rule and without altering the logic, the following rules should be applied:

1. Reordering of rules and/or conditions, before and/or after the splitting is allowed, as long as the logic remains unchanged.

2. Conditions and/or actions which become inapplicable after the splitting, may be excluded.

3. Every decision table (except the last one) contains an additional rule which is an else rule, and an additional action for concatenation linkage purpose (in the case of splitting into two, only the first decision table contains the linkage).

The resulting decision tables after the splitting are shown below.

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>1</th>
<th>...</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>(σ_1)</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td></td>
<td>..</td>
<td></td>
<td>..</td>
</tr>
<tr>
<td>(σ_2)</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td></td>
<td>..</td>
<td></td>
<td>..</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_1)</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td></td>
<td>..</td>
<td></td>
<td>..</td>
</tr>
<tr>
<td>(a_r)</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td></td>
<td>..</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. III-1-4
Splitting into more than two decision tables is done in the same way, keeping the same rules.

Note that any decision table can be split by rules up to $p$ concatenated decision tables, each containing a single rule and an else rule for linkage (except the last one, which contains a single rule plus the original else rule if it existed).

**Example III-1-4:** "Splitting by Rule"

In this example we shall split a given decision table into two along the 16\textsuperscript{th} rule.

It should be noted that in the process of splitting, we excluded the second condition $\sigma_2$ and the second action $a_2$ from $T_{11}$, since they originally had only don't care entries in their condition entry and action entry matrices up to number 16. This, obviously, did not change the logic.
TABLE T₇

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>16</th>
<th>17</th>
<th>...</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ₁</td>
<td>c₁₁</td>
<td>c₁₂</td>
<td></td>
<td></td>
<td></td>
<td>c₁₁⁷</td>
<td></td>
</tr>
<tr>
<td>σ₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c₂¹⁷</td>
<td></td>
</tr>
<tr>
<td>σ₃</td>
<td>c₃₁</td>
<td>c₃₂</td>
<td></td>
<td></td>
<td></td>
<td>c₃₁⁶</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>σᵣ</td>
<td>cᵣ₁</td>
<td>cᵣ₂</td>
<td></td>
<td></td>
<td></td>
<td>cᵣ₁⁶</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
<th>a₁</th>
<th>a₁₁</th>
<th>a₁₂</th>
<th>...</th>
<th>a₁₁⁶</th>
<th>a₁₁⁷</th>
<th>...</th>
<th>a₁ᵖ</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a₂ᵖ</td>
</tr>
<tr>
<td>a₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a₂ᵖ</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aᵣ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>aᵣᵖ</td>
</tr>
</tbody>
</table>

Splitting along the 16th rule we get:

TABLE T₁₁

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ₁</td>
<td>c₁₁</td>
<td>c₁₂</td>
<td>...</td>
<td>c₁₁⁶</td>
<td>E</td>
</tr>
<tr>
<td>σ₃</td>
<td>c₃₁</td>
<td>c₃₂</td>
<td>...</td>
<td>c₃₁⁶</td>
<td>L</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>σᵣ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
<th>a₁</th>
<th>a₁₁</th>
<th>a₁₂</th>
<th>...</th>
<th>a₁₁⁶</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a₃</td>
<td></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>
</tr>
<tr>
<td>aᵣ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GO TO T₁₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

TABLE T₁₂

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>17</th>
<th>...</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ₁</td>
<td>c₁₁⁷</td>
<td></td>
<td></td>
</tr>
<tr>
<td>σ₂</td>
<td>c₂¹⁷</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>σᵣ</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
<th>a₁</th>
<th>a₁₁⁷</th>
<th>...</th>
<th>a₁ᵖ</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₂</td>
<td></td>
<td>a₂¹⁷</td>
<td></td>
<td>a₂ᵖ</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aᵣ</td>
<td></td>
<td>aᵣ₁⁷</td>
<td></td>
<td>aᵣᵖ</td>
</tr>
</tbody>
</table>

Fig. III-1-6: Splitting by Rules.

Splitting of a decision table by rules is definitely not recommended, unless it is absolutely necessary to do so. Splitting a
decision table by rules, into a set of \( K \) concatenated decision tables, multiplies the number of conditions that have to be tested by a factor of \( Q \) where \( 1 < Q < K \).

The splitting of a decision table by its conditions is also equivalent to breaking a given decision table into a set of concatenated decision tables. The actual splitting can be done only if the condition entry matrix \( C \) can be partitioned into blocks, such that all nondiagonal blocks (cells) contain only don't care entries as in Fig. III-1-7.

A partitioned condition entry matrix with don't care entries in nondiagonal blocks.
SPLITTING BY CONDITIONS

Given a decision table with a partitioned condition entry matrix, in which all the non-diagonal blocks contain only don't care entries, and with K diagonal blocks:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>1_l</th>
<th>...</th>
<th>1_k</th>
<th>...</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_1 )</td>
<td>( c_{11} )</td>
<td>( c_{12} )</td>
<td>...</td>
<td>( c_{1l} )</td>
<td>...</td>
<td>( c_{1k} )</td>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \sigma_{j1} )</td>
<td>( c_{j1} )</td>
<td>( c_{j2} )</td>
<td>...</td>
<td>( c_{jl} )</td>
<td>...</td>
<td>( c_{jk} )</td>
<td>...</td>
<td>( c_{jp} )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \sigma_{jk} )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( \sigma_q )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
<td>...</td>
<td>( \vdots )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
<th>( a_{11} )</th>
<th>( a_{12} )</th>
<th>...</th>
<th>( a_{1p} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_2 )</td>
<td>( a_{21} )</td>
<td>( a_{22} )</td>
<td>...</td>
<td>( a_{2p} )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( a_r )</td>
<td>( a_{r1} )</td>
<td>( a_{r2} )</td>
<td>...</td>
<td>( a_{rp} )</td>
</tr>
</tbody>
</table>

Fig. III-1-8.

The result of splitting the above given table by conditions will be K concatenated decision tables. The set of conditions \( \sigma_1, \sigma_2, \ldots, \sigma_q \) is broken into K sets of \( \sigma_1, \ldots, \sigma_{j1}, \sigma_{j1+1}, \ldots, \sigma_{jk}, \ldots, \sigma_{jk}, \ldots, \sigma_q \). The decision table will be represented by a set of concatenated decision tables; every one containing one set of actions respectively, and all the actions. The linkage is
done by concatenation.

### TABLE T_{11}

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>1_{1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_{1}</td>
<td>c_{11}</td>
<td>c_{12}</td>
<td>...</td>
<td>c_{i1}</td>
<td>ELSE</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a_{j1}</td>
<td>c_{j1}</td>
<td>c_{j2}</td>
<td>...</td>
<td>c_{j1}</td>
<td>ELSE</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_{1}</td>
<td>a_{11}</td>
<td>a_{12}</td>
<td>...</td>
<td>a_{11}</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a_{r}</td>
<td>a_{r1}</td>
<td>a_{r2}</td>
<td>...</td>
<td>a_{r1}</td>
<td></td>
</tr>
<tr>
<td>GO TO T_{12}</td>
<td>-</td>
<td>-</td>
<td></td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

### TABLE T_{12}

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>1_{2}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ELSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ELSE</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>GO TO T_{13}</td>
<td>-</td>
<td>-</td>
<td></td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

### TABLE T_{13}

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
<th>1</th>
<th>...</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_{j1}</td>
<td>c_{j1}</td>
<td></td>
<td>...</td>
<td>c_{j1}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>a_{o1}</td>
<td>c_{o1}</td>
<td></td>
<td></td>
<td>c_{o1}</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_{1}</td>
<td></td>
<td>...</td>
<td>...</td>
<td>a_{1p}</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>a_{2p}</td>
</tr>
<tr>
<td>a_{r}</td>
<td></td>
<td>...</td>
<td>...</td>
<td>a_{rp}</td>
</tr>
</tbody>
</table>

Fig. III-1-9: Splitting by Conditions
Note that the splitting procedure can be executed only in special cases, e.g., whenever the problem could have been originally expressed by a set of concatenated decision tables. Whenever re-arranging rules does not alter the logic, it is possible to start from a decision table in which the condition entry matrix does not allow splitting by conditions. Then, after reordering the rules and the conditions, to generate a decision table in which splitting by conditions is possible. (As we shall see in section II-5 it is possible to do the same by changing the set of rules by merging rules, or as we have seen before, by splitting rules).

Another case in which splitting by conditions is possible, is of an extended condition entry decision table which is translated into a limited entry decision table, e.g.,

**Example III-1-5:**

Given the extended condition entry decision table:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF A =</td>
<td>B C D</td>
</tr>
<tr>
<td>IF P =</td>
<td>Q R T</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>: : : :</td>
</tr>
<tr>
<td>a₂</td>
<td>: : : :</td>
</tr>
</tbody>
</table>

Fig. III-1-10

The equivalent limited entry decision table is:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = B</td>
<td>Y - -</td>
</tr>
<tr>
<td>P = Q</td>
<td>Y - -</td>
</tr>
<tr>
<td>A = C</td>
<td>- Y -</td>
</tr>
<tr>
<td>P = R</td>
<td>- Y -</td>
</tr>
<tr>
<td>A = D</td>
<td>- - Y</td>
</tr>
<tr>
<td>P = T</td>
<td>- - Y</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>: : : :</td>
</tr>
<tr>
<td>a₂</td>
<td>: : : :</td>
</tr>
</tbody>
</table>

Fig. III-1-11
Note that splitting by rules and excluding inapplicable conditions would have resulted in the same set of concatenated decision tables.

Another type of decision table splitting by its conditions is also possible. To differentiate it from the previous one, we shall refer to it as "Splitting by Branching". In this case, any decision table may be split with respect to any one of its conditions, providing the order of the rules is immaterial.

**SPLITTING BY BRANCHING**

Splitting by branching can be done if and only if, the order of the rules in a decision table that has to be split is immaterial and reordering does not change the logic. Otherwise, splitting by branching can be done only with respect to the conditions $c_j$, in which the condition row entries $\rho_j$ are sorted by values from $V_j$ and do not contain don't care entries.

Assuming that reordering of rules is allowed, the splitting is done as follows:

If the splitting is done with respect to a limited condition entry the result will be three different tables. The first table contains only the condition which controls the splitting, transferring control to the second or the third decision table, depending on the outcome of this condition testing. In the second and the third tables, however, that condition is absent and also there are less
Note that the action $a_2$ was excluded in $T_{12}$ and the conditions $\sigma_3$ was excluded in $T_{13}$. Every rule which had a don't care entry in its first condition entry row $\rho^1$, appears in $T_{12}$ and $T_{13}$:

If the splitting is done with respect to an extended condition entry the result will be $n_j + 1$ decision tables, otherwise, everything else is done by complete analogy.

Note that this process of splitting by branching can be continued with each subtable until the resulting set of subtables contains subtables with only one condition or one rule (and one else rule if the else rule was included in the original table). Tests for ambiguity and completeness in the final stage of splitting become trivial.

Splitting by branching can also be done with respect to more
rules in each. If the else rule was included in the original table, the second and the third decision tables will contain it. Conditions and actions which become inapplicable in the process of splitting may be excluded.

Assuming that the splitting is done with respect to the first condition then the actual splitting is done as illustrated in Fig. III-1-13 below.

<table>
<thead>
<tr>
<th>TABLE T₁</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITIONS</td>
<td>1</td>
</tr>
<tr>
<td>σ₁</td>
<td>Y</td>
</tr>
<tr>
<td>σ₂</td>
<td>c₂₁</td>
</tr>
<tr>
<td>c₂₃₁</td>
<td>-</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>-</td>
</tr>
<tr>
<td>a₂</td>
<td>-</td>
</tr>
<tr>
<td>a₃</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
</tr>
<tr>
<td>aₚ</td>
<td>-</td>
</tr>
</tbody>
</table>

Fig. III-1-12
than one condition as illustrated in the following example.

Example III-1-6: Splitting by branching with respect to two conditions

<table>
<thead>
<tr>
<th>TABLE T₁</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITIONS</td>
<td>1</td>
</tr>
<tr>
<td>c₁</td>
<td>Y</td>
</tr>
<tr>
<td>c₂</td>
<td>Y</td>
</tr>
<tr>
<td>c₃</td>
<td>c₃₁</td>
</tr>
<tr>
<td>c₄</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>aᵣ</td>
</tr>
</tbody>
</table>

Fig. III-1-14

After splitting with respect to c₁ and c₂ we get the following set of tables.

<table>
<thead>
<tr>
<th>TABLE T₁₁</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITIONS</td>
<td>2</td>
</tr>
<tr>
<td>c₁</td>
<td>Y</td>
</tr>
<tr>
<td>c₂</td>
<td>Y</td>
</tr>
<tr>
<td>GO TO</td>
<td>T₁₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T₁₂</th>
<th>RULE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON.</td>
<td>1</td>
</tr>
<tr>
<td>c₃</td>
<td>c₃₁</td>
</tr>
<tr>
<td>-</td>
<td>c₄₂</td>
</tr>
<tr>
<td>ACT.</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>.</td>
</tr>
<tr>
<td>...</td>
<td>.</td>
</tr>
<tr>
<td>aᵣ</td>
<td>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T₁₃</th>
<th>RULE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON.</td>
<td>2</td>
</tr>
<tr>
<td>c₃</td>
<td>-</td>
</tr>
<tr>
<td>c₄</td>
<td>c₄₃</td>
</tr>
<tr>
<td>ACT.</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>.</td>
</tr>
<tr>
<td>...</td>
<td>.</td>
</tr>
<tr>
<td>aᵣ</td>
<td>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T₁₄</th>
<th>RULE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON.</td>
<td>5</td>
</tr>
<tr>
<td>c₃</td>
<td>c₃₅</td>
</tr>
<tr>
<td>c₄</td>
<td>c₄₅</td>
</tr>
<tr>
<td>ACT.</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>.</td>
</tr>
<tr>
<td>...</td>
<td>.</td>
</tr>
<tr>
<td>aᵣ</td>
<td>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T₁₅</th>
<th>RULE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON.</td>
<td>7</td>
</tr>
<tr>
<td>c₃</td>
<td>-</td>
</tr>
<tr>
<td>c₄</td>
<td>c₄₇</td>
</tr>
<tr>
<td>ACT.</td>
<td></td>
</tr>
<tr>
<td>a₁</td>
<td>.</td>
</tr>
<tr>
<td>...</td>
<td>.</td>
</tr>
<tr>
<td>aᵣ</td>
<td>.</td>
</tr>
</tbody>
</table>

Fig. III-1-15
Even though the splitting procedures introduced in this section were presented separately, in practice they can be utilized in some combined way.

III-1-2 MERGING OF DECISION TABLES.

Merging of decision tables refers to the combination of two or more decision tables into one decision table. The question that must be ascertained first, is whether or not two decision tables (or more) can be merged without producing ambiguity and without changing the logical function of the original decision tables.

From this point of view, one can state that, given a set of decision tables, which was derived by using one or more of the splitting procedures, introduced in section III-1-1, they can be merged. The problem is that there is no mechanized procedure for testing whether the decision tables were derived by splitting a simple decision table or not. Therefore, even in this case, no generalized merging procedure can be given.

Now, with considering any two decision tables, it was found that even under severe restrictions there is no generalized procedure which will not produce, in general, ambiguities or other logical errors. The only statement that can be made is that merging of two decision tables (or more) can be considered only on an individual basis.
III-2 RELATIONS BETWEEN DECISION TABLES.

In this section we shall define the different relations possible between any two decision tables namely; isomorphism, equivalence, and coverage. As we shall see in the next section the equivalence relation is of special importance mainly in practical usage.

III-2-1 ISOMORPHISM AND EQUIVALENCE OF DECISION TABLES.

Definition III-2-1; Two decision tables are called isomorphic if:

1. They have the same set of conditions in the conditions stub, (not necessarily in the same order).
2. They have identical action stubs.
3. They have the same set of rules with the same actions associated respectively.
4. And by reorganizing the rules and the conditions in one decision table, providing the logic remains unchanged, it can be transformed into the second.

The above definition of isomorphism defines an equivalence relation. The only problem is how to check whether reorganizing of the rules changes the logic of a table or not. If the conditions are unrelated, the procedure for checking isomorphism can be mechanized.
Checking two decision tables (with dependent conditions) for isomorphism can be accomplished manually. This is done by checking after every rule reorganization to see that the logic was not changed. Another way is simply to check if all possible combinations of data will result in the same actions in both tables.

**Definition III-2-2:** Two decision tables $T_1$ and $T_2$ are said to be simply equivalent if:
1. They have the same set of conditions.
2. They have identical action stubs.

And as well, we can replace one decision table by the other without changing the logical function of the total system.

We shall introduce at this point another rule operation which, together with the rule splitting operations that were introduced in II-2-1, will simply allow us to treat equivalent tables more systematically.

**Definition III-2-3:** "Rule Combination" - Let $\{\gamma_{n_1}, \gamma_{n_2}, \ldots, \gamma_{n_l}\}$ be a subset of rules in a decision table with identical actions $\delta_{n_u} = \delta_{n_v}$, $u, v = 1, 2, \ldots, j$. Let $V_j = \{V_{j1}, V_{j2}, \ldots, V_{jn}\}$ be the set of values associated with the $j$th condition. Then if:
1. The subset contains rules which are identical except for the $j$th entry; and
2. Every element $V_{ji} \in V_j$ appears at least in one of the rules ($n_i > n_j$).
This subset of rules can be combined into one rule in which the $j^\text{th}$ entry is a don't care entry.

**Example:** "Rule Combination"

$V_j = \{1, 2, 3, 4\}$

Now it becomes quite obvious how two decision tables can be simply equivalent. To illustrate this, we shall introduce the following example.

**Example III-2-1:** - "Simple equivalent tables"

Suppose the following table is given

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>E</td>
</tr>
<tr>
<td>$c_2$</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>L</td>
</tr>
<tr>
<td>$c_3$</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>ACTIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Fig. III-2-1
It is easy to see that we can combine rules 1 and 2 or rules 2 and 3 giving two different decision tables which are simply equivalent by definition II-4-2.

\[
\begin{array}{|c|c|c|c|}
\hline
\text{CONDITIONS} & \text{RULE NO.} & 1 & 2 & 3 \\
\hline
\text{c}_1 & Y & Y & E \\
\text{c}_2 & - & N & E \\
\text{c}_3 & Y & N & E \\
\hline
\text{ACTIONS} & & X & X & X \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{CONDITIONS} & \text{RULE NO.} & 1 & 2 & 3 \\
\hline
\text{c}_1 & Y & Y & E \\
\text{c}_2 & Y & N & E \\
\text{c}_3 & Y & - & E \\
\hline
\text{ACTIONS} & & X & X & X \\
\hline
\end{array}
\]

Fig. III-2-2

For checking if two decision tables with unrelated conditions are simply equivalent the following procedure can be used:

**Procedure III-2-1:** Given two decision tables \( T_1 \) and \( T_2 \).

**STEP 1:** Check if the conditions are the same

1. If they are different the tables are not simply equivalent, END.
2. GO to STEP 2.

**STEP 2:** Check if the actions stubs are identical

1. If not, the tables are not simply equivalent, END.
2. If yes GO to STEP 3

**STEP 3:** Transfer each table \( T_i \) (\( i = 1,2 \)) to a completely specified condition entry decision table \( T_i^c \). GO to STEP 4.

**STEP 4:** Check if \( T_1^c \) and \( T_2^c \) are isomorphic.

1. If Yes, the tables are simply equivalent, END.
2. If Not, the tables are not simply equivalent, END.
Checking for simple equivalence in the general case, when the decision tables may include related rules, may be considered only on an individual basis. The only mechanized procedure that can be applied in every case is simply generating all possible combinations of data and checking on both tables to see if the resulting actions are the same for every combination. But even in this case, since many combinations might be logically, or data inconsistent, the results of the test may not prove a thing.

**Definition III-2-4:** Two decision tables are called equivalent if, and only if, we can replace one table by the other without changing the logical function of the total system.

In order to introduce a few comprehensive examples of equivalent decision tables, we shall define a transformation of an extended condition entry into a set of limited condition entries.

**Definition III-2-5:** Let \( C_j \) be an extended condition associated with the set of values \( V_j = \{V_{j1}, V_{j2}, \ldots, V_{jn} \} \) then the transformation of \( C_j \) and the associated row \( \rho_j \) of condition entries into a set of limited conditions, and limited condition entries is done as follows:

1. The extended condition \( C_j \) is transformed into \( n_j \) limited conditions (obviously related).
2. Every don't care entry in \( c_j \) will result in a complete column of don't care entries after the transformation.
3. If as a result of the transformation, we get a condition \( c_p \)
associated with a row \( \rho^p \) full of don't care entries, it can be suppressed.

**Example:**

Given the extended condition \( A \)

\[
A = 1 \quad 2 \quad 3 \quad 4 \quad 2
\]

associated with the set \( V_j = \{1, 2, 3, 4, 5\} \)

then after the transformation we get

\[
1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6
\]

\[
A = 1 \quad \quad Y \quad - \quad - \quad - \quad -
\]

\[
A = 2 \quad - \quad Y \quad - \quad - \quad Y
\]

\[
A = 3 \quad - \quad - \quad Y \quad - \quad -
\]

\[
A = 4 \quad - \quad - \quad - \quad Y \quad -
\]

\[
A = 5 \quad - \quad - \quad - \quad - \quad -
\]

Obviously, the 5\(^{th}\) condition may be suppressed, column number 5 is the result of having a don't care entry in the original row. Columns 2 and 6 must both be included.

Attention should be paid to the fact that the number of columns after the transformation remains the same and the number of limited entry conditions is \( n_j \) or less. Therefore, this transformation does not affect the action entry matrix.

**Definition III-2-6:** Let \( A_k \) be an extended action associated with the set \( W_k = \{w_{k1}, w_{k2}, \ldots, w_{kn_k}\} \) then the transformation of \( A_k \) and the associated action entries row into a set of limited entry actions and limited action entries, is done as follows:
1. The extended action $A_k$ is transformed into $n_k$ limited actions.
2. Every blank entry in the action entries row results in a complete columns of blanks.

Example:

Given the extended action:

\[
\text{GO TO \hspace{0.5cm} L1 \hspace{0.5cm} L2 \hspace{0.5cm} L3 \hspace{0.5cm} - \hspace{0.5cm} L2 \hspace{0.5cm} - -}
\]

then after the transformation we get

\[
\text{GO TO \hspace{0.5cm} L1 \hspace{0.5cm} X}
\]

\[
\text{GO TO \hspace{0.5cm} L2 \hspace{0.5cm} X \hspace{0.5cm} X}
\]

\[
\text{GO TO \hspace{0.5cm} L3 \hspace{0.5cm} X}
\]

Here again the number of columns remains the same and the number of limited entry actions is exactly $n_k$. Therefore this transformation does not effect in any way the condition entry matrix.

In the following example we shall see two equivalent decision tables and using the transformations we just defined, we shall prove that they are equivalent. Unfortunately, a general procedure for testing for equivalence cannot be given. In general case, the equivalence or the difference between two decision tables may even depend on information not explcitely stated in the decision tables under consideration, and again, every case must be treated on an individual basis.

Example:

The following two decision tables are equivalent


<table>
<thead>
<tr>
<th>TABLE T₁</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONS</td>
<td>1</td>
</tr>
<tr>
<td>A =</td>
<td>1</td>
</tr>
<tr>
<td>c₂</td>
<td>Y</td>
</tr>
<tr>
<td>c₃</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO TO L₁ X</td>
</tr>
<tr>
<td>GO TO L₂ X X</td>
</tr>
<tr>
<td>GO TO L₃ X X</td>
</tr>
<tr>
<td>B ← A X</td>
</tr>
<tr>
<td>T ← A X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE T₂</th>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONS</td>
<td>1</td>
</tr>
<tr>
<td>A = 1</td>
<td>Y</td>
</tr>
<tr>
<td>A = 2</td>
<td>- Y</td>
</tr>
<tr>
<td>A = 3</td>
<td>-</td>
</tr>
<tr>
<td>A = 4</td>
<td>-</td>
</tr>
<tr>
<td>c₂</td>
<td>Y</td>
</tr>
<tr>
<td>c₃</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO TO L₁</td>
</tr>
<tr>
<td>B ← A</td>
</tr>
<tr>
<td>T ← A</td>
</tr>
</tbody>
</table>

**Fig. III-2-3**

In the above given example (which is relatively simple) we can deduce the equivalence of the tables simply by showing that by using the transformations just defined, we can transform every one of them to the table given below which is equivalent to T₁ and to T₂. (The equivalence of decision tables is an equivalence relation i.e. Reflexive symmetric and transitive).

In general, two equivalent decision tables may contain completely different conditions or actions which are equivalent in some particular type of

**Table T**

<table>
<thead>
<tr>
<th>ACTIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 1</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>E</td>
</tr>
<tr>
<td>A = 2</td>
<td>- Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>L</td>
</tr>
<tr>
<td>A = 3</td>
<td>-</td>
<td>- Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>S</td>
</tr>
<tr>
<td>A = 4</td>
<td>-</td>
<td>-</td>
<td>- Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>c₂</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>c₃</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO TO L₁ X</td>
</tr>
<tr>
<td>GO TO L₂ X X</td>
</tr>
<tr>
<td>GO TO L₃ X X</td>
</tr>
<tr>
<td>B ← A X X</td>
</tr>
<tr>
<td>T ← A X</td>
</tr>
</tbody>
</table>

**Fig. III-2-4**
application. If, for example, the variable ALPHA is redefined some-
were as Beta (in COBOL) then a condition such as "IF ALPHA > 0" is 
logically equivalent to the statement "IF BETA > 0". But it is 
very difficult to know just from the information given in the deci-
sion tables. Moreover, in ALGOL, for example, if a statement like 
"A ← B ← C" is executed previous to executing the same decision 
table, then a condition like "IF A > 0" is equivalent to the condi-
tion "IF B > 0". But this equivalence is not apparent and requires 
additional knowledge not included in the decision tables.

III-2-2 MINIMIZATION OF DECISION TABLES.

Minimization of decision tables can be done along three lines: 
Rule minimization, done by rule combination; conditions stub mini-
mization, done by transforming sets of limited entry conditions into 
extended entry conditions and action stub minimization, done by 
transformation of sets of limited entry actions into extended entry 
actions.

RULE MINIMIZATION: Here we try to minimize a given deci-
sion table by minimizing the number of rules included in the condi-
tion entry matrix by combining or merging sets of rules whenever 
possible.

A necessary condition for rule combination is identical 
actions associated with the rules we are trying to combine. There-
fore the first step in minimizing the table by rules is partitioning 
the set of rules into sub-tables with identical actions. If the
partitioning requires reorganizing rules then afterwards additional tests should be conducted to verify that the logic was not changed. If, by reorganizing the rules, the logic was changed and the original order of the rules is significant, then only partial partitioning is possible. In some cases some reorganization of rules is possible without altering the logic of the decision tables and should be done, if necessary, before partitioning. The goal, in every case, is to reach a partition with the smallest number of sub-sets possible.

Suppose the following decision table is given:

\[
\begin{array}{cccccccccc}
\Sigma & \gamma^1 & \gamma^2 & \gamma^3 & \gamma^4 & \gamma^5 & \gamma^6 & \gamma^7 & \gamma^8 & \gamma^9 & \gamma^{10} \\
\delta^1 & \delta^2 & \delta^3 & \delta^4 & \delta^5 & \delta^6 & \delta^7 & \delta^8 & \delta^9 & \delta^{10} \\
\end{array}
\]

where \( \delta^1 = \delta^2 = \delta^6 \)

\( \delta^3 = \delta^4 = \delta^5 = \delta^7 = \delta^8 \)

\( \delta^9 = \delta^{10} \)

then, if reorganization of rules is possible, the partitioning will give:

\( \{\gamma^1, \gamma^2, \gamma^6\} ; \{\gamma^3, \gamma^4, \gamma^5, \gamma^7, \gamma^8\} ; \{\gamma^9, \gamma^{10}\} \)

if the original order of the rules is significant, the partitioning will give:

\( \{\gamma^1, \gamma^2\} ; \{\gamma^3, \gamma^4, \gamma^5\} ; \{\gamma^6\} ; \{\gamma^7, \gamma^8\} ; \{\gamma^9, \gamma^{10}\} \)

if, for example, some reorganizing should take place without chan-
ging the logic, for example if shifting \( \gamma^6 \) to the 8\textsuperscript{th} position was possible, then the partioning will give:

\[
\{\gamma^1, \gamma^2\} ; \{\gamma^3, \gamma^4, \gamma^5, \gamma^7, \gamma^8\} ; \{\gamma^6\} ; \{\gamma^9, \gamma^{10}\}
\]

The second step in rule minimization is to combine within every subset, as many rules as possible. To do that we shall partition again every sub-set by condition entry rows. As a first step in partioning again, we have to find which are the possible condition entry rows that can be used. If the i\textsuperscript{th} sub-set includes \( N_i \) rules, then only the condition entry row associated with condition \( \sigma_j \) which, in turn, is associated with the set \( V_j = \{V_{j1}, V_{j2}, \ldots, V_{jn_j}\} \) where \( n_j < N_i \), should be considered. For simplicity, we shall call the number \( n_j \) associated with the set \( V_j \) the index of the condition \( \sigma_j \).

We start by trying to partition every sub-set with respect to the largest index \( n_j < N_i \). Every \( n_j \) rules or more, included in the same sub-set, will form a new sub-set in the secondary partition if the j\textsuperscript{th} condition entry row contains the complete set \( V_j = \{V_{j1}, V_{j2}, \ldots, V_{jn_j}\} \) and all the other condition row entries contain identical values, respectively. Such a sub-set is called COMPLETE. If the original order of the rules is significant, then the rules of that sub-set must appear in a continuous sequence.

Now every complete sub-set of rules may be combined into a single incompletely specified rule with respect to the j\textsuperscript{th} entry*.  

* See also the definition of rule combination; Definition III-2-3
Here again we must differ between two cases. If a sub-set can be partitioned into a set of complete sub-sets, then the secondary partition is called COMPLETE; otherwise, PARTIAL SECONDARY PARTITION.

A complete secondary partition with respect to any index \( n_j < N_1 \) will decrease the number of rules in the sub-set by a factor of \( n_j \).

That procedure of secondary partitioning and rule combination, can be continued with respect to the other conditions until no more secondary partitioning is possible. The following example will clarify the procedure of rule minimization.

**Example:**

Let the \( \gamma^1, \gamma^2, \ldots, \gamma^{16} \) be a sub-set of rules with identical actions as given below

\[
\{\gamma^1, \gamma^2, \ldots, \gamma^{16}\} = \begin{bmatrix}
    \text{a a a a} & \text{b b b b} & \text{c c c c} & \text{d d d d} \\
    \text{Y Y N N} & \text{Y Y N N} & \text{Y Y N N} & \text{Y Y N N} \\
    \text{Y N Y N} & \text{Y N Y N} & \text{Y N Y N} & \text{Y N Y N} \\
    \text{e e e e} & \text{e e e e} & \text{e e e e} & \text{e e e e}
\end{bmatrix}
\]

Where the first condition entry row is associated with the index \( n_1 = 4 \) (and \( V_1 = \{V_{11}, V_{12}, V_{13}, V_{14}\} = \{a, b, c, d\} \)) and the next two are limited condition entry rows associated with the index \( n_2 = n_3 = 2 \) (and \( V_2 = V_3 = \{Y, N\} \)).

If the original order of the rules is not significant, starting from the largest index \( n_1 = 4 \), we can derive the following complete secondary partition:
\{\gamma^1, \gamma^5, \gamma^9, \gamma^{13}\} ; \{\gamma^2, \gamma^6, \gamma^{10}, \gamma^{14}\} ; \{\gamma^3, \gamma^7, \gamma^{11}, \gamma^{15}\} ; \\
\{\gamma^4, \gamma^8, \gamma^{12}, \gamma^{16}\}.

and after rule combination with respect to the above secondary partition, the following set of rules will result.

\[
\{\gamma_1^1, \gamma_1^2, \gamma_1^3, \gamma_1^4\}^* = \begin{bmatrix}
Y & Y & N & N \\
Y & N & Y & N \\
e & e & e & e
\end{bmatrix}
\]

As can be easily seen, the number of rules is decreased by a factor \(n_1 = 4\) (from 16 to 4), because the secondary partition is complete.

Now we can go one step farther and try to find another partition with the largest index \(n_2\) possible, which is, in this case, \(n_2 = 2\), and the partition is:

\(\{\gamma_1^1, \gamma_1^2\} ; \{\gamma_1^3, \gamma_1^4\}\)

after rule combination we get:

\[
\{\gamma_2^1, \gamma_2^2\} = \begin{bmatrix}
- & Y \\
- & N \\
e & e
\end{bmatrix}
\]

In the final step we arrive at a single rule which is:

* The notation \(\gamma_n^i\) is used to denote the \(i^{\text{th}}\) rule after \(n\) rule combination took place
\{ \gamma_3^1 \} = \left[ \begin{array}{c} 1 \\
- \\
e \end{array} \right]

Note: The above example was so designed as to give, in every step, a complete secondary partition. In practical tables, this is not always the case. Sometimes it will be even difficult to decide which index to use for partitioning (if they are equal) as for example the following case:

\{ \gamma_1^1, \gamma_2^2, \gamma_3^3 \} = \left[ \begin{array}{ccc} y & y & n \\
N & Y & N \\
y & Y & Y \end{array} \right]

By using \( n_1 = 2 \) we get \{ \gamma_1^1 \} ; \{ \gamma_2^2, \gamma_3^3 \}, and by using \( n_2 = 2 \) we get \{ \gamma_1^1, \gamma_2^2 \} ; \{ \gamma_3^3 \}. As a result, whenever the secondary partitions are not complete, the above described procedure may not necessarily lead to the minimal number of rules, but will always decrease the number of rules. In such a case, the derivation of the minimal number of rules can be done by exhausting all possibilities, trying in every step, to find a complete secondary partition with respect to the maximal index.

**CONDITION STUB and ACTION STUB MINIMIZATION.**

The condition stub and action stub minimization are the opposite transformations of the transformations defined in III-2-5/6. By their very nature they are possible only in special cases. The usefulness of that kind of minimization is mainly for documentation
purposes and whenever a precompiler, interpreter or compiler, which accept ECE-EAE decision tables, is available. The procedure of minimizing the condition stub or the action stub (wherever possible) is simple and done step by step in the opposite order as defined in definitions III-2-5 and III-2-6 and for that reason will be omitted.

Note: that the nature of rule minimization is different from the condition stub and action stub minimization. In rule minimization we actually decrease the size of the decision table (not only physically) by reducing the number of rules and in terms of electronic data processing we decrease, in general, the compile time and run time. The condition stub and action stub minimization on the other hand decreases the size of the decision table only physically but usually increases compile or procompiler time due to the increase of complexity and does not save, in general, run time.
CHAPTER IV. CONVERSION METHODS

The methods used in converting decision tables to computer programs (either manual or automatic) fall into three main categories, depending upon the translation method. In this chapter we shall introduce the TREE METHOD, the MASK METHOD and the COMPUTED GO TO METHOD. The purpose of this chapter is to present existing procedures corresponding to the above methods and to print out their advantages, disadvantages and the limitations.

IV-1 THE TREE METHOD:

In this section, the tree method for conversion of decision tables to computer programs is described. Many variations of this method have been published, and based upon this method a large number of precompilers, interpreters and compilers have been implemented. All the procedures and algorithms based upon the tree technique were designed mainly for IS-LCE decision tables. At the end of the section we shall analyse the advantages and disadvantages of the tree method.

IV-1-1 THE TECHNIQUE:

The simplest technique for conversion of decision tables to computer programs is to translate the decision table into an equivalent flow chart, rule by rule, based upon "if... then..." relation-
ships. To show this we shall use the decision table given in Fig. IV-1-1:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>c₂</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>c₃</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>a₂</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>a₃</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Fig. IV-1-1

The corresponding flow chart is shown in Fig. IV-1-2. Note that this technique produces as many branch points as the number of condition entries which are not "don't care" entries.
In this section the conversion is presented using a flow chart to avoid the explicit use of a given programming language. Once a flow chart is obtained it can be written in assembly or any other high level language.

As seen from Fig. IV-1-2, in this translation a given condition is tested for every rule that includes it. For example condition $c_1$ is tested in every rule except the else rule. The "tree method" alleviates this by eliminating unnecessary redundancy. This is shown in the "decision tree" in Fig. IV-1-3.

![Decision Tree Diagram](image)

**Fig. IV-1-3: Decision Tree**

Each condition is tested never more than once and, therefore, the number of tests in a decision tree necessary to satisfy a particular condition, never can exceed the total number of conditions in the condition stub of a decision table.

In comparing the flow chart in Fig. IV-1-2 with the Decision Tree in Fig. IV-1-3, we can see that, in the worst cases, the former
must test eight times, while the latter requires three. Furthermore, a saving of over 50\% is realized in storage required utilizing the Decision Tree Technique. Thus the superiority of the Tree technique over the usual flow chart a certainty.

Note that only if the order in which the conditions are tested must be kept the same as in the condition stub, then the Decision Tree is unique. In general the form of the Decision Tree for any decision table is not unique and depends on the order in which the conditions are tested:

1. If, in every level of the decision tree, we test the same condition (if required), the number of different decision trees for a given decision table is equal to the number of permutations of the q conditions and therefore equal to \( q! \).

2. If for every branch in the decision tree we are free to choose the next condition to be tested (in every level), then the number of decision trees for a given decision table is much larger. In this case, the upper limit of the number of different decision trees for a given decision table is:

\[
q! \left[ \left( q-1 \right)! \right]^2 \left[ \left( q-2 \right)! \right]^3 \ldots \left[ 2! \right]^{q-1} = \prod_{u=1}^{q} (u!)^{q-u}
\]

This upper limit is obtained whenever the decision table does not include don't care conditions.

Even though many of the outcoming decision trees may look similar to each other, but taking into account that different tests may have different program code storage requirements and different
run times, the different trees may differ in total storage or run
time requirements.

**IV-1-2 EFFICIENCY CONSIDERATIONS.**

In converting decision tables to computer programs the main
goal is to generate the most efficient program code in the least
amount of conversion time (by conversion time we mean the precompile,
the interpret or the compile time). The efficiency of the program
code is measured in terms of:

1. Computer run time.
2. Computer storage required.

As usual, we expect to use any program for some length of
time before any changes or updates are necessary. For this reason
the minimization of the conversion time is of less or importance
and is not considered in this discussion.

The tree method can be used in many variations, and

generate program code which takes minimum run time or minimum
storage or a desired combination of both. This is done by choosing
one of the many decision trees corresponding to a given decision
table.

In the rest of this section we shall introduce briefly some
of the most advanced techniques. The different techniques using
the tree method fall in two categories:

1. Techniques for optimization of standard decision tables.
2. Techniques that require additional information not usually included in standard decision tables, such as apriori knowledge on data distribution, or the amount of code generated for a given condition etc.

PRESS'S [A5] PROCEDURE.

Press's procedure falls into the first category and can be used for hand-coding or as an algorithm for a decision tables processor. The leading idea is to choose, in every level and for every branch in the decision tree, the next condition to be tested, called by Press the "key row" or "key condition", based upon certain criteria. The table is then split by branching as defined in III-1-1.

The actual procedure, as stated by Press, is given below. After every step a brief explanation is also included.

**STEP 1:** CHOOSE THE KEY CONDITION FROM THE SET OF ROWS (condition entry rows \( p^1 \)) WITH THE MINIMUM NUMBER OF DON'T CARE ENTRIES.

As defined in III-1-1 splitting a table by branching on a given condition \( c_k \) that has \( y \) "Y" entries, \( x \) "N" entries and \( i \) don't care entries, results in two sub-tables with \( q-1 \) conditions and with \( x+i \) and \( y+i \) rules respectively. Choosing the key condition as the one with the minimum number of don't care entries obviously leads to smallest sub-table.

**STEP 2:** IF ONE OF THE ROWS CONTAINS ALL Y'S IN ALL ENTRIES DISCRIMINATE ON THAT ROW.
This can be best seen by the following example.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁</td>
<td>Y</td>
<td>N</td>
<td>ELSE</td>
</tr>
<tr>
<td>c₂</td>
<td>Y</td>
<td>Y</td>
<td>ELSE</td>
</tr>
</tbody>
</table>

In this example, both rules c₁ and c₂ are qualified to be the key condition by step 1. Choosing c₂ as a key condition, results in a decision tree which requires less storage.

**STEP 3: DISCRIMINATE ON THE CONDITION ROW WHICH MAXIMIZES CS₁ WHERE**

\[
CS₁ = \sum_{j=1}^{q} \rho_{ij} - \rho_{ii}
\]

AND WHERE q EQUALS THE NUMBER OF CONDITIONS IN THE TABLE.

The first two steps of the procedure can be applied by looking at the condition entry rows individually. In the third step of the procedure a look ahead strategy is applied by taking under consideration the relation among the key condition in such a way that future sub-tables will contain condition entry rows with all Y's or all N's. To understand this look ahead strategy, con-
Consider the following:

Divide the entries of each condition entry row $\rho^i$ into two non disjoint sub-set:

1. A positive sub-set including all the Y and don't care entries.
2. A negative sub-set including all the N and don't care entries.

A condition entry row $\rho^j$ is said to be the complement of a condition entry row $\rho^i$ if corresponding to the negative elements and/or the positive element in $\rho^i$ the elements of $\rho^j$ are only Y's or only N's. The number of complementary elements, which is called the count of $\rho^i$ with respect to $\rho^j$, is denoted by $\rho^i_j$.

$$CS_i = \sum_{j=1}^{q} \rho_{ij} - \rho_{ii} = \sum_{j\neq i} \rho_{ij}$$

- 've elements

$\rho^i = \{ Y \quad Y \quad - \quad N \}$$

$\rho^j = [N \quad N \quad N \quad -]$$

+ 've elements

$\rho^i_j = 3$

Thus choosing the row with the largest $CS_i$ may lead at a future splitting to a subtable with a condition entry row with all Y's or all N's. This procedure does not produce necessarily the optimum code possible but gives a way of converting decision tables into efficient program in terms of run time and storage required.

It should be noted that the above procedure is good only for LCE non ambiguous decision tables. In this procedure no means were provided for testing for ambiguity or for completeness.
POLLACK’S [AI] PROCEDURES.

Two procedures were developed by S.L. POLLACK. One which minimizes the required computer storage space and the other which minimizes computer run time. The former falls into first category, while the second falls into the second category which requires information not usually included in a decision table.

POLLACK’S FIRST PROCEDURE: MINIMUM COMPUTER STORAGE REQUIRED

This procedure is based (as in Press) upon a selection criteria that chooses, in every level and for every branch in the decision tree, the next condition or condition entry row upon which the table or the sub-table is split by branching (see definition III-1-1). The selected condition is called by Pollack the "K condition" or the "K row". Pollack does not prove that the procedure actually leads to an optimum decision tree in terms of minimum computer storage required.

The procedure as stated by Pollack is given below. After every step whenever necessary a brief explanation is given.

STEP 1: CHECK FOR REDUNDANCY OR CONTRADICTION. IF, AT ANY STAGE, A PAIR OF RULES DOES NOT CONTAIN AT LEAST ONE Y,N PAIR IN ANY OF ITS ROWS, REDUNDANCY OR CONTRADICTION EXISTS.

Note: Pollack suggests performing the test for redundancy or contradiction when the table is reduced to sub-tables with one
condition.

STEP 2: DETERMINE THOSE ROWS THAT HAVE A MINIMUM DASH-COUNT.

A don't care entry that appears in a rule (i.e. column) containing \( r \) don't care entries is counted as \( 2^r \) dashes. For each rule, we denoted the \( 2^r \) as the column-count. In each row, the sum of the column-counts corresponding to the don't care entries in that row is denoted as dash-count. A row's dash-count is the sum of the column-count of those rules that have don't care entries in the row.

STEP 3: IF TWO OR MORE ROWS HAVE A MINIMUM DASH-COUNT, SELECT THE ROW THAT HAS THE MAXIMUM DELTA.

Delta is the absolute value of the difference of the \( Y \)-count and the \( N \)-count. The \( Y \)-count is the sum of the column-counts corresponding to the \( Y \)'s in the row. The \( N \)-count is similar.

STEP 4: DISCRIMINATE ON THE CONDITION IN ROW \( K \); CALL IT \( c_k \). THIS DISCRIMINATION HAS BRANCHES, EACH OF WHICH LEADS TO A SUB-TABLE CONTAINING ONE MORE RULES, WITH ONE ROW LESS THAN THE ORIGINAL TABLE (ROW \( K \) IS DELETED).

The \( K \) row is the selected row. The selection is based on criteria described in the first three steps. Step 4 simply splits the decision table on the selected condition \( K \) by branching.

STEP 5: IF THE BRANCH LEADS TO A SUB-TABLE CONTAINING MORE THAN ONE RULE GO BACK TO STEP 1.
Step 5 simply states that the splitting by branching is repeated until the sub-tables contain only one rule.

**STEP 6a:** IF A BRANCH LEADS TO A SUB-TABLE CONTAINING EXACTLY ONE RULE, AND IF THAT RULE CONTAINS ALL DON'T CARE ENTRIES REPLACE THE SUB-TABLE WITH THE RULE ITSELF.

**STEP 6b:** IF A BRANCH LEADS TO A SUB-TABLE WITH ONE RULE, AND THAT RULE CONTAINS ONE OR MORE DON'T CARE ENTRIES BUT DOES NOT CONTAIN ALL DON'T CARE ENTRIES, CHOOSE AS ROW K ANY ROW THAT HAS NO DON'T CARE ENTRIES.

The selected branch will indicate a sub-table with one less row in it. The opposing branch of the selected row will be indicated as an ELSE-RULE.

**STEP 6c:** IF A BRANCH DOES NOT LEAD TO A SUB-TABLE, IT LEADS TO AN ELSE-RULE.

**STEP 6d:** IF THE BRANCH LEADS TO A SUBTABLE CONTAINING ONLY ONE RULE AND THAT RULE CONTAINS ONLY ONE CONDITION WHOSE VALUE IS Y (OR N), THEN ONE BRANCH OF THE DISCRIMINATION ON THAT CONDITION LEADS TO THE RULE, THE OTHER BRANCH LEADS TO THE ELSE-RULE.

**POLLACK'S SECOND PROCEDURE: MINIMUM COMPUTER RUN TIME.**

In his second procedure, Pollack uses the same technique of splitting by branching but the criteria for choosing the K condition upon which the splitting is done by introducing a weighted
dash count. This procedure can be used if and only if the following restrictions and conditions are satisfied.

1. The table is complete, whether by an implicit ELSE-RULE, or an explicit ELSE-RULE.

2. The average probability to satisfy a rule can be provided for every rule.

3. Only a small percentage of the transactions will satisfy the else rule.

This procedure falls into the second category, for which additional information is needed. The actual procedure, as stated by Pollack, is given below and explanations are added whenever needed.

**STEP 1:** SAME AS FOR THE FIRST PROCEDURE.

**STEP 2:** DETERMINE THOSE ROWS THAT HAVE MINIMUM WEIGHTED DASH COUNT.

If we denote by \((p^i)\) the frequency of satisfying the \(i^{th}\) rule the weighted dash-count (WDC) for any row is the sum of the products \([p^i \times \text{(column count)}]\) for those column that contain a dash in that row.

**STEP 3:** IF TWO OR MORE ROWS HAVE A MINIMUM WDC, SELECT, FROM AMONG THEM, THE ROW THAT HAS THE MINIMUM DELTA. IF, AMONG THESE, THERE STILL EXISTS TWO OR MORE ROWS, SELECT THE ROW WITH THE MINIMUM DASH-COUNT. IF THERE ARE MORE THAN TWO SUCH ROWS, SELECT ANY ONE OF THEM.

**STEP 4, 5, and 6:** Same as the first procedure.
The procedure imposes the following three restrictions:

1. The decision table must be unambiguous.
2. The decision table must be complete without including an explicit else rule.
3. There is complete freedom in order in which the conditions in the decision table are being tested.

Furthermore it requires the following additional information:

\[ P_i = P(Y^i) \] the probability distribution of the rules.

\[ C_i = C(c_i) \] the cost function for testing the \( i \)th condition expressed in terms of run time.

The rules are partitioned by actions.

At the beginning of this section we found that the number of decision trees corresponding to a given decision table is

\[ \prod_{k=1}^{q} (k!)^{q-k}. \]

Every decision tree is referred to as a Sequential Testing procedure (STP). The criterion of efficiency employed is "expected processing time". The expected processing cost for an STP is the weighted sum of the path costs where the weights are the respective path probabilities. The procedure described [A6] will find the minimum expected cost STP for a given LCE decision table. This procedure uses a similar algorithm like the Branch and Bound algorithm given by Little et al. [D4] for solving the traveling Salesman Problem. The authors have developed an extension which can be used in conversion of decision tables with related conditions.
REMARKS ON THE PROCEDURES DEVELOPED BY L.T. REINWALD
AND R.M. SOLAND [A6],[A7]

A detailed description of these two procedures are excluded because of their length, but for the sake of completeness some remarks are presented. L.T. REINWALD and R.M. SOLAND developed two procedures for converting decision tables to optimal computer programs. The first converts decision tables to computer programs with minimum average run time and the second converts to programs with minimum storage required. Both procedures were developed for conversion of LCE-decision tables, and fall into the second category in which additional information is required during conversion time.

FIRST PROCEDURE: MINIMUM RUN TIME [A6]

This procedure is designed to convert decision tables of the form given in Fig. IV-1-4 to program code which on the average will require minimum run time.

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>i</th>
<th>...</th>
<th>p</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C(c_1)</td>
</tr>
<tr>
<td></td>
<td>c_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C(c_2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c_q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C(c_q)</td>
</tr>
<tr>
<td>PROBABILITIES</td>
<td>p_1</td>
<td>p_2</td>
<td>...</td>
<td>p_i</td>
<td>...</td>
<td>p_p</td>
<td></td>
</tr>
<tr>
<td>ACTIONS</td>
<td>\Delta^1</td>
<td>...</td>
<td>\Delta^\mu</td>
<td>...</td>
<td>\Delta^\nu</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. IV-1-4.
SECOND PROCEDURE: MINIMUM STORAGE REQUIRED [A7]

This procedure converts decision tables of the form given in Fig. IV-1-5 to program code with minimum storage requirements. The storage required for every individual test must be given.

![Rule Number Table]

Fig. IV-1-5

\[ S_i = S(c_i) \]

is the storage required for \( i \)th test.

The rules are partitioned according to the actions.

The algorithm is similar to the Branch and Bound algorithm given by Little et al [D4] and used in the first procedure too.

The author developed a combination of both procedures to give a total minimum cost in terms of minimum average time and storage.

IV-1-3 ADVANTAGES, DISADVANTAGES AND COMPARISON OF TREE METHODS

1. THE ADVANTAGES OF THE TREE METHODS:

a. Easy to use for CS-LCE and IS-LCE decision tables with and without related conditions.

b. If used in a pre-compiler the generated code is clear and
easy to follow, (for debugging purposes).

c. Using the last two procedures, whenever the probability function and the storage required per test is known, an optimum program code is generated.

2. THE DISADVANTAGES OF THE TREE METHODS:

a. Not easily adaptable for conversion of ECE or MCE decision tables. Any extension of existing procedures to cover MCE becomes extremely complicated.

b. The method does not include effective means for testing ambiguity and completeness, in conversion and/or run time.

c. If order of the conditions is important no optimization is possible.

d. If the order in which the rules have to be tested is significant the tree method cannot be used efficiently.

e. The last two procedures require the probability function and/or the storage and cost per test as additional information, which in most cases are unknown to the user.

3. COMPARISON.

The comparison study of the different procedures presented in this section will be done by converting two illustrative examples to decision trees using the appropriate procedures as applicable.

Example IV-1-1: Minimum Average Processing Time.

Given the decision table:
Using the minimization techniques defined in III-2-2 we can obtain the following minimized condition entry matrices (with the appropriate probability and actions):

(a) 
\[
\begin{bmatrix}
Y & N & Y & N & N \\
Y & Y & N & - & N \\
- & N & - & Y & N \\
[.15 & .20 & .30 & .20 & .15]
\end{bmatrix}
\begin{bmatrix}
\delta^1 & \delta^1 & \delta^2 & \delta' & \delta^3
\end{bmatrix}
\]

(b) 
\[
\begin{bmatrix}
Y & Y & N & N & N \\
Y & Y & N & Y & N \\
- & N & Y & Y & - \\
[.00 & .35 & .30 & .20 & .15]
\end{bmatrix}
\begin{bmatrix}
\delta^1 & \delta^1 & \delta^2 & \delta' & \delta^3
\end{bmatrix}
\]

(c) 
\[
\begin{bmatrix}
Y & N & Y & N & N \\
Y & Y & N & Y & N \\
- & N & - & Y & - \\
[.15 & .20 & .30 & .20 & .15]
\end{bmatrix}
\begin{bmatrix}
\delta^1 & \delta^1 & \delta^2 & \delta' & \delta^3
\end{bmatrix}
\]

(d) 
\[
\begin{bmatrix}
Y & Y & N & N & N \\
Y & Y & N & Y & N \\
- & N & - & Y & N \\
[.00 & .35 & .30 & .20 & .15]
\end{bmatrix}
\begin{bmatrix}
\delta^1 & \delta^1 & \delta^2 & \delta' & \delta^3
\end{bmatrix}
\]
A. To take advantage of Press's procedure, we have to apply it to each of the minimized version of the decision table. The resulting decision trees are given below.

(a)  
```
Y       N
  |       |
  v       v
Y       N
  |       |
Y       N
  |       |
Y       N
  |       |
δ¹      δ² δ³ δ⁴ δ⁵
1       2   3   4   5
```

(b)  
```
N       Y
  |       |
  v       v
Y       N
  |       |
Y       N
  |       |
Y       N
  |       |
δ⁵      δ' δ' δ' δ'
1       2   3   4   5
```

Press's criteria for choosing the condition for splitting do not give a unique result in this case. There are three different possibilities.

(a)  
```
Y       N
  |       |
  v       v
Y       N
  |       |
Y       N
  |       |
Y       N
  |       |
δ¹      δ² δ³ δ³
1       2   3   4   5
```

(d)  
```
```

Fig. IV-1-7
In the following table the expected costs of (a) (b) and (c) are given:

<table>
<thead>
<tr>
<th>PATH</th>
<th>PROBABILITY (a)</th>
<th>PROBABILITY (b)</th>
<th>PROBABILITY (c)</th>
<th>COST (a)</th>
<th>COST (b)</th>
<th>COST (c)</th>
<th>EXPECTED COST (a)</th>
<th>EXPECTED COST (b)</th>
<th>EXPECTED COST (c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.15</td>
<td>.30</td>
<td>.15</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>17.70</td>
<td>35.40</td>
<td>17.70</td>
</tr>
<tr>
<td>2</td>
<td>.30</td>
<td>.15</td>
<td>.20</td>
<td>118</td>
<td>118</td>
<td>143</td>
<td>35.40</td>
<td>17.70</td>
<td>23.60</td>
</tr>
<tr>
<td>3</td>
<td>.20</td>
<td>.35</td>
<td>.20</td>
<td>75</td>
<td>93</td>
<td>143</td>
<td>15.00</td>
<td>32.55</td>
<td>28.60</td>
</tr>
<tr>
<td>4</td>
<td>.15</td>
<td>.00</td>
<td>.30</td>
<td>143</td>
<td>143</td>
<td>118</td>
<td>21.45</td>
<td>00.00</td>
<td>35.40</td>
</tr>
<tr>
<td>5</td>
<td>.20</td>
<td>.20</td>
<td>.15</td>
<td>143</td>
<td>143</td>
<td>118</td>
<td>28.60</td>
<td>28.60</td>
<td>17.70</td>
</tr>
</tbody>
</table>

**TOTAL EXPECTED COST**

118.15 104.25 123.00

**TABLE IV-1-1**

**Note:** It should be noted that using Press's procedure, any one of the above decision trees could have been the result, depending on the choice of the minimized form of the decision table.

**B. Applying Pollack's second procedure** on the minimized versions of the decision table given in Fig. IV-1-6, the resulting decision trees for (a) and (b) are exactly the same as in the case where Press's procedure is used. For (c) we arrive at two possible decision trees as shown below in (c₁) and (c₂). Pollack's procedure gives a unique decision tree shown in (d) below.
In the following table the expected costs of (a), (b), (c₁), (c₂), and (d) are given:

<table>
<thead>
<tr>
<th>PATH</th>
<th>PROBABILITY</th>
<th>COST</th>
<th>EXPECTED COST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a b c₁ c₂ d</td>
<td>a b c₁ c₂ d</td>
<td>a b c₁ c₂ d</td>
</tr>
<tr>
<td>1</td>
<td>0.15 0.30 0.15 0.15 0.00118116118118143</td>
<td>17.70 35.40 17.70 17.70 00.00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.30 0.15 0.20 0.20 0.118116118143143</td>
<td>35.40 17.70 35.40 23.60 28.60</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.20 0.35 0.15 0.20 0.35 0.75 0.9311814393</td>
<td>15.00 32.55 17.70 28.60 32.55</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.15 0.00 0.20 0.30 0.3014314314314318118</td>
<td>21.45 00.00 28.60 35.40 35.40</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.20 0.20 0.15 0.00143143143143118143</td>
<td>28.60 28.60 28.60 17.70 00.00</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.15</td>
<td>143</td>
<td>143</td>
</tr>
<tr>
<td>TOTAL EXPECTED COST</td>
<td>118.1510425128.0012300118.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE IV-1-2**

Note: It should be noted that in this case too, the resulting decision tree could have been any one of the five different decision trees, depending on the minimized version of the decision table used.

Using the first procedure developed by L.T. Reinwald and R.M.
tree is not unique when using Press's or Pollack's procedure exhausting all possible minimal decision tables and trees, the optimum decision tree can be found. (see tables IV-1-1 and IV-1-2. Both include the minimal expected cost 104.25)

3. Press's procedure does not require any additional information such as probabilities or costs, which is an advantage over the other procedures. The procedure is simple and automatic conversion is relatively simple and efficient. To evaluate the maximum amount of expected overhead cost we shall use:

$$\frac{\text{MAXEC-MINEC}}{\text{MINEC}} \times 100 = \frac{123.00-104.25}{104.25} \times 100 \approx 19\%$$

Where: MAXEC = MAXimum Expected Cost and MINEC = MINimum Expected Cost.

The cost for a straightforward conversion is obviously 143 and the absolute maximum overhead is $\frac{143.00-104.25}{104.25} = 37.2\%$

Taking into account cases in which the costs for the individual tests cannot be supplied, Press's procedure still gives an improvement of about 37.2-19 = 18.2% in the worst case.

If the costs are given, Press's procedure can be used in an exhausting mode in which the expected cost is calculated for all the minimized decision tables. The decision tree with the minimal expected cost is the optimal sequential path resulting from Reinwald and Soland's first procedure.

4. Pollack's second procedure used in this example, makes use of the rule probabilities but does not require or uses the cost for the individual test. In this case also, the optimim deci-
Soland [A6], we obtain directly from the original decision table given in Fig. IV-1-6, the decision tree shown in Fig. IV-1-9(a) and using the modified form of that procedure we obtain decision tree shown in Fig. IV-1-9(b)

![Decision Trees](image)

(a) \hspace{1cm} (b)

**Fig. IV-1-9**

Optimal decision tree for the decision table given in Fig. IV-1-6. The expected cost is $311.9$.

Optimal decision tree resulting from the modified procedure. The expected cost is $104.25$.

At this point we are ready to analyse the results from this example:

1. Neither Press's procedure nor Pollack's could produce a unique, efficient decision tree from the original decision table given in Fig. IV-1-6. Both procedures require some additional steps, namely to minimize the original decision table.

2. Minimization techniques do not lead necessarily to the appropriate minimal decision table and therefore the resulting decision
sion tree can be found (if the costs are supplied in an exhaus-
tive mode. The maximum overhead in this case is

\[
\frac{\text{MAXEC-MINEC}}{\text{MINEC}} = \frac{128.00 - 104.25}{104.25} = 23.8\%
\]

Taking into account that the maximum overhead is 37.2% we still
save about 37.8 - 23.8 = 14% in the worst case.

5. The greatest limitation of Reinwald and Soland's first procedure
is that the rule probabilities and the costs for individual
test must be supplied. On the other hand, this procedure leads
to an optimal decision tree without going into a phase of mini-
mization. Note that for using this procedure it is required to
complete the table if it is given in an incompletely specified
form.

\[
\frac{\text{MAXEC-MINEC}}{\text{MINEC}} = \frac{111.9 - 104.25}{104.25} = 7.3\%
\]

The modified procedure leads to a decision tree with an absolute
minimal expected cost. The only weak point is that branches (or
rules) with probability 0 do not appear in the decision tree.
If these rules are logically inconsistent, then by omitting them
no harm is done. However they are data inconsistent or simply
with probability 0, and if data satisfying these rules is sub-
mitted by mistake, the result may be unpredictable.

**Example IV-1-2: Minimum Storage Requirement.**

Given the decision table:
Using again the minimization techniques defined in III-2-2, we arrive at the following simple minimized condition entry matrix (with the appropriate actions)

\[
\begin{bmatrix}
Y & N & Y & Y & N & N \\
Y & N & N & Y & N & Y \\
- & N & Y & Y & N & N \\
\delta' & \delta' & \delta^2 & \delta^2 & \delta^3 & \delta^3
\end{bmatrix}
\]

A. Using Press's procedure we obtain four decision trees, as shown in Fig. IV-1-11.

![Decision Trees](image)

The storage required is

\[
3 + 2 \times 7 + 3 \times 8 = 41
\]

(a)

The storage required is

\[
3 + 2 \times 8 + 3 \times 7 = 40
\]

(b)
The storage required is
\[ 7 + 2 \times 3 + 3 \times 8 = 37 \]
(c)

The storage required is
\[ 7 + 3 \times 3 + 2 \times 8 = 32 \]
(d)

Fig. IV-1-11

B. using Pollack's first procedure on the minimized decision table, the resulting decision trees are exactly the same as those obtained by Press's procedure.

C. Using Reinwald and Soland's procedure, the resulting tree is unique and is given in Fig. IV-1-11(d).

The results of this example lead to the following conclusion:

1. The resulting decision trees after applying Press's or Pollack's procedures are not unique and depend on the minimized versions of the decision tables. Even if there is only one minimized decision table, the resulting decision tree is not unique. If the storage required for every individual test is known, then by exhausting search the decision tree with minimum storage required can be found. But, in any case, both procedures minimize to some extent the required storage.
2. Reinwald and Soland's second procedure leads to a unique decision tree with minimum storage required. The only weak point is that the individual storage required has to be known for every test, which becomes extremely hard in high language as it is compiler dependent.

3. Reinwald and Soland's procedure is applicable for tree methods only. It is possible to obtain further reduction in the storage requirement by using the mask method.

**IV-2 THE MASK METHOD**

In this section, a fundamentally different method for conversion of decision tables to computer programs is presented. The mask method can be regarded as a semi-interpretive method. The basic technique as first described by H.W. Kirk [Bl] together with some variations, modifications and extensions will be discussed. At the end, a general comparison between the tree method and the mask method is presented.

**IV-2-1 THE TECHNIQUE.**

The technique, as Kirk [Bl] describes it, is based upon the creation of a binary image of a limited entry decision table (we shall see later that this can be extended and used for LCE and MCE decision tables, too), and a binary image vector of the results after testing a given set of conditions for given data. This binary image vector is used then to scan the binary image of the decision
table in order to find which rule is being satisfied and which ac-
tion has to be taken.

In introducing the basic Mask technique we shall use the
following IS-LCE decision table (given in Fig. IV-2-1).

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>L</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>ACTIONS</td>
<td>δ'</td>
<td>δ²</td>
<td>δ³</td>
<td>δ⁴</td>
<td>δ⁵</td>
<td>δ⁶</td>
<td>δ⁷</td>
</tr>
</tbody>
</table>

Fig. IV-2-1: Illustrative example for presenting the Mask technique

To create a binary image of a given decision table (like the
one in Fig. IV-2-1), we have to build two binary matrices: a "Decision
Matrix" and a "Mask Matrix".

The Decision Matrix: - Given a LCE decision table, the respective
decision matrix, denoted by \( D \), is constructed by the following
rule:

\[
    d_{ij} = \begin{cases} 
        1 & \text{if } c_{ij} = Y \text{ or Yes} \\
        0 & \text{otherwise} 
    \end{cases}
\]

The decision matrix which corresponds to the decision table of
Fig. IV-2-1 is:

\[
    D = [d_{ij}] = \begin{bmatrix}
        1 & 1 & 1 & 0 & 0 & 0 \\
        1 & 0 & 0 & 1 & 0 & 0 \\
        1 & 0 & 1 & 0 & 1 & 0 \\
        0 & 1 & 1 & 0 & 0 & 0
    \end{bmatrix}
\]
The Mask Matrix: - Given a LCE decision table, the respective mask matrix, denoted by M, is constructed by the following rule:

\[
m_{ij} = \begin{cases} 
0 & \text{if } c_{ij} \text{ don't care} \\
1 & \text{otherwise}
\end{cases}
\]

The mask matrix which corresponds to the decision table of Fig. IV-2-1 is:

\[
M = [m_{ij}] = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0
\end{bmatrix}
\]

These two binary matrices include all the information given in a limited condition entry matrix. In order to decide which action is to be taken when certain data is presented, upon testing the different conditions, we build concurrently a result binary vector denoted by R, which is constructed by the following rule:

\[
R_i = \begin{cases} 
1 & \text{If the result of testing } c_i \text{ is YES} \\
0 & \text{otherwise}
\end{cases}
\]

This next operation is a scanning operation in which, using the result vector and the mask matrix, we scan the decision matrix to find which rule is satisfied.

The Scanning Operation: - Starting from the first column vector in the mask matrix, we multiply it logically by the result binary vector and test for equality with the corresponding column vector in the decision matrix.

i. If \( r_i x m_{ij} = d_{ij} \) for all \( i = 1, 2, \ldots, q \) then the \( j^{th} \) rule \( y_j \) is satisfied and the action to be taken is \( o_j \).

ii. If \( r_i x m_{ij} \neq d_{ij} \) for a least one \( i, \ i = 1, 2, \ldots, q \) then the next column vector has to be tested.
iii. If upon completion of the scanning operation, no rule was satisfied, the action corresponding to the "else rule" has to be taken. If no "else rule" is included (and the decision table is not complete) a standard error procedure has to be performed.

In the basic mask technique, as originally presented by Kirk, in order to find which rule is satisfied, we first have to test all the conditions. In addition, p/2 scanning operations, on the average, are also required. Therefore, a straight application of this method as a conversion procedure will give very inefficient program with respect to run time.

On the other hand, testing every condition in a continuous sequence, and recording the result in a binary form, gives a very compact program procedure requiring the least possible amount of storage. Also, no code for branching is required and therefore some run time together with many paragraph names (or labels) are saved (very important especially for computers in which the number of paragraph names or labels per program is limited as in the Burroughs B-5500 computer).

Note that since both matrices and the result vector are binary, very little additional storage is required.

**Kirk's Modification.**

For computers which are character oriented and are lacking bit manipulation operations, Kirk [Bl] has suggested the following
modification of the mask technique. In the modified technique, only a modified decision matrix and a modified result vector are used. The decision matrix is created by the following rule:

\[
d_{ij} = \begin{cases} 
Y & \text{if } c_{ij} = Y \text{ or YES} \\
N & \text{if } c_{ij} = N \text{ or NO} \\
D \text{ or "-"} & \text{if } c_{ij} = "-" \text{ or don't care.}
\end{cases}
\]

In other words \( d_{ij} = c_{ij} \) or \( D = C \) the condition entry matrix as defined. The result vector is created by the following rule:

\[
r_i = \begin{cases} 
Y & \text{if testing the condition } c_i \text{ results in Yes} \\
N & \text{otherwise}
\end{cases}
\]

The function originally performed by the mask matrix in the basic mask technique, is replaced by moving the result vector to a work area and then only inserting a \( D \) or "-" into the corresponding position whenever a \( D \) exists in the corresponding column vector in the decision matrix. The resulting modified result vector is denoted by \( D_{ij}(r_i) \). Only then the comparison is done.

Let us use now the example given in Fig. IV-2-1. The modified decision matrix would be simply the original condition entry matrix \( C = D \).

\[
D = C = [r_{ij}] = \begin{bmatrix} 
Y & Y & Y & N & N & N \\
Y & - & - & Y & N & N \\
Y & N & Y & - & Y & N \\
N & Y & Y & - & - & N 
\end{bmatrix}
\]

And if the result vector is:

\[
R = \begin{bmatrix} 
Y \\
N \\
N \\
N
\end{bmatrix}
\]
Then:

\[ D_{11}(r_1) = \begin{bmatrix} Y \\ N \\ Y \\ N \end{bmatrix} \neq \begin{bmatrix} Y \\ Y \\ Y \\ N \end{bmatrix} = [d_{11}] = [c_{11}] \]

Therefore the data does not satisfy the first rule

\[ D_{12}(r_1) = \begin{bmatrix} Y \\ N \\ N \end{bmatrix} = [d_{12}] = [c_{12}] \]

The second rule \( \gamma^2 \) is satisfied and therefore the action \( \delta^2 \) has to be taken.

The mask method, unlike the tree method, can be easily extended and used for ECE and MCE decision tables as well. This extension was done in 1969 by R. Peel [B6] and is given next.

**PEEL'S EXTENDED MASK TECHNIQUE**

In the extended mask technique introduced by Peel [B6] the decision matrix is modified and some additional vectors containing information pertinent to extended condition are utilized. In order to illustrate extended mask method, the following IS-MCE decision table will be used.

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
<td>-</td>
<td>N</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>L</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>10</td>
<td></td>
<td>-</td>
<td>20</td>
<td>30</td>
<td>E</td>
</tr>
<tr>
<td>ACTIONS</td>
<td>( \delta' )</td>
<td>( \delta^2 )</td>
<td>( \delta' )</td>
<td>( \delta^4 )</td>
<td>( \delta^5 )</td>
<td>( \delta^6 )</td>
<td>( \delta^7 )</td>
</tr>
</tbody>
</table>

Fig. IV-2-2: Illustrative example for presenting the extended mask technique
In the extended mask technique we first create a row vector, denote by RV, for every condition by the following rule:

\[ RV(c_j) = [V_{j1}, V_{j2}, \ldots, V_{jn_j}] \]

For the illustrative example given in Fig. IV-2-2 we have to create two such row vectors for the first and the third conditions which are extended.

\[ RV(c_1) = [A, B, C] \]
\[ RV(c_3) = [20, 10, 30] \]

Then the decision matrix is built, by using two different rules, depending on the type of condition. For a limited condition \( c_i \) the corresponding row in the decision matrix is built using the rule:

\[ d_{ij}(c_i) = \begin{cases} 
0 & \text{if } c_{ij} = \text{don't care/"-"} \\
1 & \text{if } c_{ij} = \text{YES/Y} \\
2 & \text{if } c_{ij} = \text{NO/N} 
\end{cases} \]

For an extended condition \( C_i \) the corresponding row in the decision matrix is built according to the following rule:

\[ d_{ij}(C_i) = \begin{cases} 
0 & \text{if } C_{ij} = \text{don't care/"-"} \\
1 & \text{if } C_{ij} = \text{1}^{st} \text{ element in the appropriate row vector} \\
2 & \text{if } C_{ij} = \text{2}^{nd} \text{ element in the appropriate row vector} \\
\text{etc...} 
\end{cases} \]

Using these rules the decision matrix for the decision table given in Fig. IV-2-2 is:

\[ D = \begin{bmatrix} 
1 & 2 & 1 & 3 & 3 & 2 \\
1 & 0 & 2 & 0 & 1 & 2 \\
1 & 2 & 0 & 0 & 1 & 3 
\end{bmatrix} \]
The mask matrix is created by the same rule as in the basic mask technique and therefore the corresponding mask matrix is:

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

The result vector contains, as before, a single element for each condition. For limited conditions, the result value after the test is:

\[
r_i(c_i) = \begin{cases} 
1 & \text{if the result is YES} \\
2 & \text{if the result is NO} 
\end{cases}
\]

For extended conditions the data is first compared against each element in the appropriate row vector, and the value is assigned according to the following rule.

\[
r_i(c_i) = \begin{cases} 
0 & \text{if no successful compare results.} \\
1 & \text{if the first comparison is successful.} \\
2 & \text{if the second comparison is successful.} \\
& \text{etc ...}
\end{cases}
\]

The scanning procedure is done in the same way as in the basic mask technique. As shown by Peel, the mask method can be easily adapted and used for all types of decision tables defined in chapter II. This is its big advantage. In the next section we shall analyse the efficiency of the mask technique and introduce an additional variation which will increase slightly the storage required, but increase the efficiency of the technique in run time.

**EFFICIENCY CONSIDERATIONS**

Here, as in the tree technique, the efficiency of the method
or the technique would be measured in terms of: Conversion time, Run
time and Storage requirement.

In addition, the flexibility of the method to include pro-
cessing of ECE and MCE decision tables, without overcomplicating
the techniques and increasing the conversion time significantly,
should be also considered as a strong point in measuring the effi-
ciency of the method.

Conversion time can be kept to a minimum due to the simpli-
city of the method and the different techniques. The modified Kirk
technique for example can make use of the original decision table
as part of the converted tables. Therefore the conversion time
using the mask method, as presented so far, can be classified as low
(the exact efficiency must be measured on an individual basis and
depends on the skill and ability of the system programmer writing
the processor).

Storagewise, the method allows implementation of techniques
which will give program code with minimum storage requirements re-
lative to any tree technique which attempts to do the same. As an
example if we calculate again the storage requirement for the deci-
sion table given in Fig. IV-1-10 we get: \( S(c_1) = 3+7+8 = 15 \) plus
some small additional storage for the different tables and vectors.
Using the best tree technique the minimum storage required is 32.

The only disadvantage of the mask method lies in the fact
that the efficiency measured in terms of run time is less than in
any tree technique.

Kirk, himself, noticed this drawback of his method and suggested splitting large decision tables and taking into account, in every partial decision table, the rule probabilities. The run time saved in this way is relatively small and the main advantage of decision tables as a tool for describing complex logical problems in a single tabular form is lost.

P.J.H. King [B2] is the only one who has given serious considerations to the mask technique and suggested a modification which is called by King "The interrupted rule mask procedure".

THE INTERRUPTED RULE MASK PROCEDURE

As stated before, the only drawback of the different mask techniques and procedures is the inefficiency in run time. King, in his techniques attempts to improve the run time by taking into account the rule probabilities and the relative cost for evaluating the different conditions.

We shall introduce the interrupted rule mask procedure through the illustrative example given in Fig. IV-2-3. King developed the procedure only for LCE decision tables. For the sake of simplicity we shall also keep this restriction although that procedure can be extended and combined with Peel's procedure for ECE and MCE decision tables.
If we express by a flow chart the basic mask technique, we get the flow chart presented in Fig. IV-2-4(a). In this flow chart we first test the conditions and then scan the rules.

Fig. IV-2-3

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULE NUMBER</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁</td>
<td>Y Y N N N E</td>
<td>c(c₁)=.6</td>
</tr>
<tr>
<td>c₂</td>
<td>N Y Y Y Y L</td>
<td>c(c₂)=.1</td>
</tr>
<tr>
<td>c₃</td>
<td>- Y N Y Y S</td>
<td>c(c₃)=.3</td>
</tr>
<tr>
<td>c₄</td>
<td>- - - N Y E</td>
<td>c(c₄)=.3</td>
</tr>
</tbody>
</table>

PROBABILITIES: 1 .1 .4 .2 .1 .1

| ACTIONS | δ' δ² δ³ δ⁴ δ⁵ δ⁶ |

Fig. IV-2-4.
However, it can be easily seen that in order to decide if \( \gamma^1 \) is satisfied, it is enough if we test only first two conditions, and in order to decide whether \( \gamma^2 \) is satisfied, only the first three conditions should be tested. This is expressed in Fig. IV-2-4(b).

The procedure described in IV 2-4(b) is referred to as the interrupted rule mask technique.

As suggested by King [B2], the order of the tests of the different conditions and scanning of the different rules can be specified in a 2\( x \)\( (q+p) \) array where \( q \) is the number of conditions and \( p \) the number of rules (excluding the else rule). For example, the order of operations for the flow chart in Fig. IV-2-4(a) and Fig. IV-2-4(b) is given in Fig. IV-2-5 (a) and (b) respectively.

\[
\begin{bmatrix}
  c_1 & c_2 & c_3 & c_4 & - & - & - & - & - & - \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  c_1 & c_2 - c_3 - - c_4 - - \\
  - & - & \gamma^1 - \gamma^2 \gamma^3 - \gamma^4 \gamma^5 \\
\end{bmatrix}
\]

**Fig. IV-2-5**

Using the additional information supplied in the decision table given in Fig. II-2-3 we are in a position to calculate the average expected cost in terms of run time. Let denote by \( S_c \) the time (or cost) required for scanning a rule. Then the average cost for (a) and (b) are:

\[
T_a = (0.6+1.1+3.3) + S_c \times 1 + 2S_c \times 0.1 + 3S_c \times 0.4 + 4S_c \times 0.2 + 5S_c \times 0.1 = 1.30 + 2.8S_c
\]

\[
T_b = (0.6+1+S_c) \times 0.1 + (0.6+1+3+2S_c) \times 0.1 + (0.6+1+3+3S_c) \times 0.4 +
\]

\[
+ (0.6 + 0.1 + 0.3 + 0.3 + 4 Sc) \times 0.2 + (0.6 + 0.1 + 0.3 + 5 Sc) \times 0.1
\]
\[
= 0.96 + 28 Sc
\]

Both expressions contain the constant 2.8Sc which depends on the way the scanning procedure was implemented, and remains a constant, independent of the order in which the different operations are performed. On the other hand, the first portion of the cost is different and is reduced in the interrupted rule mask technique.

Let us introduce the following vector notation.

\[
P = \begin{bmatrix}
P_1 \\
P_2 \\
\vdots \\
P_p
\end{bmatrix}, \quad \mathcal{C} = \begin{bmatrix}
C(c_1)
\\
C(c_2)
\\
\vdots \\
C(c_q)
\end{bmatrix}, \quad A = \begin{bmatrix}
a_{11} & \cdots & a_{p1} \\
\vdots & & \vdots \\
a_{1q} & \cdots & a_{pq}
\end{bmatrix}, \quad \mathcal{V} = \begin{bmatrix}
1 \\
2 \\
\vdots \\
p
\end{bmatrix}
\]

Where \(\mathcal{P}\) is the vector of probabilities, \(\mathcal{C}\) the cost vector and \(A\) is a matrix in which every column vector conditions that have to be tested for the corresponding rule are represented by 1's. Finally let's denote by \(\mathcal{G}\), a vector of the probabilities rearranged in the order in which the rules are tested.

Then:

\[
T = \mathcal{C}^T \mathcal{A} \mathcal{P} + P \left\{ \sum_{i=1}^{q} C(c_i) + P Sc \right\} + Sc \mathcal{G}^T \mathcal{V}
\]

The middle term in the above expression is constant, and therefore the procedure with the minimum average cost is the one for which \(\mathcal{C}^T \mathcal{A} \mathcal{P} + Sc \mathcal{G}^T \mathcal{V}\) is minimum.

King does not suggest any numerical or analytical procedure
to solve this problem (except an exhausting search which is of limited usage in the case of large tables). Instead, he offers four different strategies, in which the additional information of cost and probability is used in a limited mode and do not lead to optimum procedures. More detailed discussion on these techniques will be given in chapter VI.

IV-2-3 ADVANTAGES, DISADVANTAGES AND COMPARISON OF THE MASK METHODS

1. THE ADVANTAGES OF THE MASK METHODS:

   a. Easy to use for any kind of decision tables with and without related conditions.
   b. If used in a pre-complier, a compiler or an interpreter, the conversion time is very low.
   c. The generated program code requires minimum storage space.
   d. If the order in which the rules that have to be tested is significant, the mask method is the one to be used.
   e. Does not require manual translation from BCE and MCE decision tables to LCE decision tables.
   f. The mask method lends itself to efficient testing for ambiguity and completeness in conversion time and/or run time. (See chapter V).

2. THE DISADVANTAGES OF THE MASK METHODS

   a. If used in a pre-compiler or a compiler, the generated code can be some times confusing and hard to follow (when debu-
gging is neccessary).

b. Less efficient in terms of run time compared to the tree method.

c. The scanning operation adds a significant factor to the inefficiency in run time.

3. COMPARISON.

At this point, general discussion rather than analytical or numerical comparison is done. This because the complete presentation of the mask method together with different techniques for optimizing will be given in chapter VI.

In general, the mask method can be considered superior to the tree method. The mask method is easier to implement in an automatic translator or compiler, even when used with most complicated decision tables. Using the mask method in an advanced precompiler or interpreter can give us the possibility of skipping the function of the programmer; going from the system analyst straight to the computer, sacrificing, in some cases, more machine time which is becoming cheaper and cheaper. As a semi-interpretive method, it can be applied in time sharing and real time systems.

IV-3 THE COMPUTED GO TO METHOD:

This last method is also a semi-interpretive method and with some modifications it could be used in interpreters for time sharing systems. The method was first introduced by C.G. Veinott [Cl] and will be referred to as "Veinott's Method", or "Veinott's Technique". 
Veinott's method is based upon the well known computed GO TO in COBOL, FORTRAN and ALGOL.

**IV-3-1 THE VEINOTT'S TECHNIQUE**

The Veinott's technique differ from the mask technique, in the recording method of the results after testing, and in the scanning function required to find the right action. We shall introduce the technique through an example of Fig IV-2-1. The basic idea in Veinott's technique is to calculate a unique number for every possible combination of conditions, with a single restriction that the resulting number are in an unbroken sequence; such that they can be used as the value of a branching variable.

As a first step we shall assign a weighted value $2^{j-1}$ to the $j$th condition and extend the decision table to its completely specified form. The resulting decision table (which is equivalent to the one given in Fig. IV-2-1) is:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>W VAL.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>1</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_2$</td>
<td>2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_3$</td>
<td>4</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$c_4$</td>
<td>8</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>ACTIONS</td>
<td>$\delta$</td>
<td>$\delta^2$</td>
<td>$\delta^3$</td>
<td>$\delta^4$</td>
<td>$\delta^5$</td>
<td>$\delta^6$</td>
<td>$\delta^7$</td>
<td>$\delta^8$</td>
<td>$\delta^9$</td>
<td>$\delta^{10}$</td>
<td>$\delta^{11}$</td>
<td>$\delta^{12}$</td>
<td>$\delta^{13}$</td>
<td>$\delta^{14}$</td>
<td>$\delta^{15}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fig. IV-3-1**

It should be noted, that in this case, it was relatively easy to derive the completely specified decision table with the correct
number of rules because the original decision table was unambiguous and complete. For decision tables in which apparent ambiguity (in particular, when some or all of the conditions are related) the process of going from a decision table given in IS form to a decision table which is in CS form can be more complicated, ambiguous and cannot be done using an automated procedure. In order to simplify the procedure, avoid ambiguity and keep completeness, it is preferable to arrange the rules in an increasing sequence denoting by "X", wherever the result of a test should be "Y" and leaving the entry blank when "N" should appear. The reformatted decision table is in Fig. IV-3-2

| CONDITIONS | W VAL. | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----------|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| c₁        | 1      | X |   | X |   | X |   |   |   | X |   | X   |   |   |   |   |   |
| c₂        | 2      |   | X | X | X | X |   |   |   | X | X   | X   |   |   |   |   |   |
| c₃        | 4      |   |   | X | X | X |   |   | X | X | X   | X   |   |   |   |   |   |
| c₄        | 8      |   |   |   | X | X | X | X | X | X | X   | X   |   |   |   |   |   |
| ACTIONS   |        | δ⁶ | δ⁷ | δ⁴ | δ⁷ | δ⁵ | δ⁷ | δ⁴ | δ⁷ | δ⁵ | δ⁷ | δ⁴ | δ⁷ | δ⁵ | δ⁷ | δ⁴ |

Fig. IV-3-2

The technique now uses a result vector R with q entries in which the outcome results for the respective conditions tests are recorded, "1" if the outcome is yes, and "0" otherwise. The next operation is to compute a single number for the satisfied rule, and the last operation is to branch to the appropriate action to be taken. If we assign labels L₁, L₂, L₃ etc to the different actions the last two operations can be done in COBOL, FORTRAN IV, ALGOL and PL/1 as follows:
In COBOL:

```
COMPUTE JUMP = 1 + R(1) + 2*R(2) + 4*R(3)
GO TO L6 L7 L4 L7 L5 L7 L4 L1 L7 L2 L4 L2 L5 L3 L4 L3 DEPENDING ON JUMP.
```

In FORTRAN:

```
JUMP = 1 + R(1) + 2*R(2) + 4*R(3)
GO TO (L6,L7,L4,L7,L5,L7,L4,L1,L7,L2,L4,L2,L5,L3,L4,L3,JUMP
```

In FORTRAN IV:

```
GO TO (L6,L7,L4,L7,L5,L7,L4,L1,L7,L2,L4,L2,L5,L3,L4,L3)
```

In ALGOL or PL/1:

```
SWITCH SW = L6,L7,
::;
JUMP = 1 + R(1) + 2*R(2) + 4*R(3);
GO TO SW (JUMP)
```

or

```
SWITCH SW = L6,L7,
::;
GO TO SW (1 + R(1) + 2*R(2) + 4*R(3));
```

Using this technique, the problem of ambiguous and incomplete decision tables does not arise in the case of limited condition entry decision tables.

The same technique can be applied for ECE and MCE decision
tables. The technique generates again for every rule, a unique number which is used as a branching variable with the same restriction as for the case of LCE decision tables. Let's assume now that the table contains q conditions and every condition $\sigma_j$ is associated with the set $V_{ji}, \ldots, V_{jn_j}$ of $n_j$ vals. Then the $j^{th}$ position in the result vector $R(j)$ can take the values 0, 1, 2, ..., $n_j$, depending on the outcome of the rest of the $j^{th}$ condition.

Now, the branching variable is calculated according to the following formula:

$$\text{JUMP} = 1 + R(1) + n_1 R(2) + n_1 n_2 R(3) + \ldots \ldots + R(q) \prod_{j=1}^{q-1} n_j$$

Everything else remains the same as in the case of LCE decision tables. This technique implies again that the ECE or MCE decision table has to be given in an CS-LCE decision table, excluding the impossible combinations, due to the resulting related limited conditions, after the conversion from MCE to LCE.

The following example given in Fig. IV-3-3 will clarify the utilization of Veinott's technique used for extended condition entry decision tables.

<table>
<thead>
<tr>
<th>RULE NUMBER</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITIONS</td>
<td></td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>$\sigma_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>ACTIONS</td>
<td>$\delta^0$</td>
<td>$\delta^1$</td>
<td>$\delta^2$</td>
<td>$\delta^3$</td>
<td>$\delta^4$</td>
<td>$\delta^5$</td>
<td>$\delta^6$</td>
<td>$\delta^7$</td>
<td>$\delta^8$</td>
<td>$\delta^9$</td>
<td>$\delta^{10}$</td>
<td></td>
</tr>
</tbody>
</table>

Fig. IV-3-3
The condition \( c_1 \) is associated with the set \( a, b, c \) with \( n_1 = 3 \)
The condition \( c_2 \) is associated with the set \( 1, 2, 3, 4 \) with \( n_2 = 4 \)

The decision table in Fig. IV-3-3 is already given in a completely specified form. In order to make use of the Veinott procedure for extended condition entry decision tables, the table should be converted into the following form:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1(a) )</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( c_1(b) )</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( c_1(c) )</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( c_2(1) )</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( c_2(2) )</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( c_2(3) )</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>2</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>( c_2(4) )</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>3</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>ACTIONS</td>
<td>( \delta^0 )</td>
<td>( \delta^1 )</td>
<td>( \delta^2 )</td>
<td>( \delta^3 )</td>
<td>( \delta^4 )</td>
<td>( \delta^5 )</td>
<td>( \delta^6 )</td>
<td>( \delta^7 )</td>
<td>( \delta^8 )</td>
<td>( \delta^9 )</td>
<td>( \delta^{10} )</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. IV-3-4**

It can be easily seen that the table does not contain all possibilities; only those which are included in the original CS-ECE decision table.

**IV-3-2 EFFICIENCY CONSIDERATIONS.**

If we limit the technique to CS-LCE-LAE nonambiguous and complete decision tables, rather than MCE-MAE decision tables then the conversion time required (manually or automatically) is low. In other words if the decision tables are such that most of the work is done ahead of conversion time then obviously the conversion time is
low. On the other hand if we allow any kind of decision tables up to IS-MCE-MAE then the conversion time will be increased significantly. In some cases in particular when IS-MCE decision tables which include apparent ambiguity but are incomplete, automated conversion becomes impossible.

The run time of the generated code using this technique obviously is maximum for testing all conditions. In addition some computation time is required for computing the proper rule number and executing a computed GO TO, which is, in general, relatively less than the time required for the scanning function in the mask techniques. In the case of ECE decision table, first they have to be converted to LCE. In run time for every condition which was originally extended $c_j$, $n_j$ tests are made disregarding the fact that they are dependant.

Obviously, the storage required for the generated code is minimal as in the case of the mask method and may be even less.

Finally, it should be noted that, in terms of efficiency, this method is not recommended.

**IV-3-3 ADVANTAGES AND DISADVANTAGES**

1. **THE ADVANTAGES OF THE VEINOTT'S METHOD**
   a. Simple to use as a manual conversion technique.
   b. If used in pre-compiler or interpreter, within the limitations mentioned before the conversion time is relatively small.
c. The generated code requires minimum storage.

2. THE DISADVANTAGES OF THE VEINOTT'S METHOD

a. The method generates inefficient code with respect to run time.

b. Does not allow natural extensions to ECE decision tables in the real sense.

c. Cannot deal with apparent ambiguities.

d. If the order in which the rules have to be tested is significant there is no way to do so using Veinott's method.

e. Uses in every case $2^q$ branch tables, even in cases where this is unnecessary.
CHAPTER V. TESTING AND DIAGNOSTIC METHODS

This chapter is devoted to one of the most important aspects in the usage of decision tables, namely, testing and diagnostic methods. The efficiency and usefulness of decision tables in EDP (Electronic Data Processing) becomes doubtful without incorporating advanced and efficient methods and/or procedures for testing decision tables and generating, if necessary, comprehensive diagnostic messages. By testing, we mean testing for ambiguity of any kind and testing for completeness.

The testing of tables for ambiguity and completeness can be done at two different times; prior to conversion to computer programs and/or at run time. If a decision tables processor is used in time sharing or real time in an interpretive mode, the possibility of checking at run time is very desirable feature. Different approaches have been suggested and taken. The most common approach, defined by Pollack [A9], which assumes unrelated conditions in the condition stub, can be implemented in any LCE decision table processor, with relative simplicity, using the definitions for ambiguity given in II-2-2. If testing for completeness is required, one may use, also with relative simplicity, the definition given in II-2-11. Many of the existing decision table processors ignore completely the test for completeness, and that automatically adds the requirement of incorporating the ELSE RULE in every decision table. It should be noted that if definitions II-2-5 and II-2-6 are used as guide
rules for writing testing and diagnostic procedures (even with related conditions) the resulting tables and programs are correct (see the decision tables in Fig. II-2-2 and Fig. II-2-3). These definitions for ambiguity were used in many LCE decision table processors.

In general the testing and diagnostic methods fall into two main categories:
2. Testing methods in run time.

V-1. TESTING AND DIAGNOSTIC METHODS IN CONVERSION TIME.

In this section we shall introduce necessary and sufficient conditions for ambiguity in LCE and MCE decision tables which can serve as guide rules in writing and diagnostic procedures in conversion time. The conditions are based on the definitions given in sections II-1 and II-2. We shall start with LCE decision tables with unrelated conditions, extend it to LCE decision tables with related and unrelated conditions. Finally we shall consider MCE decision tables. It should be noted that P.H.J. King was the first to give a serious consideration to this subject and he introduced necessary and sufficient condition for testing LCE decision tables.

Before going any farther let us introduce the following definitions and notations:
1. We denote by $d_i$ the $i$th column of the decision matrix defined in IV-2-1.
2. We denote by $m_j$ the $j$th column vector of the mask matrix defined
in IV-2-1.

3. We denote by $I$ a unity vector, a $q$ dimensional column vector in which every component equals 1.

**Definition V-1-1:** Given two $q$ dimensional vectors $m_i$ and $d_j$, the intersection $m_i \cap d_j$ is the row by row logical intersection and the union $m_i \cup d_j$ is the row by row logical union.

Bessen upon definitions II-2-5 and II-2-6, necessary and sufficient conditions for ambiguity are derived in the following theorems.

**Theorem V-1-1:** Given a LCE decision table with unrelated conditions, the necessary and sufficient conditions for redundancy between two rules $\gamma^i$ and $\gamma^j$ are:

1. $m_i \cap d_j = m_j \cap d_i$
2. $\delta^i = \delta^j$

Where $m_i$, $m_j$ are the $i^{th}$ and the $j^{th}$ vectors from the mask matrix, and $d_i$, $d_j$ are the $i^{th}$ and the $j^{th}$ vectors from the decision matrix associated with the given LCE decision table.

**Proof:** First let us prove that the conditions are necessary. The necessity of the second condition is obvious from the definition of redundancy given in II-2-5.

From the definitions of the LCE decision matrix and the LCE mask matrix, the set of conditions that has to be tested in order to satisfy $\gamma^i$, $\gamma^j$ or both, is given by the 1's in the vector $m_i \cup m_j$.
\[ m_j \cap (m_i \cap d_j) = m_j \cap d_i \]

\[ (m_i \cap m_j) \cap d_j = m_i \cap (m_j \cap d_j) = m_i \cap d_j \]

if ambiguity exists \( m_i \cap d_j = m_j \cap d_i \)

and if the ambiguity is redundancy \( \delta^i = \delta^j \)

This concludes the proof that the conditions are necessary.

To prove that the conditions are sufficient we reverse the steps used in proving the necessity of the conditions. Therefore if the two conditions hold, so does:

\[ (m_i \cap m_j) \cap d_i = (m_i \cap m_j) \cap d_j \]

and therefore redundancy exists.

**Corollary V-1-l:** If, for a given LCE decision table, two rules \( \gamma^i \) and \( \gamma^j \) are found to satisfy the two conditions stated in theorem V-1-l, then redundancy will occur whenever the data satisfies the conditions represented by the \( l^s \) in the column vector, \( (m_i \cup m_j) \cap (d_i \cup d_j) \) and every 0 in \( (m_i \cup m_j) \) represents an irrelevant or a don't care condition.

This corollary simply states that if two rules \( \gamma^i \) and \( \gamma^j \) both have to be satisfied, the conditions relevant to \( \gamma^i \) or \( \gamma^j \) or both are given by the \( l^s \) in the column vector \( (m_i \cup m_j) \). This set can be divided into three disjoint sets (as shown in theorem VI-1-l) which, when united, give:
This set of conditions can be divided into three disjoint subsets:

1. Conditions relevant to $\gamma^i$ but not to $\gamma^j$, represented by the \(1^s\)'s in the column vectors \(m_i \cap (I-m_j)\), respectively.

2. Conditions relevant to $\gamma^j$ but not to $\gamma^i$, represented by the \(1^s\)'s in the column vector \(m_j \cap (I-m_i)\), respectively.

3. Conditions relevant to $\gamma^i$ and to $\gamma^j$, represented by the \(1^s\) in the column vector \(m_i \cap m_j\), respectively.

To satisfy $\gamma^i$, the conditions designated by the \(1^s\) in the column vector \(m_i \cap (I-m_j) \cap d_i\) have to be satisfied. (This is a subset of the set of conditions that has to be satisfied in order to satisfy $\gamma^i$. The complete set correspondence to the \(1^s\) in the column vector \(m_i \cap d_i\).

Note that this does not prevent the satisfaction of rule $\gamma^j$ also.

Similarly to satisfy the rule $\gamma^j$, the conditions designated by the \(1^s\) in the column vector \(m_j \cap (I-m_i) \cap d_j\) have to be satisfied. That, again, does not prevent the satisfaction of the rule $\gamma^i$ also.

Using the third subset \(m_i \cap m_j\), in order to satisfy the rule $\gamma^i$, the conditions designated by the \(1^s\) in the column vector \((m_i \cap m_j) \cap d_i\) have to be satisfied, and to satisfy the rule $\gamma^j$, the conditions designated by the \(1^s\) in the column vector \((m_i \cap m_j) \cap d_j\) have to be satisfied.

If ambiguity exists:

\[
(m_i \cap m_j) \cap d_i = (m_i \cap m_j) \cap d_j
\]

but by definition \(d_i \subseteq m_i\) and \(d_j \subseteq m_j\); therefore:

\[
(m_i \cap m_j) \cap d_i = (m_j \cap m_i) \cap d_i =
\]
\[ \{m_j \cap (I-m_i) \} \cup \{m_i \cap (I-m_j) \} \cup \{m_j \cap m_i \} \cup \{d_i \cap d_j \} \]

\[ = (m_i \cup m_j) \cap (d_i \cup d_j) \]

**Theorem V-1-2:** Given a LCE decision table with unrelated conditions, the necessary and sufficient conditions for contradiction between two rules \( \gamma^i \) and \( \gamma^j \) are:

1. \( m_i \cap d_j = m_j \cap d_i \)
2. \( \delta^i \neq \delta^j \)

Where \( m_i, m_j, d_i \) and \( d_j \) have the same meaning as in theorem V-1-1. The proof of this theorem is completely analogous to the proof in theorem V-1-1 and thus not included.

**Corollary V-1-2:** If, for a given LCE decision table, two rules \( \gamma^i \) and \( \gamma^j \) are found to satisfy the two conditions stated in theorem V-1-2, then contradiction will occur whenever the data satisfies the conditions represented by the 1's in the column vector:

\[ (m_i \cup m_j) \cap (d_i \cup d_j) \]

These two theorems can actually serve as guide rules for implementation of an efficient and comprehensive procedures for testing LCE decision tables (with unrelated conditions) for ambiguity.

Actually, the same criteria for testing ambiguity of LCE decision tables with unrelated conditions can be used for testing ambiguity for LCE decision tables with related and unrelated conditions. This is stated in the following theorems.
Theorem V-1-3: - Given an LCE decision table with related and unrelated conditions, the necessary and sufficient conditions for possible redundancy between two rules \( \gamma^i \) and \( \gamma^j \) are:

1. \( m_i \cap d_j = m_j \cap d_i \)
2. \( \delta^i = \delta^j \)

Actual redundancy exists if, and only if, there exists consistent data which satisfies the conditions represented by the \( 1^s \)'s in the column vector \((m_i \cup m_j) \cap (d_i \cup d_j)\).

The proof of this theorem is completely analogous to the proof of theorem V-1-1 and corollary V-1-1 and therefore not included.

Theorem V-1-4: - Given a LCE decision table with related and unrelated conditions, the necessary and sufficient conditions for possible contradiction between two rules \( \gamma^i \) and \( \gamma^j \) are:

1. \( m_i \cap d_j = m_j \cap d_i \)
2. \( \delta^i \neq \delta^j \)

Real contradiction exists if, and only if, there exists consistent data which satisfies the conditions represented by the \( 1^s \)'s in the column vector \((m_i \cup m_j) \cap (d_i \cup d_j)\).

The proof of this theorem is again completely analogous to the proof of theorem V-1-1 and corollary V-1-1 and thus not included.

**Example V-1-1**: - Given the rules \( \gamma^i = \begin{bmatrix} Y \\ Y \\ -N \\ - \end{bmatrix} \) and \( \gamma^j = \begin{bmatrix} Y \\ Y \\ N \\ N \end{bmatrix} \)

the respective decision and mask vectors are:
\[ m_i = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \quad d_i = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad m_j = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \quad d_j = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \]

\[ m_i \cap d_j = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cap \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \cap \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = m_j \cap d_i \]

\[ \delta^i = \delta^j; \text{ then:} \]
If the decision table contains unrelated conditions, a definite redundancy exists. (Theorem V-1-1)

If the decision table contains related (and unrelated) conditions (as well), a possible redundancy exists. (Theorem V-1-3)

\[ \delta^i \neq \delta^j; \text{ then:} \]
If the decision table contains unrelated conditions, a definite contradiction exists. (Theorem V-1-2)

If the decision table contains related (and unrelated) conditions (as well), a possible redundancy exists. (Theorem V-1-4)

Based upon corollaries V-1-1, V-1-2, and the last condition theorems V-1-3 and V-1-4, the ambiguity will occur with the following result:

\[ (m_i \cup m_j) \cap (d_i \cup d_j) = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cup \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \cap \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \cup \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \cap \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} Y \\ Y \\ N \\ N \end{bmatrix} = R \]

: can be easily seen that \( R \in \{ \omega^i \} \) as well as \( R \in \{ \omega^j \} \).

The last four theorems can be extended, with some modification
and definition, to cover the MCE decision table. In order to introduce these extensions we shall define the following two operations.

**Definition V-1-2:** - Given a mask vector \( m_i \) and a decision vector \( d_j \) of the same order, the multiplication \( m_i \times d_j \) is a vector whose components are the respective row by row multiplication

\[
R_{ik} = m_{ik} \times d_{jk} \; ; \; k = 1, 2, \ldots, q
\]

This operation has the following property

\[
(m_i \cap m_j) \times d_j = m_i \times (m_j \times d_j)
\]

**Definition V-1-3:** - The operator \( \ast \) has the following meanings:

\[
d_{ik} \ast d_{jk} = \begin{cases} 
d_{ik} & \text{if } d_{ik} = d_{jk} \\
0 & \text{if } d_{ik} = 0 \text{ and/or } d_{jk} = 0 \\
\text{not defined otherwise.} & 
\end{cases}
\]

the operator \( \ast \) between two vectors \( d_i \) and \( d_j \) is defined as if it is between every one of the corresponding components.

**Theorem V-1-5:** - Given an MCE decision table with unrelated conditions, these necessary and sufficient conditions for redundancy between two rules \( \gamma^i \) and \( \gamma^j \) are:

1. \( m_i \times d_j = m_j \times d_i \)
2. \( \delta^i = \delta^j \)
where \( m_i \), \( m_j \) are the \( i \)th and the \( j \)th vectors from the MCE mask matrix and \( d_i \), \( d_j \) are the \( i \)th and the \( j \)th vectors from the MCE decision matrix associated with a given MCE decision table.

**Proof:** First let us prove that the conditions are necessary. The necessity of the second condition is obvious from the definition of redundancy for MCE decision tables.

From the definitions of the MCE decision matrix and the MCE mask matrix the set of conditions that has to be tested in order to satisfy \( \gamma^i \), \( \gamma^j \) or both is given by the 1's in the vector \( m_i \cup m_j \). This set of conditions can be divided into three disjoint subsets:

1. Conditions relevant to \( \gamma^i \) but not to \( \gamma^j \), represented by the 1's in the column vector \( m_i \cap (I-m_j) \) respectively.
2. Conditions relevant to \( \gamma^j \) but not to \( \gamma^i \), represented by the 1's in the column vector \( m_j \cap (I-m_i) \) respectively.
3. Conditions relevant to \( \gamma^i \) and to \( \gamma^j \) represented by the column vector \( m_i \cap m_j \) respectively.

To satisfy \( \gamma^i \), the conditions designated by the non-zero entries in the column vector \( [m_i \cap (I-m_j)] \times d_i \) have to be satisfied (This is a subset of the set of conditions that has to be satisfied in order to satisfy \( \gamma^i \). The complete set corresponds to the non-zero entries in the column vector \( m_i \times d_i \)). Note that this does not prevent the satisfaction of the rule \( \gamma^j \) also. Similarly to satisfy the rule \( \gamma^j \) the conditions designated by the non-zero entries in the column vector \( [m_j \cap (I-m_i)] \times d_j \) have to be satisfied. That, again, does not prevent the satisfaction of the rule \( \gamma^i \) also.
From the third subset \( m_i \cap m_j \), in order to satisfy the rule \( \gamma^i \), the conditions designated by the non-zero entries in the column vector \((m_i \cap m_j) x d_i\) have to be satisfied, and to satisfy the rule \( \gamma^j \) the conditions designated by the non-zero entries in the column vector \((m_i \cap m_j) x d_j\) have to be satisfied.

If ambiguity exists
\[(m_i \cap m_j) x d_i = (m_i \cap m_j) x d_j\]

But the set of the non-zero entries in \( d_j \) is a proper subset of the non-zero entries in \( m_j \), and the set of the non-zero entries in \( d_i \) is a proper subset of the non-zero entries in \( m_i \) therefore
\[
(m_i \cap m_j) x d_i = (m_j \cap m_i) x d_i = m_j x (m_i x d_i) = m_j x d_i
\]
\[
(m_i \cap m_j) x d_j = m_i x (m_j x d_j) = m_i x d_j
\]
If ambiguity exists \( m_i x d_j = m_j x d_i \) and if the ambiguity is redundancy \( \delta^i = \delta^j \)

This concludes the proof that the conditions are necessary.

To prove that the conditions are sufficient we reverse the steps used in proving the necessity of the conditions. Therefore if the two conditions hold so does:
\[(m_i \cap m_j) x d_i = (m_i \cap m_j) x d_j\]
and therefore redundancy exists.

**Corollary V-1-3:** - If, for a given MCE decision table, two rules \( \gamma^i \) and \( \gamma^j \) satisfy the conditions stated in theorem V-1-5; then redundancy will occur whenever the conditions represented by the vector:

\[
(m_i \cup m_j) \times (d_i^* \cup d_j)
\]

are satisfied.

The meaning of \( d_i^* \cup d_j \) is as previously defined in definition V-1-3.

**Proof:** This corollary states that if two rules \( \gamma^i \) and \( \gamma^j \) both have to be satisfied, the conditions relevant only to \( \gamma^i \) have to be satisfied, the conditions relevant only to \( \gamma^j \) have to be satisfied and the conditions relevant to both \( \gamma^i \) and \( \gamma^j \) have to be satisfied by the same values respectively. The set of all the conditions that have to be satisfied can be divided into three disjoint sets (as shown in theorem VI-1-5) which when united give:

\[
\{[m_j \cap (I - m_i)] \times d_j\} \cup \{[m_i \cap (I - m_j)] \times d_i\} \cup \{[m_j \cap m_i] \times [d_i^* \cup d_j]\}
\]

\[
= (m_i \cup m_j) \times (d_i^* \cup d_j)
\]

Note that if the conditions of theorem V-1-5 hold \( d_i^* \cup d_j \) is always defined and the operation \( \cup \) is valid and can be applied.

**Theorem V-1-6:** - Given an MCE decision table with unrelated conditions, the necessary and sufficient conditions for contradiction between two rules \( \gamma^i \) and \( \gamma^j \) are:

1. \( m_i \times d_j = m_j \times d_i \)
2. \( \delta^i \neq \delta^j \)
Proof: The proof of this theorem is completely analogous to the proof of theorem V-1-5 and thus omitted.

**Corollary V-1-4:** - If, for a given MCE decision table two rules $\gamma^i$ and $\gamma^j$ satisfy the conditions stated in theorem V-1-6; then contradiction will occur whenever the data satisfies the conditions represented by the vector:

$$(m_i \text{Um}_j) \times (d_i \text{Ud}_j).$$

Proof: The proof of this corollary is completely analogous to the proof of corollary V-1-3 and thus not included.

**Theorem V-1-7:** - Given a MCE decision table with related and unrelated conditions, the necessary and sufficient conditions for possible redundancy between two rules $\gamma^i$ and $\gamma^j$ are:

1. $m_i \times d_j = m_j \times d_i$
2. $\delta^i = \delta^j$

Real redundancy exists if and only if there exists consistent data which satisfy the conditions represented by the vector:

$$(m_i \text{Um}_j) \times (d_i \text{Ud}_j).$$

Proof: The proof of this theorem is completely analogous to the proof of theorem V-1-5 and corollary V-1-3 and therefore not included.

**Theorem V-1-8:** - Given a MCE decision table with related and unrelated conditions, the necessary and sufficient conditions for
possible contradiction between two rules $\gamma^i$ and $\gamma^j$ are:

1. $m_i \times d_j = m_j \times d_i$
2. $\delta^i \neq \delta^j$.

Real contradiction exists if and only if, there exists consistent data which satisfy the conditions represented by the vector:

$$(m_i \cup m_j) \times (d_i \cup d_j).$$

Proof: The proof of this theorem is completely analogous to the proof of theorem V-1-5 and corollary V-1-3 and therefore not included.

Example V-1-2: - Given the following MCE decision table:

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>1</th>
<th>2</th>
<th>3, 4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>27</td>
<td>27</td>
<td>30</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>$c_2$</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$c_3$</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ACTIONS</td>
<td>$\delta'$</td>
<td>$\delta^2$</td>
<td>$\delta^3$</td>
<td>$\delta^4$</td>
<td>$\delta^5$</td>
<td>$\delta^6$</td>
<td>$\delta^7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RULE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3, 4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

$V_1 = [27, 30, 35, 40]$

$V_2 = [=, <, >]$

$$D = \begin{bmatrix} 1 & 1 & 2 & 3 & 3 & 3 & 4 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 2 & 3 \end{bmatrix}, \quad M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$m_1 \times d_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = m_2 \times d_1$$
Since $\gamma^1$ and $\gamma^2$ stand for two different actions, and the actual conditions are not explicitly stated, a possible contradiction exists. If there exists consistent data which satisfy the conditions represented by the following vector, real contradiction exists:

\[(m_1 U m_2) \times (d_1 \hat{U} d_2) = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 27 \\ Y \end{bmatrix} \]

It can easily be verified that data which satisfies $[27,Y,-]$ satisfies both $\gamma^1$ and $\gamma^2$.

Testing for the $4^{th}$ and the $5^{th}$ rules we get:

\[m_4 \times d_5 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 0 \\ 0 \\ 0 \end{bmatrix} = m_5 \times d_4 \]

And since $\gamma^4$ and $\gamma^5$ have identical actions, a possible redundancy for data which satisfies the conditions represented by the following vector, exists:

\[(m_4 U m_5) \times (d_4 \hat{U} d_5) = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 3 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 3 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 35 \\ Y \end{bmatrix} \]

Data which satisfies $[35,Y,-]$ satisfies both $\gamma^4$ and $\gamma^5$.

**Conclusions:** Any one of the previous theorems can serve as a guide rule in implementing testing procedures for any kind of decision tables, starting with LCE decision tables with unrelated conditions and going up to MCE decision tables with related and unrelated conditions.
Since in every case, real or apparent ambiguity should result in the generation and printing of the appropriate messages, and since the most flexible decision tables for actual use are the MCE decision tables with related and unrelated conditions, the use of theorem V-1-8 is recommended. A good decision table processor should not stop the conversion if possible ambiguity is found. It should continue the conversion generating the appropriate messages and diagnostics, leaving the final decision to the systems analyst. If, after reviewing the generated messages, actual ambiguity is found then an appropriate correction and an additional conversion pass should be made.

The testing procedures which can be designed, using the former theorems, are most adaptable to the mask method, in which the same mask and decision matrices can be used. The same procedures can be adapted for testing, even if the tree methods are used; this at the expense of some additional storage and conversion time.

V-2 TESTING AND DIAGNOSTIC METHODS IN RUN TIME

In the last section, we dealt with possible ambiguity, because in actual use of decision tables, we use MCE decision tables with related and unrelated conditions. We were only able to comment on the data (if any) for which ambiguity could occur. This, because at conversion time, we are not able to judge whether or not this type of data is consistent and/or possible. The big advantage of the previously discussed testing and diagnostic methods, is the fact
that at conversion time, they can be used with the mask method as well as with the tree method.

The main problem is that a possible ambiguity cannot always be identified, by the user, as a real ambiguity or not. A simple example, given in Fig. V-2-1 shows that redundancy exists between the rules $\gamma^1$ and $\gamma^2$, and possible contradiction exists between the rules $\gamma^3$ and $\gamma^4$. But whether the ambiguity contained in this table is real or not it is sometimes quite difficult to decide.

The present section will deal with testing and diagnostic methods at run time. These procedures were first published in an algorithmic form by C.R. Muthukrishnan and V. Rajaraman [B7] and can serve as a complementary way to the previously suggested methods to solve completely the problem of ambiguity in decision table logic. Note that these algorithms can be used with masked method only.

The following two algorithms can serve as conversion techniques as well as testing and diagnostic techniques.

**ALGORITHM V-2-1:** - For converting and testing LCE decision table.

**STEP 1:** GENERATE A MASK MATRIX M BY SUBSTITUTING THE FOLLOWING CODES FOR THE CONDITION ENTRIES IN THE CONDITION ENTRY MATRIX.
\[ M_{ij} = \begin{cases} 
1 & \text{if } c_{ij} = Y \\
0 & \text{if } c_{ij} = N \\
1 & \text{if } c_{ij} = "-" \text{ or don't care.}
\end{cases} \]

For example, the corresponding mask matrix for the table given in Fig. V-2-1 is:

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

**STEP 2:** FOR GIVEN DATA, THE CONDITIONS ARE TESTED AND THE RESULTS ARE RECORDED IN A RESULT VECTOR D USING THE SAME CODES AS IN STEP 1.

**STEP 3:** SELECT THE ROWS OF M WHICH CORRESPOND TO THE 1 ENTRIES IN D, AND LOGICALLY "AND" THE RESULTING ELEMENTS IN A GIVEN POSITION TO A RESULT ROW VECTOR R OF ORDER P.

**STEP 4:** IF R_j = 0, j = 1, 2, ..., P, THE "ELSE RULE" APPLIES. IF R_k = 1 for a unique k, THEN \( \gamma^k \) RULE APPLIES. IF MORE THAN ONE ELEMENT is equal to 1, A REAL AMBIGUITY BETWEEN THE CORRESPONDING RULES EXISTS; REDUNDANCY IF THE CORRESPONDING ACTIONS ARE THE SAME AND CONTRADICTION IF THE CORRESPONDING ACTIONS ARE DIFFERENT.

**Example V-2-1:** - Using the decision table given in Fig. V-2-1;
1. Suppose $c_1$ is false, $c_2$ true and $c_3$ is false; then:

\[
D = \begin{bmatrix}
0 \\
1 \\
0 \\
1
\end{bmatrix}
\]

The corresponding rows in $M$ are

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0
\end{array}
\]

$R = [0 \ 1 \ 0 \ 0]$  

which points to the fact that $\gamma^2$ is satisfied and there is no ambiguity.

2. If $c_1$ is true, $c_2$ is true and $c_3$ is false; then:

\[
D = \begin{bmatrix}
1 \\
0 \\
1 \\
0 \\
1
\end{bmatrix}
\]

the corresponding rows in $M$ are

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0
\end{array}
\]

$R = [1 \ 1 \ 0 \ 0]$  

which points out that ambiguity between $\gamma^1$ and $\gamma^2$ exists and since they have the same action, redundancy exists.

3. If $c_1$ is true, $c_2$ is true and $c_3$ is true; then:

\[
D = \begin{bmatrix}
1 \\
0 \\
0 \\
1
\end{bmatrix}
\]

the corresponding rows in $M$ are

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1
\end{array}
\]

$R = [0 \ 0 \ 1 \ 1]$
\[ D_{ij} = \begin{cases} 1 & \text{if the } i^{th} \text{ condition is satisfied in } \gamma^j \\ 0 & \text{otherwise (including the don't care case)} \end{cases} \]

**STEP 3:** LET \( T = D \) OR \( M \)

That is \( t_{ij} = d_{ij} \) OR \( m_{ij} \)

**STEP 4:** LOGICALLY "AND" THE RESULTING ELEMENTS OF THE MATRIX \( T \) COLUMNWISE.

\[
\bigcap_{i=1}^{q} t_{ij} = R_{j}.
\]

THE RESULTING ROW VECTOR \( R \) HAS \( 1 \)'s IN EVERY POSITION FOR WHICH THE DATA SATISFIES THE RESPECTIVE RULE.

**Example V-2-3:** Referring to the MCE decision table given in Fig. V-1-1;

1. Suppose \( c_1 \) is 35, \( c_2 \) is true and \( c_3 \) is "-" then

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

\[
D \text{ or } M = \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0
\end{bmatrix} \quad \text{or} \quad \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}
\]

\[
R = [0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]
\]

which points out that ambiguity between \( \gamma^4 \) and \( \gamma^5 \) exists.

Since the actions of these two rules are different, contradiction exists. Note that this was discovered in Example V-1-2 as a possible ambiguity.
which points out that ambiguity between $\gamma^3$ and $\gamma^4$ exists and since they have different actions, the rules contain contradiction.

The proof of the algorithm is based on the fact that in order to satisfy a certain rule the corresponding outcome after the AND operation must be "1". Since the code for a don't care entry in the condition entry matrix is \( \frac{1}{1} \) this will not affect result of the AND operation.

**Algorithm V-2-2**: For Conversion and Testing MCE Decision Tables. The steps of this algorithm will be illustrated, whenever it is felt necessary, by applying them to the MCE decision table given in Fig. V-1-1.

**Step 1**: Generate a qxp mask matrix $M$ by substituting the following codes for the condition entries $c_{ij}$ in the condition entry matrix.

$$m_{ij} = \begin{cases} 0 & \text{if } c_{ij} = Y \text{ or } c_{ij} = N \text{ or } v_{ij} \\ 1 & \text{otherwise} \end{cases}$$

The corresponding $M$ matrix to the decision table given in Fig. V-1-1 is:

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

**Step 2**: For given data perform the tests and generate a qxp matrix using the following code.
2. Suppose $c_1 = 30$, $c_2$ is false and $c_3$ is ">"; then:

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

\[
D \text{ or } M = \begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \text{ or } \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[R = [0 0 1 0 0 0 0] \]

This points out that the $\gamma^3$ is satisfied.

**Note:** Conditions that have the same value in the condition entry matrix do not have to be tested more than once. For example, as soon as we found that $c_1$ is satisfied by 35 we can insert 1 in $c_{14}$, $c_{15}$ and $c_{16}$.

Now we shall show that with some small modification the same algorithm can be used for testing at conversion time and at run time.

**MODIFICATION FOR ALGORITHM V-2-1:**

The only modification that has to be made in order to use this algorithm at conversion time for checking possible ambiguities, is in the 2nd step. Instead of generating a result vector after testing real data, we can generate in a loop all the possible outcomes ($2^q$) of result vectors and apply them to the mask matrix in the same manner as described in algorithm V-2-1. This will give us another
way to indicate possible ambiguities at conversion time.

MODIFICATION FOR ALGORITHM V-2-2:

Here the modification can be made in the same way as proposed previously for algorithm V-2-1. But here, instead of generating a possible result vector, we have to generate in a loop, a possible result matrix. In this case the number of possible matrices becomes very large (\( \prod_{j=1}^{q} n_j \) — see definition II-2-11) and time consuming. Therefore, for MCE decision tables the use of theorems V-1-7 and V-1-8 is highly recommended.

REMARKS:

1. The two algorithms presented above are again interpretive in nature, and therefore more suited for interpreters to be used in time sharing or real time systems.

2. Both algorithms require testing all the conditions before deciding which rule is being satisfied and whether or not ambiguity of any kind exists. Therefore, a price of longer run time is paid for the advantage of being able to test for ambiguities at run time.

3. Both algorithms can be modified as suggested by King [B2] in his interrupted rule mask procedure, for the Kirk's procedure. But by doing so, we lose the ability of detecting ambiguities in run time since, in this procedure, the first rule satisfied will be taken. On the other hand this provides a way
to save run time.

4. In algorithm V-1-2 we could eliminate the $T = (D \text{ or } M)$ operation by initiating $D$ to the values of $M$ and save some time by eliminating further testing of conditions for a rule $\gamma_i^j$ as the condition entry $c_{ij}$ is found to be unsatisfied.

5. These two algorithms, as the other, discussed in the mask method, in chapter IV, are very efficient as far as the required storage is concerned.

**V-3 TESTING AND DIAGNOSTIC METHODS IN CONVERSION TIME AND IN RUN TIME**

In this section we shall introduce another two algorithms capable to perform testing in conversion time and run time as well as to serve as conversion algorithms. The use of conversion algorithms such as described in this section will allow conversion together with debugging runs to be run in a continuous sequence giving all the necessary diagnostics for fast and efficient debugging.

**ALGORITHM V-2-3:** For conversion of LCE decision tables (with and without related conditions) and testing in conversion time and run time whenever the table contains possible ambiguities. This algorithm is based upon the mask technique given in IV-2-1 and theorems V-1-3, V-1-4.

The steps of the algorithm are divided into two groups, the first group is executed in conversion time while the second, based upon the first, is executed in run time.
GROUP 1: IN CONVERSION TIME

STEP 1: GENERATE A DECISION MATRIX D BY SUBSTITUTING THE FOLLOWING CODES FOR THE CONDITION ENTRIES IN THE CONDITION ENTRY MATRIX.

\[ d_{ij} = \begin{cases} 1 & \text{if } c_{ij} = \text{Y or YES} \\ 0 & \text{otherwise} \end{cases} \]

STEP 2: GENERATE A MASK MATRIX M BY SUBSTITUTING THE FOLLOWING CODES FOR THE CONDITION ENTRIES IN THE CONDITION ENTRY MATRIX.

\[ m_{ij} = \begin{cases} 0 & \text{if } c_{ij} = \text{"-" or don't care} \\ 1 & \text{otherwise} \end{cases} \]

STEP 3: CHECK FOR EVERY TWO RULES \( \gamma^i, \gamma^j \) IF \( m_i \cap d_j = m_j \cap d_i \) AND IF \( \delta^i = \delta^j \) GENERATE A DIAGNOSTIC MESSAGE FOR POSSIBLE REDUNDANCY BETWEEN RULES \( \gamma^i \) AND \( \gamma^j \) FOR DATA WHICH SATISFYS THE CONDITIONS REPRESENTED BY THE 1's IN THE COLUMN VECTOR \( (m_i \cup m_j) \cap (d_i \cup d_j) \). GENERATE A TEST RESULT VECTOR TR IDENTICAL TO \( (m_i \cup m_j) \cap (d_i \cup d_j) \) WITH AN ADDITIONAL TAG \( (i, j, k) \) WHERE \( i, j \) ARE THE RULE NUMBERS AND

\[ k = \begin{cases} 0 & \text{FOR REDUNDANCY} \\ 1 & \text{FOR CONTRADICTION} \end{cases} \]

TO BE USED IN RUN TIME. NOTE THAT HERE \( k = 0 \). STORE THIS RECORD IN A TABLE.

2. IF \( m_i \cap d_j = m_j \cap d_i \) AND \( \delta^i \neq \delta^j \) GENERATE A DIAGNOSTIC MESSAGE FOR POSSIBLE CONTRADICTION BETWEEN RULES \( \gamma^i \)
AND \( \gamma_j \) FOR DATA WHICH SATISFYS THE CONDITIONS REPRESENTED BY THE 1's IN THE COLUMN VECTOR \((m_i \cup m_j) \cap (d_i \cup d_j)\). GENERATE A TEST RESULT VECTOR TR IDENTICAL TO \((m_i \cup m_j) \cap (d_i \cup d_j)\) WITH AN ADDITIONAL TAG \((i,j,k)\) AS FOR 1, HERE \(k = 1\).

STORE THAT VECTOR IN A TABLE.

**GROUP 2: IN RUN TIME**

**STEP 4:** FOR GIVEN DATA, THE CONDITIONS ARE TESTED AND THE RESULTS ARE RECORDED IN A RESULT VECTOR RV USING THE SAME CODES AS FOR THE DECISION MATRIX.

**STEP 5:** IF THE TABLE IN WHICH THE TR VECTORS ARE STORED IN CONVERSION TIME IS NOT EMPTY COMPARE THE RESULT VECTOR RV WITH EVERY VECTOR IN THE TABLE. FOR EVERY MATCH GENERATE A RUN TIME DIAGNOSTIC MESSAGE LISTING THE AMBIGUOUS DATA, REDUNDANCY IF \(k = 0\), CONTRADICTION IF \(k = 1\). STATE ALSO THE RULE NUMBER i AND j.

**STEP 6:** IF THE TEST RESULT VECTOR TABLE IS EMPTY OR NO MATCH OCCUR IN STEP 5 PERFORM THE SCANNING OPERATION (DESCRIBED IN IV-2-1) IN ORDER TO FIND WHICH RULE IS SATISFIED AND TAKE THE APPROPRIATE ACTION.

**ALGORITHM V-2-4:** - For conversion of MCE decision tables (with and without related conditions) and testing in conversion time and run time whenever the table contains possible ambiguities. This algorithm is based upon Peel's Extended Mask Technique given in page 92 and theorems V-1-7, V-1-8.
The steps of this algorithm are also divided into two groups, the first group is executed in conversion time and the second in run time.

GROUP 1: IN CONVERSION TIME.

STEP 1: GENERATE A DECISION MATRIX D AS IN PEEL'S EXTENDED MASK TECHNIQUE.

STEP 2: GENERATE A MASK MATRIX M AS IN PEEL'S EXTENDED MASK TECHNIQUE.

STEP 3: CHECK FOR EVERY TWO RULES \( \gamma^i, \gamma^j \) IF \( m_i \times d_j = m_j \times d_i \)

1. IF \( m_i \times d_j = m_j \times d_i \) AND IF \( \delta^i = \delta^j \) GENERATE A DIAGNOSTIC MESSAGE FOR POSSIBLE REDUNDANCY BETWEEN RULES \( \gamma^i \) AND \( \gamma^j \) FOR DATA WHICH SATISFIES THE CONDITIONS REPRESENTED BY THE VECTOR \( (m_i \cup m_j) \times (d_i \not\subseteq d_j) \). GENERATE A TEST RESULT VECTOR TR IDENTICAL TO \( (m_i \cup m_j) \times (d_i \not\subseteq d_j) \) WITH AN ADDITIONAL TAG \( (i, j, k) \) SAME AS IN STEP 3(1) IN ALGORITHM V-2-3, AND STORE IT IN A TABLE.

2. IF \( m_i \times d_j = m_j \times d_i \) AND \( \delta^i \neq \delta^i \) GENERATE A MESSAGE FOR POSSIBLE CONTRADICTION BETWEEN RULES \( \gamma^i \) AND \( \gamma^j \) FOR DATA WHICH SATISFIES THE CONDITIONS REPRESENTED BY THE VECTOR \( (m_i \cup m_j) \times (d_i \not\subseteq d_j) \). GENERATE A TEST RESULT VECTOR TR SAME AS IN STEP 3(2) IN ALGORITHM V-2-3, AND STORE IT IN A TABLE.

GROUP 2: IN RUN TIME.

STEP 4: FOR GIVEN DATA, THE CONDITIONS ARE TESTED AND THE RESULTS
ARE RECORDED IN A RESULT VECTOR RV USING THE SAME CODES AS FOR THE RESULT VECTOR IN PEEL'S EXTENDED MASK TECHNIQUE.

STEP 5: SAME AS STEP 5 IN ALGORITHM V-1-3.

STEP 6: SAME AS STEP 6 IN ALGORITHM V-1-3.

REMARKS:

1. The main advantage in the last two algorithm is the fact that they incorporate in one; conversion, conversion time testing and run time testing almost without increasing the conversion time or the program run time and storage required compare to Kirk's mask technique or Peel's extended mask technique.

2. The only disadvantage is that Kirk's mask technique as well as Peel's extended mask technique require the testing of every condition which is inefficient in run time.

3. One way to make use of these last algorithms is by inclusion two additional rules denoted by A (for ambiguity) and R (for redundancy) which when satisfied specify the actions to be taken in either case.
CHAPTER VI. PROPOSED CONVERSION METHOD

In this chapter, an attempt is made to propose an efficient and flexible conversion method based upon the experience accumulated in the previous chapters. We can clearly state that none of the previously suggested methods, procedures and algorithms, for conversion of decision tables to program code and for checking ambiguity and completeness, can be used separately in a real attempt to write a good decision table processor. We shall first state the main requirements of a good processor in order of priority. This list of requirements will be the guiding list and the reason standing behind the suggested approach. Finally, we shall give an example in which the conversion will be made manually as if done by a preprocessor converting the tables into COBOL program code.

VI-1 REQUIREMENTS:

1. Minimum requirement limitations and restrictions imposed on the user.
2. Efficient conversion time checking procedures and a comprehensive system of diagnostic messages.
3. Efficient program code with optimal combinations of minimal run time and storage requirements.
4. Possible run time checking procedures for debugging mode runs.
5. Possible inclusion of additional information (such as probability distribution of the rules and storage required) for optimization
of the program code generated.

6. Possible elimination of the programming job.

7. Efficient decision table processor.

The most essential requirement, as stated above, is maximum flexibility. Fulfilling this requirement can eventually lead to possible elimination of the programming job. Using third generation computers, where machine time and memory are becoming cheaper and cheaper, and the price paid for programming becomes more and more expensive, these requirements state that the reduction and/or the elimination of the programming job is most desirable even if this means relatively longer conversion time and execution time. We would like to design and implement a decision table translator which can cope with the following functions:

1. Accept MCE decision tables, with and without auxiliary information. Easy to learn and easy to use.

2. Be able to copy standard portions of program code from standard library.

3. Have the necessary interface to database generator.

4. Generate as efficient program code as possible, without imposing heavy restrictions or requirements upon the user.

5. Have efficient procedure for testing and generating diagnostic messages in conversion time, and upon requirement in run time as well, in a special debugging mode run.


7. The decision table processor should be written in a modular form
(as far as possible) to allow easy modification by the potential user.

VI-2 SUGGESTED DECISION TABLE COBOL PRE-COMPILE SYNTACTIC STRUCTURE

In this section we shall deal with the formation of rules for writing decision tables utilizing the COBOL language. The COBOL language was chosen in this particular example because this is the most useful language in today's computer applications. Besides a decision table processor written in a high level language is relatively machine independent. As stated before, an actual pre-compiler was not written because of lack of resources, but the presented suggestions can serve as nuclei in implementing one, and were used to generate the example of this chapter.

A COBOL pre-compiler must be able to accept any COBOL program intermixed with decision tables as input, from which an efficient COBOL program will be generated.

The syntax of the precompiler may contain few reserved words and symbols which must not be reserved nor have any specific meaning in any standard COBOL language specifications (COBOL 60, COBOL 61, COBOL 65, USASI and CIDASYL COBOL).

The pre-compiler must be able to accept three types of input data (or statements).
1. CONTROL STATEMENTS.

2. DECISION TABLES.

3. STANDARD COBOL STATEMENTS.

In the rest of the section we will make use of the Bachus normal form (with some modifications) to describe the syntax required by the pre-compiler. The conversions used are:

a. `< >` left and right broken brackets are used to contain a variable.

b. `::=` Means "IS DEFINED AS".

c. Means "OR"

d. `[ ]` Left and right brackets are used to indicate "AND"

Note: Some of the following is Burroughs oriented since I am most familiar with the Burroughs computers.

**CONTROL STATEMENTS**

Control statements can be supplied to the precompiler by means of punch cards and/or card images on magnetic tape or disk. The control statements supply important information to the precompiler as well as set and reset different options. Most of the control options are optional and must have a default value. The general format of a control card is as follows:

**COLUMN:**

1-6 STANDARD COBOL CARD SEQUENCE

7 BLANK

8 # (CONTROL CARD ID)
9-72 CONTROL OPTIONS (in free format)
73-80 STANDARD COBOL IDENTIFICATION.

SYNTAX:

CONTROL OPTION :: = INPUT OPTIONS<OUTPUT OPTIONS>
                      <TESTING OPTIONS><EFFICIENCY OPTIONS>
                      <DEBUGGING OPTIONS><WORK-FILES OPTIONS>
                      <COMPILE OPTIONS>

INPUT OPTIONS :: = MAIN INPUT OPTIONS<LIBRARY OPTIONS>

MAIN INPUT OPTIONS :: = CARD | TAPE | DISK | EMPTY

LIBRARY OPTIONS :: = EMPTY | TAPELIB | DISKLIB

OUTPUT OPTIONS :: = LISTING OPTIONS<CARD-OUTPUT OPTIONS>
                      TAPE-OUTPUT OPTIONS<DISK-OUTPUT OPTIONS>
                      XREF OPTIONS

LISTING OPTIONS :: = REPORT OPTIONS<EDITING OPTIONS>

REPORT OPTIONS :: = EMPTY|LIST|LIST-1

EDITING OPTIONS :: = EMPTY|LINE OPTIONS<PAGE OPTIONS>

LINE OPTIONS :: = EMPTY|SINGLE|DOUBLE

PAGE OPTION :: = EMPTY|NEWPAGE

CARD-OUTPUT OPTIONS :: = EMPTY|PUNCH|PUNCH-1

TAPE-OUTPUT OPTIONS :: = EMPTY|NEWTAPE|NEWTAPE-1

DISK-OUTPUT OPTIONS :: = EMPTY|NEWDISK|NEWDISK-1

TREE OPTIONS :: = EMPTY|XREF

TESTING OPTIONS :: = EMPTY|CT-TEST|RT-TEST

EFFICIENCY OPTIONS :: = EMPTY FREQUENCY OPTIONS<MEMORY OPTIONS>
                      <MINIMIZATION OPTIONS><COST OPTIONS>
<FREQUENCY OPTIONS> :: = EMPTY|FREQ.
<MEMORY OPTION> :: = EMPTY|MEM.
<MINIMIZATION OPTIONS> :: = EMPTY|MINIMIZE
<COST OPTIONS> :: = EMPTY|COST
<DEBUGING OPTIONS> :: = EMPTY|DEBUG

<WORK-FILES OPTIONS> :: = EMPTY|WD3|WD2TP1 | WD1TP2 | TP3
<COMPILE OPTIONS> :: = EMPTY|SYNCOM|RUNCOM|LIBCOM

REMARKS:

1. Control cards can be inserted anywhere in the input string to reset different options with one exception. The work-files option may appear only once in the first set of control cards in front of the pre-compiler input.

2. All the options can be pre-set at the time of the pre-compiler compilation. A suggested list of pre-set values is: CARD, LIST, SINGLE, CT-TEST, TP3, SYNCON.

3. If necessary more than one control card may be used.

SEMANTICS:

The functions of the different control options available are as follows:

INPUT OPTIONS: The precompiler must be able to accept its primary input form cards, tape or decision tables and may be copied from a standard library.

OUTPUT OPTIONS: Up to four different output files may be produced
simultaneously. LIST stands for a listing of the generated program excluding the original decision tables. LIST-1 stands for a complete listing including the decision tables listed as notes for documentation purposes. SINGLE and DOUBLE stand for single and double line skipping, NEWPAGE stands for a new page for every new decision tables. PUNCH and PUNCH-1, NEWTAPE and NEWTAPE-1, DISK and NEWDISK-1 are the same as LIST and LIST-1. The XREF option allows the user to receive cross-reference of the generated program. On request test for ambiguity can be done in conversion time or different procedures will be incorporated to do so in run time as well. This is a debugging mode of operation and as soon as the user is sure that the program is free of ambiguities, it should perform another conversion and compile pass to return back to normal mode of operation. Using the RT-TEST option additional and inefficient code will be incorporated to perform the required test.

EFFICIENCY OPTIONS: If the probability distribution of the rules and the storage required for testing the different conditions is known, they may be supplied to the pre-compiler and taken into account in the conversion pass, on request, to generate a more effi-
cient code whenever possible. Possible minimization of tables should be also available.

**DEBUGGING OPTIONS:** If the DEBUG option is set together with LIST-1 some additional notes (including the mask and the decision matrices) will be listed for every decision table.

**WORKING-FILES OPTIONS:**

**COMPILE OPTIONS:**

Three different options are available. The pre-compiler may start the COBOL compiler, after conversion, for syntax check, for compile and run or for compile for library.

**DECISION TABLES; PROGRAM FORMAT:**

The format of the program must be according to the COBOL requirement except from the control cards which can appear in one place and the decision tables which are inserted in the procedure division as sections and referred to by an appropriate COBOL statement (GO TO, PERFORM or simply in line code).

The IDENTIFICATION DIVISION and the ENVIRONMENT DIVITION are treated as in any standard COBOL program. The DATA DIVISION in a program containing decision tables must contain every data name, constant or conditional variable referenced in the procedure division code or decision tables. Inconsistencies will be discovered only in the compilation pass.
CODE AND TABLE LINKAGE

In the procedure division COBOL statement can be intermixed with decision tables. Every decision table is translated into a COBOL section code and decision tables can be linked by any of the linkages described in chapter III.

FORMAT OF THE TABLES

The suggested format for a decision table is demonstrated in the table given at the end of this chapter. Due to the limitations, in size, of punched cards (80 columns) a special way was designed for extended condition entries and extended action entries.

The program generated from a decision table is illustrated in the example given at the end of this chapter.

ALGORITHMS WHICH SHOULD BE USED

An efficient decision table precompiler must be able to use different procedures with different decision tables and according to the user request. Therefore the following is suggested:

1. For standard LCE-MAE-decision tables: Press's or Pollack's procedures should be used if run-time efficiency is desired, and Kirk's mask procedure if minimum storage is desired.

2. For LCE-MAE decision tables where the cost for testing different conditions together with the rule probability distribution is supplied: Reinwald and Soland's procedure is to be used if run
time efficiency is desired; otherwise, the interrupted rule mask procedure (by King) should be used.

Note: In every one of the previous cases the procedures given by section V-2 should be used if run time diagnostics are required.

3. For MCE-MAE decision tables with and without cost and probability information, a special procedure is suggested here (and demonstrated at the end of this chapter).

**STEP 1:** Sort the conditions into two groups, LCE and ECE.

**STEP 2:** Use one of the tree procedures suggested in 1 or 2 for the LCE (depending on the type of the table) to partition the ECE into groups. The table is split by branching.

**STEP 3:** If cost and probability is given, use interrupted rule mask procedure accordingly; otherwise, use interrupted rule mask procedure dictated by the don't care entries.

**STEP 4:** Unsuccessful tests in any of the ECE groups leads to the ELSE RULE (if any); otherwise, to a standard error exit.

Note: If probabilities are given, the probability of every branch in the LCE is the sum of the probabilities of the rules in the appropriate group.

This procedure cannot produce run time diagnostics but can give the possible ambiguities included in the table.

If run time diagnostics are required, the procedures described in section V-2 must be used.
VI-3 DECISION TABLE PRE-COMPILED GENERAL BLOCK DIAGRAM
AND FILE CHART.

The first of the following two diagrams is a macro flow chart
describing the different modules of the suggested COBOL pre-compiler.
No doubt that the real implementation will probably require signifi-
cant modifications even to this general block diagram.

The file chart suggests a possible way of file utilization.
All the files are described as disk files, actually any kind of
combination of disk files and magnetic tape files can be used.
DECISION TABLE COBOL PRE-COMPILER

FILE CHART

INPUT CARD

MODULE I

SOURCE LIBRARY

MODULE II

FILE-A

FILE-B

MODULE IV

FILE-A CONCATENATED TO FILE B

MODULE III

NEWTAPE-NEWTAPE-1

PUNCH PUNCH-1

LIST LIST-1 XREF

COBOL COMPILER
VI-4  EXAMPLE:

The following example was designed to illustrate a possible use of the suggested COBOL pre-compiler. The example does not contain all the suggested options but can serve as sample. The example contains the following:

1. Input to the (suggested) precompiler.
2. LIST-1 output from the pre-compiler which was prepared by simulating the pre-compiler operations manually.
3. An actual compilation of the generated program (done on the Burroughs B-5500 computer).
4. Test data.
5. Run.

Note that run time diagnostics were not demonstrated.

VI-4-1  PRE-COMpiler INPUT.

The pre-compiler input was designed to illustrate the simplicity of the input, and the suggested format for writing decision tables with extended condition entry and extended action entry.
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td># CARD TAPELIB LIST=1 SINGLE XREF CT=TEST FREQ COST RUNCMD.</td>
</tr>
<tr>
<td>00010</td>
<td>IDENTIFICATION DIVISION.</td>
</tr>
<tr>
<td>00012</td>
<td>PROGRAM-ID. &quot;TEST MCE-MAE&quot;.</td>
</tr>
<tr>
<td>00013</td>
<td>AUTHOR. A. PINHAS.</td>
</tr>
<tr>
<td>00014</td>
<td>DATE-WRITTEN 29 DEC. 1970.</td>
</tr>
<tr>
<td>00015</td>
<td>DATE-COMPILED.</td>
</tr>
<tr>
<td>00016</td>
<td>REMARKS. THIS PROGRAM WAS WRITTEN MANUALLY SIMULATING A.</td>
</tr>
<tr>
<td>00017</td>
<td>SUGGESTED DECISION TABLE PRE-COMPILER, THE PRE.</td>
</tr>
<tr>
<td>00018</td>
<td>COMPIATION PASS WAS PREVIOUSLY DONE WITH THE.</td>
</tr>
<tr>
<td>00019</td>
<td>FOLLOWING OPTIONS SET:</td>
</tr>
<tr>
<td>00021</td>
<td>1. INPUT FROM CARDS.</td>
</tr>
<tr>
<td>00022</td>
<td>2. OUTPUT LIST=1 PUNCH=1.</td>
</tr>
<tr>
<td>00023</td>
<td>3. TEST. CT=TEST.</td>
</tr>
<tr>
<td>00024</td>
<td>4. FREQ.</td>
</tr>
<tr>
<td>00025</td>
<td>5. COST.</td>
</tr>
<tr>
<td>00026</td>
<td>ENVIRONMENT DIVISION COPY ENV-DIV=1 FROM TAPELIB.</td>
</tr>
<tr>
<td>00027</td>
<td>DATA DIVISION COPY DAT-DIV=1 FROM TAPELIB.</td>
</tr>
<tr>
<td>00029</td>
<td>PROCEDURE DIVISION.</td>
</tr>
<tr>
<td>00030</td>
<td>INIT.</td>
</tr>
<tr>
<td>00031</td>
<td>OPEN INPUT CARDS.</td>
</tr>
<tr>
<td>00032</td>
<td>OPEN OUTPUT LISTING. PERFORM HEADINGS.</td>
</tr>
<tr>
<td>00033</td>
<td>READ-LOOP.</td>
</tr>
<tr>
<td>00034</td>
<td>READ CARDS AT END GO TO LAST.</td>
</tr>
<tr>
<td>00036</td>
<td>PERFORM IS-MCE-MAE-DT.</td>
</tr>
<tr>
<td>00038</td>
<td>PERFORM WRITE-LISTING.</td>
</tr>
<tr>
<td>00039</td>
<td>GO TO READ-LOOP.</td>
</tr>
<tr>
<td>00109</td>
<td>TABLE=ID. &quot;IS-MCE-MAE-DT&quot; MCE-MAE.</td>
</tr>
<tr>
<td>00110</td>
<td>R11 0000000001111111112222222223E123456789012345678901234567890L</td>
</tr>
<tr>
<td>00112</td>
<td>R12 123456789012345678901234567890L</td>
</tr>
<tr>
<td>00113</td>
<td>CONDITIONS</td>
</tr>
<tr>
<td>00114</td>
<td>L IF G=&quot;PINK&quot; Y-YNYY-NYNN=YYN=YNN-YYLY=NYN= 50</td>
</tr>
<tr>
<td>00121</td>
<td>L IF (X*2+3)=97 YNNYNYNNYNYYNNYNYYNNYNYYNNYYNNYYNYYNNYYNNYYNYYNNYYNYYNNYY 68</td>
</tr>
<tr>
<td>00122</td>
<td>E IF C 15 NUMERIC Y-YYYY-Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00123</td>
<td>ALPHARETIC Y-YYYY-Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00124</td>
<td>ALPHANUMERIC Y-YYYY-Y-YYYY-Y-YYYY-Y-/YYYY-Y-</td>
</tr>
<tr>
<td>00125</td>
<td>E IF COULR &quot;BLUE&quot; Y-YYYY-Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00126</td>
<td>&quot;RED&quot; Y-YYYY-Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00127</td>
<td>&quot;GREEN&quot; Y-YYYY-Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00128</td>
<td>&quot;WHITE&quot; Y-YYYY-Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00129</td>
<td>L IF VALUE=I YNN-YNY-NY=NYY=N=Y-NN=NY= 25</td>
</tr>
<tr>
<td>00130</td>
<td>E IF NUMBER IS POSITIVE Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00131</td>
<td>NEGATIVE Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00132</td>
<td>ZERO Y-YYYY-Y-YYYY-Y-YYYY-Y-</td>
</tr>
<tr>
<td>00133</td>
<td>ACTIONS</td>
</tr>
<tr>
<td>00135</td>
<td>L MOVE &quot;0&quot; TO R11 XXXXXXXX</td>
</tr>
<tr>
<td>00137</td>
<td>L MOVE &quot;1&quot; TO R11 XXXXXXXX</td>
</tr>
<tr>
<td>00139</td>
<td>L MOVE &quot;2&quot; TO R11 XXXXXXXX</td>
</tr>
<tr>
<td>00141</td>
<td>L MOVE &quot;3&quot; TO R11 XXXXXXXX</td>
</tr>
</tbody>
</table>
02816 WRITE PRINTER-LINE FROM GROUP=CNT-LINE AFTER ADVANCING 2
02817 WRITE PRINTER-LINE FROM RULE-COUNT-LINE=1 AFTER ADVANCING 2
02818 WRITE PRINTER-LINE FROM RULE-COUNT-LINE=2 AFTER ADVANCING 2
02819 WRITE PRINTER-LINE FROM RULE-COUNT-LINE=3 AFTER ADVANCING 2
02820 WRITE PRINTER-LINE FROM RULE-COUNT-LINE=4 AFTER ADVANCING 2
02821 WRITE PRINTER-LINE FROM RULE-COUNT-LINE=5 AFTER ADVANCING 2
02822 CLOSE CARDIN LISTING WITH RELEASE.
02823 STOP RUN.
99999 END-OF-JOB.
VI-4-2 COBOL PRE-COMPILED OUTPUT.

The pre-compiler output includes LIST-1, in which the original decision table together with conversion time diagnostics are listed under COBOL NOTES, as well as a cross reference of the program.
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101</td>
<td>IDENTIFICATION DIVISION.</td>
<td></td>
</tr>
<tr>
<td>00102</td>
<td>PROGRAM-ID. &quot;TEST_MCE-MAE&quot;.</td>
<td></td>
</tr>
<tr>
<td>00103</td>
<td>AUTHOR. A. PINOAS</td>
<td></td>
</tr>
<tr>
<td>00104</td>
<td>DATE-WRITTEN 20, DEC 1979.</td>
<td></td>
</tr>
<tr>
<td>00105</td>
<td>DATE-COMPiled.</td>
<td></td>
</tr>
<tr>
<td>00106</td>
<td>REMARKS. THIS PROGRAM WAS WRITTEN MANUALLY SIMULATING A</td>
<td></td>
</tr>
<tr>
<td>00107</td>
<td>SUGGESTED DECISION TABLE PRE-COMPILED, THE PRE-</td>
<td></td>
</tr>
<tr>
<td>00108</td>
<td>COMPIlation PASS WAS PRESUMABLY DONE WITH THE</td>
<td></td>
</tr>
<tr>
<td>00109</td>
<td>FOLLOWING OPTIONS SET:</td>
<td></td>
</tr>
<tr>
<td>00110</td>
<td>1. INPUT FROM CARDS</td>
<td></td>
</tr>
<tr>
<td>00111</td>
<td>2. OUTPUT LIST=1 PUNCH=L</td>
<td></td>
</tr>
<tr>
<td>00112</td>
<td>3. TEST C/E TEST</td>
<td></td>
</tr>
<tr>
<td>00113</td>
<td>4. FREQ</td>
<td></td>
</tr>
<tr>
<td>00114</td>
<td>5. COST</td>
<td></td>
</tr>
<tr>
<td>00115</td>
<td>ENVIRONMENT DIVISION.</td>
<td></td>
</tr>
<tr>
<td>00116</td>
<td>CONFIGURATION SECTION.</td>
<td></td>
</tr>
<tr>
<td>00117</td>
<td>SOURCE-COMPUTER, R=5500.</td>
<td></td>
</tr>
<tr>
<td>00118</td>
<td>OBJECT-COMPUTER, R=5500.</td>
<td></td>
</tr>
<tr>
<td>00119</td>
<td>INPUT-OUTPUT SECTION.</td>
<td></td>
</tr>
<tr>
<td>00120</td>
<td>FILE-CONTROL.</td>
<td></td>
</tr>
<tr>
<td>00121</td>
<td>SELECT CARDFIN ASSIGN TO CARD-READER.</td>
<td></td>
</tr>
<tr>
<td>00122</td>
<td>SELECT LISTING ASSIGN TO PRINTER.</td>
<td></td>
</tr>
<tr>
<td>00123</td>
<td>DATA DIVISION.</td>
<td></td>
</tr>
<tr>
<td>00124</td>
<td>FILE SECTION.</td>
<td></td>
</tr>
<tr>
<td>00125</td>
<td>FD CARDIN</td>
<td></td>
</tr>
<tr>
<td>00126</td>
<td>VALUE OF ID IS &quot;CARDS&quot;</td>
<td></td>
</tr>
<tr>
<td>00127</td>
<td>DATA RECORD IS INPUT-CARD.</td>
<td></td>
</tr>
<tr>
<td>00128</td>
<td>01 INPUT-CARD</td>
<td></td>
</tr>
<tr>
<td>00129</td>
<td>SIZE 80.</td>
<td></td>
</tr>
<tr>
<td>00130</td>
<td>02 A</td>
<td>PICTURE 99.</td>
</tr>
<tr>
<td>00131</td>
<td>02 B</td>
<td>PICTURE 99.</td>
</tr>
<tr>
<td>00132</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00133</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00134</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00135</td>
<td>02 E</td>
<td>PICTURE 99.</td>
</tr>
<tr>
<td>00136</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00137</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00138</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00139</td>
<td>02 G</td>
<td>PICTURE 99.</td>
</tr>
<tr>
<td>00140</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00141</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00142</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00143</td>
<td>02 X</td>
<td>PICTURE 99.</td>
</tr>
<tr>
<td>00144</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00145</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00146</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00147</td>
<td>02 NOUER</td>
<td>PICTURE 99.</td>
</tr>
<tr>
<td>00148</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00149</td>
<td>02 VAL</td>
<td>PICTURE XX.</td>
</tr>
<tr>
<td>00150</td>
<td>02 NOUER</td>
<td>PICTURE 99.</td>
</tr>
<tr>
<td>00151</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00152</td>
<td>02 FILLER</td>
<td>PICTURE X(5).</td>
</tr>
<tr>
<td>00153</td>
<td>02 RULR-ID</td>
<td>PICTURE X(4).</td>
</tr>
<tr>
<td>00154</td>
<td>02 RULE-ID</td>
<td>PICTURE X(4).</td>
</tr>
<tr>
<td>00155</td>
<td>02 LISTING</td>
<td>VALUE OF ID IS &quot;REPORT&quot;.</td>
</tr>
<tr>
<td>00156</td>
<td>02 DATA RECORD IS PRINTER-LINE.</td>
<td></td>
</tr>
<tr>
<td>00157</td>
<td>01 PRINTER-LINE</td>
<td>SIZE 132.</td>
</tr>
</tbody>
</table>
00616 02 COUNTER-1 PICTURE 9(3) VALUE 0.
00617 02 FILLER PICTURE X(17) VALUE " COUNTER-2 = ".
00618 02 COUNTER-2 PICTURE 9(3) VALUE 0.
00619 02 FILLER PICTURE X(17) VALUE " COUNTER-3 = ".
00620 02 COUNTER-3 PICTURE 9(3) VALUE 0.
00621 02 FILLER PICTURE X(17) VALUE " COUNTER-4 = ".
00622 02 COUNTER-4 PICTURE 9(3) VALUE 0.
00623 02 FILLER PICTURE X(17) VALUE " COUNTER-5 = ".
00624 02 COUNTER-5 PICTURE 9(3) VALUE 0.
00625 02 FILLER PICTURE X(17) VALUE " COUNTER-6 = ".
00701 02 COUNTER-6 PICTURE 9(3) VALUE 0.
00702 02 FILLER PICTURE X(3) VALUE SPACES.
00703 01 RULE=COUNT-LINE=2.
00704 02 FILLER PICTURE X(22) VALUE " COUNTER-7 = ".
00705 02 COUNTER-7 PICTURE 9(3) VALUE 0.
00706 02 FILLER PICTURE X(17) VALUE " COUNTER-8 = ".
00707 02 COUNTER-8 PICTURE 9(3) VALUE 0.
00708 02 FILLER PICTURE X(17) VALUE " COUNTER-9 = ".
00709 02 COUNTER-9 PICTURE 9(3) VALUE 0.
00710 02 FILLER PICTURE X(17) VALUE " COUNTER-10 = ".
00711 02 COUNTER-10 PICTURE 9(3) VALUE 0.
00712 02 FILLER PICTURE X(17) VALUE " COUNTER-11 = ".
00713 02 COUNTER-11 PICTURE 9(3) VALUE 0.
00714 02 FILLER PICTURE X(17) VALUE " COUNTER-12 = ".
00715 02 COUNTER-12 PICTURE 9(3) VALUE 0.
00716 02 FILLER PICTURE X(3) VALUE SPACES.
00717 01 RULE=COUNT-LINE=3.
00718 02 FILLER PICTURE X(22) VALUE " COUNTER-13 = ".
00719 02 COUNTER-13 PICTURE 9(3) VALUE 0.
00720 02 FILLER PICTURE X(17) VALUE " COUNTER-14 = ".
00721 02 COUNTER-14 PICTURE 9(3) VALUE 0.
00722 02 FILLER PICTURE X(17) VALUE " COUNTER-15 = ".
00723 02 COUNTER-15 PICTURE 9(3) VALUE 0.
00724 02 FILLER PICTURE X(17) VALUE " COUNTER-16 = ".
00725 02 COUNTER-16 PICTURE 9(3) VALUE 0.
00801 02 FILLER PICTURE X(17) VALUE " COUNTER-17 = ".
00802 02 COUNTER-17 PICTURE 9(3) VALUE 0.
00803 02 FILLER PICTURE X(17) VALUE " COUNTER-18 = ".
00804 02 COUNTER-18 PICTURE 9(3) VALUE 0.
00805 02 FILLER PICTURE X(3) VALUE SPACES.
00806 01 RULE=COUNT-LINE=4.
00807 02 FILLER PICTURE X(22) VALUE " COUNTER-19 = ".
00808 02 COUNTER-19 PICTURE 9(3) VALUE 0.
00809 02 FILLER PICTURE X(17) VALUE " COUNTER-20 = ".
00810 02 COUNTER-20 PICTURE 9(3) VALUE 0.
00811 02 FILLER PICTURE X(17) VALUE " COUNTER-21 = ".
00812 02 COUNTER-21 PICTURE 9(3) VALUE 0.
00813 02 FILLER PICTURE X(17) VALUE " COUNTER-22 = ".
00814 02 COUNTER-22 PICTURE 9(3) VALUE 0.
00815 02 FILLER PICTURE X(17) VALUE " COUNTER-23 = ".
00816 02 COUNTER-23 PICTURE 9(3) VALUE 0.
00817 02 FILLER PICTURE X(17) VALUE " COUNTER-24 = ".
00818 02 COUNTER-24 PICTURE 9(3) VALUE 0.
00819 02 FILLER PICTURE X(3) VALUE SPACES.
00920 01 RULE=COUNT-LINE=5.
00821 02 FILLER_PICTURE X(22) VALUE " COUNTER=25 = ",
00822 02 COUNTER=25 PICTURE 9(3) VALUE 0.
00823 02 FILLER_PICTURE X(17) VALUE " COUNTER=25 = ".
00824 02 COUNTER=26 PICTURE 9(3) VALUE 0.
00825 02 FILLER_PICTURE X(17) VALUE " COUNTER=26 = ".
00826 02 COUNTER=27 PICTURE 9(3) VALUE 0.
00827 02 FILLER_PICTURE X(17) VALUE " COUNTER=27 = ".
00828 02 COUNTER=28 PICTURE 9(3) VALUE 0.
00829 02 FILLER_PICTURE X(17) VALUE " COUNTER=28 = ".
00830 02 COUNTER=29 PICTURE 9(3) VALUE 0.
00831 02 FILLER_PICTURE X(17) VALUE " COUNTER=29 = ".
00832 02 COUNTER=30 PICTURE 9(3) VALUE 0.
00833 02 FILLER_PICTURE X(3) VALUE SPACES.
00901 01 TABLE-1.
00902 02 RESULT-VECTOR-1.
00903 03 RV=1 PICTURE 9 OCCURS 5 TIMES.
00904 02 WORK-VECTOR-1.
00905 03 RV=1 PICTURE 9 OCCURS 5 TIMES.

01001 PROCEDURE DIVISION.
01002 INIT.
01003 OPEN INPUT CARDIN.
01004 OPEN OUTPUT LISTING..PERFORM..HEADINGS.
01005 READ-LOOP.
01006 READ CARDIN..AT END GO TO LAST.
01007 PERFORM IS-MCE-MAE-DT.
01008 PERFORM WRITE-LISTING.
01009 GO TO READ-LOOP.
01010 IS-MCE-MAE-DT SECTION.
01011 IS-MCE-MAE-DT-NOTFS.
01012 NOTE THE ORIGINAL DECISION TABLE TOGETHER WITH THE REQUIRED
01013 DIAGNOSTICS (ACCORDING TO THE SPECIFIED OPTIONS IN THE
01014 CARD) ARE LISTED HERE FOR YOUR INSPECTION. IF CORRECT=
01015 IONS ARE NECESSARY EXECUTE ANOTHER PRE-COMPLE RUN.
01016 TABLE-10..IS-MCE-MAE-DT MCE-MAE.
01017 RL1 00000000011111111111122222222223E
01018 RL2 123456789012345678901234567890L
01019 CONDITIONS
01020 L IF G="PINK" YNYNYYNYYNNYYNYNYYNYYNYY
01021 E IF A***R YYY YY YY YY YY YY YY
01022 E IF E = YY YY YY YY YY YY YY YY
01023 YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
IS* MCE-MAE-DT-OPIONS

INDEX FOR THIS DECISION TABLE. THE FOLLOWING OPTIONS WERE SET:

1 Input Options: CARD TAPELIB
2 Output Options: LIST=1, SINGLE

For other options the default values were used.

IS* MCE-MAE-DT-DIAGNOSTICS

NOTE 1 POSSIBLE REDUNDANCIES:
NONE

2 POSSIBLE CONTRADICTIONS:

FOR RULE: NO-1 NO-3 NO-18 NO-3 NO-3
AND RULE: NO-9 NO-18 NO-24 NO-29 NO-26

< < < < <

20 30 15 15 35

Y Y N N

NUMERIC ALPHABETIC ALPHABETIC NUMERIC ALPHANUCER

RED GREEN GREEN  -

Y N N -

POSITIVE POSITIVE POSITIVE - NEGATIVE

FOR RULE: NO-6 NO-19
AND RULE: NO-14 NO-26

Y Y

> =

30 20

N N

NUMERIC ALPHABETIC

RED GREEN

- - POSITIVE

3 COMPLETNESS ACHIEVED BY THE ELSE RULE

4 GENERAL: THE SUM OF THE PROBABILITY IS NOT 1

*** NO FATAL ERRORS WERE FOUND
*** IN OPTIMIZING THE FOLLOWING PROCEDURES WERE USED:
1 CONDITION PARTITIONING BY LCE FOLLOWED BY ECE
2 TABLE SPLITTNG BY BRANCHING DONE ON THE LCE
3 OPTIMIZING THE LCE BY A TREE METHOD
4 OPTIMIZING THE ECE FIRST BY PROBABILITIES FOLLOWED BY
   COST.

IS=MCE=MAE=DT-SPLIT.
IF (X**2 = .3) = .97 NEXT SENTENCE.
ELSE IF G="PINK" GO TO IS=MCE=MAE=DT-GRP=3.
ELSE IF VALUE=1 GO TO IS=MCE=MAE=DT-GRP=5.
ELSE IF VALUE=1 NEXT SENTENCE.
ELSE IF G="PINK" GO TO IS=MCE=MAE=DT-GRP=2.
ELSE IF G="PINK" GO TO IS=MCE=MAE=DT-GRP=1.
ELSE GO TO IS=MCE=MAE=DT-GRP=4.

IS=MCE=MAE=DT-CON=2.
IF A < B MOVE 1 TO RV=1(1) ELSE
IF A = B MOVE 2 TO RV=1(1) ELSE
MOVE 3 TO RV=1(1).

IS=MCE=MAE=DT-CON=3.
IF E = 20 MOVE 1 TO RV=1(2) ELSE
IF E = 15 MOVE 2 TO RV=1(2) ELSE
IF E = 30 MOVE 3 TO RV=1(2) ELSE
IF E = 35 MOVE 4 TO RV=1(2).

IS=MCE=MAE=DT-CON=5.
IF C IS NUMERIC MOVE 1 TO RV=1(3) ELSE
IF C IS ALPHABETIC MOVE 2 TO RV=1(3) ELSE
MOVE 3 TO RV=1(3).

IS=MCE=MAE=DT-CON=6.
IF COULOR = "BLUE" MOVE 1 TO RV=1(4) ELSE
IF COULOR = "RED" MOVE 2 TO RV=1(4) ELSE
IF COULOR = "GREEN" MOVE 3 TO RV=1(4) ELSE
IF COULOR = "WHITE" MOVE 4 TO RV=1(4).

IS=MCE=MAE=DT=CON=8.
IF NUMBER IS POSITIVE MOVE 1 TO RV=1(5) ELSE
IF NUMBER IS NEGATIVE MOVE 2 TO RV=1(5) ELSE
MOVE 3 TO RV=1(5).

IS=MCE=MAE=DT-GRP=1.
MOVE ALL ZEROS TO RESULT-VECTOR.
PERFORM IS=MCE=MAE=DT-CON=2.
PERFORM IS=MCE=MAE=DT-CON=3.
PERFORM IS=MCE=MAE=DT-CON=8.
PERFORM IS=MCE=MAE=DT-CON=8.
MOVE RESULT-VECTOR 1 TO WORK-VECTOR.
MOVE ZERO TO RV=1(3) RV=1(5).
IF WORK-VECTOR-1 = "33020" + RULE-20

MOVE "2" TO RL1
MOVE "0" TO RL2
ADD 1 TO CNT1
ADD 1 TO COUNTER-1
GO TO IS-MCE-MAE-DT-END*

PERFORM IS-MCE-MAE-DT-CON-5.

MOVE RESULT-VECTOR-1 TO WORK-VECTOR-1

MOVE ZERO TO WY=1(4)

IF WORK-VECTOR-1 = "11101" + RULE-01

MOVE "0" TO RL1
MOVE "1" TO RL2
ADD 1 TO CNT1
ADD 1 TO COUNTER-1
GO TO IS-MCE-MAE-DT-END*

MOVE RESULT-VECTOR-1 TO WORK-VECTOR-1

IF WORK-VECTOR-1 = "22321" + RULE-03

MOVE "3" TO RL2
ADD 1 TO CNT1
ADD 1 TO COUNTER-3
GO TO IS-MCE-MAE-DT-END*

MOVE RESULT-VECTOR-1 TO WORK-VECTOR-1

MOVE ZERO TO WY=1(1)

IF WORK-VECTOR-1 = "01121" + RULE-09

MOVE "0" TO RL1
MOVE "0" TO RL2
ADD 1 TO CNT1
ADD 1 TO COUNTER-9
GO TO IS-MCE-MAE-DT-END*

MOVE RESULT-VECTOR-1 TO WORK-VECTOR-1

MOVE ZERO TO WY=1(2)

IF WORK-VECTOR-1 = "30232" + RULE-16

MOVE "1" TO RL1
MOVE "6" TO RL2
ADD 1 TO CNT1
ADD 1 TO COUNTER-16
GO TO IS-MCE-MAE-DT-END*

IS-MCE-MAE-DT-GRP-2.

MOVE ALL ZEROS TO RESULT-VECTOR-1.

PERFORM IS-MCE-MAE-DT-CON-2.

PERFORM IS-MCE-MAE-DT-CON-5.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT-VECTOR-1 = "10031" + RULE-18

MOVE "1" TO RL1
MOVE "8" TO RL2
ADD 1 TO CNT2
ADD 1 TO COUNTER-18
GO TO IS-MCE-MAE-DT-END*

PERFORM IS-MCE-MAE-DT-CON-3.

MOVE RESULT-VECTOR-1 TO WORK-VECTOR-1

MOVE ZERO TO WY=1(3) WY=1(4) WY=1(5)

IF WORK-VECTOR-1 = "31000" + RULE-25

MOVE "2" TO RL1
MOVE "5" TO RL2.
ADD 1 TO CNT2.
ADD 1 TO COUNTER=25.
GO TO IS=ME=MAE=DT-END.
MOVE_RESULT=VECTOR=1 TO WORK=VECTOR=1
MOVE ZERO TO WV=1(3) WV=1(5)
IF WORK=VECTOR=1 = "240200" + RULE=02
MOVE "0" TO RL1.
MOVE "1" TO RL2.
ADD 1 TO CNT2.
ADD 1 TO COUNTER=2.
GO TO IS=ME=MAE=DT-END.
MOVE_RESULT=VECTOR=1 TO WORK=VECTOR=1.
MOVE ZERO TO WV=1(3) WV=1(4) WV=1(5)
IF WORK=VECTOR=1 = "220000" + RULE=30
MOVE "3" TO RL1.
MOVE "0" TO RL2.
ADD 1 TO CNT2.
ADD 1 TO COUNTER=30.
GO TO IS=ME=MAE=DT-END.
IF RESULT=VECTOR=1 = "230434" + RULE=07.
MOVE "0" TO RL1.
MOVE "7" TO RL2.
ADD 1 TO CNT2.
ADD 1 TO COUNTER=7.
GO TO IS=ME=MAE=DT-END.
PERFORM IS=ME=MAE=DT=CON=5.
MOVE_RESULT=VECTOR=1 TO WORK=VECTOR=1
MOVE ZERO TO WV=1(4)
IF WORK=VECTOR=1 = "122016" + RULE=24
MOVE "2" TO RL1.
MOVE "4" TO RL2.
ADD 1 TO CNT2.
ADD 1 TO COUNTER=24.
GO TO IS=ME=MAE=DT-END.
MOVE_RESULT=VECTOR=1 TO WORK=VECTOR=1
MOVE ZERO TO WV=1(1)
IF WORK=VECTOR=1 = "032310" + RULE=13
MOVE "1" TO RL1.
MOVE "3" TO RL2.
ADD 1 TO CNT2.
ADD 1 TO COUNTER=13.
GO TO IS=ME=MAE=DT-ELSE-ELSE.
IS=ME=MAE=DT=GRP=3.
MOVE ALL ZEROS TO RESULT=VECTOR=1.
MOVE "0" TO RL1.
MOVE "2" TO RL2.
ADD 1 TO CNT2.
ADD 1 TO COUNTER=6.
GO TO IS=ME=MAE=DT-END.
02424  MOVE 'M5' TO RL2.
02425  ADD 1 TO CNTA.
02426  ADD 1 TO COUNTER-15.
02427  GO TO IS=MCE=MAE=DT-END.
02428  PERFORM IS=MCE=MAE=DT-CON=6.
02429  IF RESULT-VECTOR-1 = "00222"  + RULE-11
02430  MOVE '1' TO RL1.
02431  MOVE '1' TO RL2.
02432  ADD 1 TO CNTA.
02433  ADD 1 TO COUNTER-11.
02434  GO TO IS=MCE=MAE=DT-END.
02435  MOVE ZEROS TO RV=1(4).
02436  IF RESULT-VECTOR-1 = "00103"  + RULE-27
02437  MOVE '2' TO RL1.
02438  MOVE '7' TO RL2.
02439  ADD 1 TO CNTA.
02440  ADD 1 TO COUNTER-27.
02441  GO TO IS=MCE=MAE=DT-END.
02442  IS=MCE=MAE=DT-GRP=5.
02443  MOVE ALL ZEROS TO RESULT-VECTOR-1.
02444  PERFORM IS=MCE=MAE=DT-CON=2.
02445  PERFORM IS=MCE=MAE=DT-CON=3.
02446  PERFORM IS=MCE=MAE=DT-CON=6.
02447  PERFORM IS=MCE=MAE=DT-CON=8.
02448  IF RESULT-VECTOR-1 = "31022"  + RULE-12
02449  MOVE '1' TO RL1.
02450  MOVE '9' TO RL2.
02451  ADD 1 TO CNTA.
02452  ADD 1 TO COUNTER-12.
02453  GO TO IS=MCE=MAE=DT-END.
02454  IF RESULT-VECTOR-1 = "23033"  + RULE-16
02455  MOVE '1' TO RL1.
02456  ADD 1 TO CNTA.
02457  ADD 1 TO COUNTER-16.
02458  GO TO IS=MCE=MAE=DT-END.
02459  IF RESULT-VECTOR-1 = "13012"  + RULE-18
02460  MOVE '1' TO RL1.
02461  ADD 1 TO CNTA.
02462  ADD 1 TO COUNTER-20.
02463  GO TO IS=MCE=MAE=DT-END.
02464  IF RESULT-VECTOR-1 = "23019"  + RULE-20
02465  MOVE '1' TO RL1.
02466  ADD 1 TO CNTA.
02467  ADD 1 TO COUNTER-24.
02468  GO TO IS=MCE=MAE=DT-END.
02469  IF WORK-VECTOR = "22041"  + RULE-10
02470  BRAVE 'E' TO RL1.
02471  ADD 1 TO CNTA.
02472  ADD 1 TO COUNTER-28.
02473  GO TO IS=MCE=MAE=DT-END.
02474  IF WORK-VECTOR = "02032"  + RULE-22
02475  BRAVE 'E' TO RL1.
02476  ADD 1 TO CNTA.
02477  ADD 1 TO COUNTER-32.
02478  GO TO IS=MCE=MAE=DT-END.
02479  IF WORK-VECTOR = "02032"  + RULE-22
02480  BRAVE 'E' TO RL1.
02481  ADD 1 TO CNTA.
02482  ADD 1 TO COUNTER-36.
02483  GO TO IS=MCE=MAE=DT-END.
02484  IF WORK-VECTOR = "02032"  + RULE-22
02485  BRAVE 'E' TO RL1.
02486  ADD 1 TO CNTA.
02487  ADD 1 TO COUNTER-40.
02704   ADD 1 TO COUNTER=22.
02705   GO TO IS-MCE-MAE-DT-END.
02706   MOVE RESULTVECTOR=1 TO WORKVECTOR=1.
02707   MOVE ZERO TO WV-1(2).
02708   IF WORKVECTOR=1 = "10022" + RULE=04
02709   MOVE "0" TO RL1.
02710   MOVE "4" TO RL2.
02711   ADD 1 TO CNT5.
02712   ADD 1 TO COUNTER=8.
02713   GO TO IS-MCE-MAE-DT-END.
02714   IF RESULTVECTOR=1 = "14033" + RULE=21
02715   MOVE "2" TO RL1.
02716   MOVE "1" TO RL2.
02717   ADD 1 TO CNT5.
02718   ADD 1 TO COUNTER=21.
02719   GO TO IS-MCE-MAE-DT-END.
02720   MOVE ZERO TO RV-1(11).
02721   IF RESULTVECTOR=1 = "01031" + RULE=28
02722   MOVE "2" TO RL1.
02723   MOVE "8" TO RL2.
02724   ADD 1 TO CNT5.
02725   ADD 1 TO COUNTER=28.
02726   GO TO IS-MCE-MAE-DT-END.
02727   IS-MCE-MAE-DT-ELSE=RULE.
02728   MOVE "ELSE" TO RULE=NO.
02729   ADD 1 TO CNT5.
02730   GO TO IS-MCE-MAE-DT-END.
02731   IS-MCE-MAE-DT-ELSE=END.
02732   EXIT.
02733   REPORT SECTION.
02734   HEADINGS.
02735   WRITE PRINTER=LINE FROM HDR1 AFTER ADVANCING TO CHANNEL 1.
02736   WRITE PRINTER=LINE FROM HDR2 AFTER ADVANCING 2 LINES.
02737   WRITE LISTING.
02738   WRITE INPUT=CARD TO MS-CARD=IMAGE.
02739   WRITE PRINTER=LINE FROM PRINT=LINE AFTER ADVANCING 1 LINES.
02740   MOVE SPACES TO RULE=NO.
02741   LAST.
02742   WRITE PRINTER=LINE FROM GROUP=CNT-LINE AFTER ADVANCING 2.
02743   WRITE PRINTER=LINE FROM RULE=COUNT-LINE=1 AFTER ADVANCING 2.
02744   WRITE PRINTER=LINE FROM RULE=COUNT-LINE=2 AFTER ADVANCING 2.
02745   WRITE PRINTER=LINE FROM RULE=COUNT-LINE=3 AFTER ADVANCING 2.
02746   WRITE PRINTER=LINE FROM RULE=COUNT-LINE=4 AFTER ADVANCING 2.
02747   WRITE PRINTER=LINE FROM RULE=COUNT-LINE=5 AFTER ADVANCING 2.
02748   CLOSE CARDIN LISTING WITH RELEASE.
02749   STOP RUN.
999999 END-OF-JOB.
<table>
<thead>
<tr>
<th>Defined On</th>
<th>Name</th>
<th>&quot;Referenced On&quot;</th>
<th>010</th>
<th>No Records Written</th>
<th>001</th>
<th>No RRefs Written</th>
</tr>
</thead>
<tbody>
<tr>
<td>00204</td>
<td>A</td>
<td></td>
<td>01023 01702 01703</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00206</td>
<td>B</td>
<td></td>
<td>01023 01702 01703</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILE 00125</td>
<td>CARDIN</td>
<td>DUPLICATE NAME</td>
<td>01003 01006 02822</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00521</td>
<td>CNT1</td>
<td></td>
<td>01214 01812 01821 01903 01911 01919 02121</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00524</td>
<td>CNT2</td>
<td></td>
<td>01215 02006 02015 02023 02106 02112 02204</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00602</td>
<td>CNT3</td>
<td></td>
<td>01216 02216 02301 02307 02315 02323 02406 02414</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00605</td>
<td>CNT4</td>
<td></td>
<td>01217 02425 02507 02514</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00608</td>
<td>CNT5</td>
<td></td>
<td>01218 02602 02608 02614 02620 02703 02711 02717 02724</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00611</td>
<td>CNT6</td>
<td></td>
<td>01219 02805</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00214</td>
<td>COULOR</td>
<td></td>
<td>01112 01716 01717 01718 01719</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00711</td>
<td>COUNTER-10</td>
<td></td>
<td>01305 02621</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00713</td>
<td>COUNTER-11</td>
<td></td>
<td>01306 02508</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00715</td>
<td>COUNTER-12</td>
<td></td>
<td>01307 02603</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00719</td>
<td>COUNTER-13</td>
<td></td>
<td>01308 02205</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00721</td>
<td>COUNTER-14</td>
<td></td>
<td>01309 02316</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00723</td>
<td>COUNTER-15</td>
<td></td>
<td>01310 02501</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00725</td>
<td>COUNTER-16</td>
<td></td>
<td>01311 01920 02609</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00802</td>
<td>COUNTER-17</td>
<td></td>
<td>01312</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00804</td>
<td>COUNTER-18</td>
<td></td>
<td>01313 02007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00806</td>
<td>COUNTER-19</td>
<td></td>
<td>01314 02308</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00816</td>
<td>COUNTER-1</td>
<td></td>
<td>01221 01813 01822</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00810</td>
<td>COUNTER-20</td>
<td></td>
<td>01315</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00812</td>
<td>COUNTER-21</td>
<td></td>
<td>01316 02718</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00814</td>
<td>COUNTER-22</td>
<td></td>
<td>01317 02704</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00816</td>
<td>COUNTER-23</td>
<td></td>
<td>01318 02302</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
READ PARA PROCESSOR TIME = 0.000690 SECS
SCANNER PROCESSOR TIME = 0.000410 SECS
OVERALL ELAPSED TIME = 0.000690 SECS
OVERALL PROCESSOR TIME = 0.000601 SECS
OVERALL I/O TIME = 0.000466 SECS
NO. RECDS RELEASED TO SORT = 1297
VI-4-3 PROGRAM COMPILATION.

This part contains the actual COBOL compilation which can be initiated by the precompiler on an optional basis. Three different options can be used:

1. Compile for syntax (COMSYN)
2. Compile and Run (COMRUN)
3. Compile for Library (COMLIB)

Here the second option was illustrated.
B = 5500 COROL DISK COMPILER

IDENTIFICATION DIVISION.
PROGRAM-ID. "TEST MCE-MAF".
AUTHOR. A. PINHAS.
DATE-WRITTEN. 20 DEC 1970.
REMARKS. THIS PROGRAM WAS WRITTEN MANUALLY SIMULATING A
SUGGESTED DECISION TABLE PRE-COMPILER. THE PRE-
COMPILATION PASS WAS PREVIOUSLY DONE WITH THE
FOLLOWING OPTIONS SET:
1. INPUT FROM CARDS
2. OUTPUT LIST-1 PUNCH-L
3. TEST CT-TEST
4. FREQ
5. COST

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-PROGRAM. B = 5500.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT CARDS ASSIGN TO CARD-READER.

SELECT LISTING ASSIGN TO PRINTER.

DATA DIVISION.
FILE SECTION.
FD CARDIN
VALUE OF ID IS "CARDS"
DATA RECORD IS INPUT-CARD.
01 INPUT-CARD SIZE 80.
02 A PICTURE 9.
02 B PICTURE 9.
02 C PICTURE 9.
02 D PICTURE 9.
02 E PICTURE 9.
02 F PICTURE 9.
02 G PICTURE 9.
02 H PICTURE 9.
02 VALUE-1 PICTURE 9.
02 VALUE "AA", + AA PICTURE 9.
02 VALUE "B shiny", + B PICTURE 9.
02 VALUE "C shiny", + C PICTURE 9.
02 VALUE "D shiny", + D PICTURE 9.
02 VALUE "E shiny", + E PICTURE 9.
02 VALUE "F shiny", + F PICTURE 9.
02 VALUE "G shiny", + G PICTURE 9.
02 VALUE "H shiny", + H PICTURE 9.
02 VALUE "I shiny", + I PICTURE 9.
02 VALUE "J shiny", + J PICTURE 9.
02 VALUE "K shiny", + K PICTURE 9.
02 VALUE "L shiny", + L PICTURE 9.
02 VALUE "M shiny", + M PICTURE 9.
02 VALUE "N shiny", + N PICTURE 9.
02 VALUE "O shiny", + O PICTURE 9.
02 VALUE "P shiny", + P PICTURE 9.
02 VALUE "Q shiny", + Q PICTURE 9.
02 VALUE "R shiny", + R PICTURE 9.
02 VALUE "S shiny", + S PICTURE 9.
02 VALUE "T shiny", + T PICTURE 9.
02 VALUE "U shiny", + U PICTURE 9.
02 VALUE "V shiny", + V PICTURE 9.
02 VALUE "W shiny", + W PICTURE 9.
02 VALUE "X shiny", + X PICTURE 9.
02 VALUE "Y shiny", + Y PICTURE 9.
02 VALUE "Z shiny", + Z PICTURE 9.
02 VALUE "AA", + AA PICTURE 9.
0016  02  COUNTER-23  PICTURE 9(3)  VALUE 0.
0017  09  FILLER  PICTURE X(17)  VALUE  "COUNTER-24 = ".
0018  09  COUNTER-24  PICTURE 9(3)  VALUE 0.
0019  09  FILLER  PICTURE X(3)  VALUE  SPACES.
0020  01  RULE=COUNT-LINE-5.
0021  02  FILLER  PICTURE X(22)  VALUE  " COUNTER-25 = ".
0022  09  COUNTER-25  PICTURE 9(3)  VALUE 0.
0023  09  FILLER  PICTURE X(17)  VALUE  " COUNTER-26 = ".
0024  09  COUNTER-26  PICTURE 9(3)  VALUE 0.
0025  09  FILLER  PICTURE X(17)  VALUE  " COUNTER-27 = ".
0026  09  COUNTER-27  PICTURE 9(3)  VALUE 0.
0027  02  FILLER  PICTURE X(17)  VALUE  " COUNTER-28 = ".
0028  09  COUNTER-28  PICTURE 9(3)  VALUE 0.
0029  09  FILLER  PICTURE X(17)  VALUE  " COUNTER-29 = ".
0030  09  COUNTER-29  PICTURE 9(3)  VALUE 0.
0031  09  FILLER  PICTURE X(17)  VALUE  " COUNTER-30 = ".
0032  09  COUNTER-30  PICTURE 9(3)  VALUE 0.
0033  09  FILLER  PICTURE X(3)  VALUE  SPACES.
0034  01  TABLE-1.
0035  02  RESULT-VECTOR-1.
0036  03  RV-1  PICTURE 9  OCCURS 5 TIMES.
0037  09  TABLE-1.
0038  02  WORK-VECTOR-1.
0039  03  WV-1  PICTURE 9  OCCURS 5 TIMES.
0040  01  DATA-LINE.

0041  03  RV-1  PICTURE 9  OCCURS 5 TIMES.
0042  09  TABLE-1.
0043  02  WORK-VECTOR-1.
0044  03  WV-1  PICTURE 9  OCCURS 5 TIMES.
0045  01  DATA-LINE.

0101  PROCEDURE  DIVISION.
0102     INIT.
0103  PROCEDURE  DIVISION.
0104     OPEN  INPUT  CARDIN.
0105     OPEN  OUTPUT  LISTING.  PERFORM  HEADINGS.
0106  PROCEDURE  0005  SIZE  0001.
0107  READ-LOOP.
0108  PROCEDURE  0010  SIZE  0007.
0109  READ  CARDIN  AT  END  GO  TO  LAST.
0110  PERFORM  IS+MCF+MAE+DT.
0111  PERFORM  WRITE-LISTING.
0112  GO  TO  READ-LOOP.
0113  IS+MCF+MAE+DT  SECTION.
0114  IS+MCF+MAE+DT-NOTES.
0115  NOTE  THE  ORIGINAL  DECISION  TABLE  TOGETHER  WITH  THE  REQUIRED
0116  DIAGNOSTICS  (ACCORDING  TO  THE  SPECIFIED  OPTIONS  IN  THE
0117  CODE)  ARE  LISTED  HERE  FOR  YOUR  INSPECTION.  IF  CORRECT-
0118  TIONS  ARE  NECESSARY  EXECUTE  ANOTHER  PRE-COUPLE  RUN.
NOTE FOR THIS DECISION TABLE THE FOLLOWING OPTIONS WERE SET:
1 INPUT OPTIONS: CARD, TAP, EQLIB
2 OUTPUT OPTIONS: LIST, SINGLE
3 TESTING OPTIONS: CT, TEST
4 EFFICIENCY OPTIONS: FREQ, COST
5 COMPILE OPTIONS: RUN, CUM

FOR OTHER OPTIONS THE DEFAULT VALUES WERE USED.

1 POSSIBLE REDUNDANCIES;
N/A

2 POSSIBLE CONTRADICTIONS:

FOR RULE NO-1 NO-2
AND RULE NO-9 NO-13 NO-18 NO-20 NO-26
Y Y Y Y Y
< < < < <
20 30 15 15 35
Y Y Y Y N

NUMERIC ALPHABETIC ALPHABETIC NUMERIC ALPHANUMERIC
RED GREEN : GREEN = =
Y N N N

POSITIVE POSITIVE = =
NEGATIVE

FOR RULE NO-6 NO-19
AND RULE NO-14 NO-26
Y Y
> =
30 20
N N
IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.

PERFORM IS-MCE-MAE-DT-CON-6.

IF RESULT=VECTOR-1 = "10031", + RULE-18
    MOVE "1" TO RL1
    MOVE "8" TO RL2
    ADD 1 TO CNT2
    ADD 1 TO COUNTER-18
    GO TO IS-MCE-MAE-DT-END.
ADD 1 TO COUNTER=3.
GO TO IS-MCE-MAE-DT-END.
GO TO IS-MCE-MAE-DT-ELSE-RULE.
ELSE-RULE.

IS-MCE-MAE-DT-GRP=4.

MOVE ALL ZEROS TO RESULT-VECTOR=1.
PERFORM IS-MCE-MAE-DT-CON=5.
PERFORM IS-MCE-MAE-DT-CON=3.
IF RESULT-VECTOR=1="00201", + RULE=15.
MOVE "1" TO RL1
MOVE "5" TO RL2
ADD 1 TO CNT4
ADD 1 TO COUNTER=13.
GO TO IS-MCE-MAE-DT-END.
PERFORM IS-MCE-MAE-DT-CON=3.
IF RESULT-VECTOR=1="00222", + RULE=11.
MOVE "1" TO RL1
MOVE "4" TO RL2
ADD 1 TO CNT4
ADD 1 TO COUNTER=11.
GO TO IS-MCE-MAE-DT-END.
MOVE ZERO TO RV=14.
IF RESULT-VECTOR=1="00103", + RULE=27.
MOVE "2" TO RL1
MOVE "7" TO RL2
ADD 1 TO CNT4
ADD 1 TO COUNTER=27.
GO TO IS-MCE-MAE-DT-END.
ELSE-RULE.

IS-MCE-MAE-DT-GRP=3.

MOVE ALL ZEROS TO RESULT-VECTOR=1.
PERFORM IS-MCE-MAE-DT-CON=2.
PERFORM IS-MCE-MAE-DT-CON=3.
PERFORM IS-MCE-MAE-DT-CON=4.
IF RESULT-VECTOR=1="31022", + RULE=12.
MOVE "1" TO RL1
MOVE "2" TO RL2
ADD 1 TO CNT5
ADD 1 TO COUNTER=12.
GO TO IS-MCE-MAE-DT-END.
IF RESULT-VECTOR=1="23033", + RULE=16.
MOVE "1" TO RL1
MOVE "4" TO RL2
ADD 1 TO CNT5
ADD 1 TO COUNTER=16.
GO TO IS-MCE-MAE-DT-END.
IF RESULT-VECTOR=1="13012", + RULE=08.
MOVE "0" TO RL1
MOVE "6" TO RL2
ADD 1 TO CNT5
ADD 1 TO COUNTER=8.
GO TO IS-MCE-MAE-DT-END.
IF RESULT-VECTOR=1="20041", + RULE=10.
MOVE "1" TO RL1
MOVE "0" TO RL2
ADD 1 TO CNT5
ADD 1 TO COUNTER=10.
GO TO IS-MCE-MAE-DT-END.
MOVE RESULT-VECTOR=1 TO WORK-VECTOR=1.
MOVE ZERO TO WV-1(1)

IF WORK-VECTOR-1 = "02032" + RULE-22

MOVE "3" TO RL1
MOVE "4" TO RL2
MOVE 1 TO CNT5
ADD 1 TO COUNTER-22
GO TO IS-MCF-MAE-DT-END.
MOVE RESULT-VECTOR-1 TO WORK-VECTOR-1.
MOVE ZERO TO WV-1(2).

IF WORK-VECTOR-1 = "10022" + RULE-04
MOVE "0" TO RL1
MOVE "4" TO RL2
MOVE 1 TO CNT5
ADD 1 TO COUNTER-12
GO TO IS-MCF-MAE-DT-END.

IF RESULT-VECTOR-1 = "14033" + RULE-21
MOV "2" TO RL1:
MOVE "1" TO RL2
ADD 1 TO CNT5
ADD 1 TO COUNTER-21
GO TO IS-MCF-MAE-DT-END

MOVE ZERO TO WV-1(1)

IF RESULT-VECTOR-1 = "01031" + RULE-28
MOVE "2" TO RL1
MOVE "0" TO RL2
ADD 1 TO CNT5
ADD 1 TO COUNTER-28
GO TO IS-MCF-MAE-DT-END.

IS-MCF-MAE-DT-ELSE-RULE.

MOVE "ELSE" TO RULE-ND.
ADD 1 TO CNT6.
GO TO IS-MCF-MAE-DT-END.

IS-MCF-MAE-DT-ELSE-RULE + ELSE-RULE

PROCEDURE 0025 SIZE 0096
0264 0026 001
0265 0026 003
0266 0026 008
0267 0026 009

PRT 0156

REPORTS SECTION.

HEADINGS.

WRITE PRINTER-LINE FROM HDR1 AFTER ADVANCING TO CHANNEL 1.
WRITE PRINTER-LINE FROM HDR2 AFTER ADVANCING 2 LINES.
WRITE LISTING.

MOVE INPUT-CARD TO WS-CARD-IMAGE.
WRITE PRINTER-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINES.
MOVE SPACES TO RULE-END.

LAST.

WRITE PRINTER-LINE FROM GROUP-CNT-LINE AFTER ADVANCING 2.
WRITE PRINTER-LINE FROM RULE-COUNT-LINE-1 AFTER ADVANCING 2.
WRITE PRINTER-LINE FROM RULE-COUNT-LINE-2 AFTER ADVANCING 2.
WRITE PRINTER-LINE FROM RULE-COUNT-LINE-3 AFTER ADVANCING 2.
WRITE PRINTER-LINE FROM RULE-COUNT-LINE-4 AFTER ADVANCING 2.
WRITE PRINTER-LINE FROM RULE-COUNT-LINE-5 AFTER ADVANCING 2.
CLOSE PRINTER WITH RELEASE.

STOP RUN.

PROCEDURE 0029 SIZE 0013
0030 0003
0030 0007
0030 0012
0030 0016
0030 0020
0030 0024
0030 0028
0030 0030

PROCEDURE 0030 SIZE 0031
PROCEDURE 0031 SIZE 0005
VI-1-4 TEST DATA LISTING.

The test data was designed to illustrate later in the execution of the program ambiguous rules. This was done by specifying on every record the intended rule.
<table>
<thead>
<tr>
<th>7</th>
<th>7</th>
<th>35</th>
<th>BLUE</th>
<th>10</th>
<th>RED</th>
<th>13</th>
<th>CC</th>
<th>BAB</th>
<th>RULE = 0.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>30</td>
<td>BLUE</td>
<td>10</td>
<td>WHITE</td>
<td>00</td>
<td>AS</td>
<td>BAB</td>
<td>RULE = 0.07</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>BLUE</td>
<td>10</td>
<td>WHITE</td>
<td>00</td>
<td>BC</td>
<td>BAB</td>
<td>RULE = 0.07</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>BLUE</td>
<td>10</td>
<td>WHITE</td>
<td>00</td>
<td>DA</td>
<td>BAB</td>
<td>RULE = 0.07</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>PINK</td>
<td>10</td>
<td>GREEN</td>
<td>H1</td>
<td>CC</td>
<td>AAA</td>
<td>RULE = 0.13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>****</td>
<td>10</td>
<td>GREEN</td>
<td>H2</td>
<td>/U</td>
<td>888</td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>BLUE</td>
<td>10</td>
<td>GREEN</td>
<td>33</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>****</td>
<td>10</td>
<td>GREEN</td>
<td>H1</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>RED</td>
<td>10</td>
<td>GREEN</td>
<td>H2</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>RED</td>
<td>10</td>
<td>GREEN</td>
<td>H3</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>RED</td>
<td>10</td>
<td>GREEN</td>
<td>H4</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>RED</td>
<td>10</td>
<td>GREEN</td>
<td>H5</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>PINK</td>
<td>10</td>
<td>GREEN</td>
<td>34</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>PINK</td>
<td>10</td>
<td>GREEN</td>
<td>37</td>
<td></td>
<td></td>
<td>RULE = 0.18</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>BLUE</td>
<td>10</td>
<td>BLACK</td>
<td>35</td>
<td>ABC</td>
<td>RULE = 0.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>BLUE</td>
<td>10</td>
<td>GREEN</td>
<td>36</td>
<td>DEF</td>
<td>RULE = 0.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>BLUE</td>
<td>10</td>
<td>BLACK</td>
<td>37</td>
<td>GHI</td>
<td>RULE = 0.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>BLUE</td>
<td>10</td>
<td>BLACK</td>
<td>37</td>
<td>JKL</td>
<td>RULE = 0.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>BLUE</td>
<td>10</td>
<td>BLACK</td>
<td>37</td>
<td>MNO</td>
<td>RULE = 0.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>RED</td>
<td>33</td>
<td>RED</td>
<td>J2</td>
<td>BA</td>
<td>RULE = 0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>GKO</td>
<td>07</td>
<td>BLUE</td>
<td>J3</td>
<td>MA</td>
<td>RULE = 0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>SPKO</td>
<td>13</td>
<td>BLUE</td>
<td>J4</td>
<td>PH</td>
<td>RULE = 0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>RNAO</td>
<td>17</td>
<td>WHITE</td>
<td>J5</td>
<td>DR</td>
<td>RULE = 0.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>BAKO</td>
<td>23</td>
<td>WHITE</td>
<td>J6</td>
<td>KT</td>
<td>RULE = 0.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>TARO</td>
<td>75</td>
<td>RED</td>
<td>J7</td>
<td>LM</td>
<td>RULE = 0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>KARO</td>
<td>51</td>
<td>RED</td>
<td>J8</td>
<td>DP</td>
<td>RULE = 0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>TAKA</td>
<td>31</td>
<td>RED</td>
<td>J9</td>
<td>RS</td>
<td>RULE = 0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>KAKA</td>
<td>46</td>
<td>GREEN</td>
<td>00</td>
<td>T0</td>
<td>RULE = 0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>SAKA</td>
<td>64</td>
<td>GREEN</td>
<td>00</td>
<td>PP</td>
<td>RULE = 0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>DNSK</td>
<td>61</td>
<td>GREEN</td>
<td>00</td>
<td>KK</td>
<td>RULE = 0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>35</td>
<td>FORD</td>
<td>73</td>
<td>GREEN</td>
<td>00</td>
<td>RR</td>
<td>RULE = 0.21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>BLUE</td>
<td>56</td>
<td>GREEN</td>
<td>J1</td>
<td>TT</td>
<td>RULE = 0.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>RAKA</td>
<td>51</td>
<td>GREEN</td>
<td>J2</td>
<td>TT</td>
<td>RULE = 0.22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>99</td>
<td>FORO</td>
<td>99</td>
<td>GREEN</td>
<td>37</td>
<td>AC</td>
<td>RULE = 0.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>77</td>
<td>ELSE</td>
<td>77</td>
<td>PINK</td>
<td>00</td>
<td>**</td>
<td>ELSE RULE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>77</td>
<td>ELSE</td>
<td>77</td>
<td>PINK</td>
<td>00</td>
<td>**</td>
<td>ELSE RULE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>77</td>
<td>ELSE</td>
<td>77</td>
<td>PINK</td>
<td>00</td>
<td>**</td>
<td>ELSE RULE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>77</td>
<td>ELSE</td>
<td>77</td>
<td>PINK</td>
<td>00</td>
<td>**</td>
<td>ELSE RULE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>77</td>
<td>ELSE</td>
<td>77</td>
<td>PINK</td>
<td>00</td>
<td>**</td>
<td>ELSE RULE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VI-1-5 THE RUN.

The program output was designed to list every input record together with the rule number it satisfies making possible to check if the data satisfied the intended rule.
<table>
<thead>
<tr>
<th>Rule Satisfied</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>G</th>
<th>X</th>
<th>Color</th>
<th>Number</th>
<th>Value</th>
<th>C</th>
<th>Intended Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfies Rule-19</td>
<td>4</td>
<td>4</td>
<td>20</td>
<td>Pink</td>
<td>77</td>
<td>Green</td>
<td>17</td>
<td>AA</td>
<td>BCA</td>
<td>Rule = 15</td>
</tr>
<tr>
<td>Satisfies Rule-19</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>Pink</td>
<td>73</td>
<td>Green</td>
<td>17</td>
<td>AA</td>
<td>OTP</td>
<td>Rule = 19</td>
</tr>
<tr>
<td>Satisfies Rule-19</td>
<td>9</td>
<td>9</td>
<td>20</td>
<td>Pink</td>
<td>31</td>
<td>Green</td>
<td>63</td>
<td>AA</td>
<td>LSD</td>
<td>Rule = 19</td>
</tr>
<tr>
<td>Satisfies Rule-26</td>
<td>3</td>
<td>3</td>
<td>20</td>
<td>Pink</td>
<td>05</td>
<td>Red</td>
<td>10</td>
<td>AA</td>
<td>ABC</td>
<td>Rule = 26</td>
</tr>
<tr>
<td>Satisfies Rule-23</td>
<td>3</td>
<td>7</td>
<td>35</td>
<td>Pink</td>
<td>35</td>
<td>Red</td>
<td>93</td>
<td>AA</td>
<td>ABC</td>
<td>Rule = 23</td>
</tr>
<tr>
<td>Satisfies Rule-23</td>
<td>5</td>
<td>7</td>
<td>35</td>
<td>Pink</td>
<td>93</td>
<td>Red</td>
<td>33</td>
<td>AA</td>
<td>OTP</td>
<td>Rule = 23</td>
</tr>
<tr>
<td>Satisfies Rule-29</td>
<td>4</td>
<td>4</td>
<td>20</td>
<td>Pink</td>
<td>78</td>
<td>Red</td>
<td>93</td>
<td>AA</td>
<td>LSD</td>
<td>Rule = 26</td>
</tr>
<tr>
<td>Satisfies Rule-29</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>Pink</td>
<td>63</td>
<td>Red</td>
<td>10</td>
<td>AA</td>
<td>375</td>
<td>Rule = 29</td>
</tr>
<tr>
<td>Satisfies Rule-11</td>
<td>10</td>
<td>Red</td>
<td>83</td>
<td>AA</td>
<td>EFP</td>
<td>Rule = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies Rule-11</td>
<td>10</td>
<td>Red</td>
<td>78</td>
<td>AA</td>
<td>EFP</td>
<td>Rule = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies Rule-11</td>
<td>10</td>
<td>Red</td>
<td>37</td>
<td>AA</td>
<td>EFP</td>
<td>Rule = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies Rule-15</td>
<td>10</td>
<td>Red</td>
<td>95</td>
<td>AA</td>
<td>ORU</td>
<td>Rule = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies Rule-15</td>
<td>10</td>
<td>Red</td>
<td>95</td>
<td>AA</td>
<td>ORU</td>
<td>Rule = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies Rule-15</td>
<td>10</td>
<td>Red</td>
<td>95</td>
<td>AA</td>
<td>ORU</td>
<td>Rule = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies Rule-15</td>
<td>10</td>
<td>Red</td>
<td>95</td>
<td>AA</td>
<td>ORU</td>
<td>Rule = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies Rule-25</td>
<td>3</td>
<td>3</td>
<td>20</td>
<td>Pink</td>
<td>15</td>
<td>Red</td>
<td>35</td>
<td>AA</td>
<td>755</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-25</td>
<td>3</td>
<td>3</td>
<td>20</td>
<td>Pink</td>
<td>15</td>
<td>Red</td>
<td>35</td>
<td>AA</td>
<td>770</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-25</td>
<td>9</td>
<td>9</td>
<td>40</td>
<td>Pink</td>
<td>15</td>
<td>Red</td>
<td>133</td>
<td>AA</td>
<td>077</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-30</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>Pink</td>
<td>23</td>
<td>Red</td>
<td>256</td>
<td>AA</td>
<td>064</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-30</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>Pink</td>
<td>23</td>
<td>Red</td>
<td>256</td>
<td>AA</td>
<td>064</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-30</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>Pink</td>
<td>23</td>
<td>Red</td>
<td>256</td>
<td>AA</td>
<td>064</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-30</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>Pink</td>
<td>23</td>
<td>Red</td>
<td>256</td>
<td>AA</td>
<td>064</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-30</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>Pink</td>
<td>23</td>
<td>Red</td>
<td>256</td>
<td>AA</td>
<td>064</td>
<td>Rule = 06</td>
</tr>
<tr>
<td>Satisfies Rule-30</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>Pink</td>
<td>23</td>
<td>Red</td>
<td>256</td>
<td>AA</td>
<td>064</td>
<td>Rule = 06</td>
</tr>
</tbody>
</table>
CHAPTER VII: CONCLUSIONS AND SUGGESTIONS

In this thesis a general mathematical description of decision tables was introduced for the first time. This mathematical formalism served as a basis of the over-all investigation and allowed for the first time a precise treatment of the logical basis of decision tables as well as the conversion problem.

Chapter II presents the matrix description of decision tables with detailed classification. Based upon this, section II-2 introduces basic rule operations such as "simple rule splitting", "complete rule splitting" and rule intersection needed in the following discussion of the characteristics of decision tables given in II-2-2, and II-2-3. In the case of decision tables with unrelated conditions a formal theorem (II-2-1) on ambiguity was given. A precise definition of data dependent inconsistancy was also included and demonstrated. Based upon previous definition a clear definition in mathematical terms for completeness is given. In II-2-3 the characteristics of decision tables with related and unrelated conditions are analyzed and given in terms of possible ambiguity. A definition of completeness for this case is also given. To this field of decision tables with related conditions further investigation can be applied by the inclusion of any particular programming language.

Chapter III deals with a more practical aspect of decision tables, namely, manipulations of decision tables and relations
between them. In many practical cases in which the use of decision tables is applied the actual decision tables can become too large and lose clarity and effectiveness. In these cases splitting techniques should be applied. The splitting techniques developed in III-1-1 are based upon the definition of different linkages between decision tables. Three different splitting procedures were developed: splitting by rules, splitting by conditions and splitting by branching. Some of them were used later in different conversion techniques. The problem of merging decision tables was analyzed as well. Unfortunately, the only conclusion reached was that even under severe restriction there is no simple generalized procedure for merging; on which will not produce, in general, ambiguities or other logical errors. Further investigations into this subject without making use of more intelligent procedures for logical rather than structured analysis is impossible. Section III-2 deals with relations between decision tables. Few definitions for isomorphism and equivalence, based upon the previously defined rule operations and the rule combination defined in definition III-2-3 are given. The purpose of these definitions was again to supply guiding rules in the practical use of decision tables creating a way of deciding whether two tables are logically equivalent or not. Under relatively severe restriction automated procedures were developed for checking simple equivalence. This subject can be further developed and analyzed. Nevertheless, this was used in III-2-2 for development of decision tables minimization procedures.
Chapter IV includes an in depth study and analysis of the advanced conversion methods published in existing literature. The necessity of this study lies in the fact that in implementing a decision table processor, one must be very careful in choosing the conversion procedures. In this chapter every one of the different existing techniques was analysed, leading to a conclusion that not one of these procedures can be used separately for every case.

Chapter V is devoted to testing and diagnostic methods. This chapter is divided into three sections. The first section dealt with testing and diagnostic procedures at conversion time. The first set of theorems (for LCE decision tables) is based upon Chapter III and the paper published by P.H.J. King [B4] which was the first to give serious consideration to this subject. Theorems V-1-5 through V-1-8 (for MCE decision tables) were originated for the first time here. In the second section testing and diagnostic procedures at run time are presented (originally published by C.R. Muthukrishnan and V. Rajaraman [B7]), together with some extensions for conversion time. In the third section, two algorithms for LCE and MCE decision tables (with and without related conditions) for conversion and testing in conversion time and run time, are developed. These algorithms give a combined and efficient way of debugging.

In Chapter VI an attempt was made to suggest briefly a method for writing a flexible decision table COBOL pre-compiler. With respect to this Chapter it should be noted that an actual pre-compiler
package has not been produced. Also, due to lack of funds, no real pre-compiler was written and the conversion work of the collaborated example included, was done manually. The goal of designing a decision table processor which can serve as a form of direct communication language between the systems analyst and the computer was not accomplished. Most of the work was done with respect to the conversion and testing procedures. Except for suggesting the option of library and data base interface no attempt was made to suggest ways for data and errors description. This part is left open for further investigation in the subject which might lead eventually to the complete elimination of the programmer.

The purpose of this thesis was to provide a general tool and therefore it is expected that an additional number of results could have been obtained for special types of decision tables or for decision tables in special applications.

It is believed that this treatment of decision tables will provide the required tools for the potential decision table user, or the implementer of a decision table translator.
BIBLIOGRAPHY

The references marked with "*" were not available to me.

A. BIBLIOGRAPHY ON TREE TECHNIQUES.

by Solomon L. Pollack (The Rand Corporation, Santa Monica, Calif.)

[A2] A Procedure for Converting Logic Table Conditions into an Efficient Sequence of Test Instructions.
by J.F. Egler (McDonnell Aircraft Corp., St. Louis, Mo.)

[A3] Egler Procedure Refuted
by Michael Montalbano (IBM Advanced Systems Development Div., San Jose, Calif.)

by James R. Slage (Lawrence Radiation Laboratory, University of Calif., Livermore, Calif.)

by Laurence I. Press (San Fernando Valley State College, Northridge, California)

by Lewis T. Reinwald and Richard M. Soland (Research Analysis Corp., McLean, Virginia)

by Lewis T. Reinwald and Richard M. Soland.

[A8] Description of the Basic Algorithm in
by Michael D. Callahan and Hanson E. Chapman (System Development
B. BIBLIOGRAPHY ON MASK TECHNIQUES.


C. BIBLIOGRAPHY ON OTHER CONVERSION TECHNIQUES.

[C1] Programming Decision Tables in FORTRAN, COBOL or ALGOL.
by Cyril G. Veinott (Reliance Electric & Engineering Company Cleveland, Ohio)

D. GENERAL BIBLIOGRAPHY ON DECISION LOGIC AND RELATED TOPICS.

by W.V. Quine (Harvard University)

by W.V. Quine (Harvard University)

by T.A. Dolotta and E.J. McCluskey Jr. (Princeton University Princeton, New Jersey)

[D4] Simplification of Class of Boolean Functions.
by Edwin Hirshhorn (Northrop Aircraft Inc., Hawthorne, Calif.)

by John D.C. Little (Massachusetts Institute of Technology) Katta G. Murty (Indian Statistical Institute) Dura W. Sweeney (IBM Corporation) Caroline Karel (case Institute of Technology).

by C.B.B. Grindley

[D6] COBOL and Compatibility
DATAMATION, February 61, P.P. 30-34.
E. BIBLIOGRAPHY ON USAGE OF DECISION TABLES.

[E1] Decision Tables as a Tool.
by Daniel F. Langenwalter

[E2] Logic-Structur Tables
by H.N. Cantrell, J. King and F.E. H. King (General Electric Corp.
Schenectady, N.Y)

[E3] Tabular Form in Decision Logic.
by Burton Grad (IBM Corp.) Thomas J. Watson (Research Center
Yorktown Heights, N.Y)

by R.C. Nickerson (Numerical Methods Engineer, GE Comp.,
Schenecotady, New York)
Communications of the ACM, Volume 11/Number 4/November 1961
P.P. 516-520.

by Paul Dixon (Technical Staff
AUERBACH Corp.) 1962


[E7] Using Decision Structure Tables: Part One: Principals and Pre-
paration.
by D.T. Schmidt and T.F. KAVANAGH
DATAMATION FEBRUARY 1964
P.P. 42-52

[E8] Using Decision Structure Tables: Part Two: Manufacturing Applications
by D.T. Schmidt and T.F. KAVANAGH
DATAMATION MARCH 1964. P.P 48-54

by Paul Dixon (AUERBACH Corp.
Philadelphia PA.)
Computers and Automation April 1964
P.P. 14-19.

[E10] Data Documentation and Decision Tables.
by D.L. Fisher (IBM Corp., Los Gatos California)

by P.J. Vincent, 1967
[E12] Decision Tables by Martin L. Huges C.P.A.
Richard M. Shank and Elinor Svendsen Stein M.A.
MDI Publications 1968.

[E13] Life with Decision Tables at Manufacturers Life.
by J.R. Campbell
Canadian Datasystems April 1970

[E14] Decision Tables a Technique for Minimizing routine, repetitive design
by David Holstein (IBM Corp)

[E15] Decision Tables in Systems Design

by Herman McDaniel.
Editor Brandon/Systems Press Inc.
(A Collection of Articles)

[E17] Decision Table Software a Handbook.
by Herman McDaniel.
Brandon/Systems Press Inc.

[E18]*Decision Table Tutorial Using DETAB-X.

[E19]*Advanced Analysis Method for Integrated Electronic Data
by Evans D.Y. (1961)

[E20]*Tabular Form of Decision Logic.
by Grad B.
Datamation July 1961 P.P. 22

[E21]*Analysis of Decision Rules in Decision Tables.
by Pollack S.L.

[E22]*Decision Tables
by Morgan J.J.
Manage. Serv. 2(1965) P.P. 13-18

[E23]*Tabular Descriptive Language.
by Ians T.B.; and Grad B.

[E24]*Decision Table Experience on a File Maintenance System.
by Brown L.M.
Proc. Decision Tables Symposium, NYC: 75-80 (Sept. 1962)

[E25]*Place of Decision Tables and DETAB-X.
by Calkins L.W., Proc. Decision Tables Symposium, NYC: 9-12 (Sept 1962)
F. DECISION TABLES ORIENTED LANGUAGES.

[F1] TABSOL: A Fundamental Concept for System Oriented Languages.
by T.F. Kavanagh (Manufacturing Services, GE, New-York, N.Y.)

General Electric.


by G.W. Armerding Memorandum RM-3306-pr, September 1962
The Rand Corporation.

[F5] GETRAN: GECOM to COBOL Translator.
General Electric, 1966

by S.L. Pollack
The ACM 1966
by Working Group 2 on Decision Tables of the Special Interest Group on Programming Languages (SIGPLAN) of the Los Angeles Chapter of the ACM.


[F11] GECom II: GE-200 Series

GeGeneral Electric
Prepared and Published by Data Description and Transformation logic Task Forces, CODASYL.

[F12] DETAB-X: Preliminary Specifications for a Decision Table Structured Language.
by GE Computer Dept. Arizona No. CPB-147 (5M 3-61)

[F13]*GE 225 TABSOL Manual

[F14]*DETAB-X: An Improved Business-Oriented Computer Language.
by Pollack S.L.
Rand Corp. No. RM-3273-pr (Aug. 1962)

[F15]*What is DETAB-X

by Pollack S.L.

[F16]*TABSOL: Application Introduction to

by GE Computer Dept. Arizona No.
CPB-147A (5M 6-61)

[F17]*Approch to Decision Table Processor

by Wright K.R.
Proc. Decision Tables Symposium NYC: 41-44(Sept. 1962)

G. PREPROCESSORS LISTINGS.

[G1] DETAB/65 COBOL PREPROCESSOR LISTING

by Working Group 2 on Decision Tables of the Special Interest Group on Programming Languages (SIGPLAN) of the Los Angeles Chapter of the ACM.

[G2] NAYTABTRANS-C

by LTJG H.B. Towne U.S. Navy Nov. 68
VITAE

NAME: Albert Pinhas
BORN: Sofia, Bulgaria (Nov. 4, 1938)
SCHOOL: University of Tel-Aviv, Israel B. Sc. (1967)
in Applied Mathematics