Ambiguity in Limited Entry Decision Tables

P. J. H. King
University College of Wales, Aberystwyth, United Kingdom

The use of decision tables as a tool in systems analysis and for program specification is now becoming accepted. Rules on redundancy, contradiction, and completeness for limited entry tables were published in 1963. These are usually used for checking, preceded if necessary by a conversion from extended to limited entry form. Processors which automatically translate tables to more conventional program usually base their diagnostic facilities on these rules. In this paper it is suggested that these rules are unsatisfactory and that the important aspect of checking is to eliminate ambiguity from tables. Ambiguity is defined and discussed, and a procedure for producing checked-out decision tables is proposed. The theoretical basis of the algorithm used is established. The importance of well-designed diagnostic facilities in decision table processors is emphasized.

KEY WORDS AND PHRASES: decision tables, DETAB-65, systems analysis
CR CATEGORIES: 3.50, 4.19, 4.49

1. Introduction

In this paper ambiguity in limited entry decision tables is discussed and diagnostic facilities and checking procedures for dealing with it are suggested. It is assumed that the reader is familiar with the conventions and notation involved. For a general review of the subject including a bibliography, see [5]; a bibliography is also given in [3].

It is basic to decision table notation that only one of the action sets specified in a table is carried out for each piece of data. The function of the condition section, therefore, is to decide which of the distinct action sets of the table is to be carried out for a given piece of data. In a well-constructed table this will be done unambiguously. If a table is not well constructed in this sense, it will fail to determine, for some particular piece of logically permissible data, a unique action set. In this case we say the table is ambiguous. A formal definition of ambiguity can thus be given:

> A decision table is said to be ambiguous if, ignoring the else rule, there is a logically permissible piece of data which satisfies the condition entries of two rules having different action sets.

Consider the example shown in Figure 1. This is not a complete decision table as there is no action section. Also, the exact nature of the conditions is unspecified.

Whatever the conditions, it is clear that in any particular case R_1 and R_2 cannot both be satisfied. This is because R_1 requires a No outcome for C_3 whereas R_2 requires a Yes outcome. Similarly none of the rule pairs, $\{R_1, R_3\}$, $\{R_1, R_4\}$, $\{R_2, R_3\}$, and $\{R_2, R_4\}$, can be satisfied simultaneously, for in each case one rule requires a Yes outcome to a condition test for which the other requires a No. However, R_3 and R_4 are both satisfied if there is a No outcome for all three condition tests. On the basis of the information in Figure 1 there may be ambiguity between R_3 and R_4. This indicated ambiguity may be real or may be only apparent. If real, it indicates an error requiring correction as the table cannot be well formed in the sense discussed. If only apparent, no action is required. This latter will be the case if a simultaneous No outcome for all three conditions is *logically* impossible, or if the actions for the two rules are identical.

	R_1	R_2	R_3	R_4	E
C_1	Y	Y	—	N	
C_2	Y	—	N	—	
C_3	N	Y	N	—	

FIG. 1

	R_1	R_2	R_3
Age <18	Y	—	N
Age >65	—	Y	N
GO TO	1	2	3

FIG. 2

The first type of apparent ambiguity is illustrated in Figure 2. Here a Yes outcome for both conditions is clearly impossible, and the table is unambiguous. It satisfactorily determines, in all circumstances, the required action. This is because the conditions are not independent but related in a particular way. A Yes outcome to the first implies that a No would result if the second were tested. Note that this does not affect the interpretation of the dash entry for C_2 in R_1 which is still that of basic decision table format, namely that C_2 is not relevant to deciding whether R_1 holds and hence need not be tested, or if tested the result should be disregarded. That is, the interpretation of R_1 is "**if** C_1 **then** A_1." This implies nothing about the result if C_2 is, in fact, tested when R_1 holds. In the case of Figure 2 this would be a No because of the relation between the conditions. For a similar table where the conditions were independent or dependent in a way not having this implication, then testing C_2 might give either a Yes or a No.

The second type of apparent ambiguity is illustrated in Figure 3 which gives the rules for combining two digits d_1 and d_2 which may have values 0 or 1. The result is 1 if either or both is 1, and 0 only if they are both 0.

If both digits are 1 then the program derived from the table may behave as though R_2 were satisfied or as though

	R_1	R_2	R_3
$d_1 = 0$	Y	N	—
$d_2 = 0$	Y	—	N
result := 0	X	—	—
result := 1	—	X	X

FIG. 3

R_3 were satisfied depending upon the translation process used. However, as the action is the same in both cases, this is immaterial, and there is no ambiguity.

We assume that it is not practical or desirable to construct decision table processors which investigate the nature of the conditions. From the point of view of the processor, the information in the condition section of the

table is of the type shown in Figure 1. While it would be possible to deal with simply related conditions like those of Figure 2, relations between the conditions may be very complicated and not amenable to programmed investigation. The approach suggested in Section 2 is that of a dialog between the person analyzing the problem and constructing the table and the processor.

A different approach has been suggested and is given by Pollack in [7]. This assumes the entries in the condition stub are independent, and a definition of this is given. It appears, however, that this assumption is frequently disregarded when Pollack's theory is applied, and his rules for checking are taken to apply to decision tables in general. Thus his ideas are reiterated in [3] and have been used in the construction of the DETAB-65 preprocessor [2] without reference to the independence of conditions. If Pollack's rules are applied when conditions are not independent, the resulting program will still give the correct results. However, these rules are not satisfactory in these circumstances, since better checking procedures are available leading to more general decision tables giving better program.

In particular Pollack gives definitions of redundancy and contradiction for limited entry tables as follows:

1. A *redundancy* exists in a decision table if two or more rules do not have at least one Y,N pair in any of the rows and the actions specified are identical.

2. A *contradiction or logic error* exists in a decision table if two or more rules do not have at least one Y,N pair in any of the rows and the actions specified are not identical.

	R_1	R_2	R_3
Age <18	Y	—	N
Age >65	N	Y	N
GO TO	1	2	3

(a)

	R_1	R_2	R_3
Age <18	Y	N	N
Age >65	—	Y	N
GO TO	1	2	3

(b)

FIG. 4

First we consider the second of these definitions, that for "contradiction or logic error." Under this, the table of Figure 2 would be deemed to be incorrect since R_1 and R_2 break this rule. It would require to be corrected by one of the implied No's in R_1 or R_2 being explicitly stated. Thus, to be an allowable table for DETAB-65, the table should be rewritten in one of the two possible ways shown in Figure 4. At the time of constructing the table the decision as to which tables to take would be arbitrary.

A disadvantage of following this procedure and eliminating the apparent contradiction is that inefficient object programs may result. Suppose R_1 is the most frequent outcome in the context in which the program is to be used, then the flowchart of Figure 5 is the best implementation. This flowchart can be derived from Figure 2 but not from the table in Figure 4(a). This latter implies that both conditions *must* be tested to establish that R_1 holds. The interrelationship between the conditions makes this unnecessary, but this information is not available to the processor. For this reason the definition of contradiction given seems restrictive and unsatisfactory.

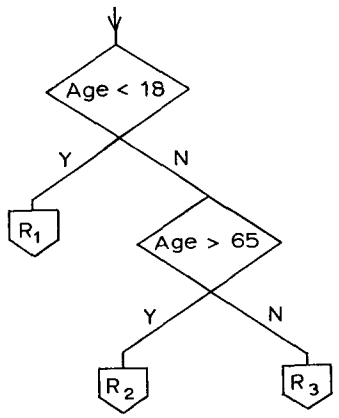

FIG. 5

We consider now the first of Pollack's definitions: that for redundancy. If we apply this to the table of Figure 3, we find that R_2 and R_3 are deemed to contain a redundancy. Pollack's procedure given in [7] requires this to be eliminated by including a Yes either for the second condition in R_2 or for the first condition in R_3. Then, when both d_1 and d_2 have the value 1, only one of the rules can be satisfied. This is unnecessary from the point of view of the table giving unambiguously the correct result. There can be disadvantage in eliminating this redundancy similar to that described for Figure 2.

A practical effect of these rules, therefore, is that programs may be obtained having longer run time than is strictly necessary. This conclusion has been illustrated by assuming that the condition portion of the table is to be converted into a branching network of tests. It also applies when the method of translating tables is by an interrupted rule mask technique of the type described in [4]. When tabular features are used in real-time programming this might be particularly important. The foregoing has used only very simple examples to illustrate the considerations involved. The disadvantage indicated becomes more important with larger tables and more complex situations.

A second aspect of Pollack's rules is that they impose on the systems analyst/programmer unnecessary, burdensome, and what may seem to him, obscure procedures. From his point of view the tables in Figures 2 and 3 are quite clear and satisfactory in their procedure specification. It is the author's view, therefore, that decision table facilities should allow these types of table.

We have argued that redundancy as defined by Pollack should, in general, be permitted and that his definition of

"contradiction or logic error" is unsatisfactory. If these views are accepted then some other form of checking procedure is required, and proposals for this are made in Section 2. We suggest that it is sufficient to eliminate ambiguity in the sense already defined. It is clear that this is not always possible without investigating the nature of the conditions. It has been suggested that processors should not do this, and in some circumstances therefore they cannot determine that a table is unambiguous. In these cases it will be concluded that, for certain outcomes, there is possible ambiguity. This will then have to be resolved by investigating relationships between the various conditions. For the sake of clarity we give a formal definition of the term "possible ambiguity" as used here:

> A decision table is said to have possible ambiguity if it cannot be decided whether or not it is ambiguous without investigating the nature of its conditions and the relationships between these conditions.

2. A Procedure for Decision Table Checking

The procedure proposed here for producing checked-out decision tables is that, when constructing the condition section, the analyst/programmer should aim at the minimum number of entries necessary to define adequately the action of the program. Thus, if a Yes entry to one condition implies necessarily a No outcome to another condition, he should not enter this, but indicate the outcome as nonpertinent. This avoids specifying tests which would be incorporated in the program but which are logically unnecessary. This should produce better programs.

A decision table processor will naturally have various diagnostic tests. It is suggested that these should include a test of each table for ambiguity. If none is possible, then a comment will be produced to this effect. If any possible ambiguities are found, then a tabulation is produced showing all outcomes for which these may occur. It is then the responsibility of the originator of the decision table to check this tabulation. He must ensure that all outcomes shown are logically impossible by virtue of relationships between the various conditions. In this case the compilation may proceed without any further action. If, however, any of the listed outcomes are logically possible then this indicates error in the table. The construction of a test for possible ambiguity is relatively simple if the program is based on the method outlined in Section 3.

Figure 6 shows the output from such a program for the tables previously discussed, 6(a) being that for Figure 2 and 6(b) that for Figure 3. The action required from the originator of the table in the first case is to check that the Yes/Yes outcome is logically impossible and thus to confirm that the table is satisfactory. No action at all is required in the second case as the table is quite unambiguous. More significant examples are in Section 4.

It is worth noting that, when checked out, the tabulation of possible ambiguities is the set of a priori excludable combinations required for the extension of the algorithms given by Reinwald and Soland in [8, 9].

3. Basis of the Program

The condition section of a limited entry decision table can be represented by two binary matrices: the mask matrix, M, and the decision matrix, D. This representation was originally presented by Kirk [6] and is described and discussed in detail by King [4] where the matrices for the table of Figure 1 are given. If a table has m conditions and n rules then these matrices have m rows and n columns.

Let $m_1, m_2, \cdots, m_n$ represent the successive columns of M, and $d_1, d_2, \cdots, d_n$ the columns of D. Each of the m_i and d_j is a binary vector of m elements. A column of the mask matrix specifies a subset of the conditions; thus m_2 specifies the conditions pertinent to R_2, a 1 indicating a pertinent condition, a 0 a nonpertinent condition. A column of the decision matrix specifies in the positions relating to pertinent conditions the required outcomes for the rule to be satisfied. Thus d_2 indicates the required outcomes for the pertinent conditions specified by m_2, a 1 indicating a Yes outcome and a 0 a No outcome. In this way an ordered pair of binary vectors $\{m_i, d_i\}$ specifies the condition test outcomes necessary for R_i to be satisfied, m_i specifying the conditions involved and d_i the outcomes for those conditions. Note that positions of d_i relating to nonpertinent conditions are zero. This is from the construction of the matrix D and is necessary for some of the manipulation below. It is not necessary, however, for the interpretation of an ordered vector pair as a set of outcomes, the positions in the second vector corresponding to zeros in the first being of no consequence in this interpretation.

In what follows the Boolean operators $\cap$ and $\cup$ are used in the conventional way to denote the combination of binary vectors on an element-by-element basis as specified

```
(a)        RULE          1  2  3
      AGE LT 18          Y  -  N
      AGE GT 65          -  Y  N

          GO TO          1  2  3

APPARENT AMBIGUITY FOR RULE  1
                 AND RULE    2

      AGE LT 18          Y
      AGE GT 65          Y

(b)        RULE          1  2  3
        D1 EQ Ø          Y  N  -
        D2 EQ Ø          Y  -  N

     RESULT = Ø          X  -  -
     RESULT = 1          -  X  X

TABLE IS UNAMBIGUOUS
```

FIG. 6

in Table I. We denote by e a vector consisting of all 1's. $A_1, A_2, \cdots, A_n$ denote the action sets corresponding to the rules $R_1, R_2, \cdots, R_n$.

We show that there is possible ambiguity (in the sense previously defined) between R_j and R_k if and only if

$$m_j \cap d_k = m_k \cap d_j \tag{1}$$

and

$$A_j \neq A_k . \tag{2}$$

The second of these two conditions is clear; possible ambiguity cannot arise between two rules if the corresponding

TABLE I

Elements in Operands	*Result with* $\cap$	*Result with* $\cup$
0 0	0	0
0 1	0	1
1 0	0	1
1 1	1	1

action sets are identical. In this case, even if a given set of data satisfies the condition entries of both rules, the action taken by the program will be unique. In the discussion of the first condition we therefore assume the second to hold.

The condition test outcomes for R_j to be satisfied are $\{m_j, d_j\}$, and the outcomes for R_k to be satisfied are $\{m_k, d_k\}$. The set of conditions pertinent to either R_j or R_k or to both is $m_j \cup m_k$. This set of conditions divides into three disjoint groups: those pertinent to R_j but not to R_k; those pertinent to R_k but not to R_j; and those pertinent to both. These groups are specified by the vectors $m_j \cap (e - m_k)$, $m_k \cap (e - m_j)$, and $m_j \cap m_k$ respectively. Now for the first group, if R_j is to be satisfied, the outcomes must be $\{m_j \cap (e - m_k), d_j\}$, and these outcomes do not prevent R_k from also being satisfied. Similarly for R_k to be satisfied, the outcomes for the second group must be $\{m_k \cap (e - m_j), d_k\}$ which do not prevent R_j from being satisfied. However for the third group, those pertinent to both R_j and R_k, we must have outcomes $\{m_j \cap m_k, d_j\}$ if R_j is to be satisfied, and outcomes $\{m_j \cap m_k, d_k\}$ if R_k is to be satisfied. If there is to be the possibility of ambiguity these must be identical, i.e. we must have

$$\{m_j \cap m_k, d_j\} = \{m_j \cap m_k, d_k\} \tag{3}$$

or equivalently

$$m_j \cap m_k \cap d_j = m_j \cap m_k \cap d_k \tag{4}$$

Now the positions in d_j containing a 1 are a subset of the positions in m_j containing a 1 by the construction of the vectors. Hence

$$m_j \cap d_j = d_j$$

and therefore

$$m_j \cap m_k \cap d_j = m_k \cap d_j \tag{5}$$

similarly

$$m_j \cap m_k \cap d_k = m_j \cap d_k \tag{6}$$

and hence (4) is equivalent to (1). Thus (1) and (2) *must necessarily hold* if there is to be possible ambiguity.

The argument used in the foregoing is reversible: i.e. if (1) holds then, by virtue of (5) and (6), (4) and hence (3) must also hold; i.e. the outcomes required for conditions common to R_j and R_k are identical. Then if the actions are different, ambiguity is possible; i.e. if (1) and (2) hold, ambiguity is possible.

Given the possibility of ambiguity, this will be encountered for a set of data having condition test outcomes

$$\{m_j \cup m_k, d_j \cup d_k\} \tag{7}$$

In the above discussion it was shown that for R_j and R_k both to be satisfied the conditions relevant to either or both, $m_j \cup m_k$, can be divided into three disjoint subsets and that the required outcomes for these are $\{m_j \cap (e - m_k), d_j\}$, $\{m_k \cap (e - m_j), d_k\}$, and $\{m_j \cap m_k, d_j\}$. These three subset outcomes combine to give the outcomes specified in (7).

The program operates by comparing every pair of rules for possible ambiguity using the conditions (1) and (2). Thus the loop structure is of the form

```
for j := 1 step 1 until n − 1 do
  for k := j + 1 step 1 until n do ···
```

If, for a pair of values of j and k, (1) and (2) are found to hold, then the values of j and k together with the outcomes giving rise to ambiguity specified by (7) are recorded for inclusion in the tabulation of apparent ambiguities.

4. Some Illustrative Results

Examples of output from a decision table processor of the type described have been given in Figure 6. Although useful as illustrations, they are trivial from a practical viewpoint. In this section are two more meaningful examples.

In many business data processing situations the conditions are highly related. For example, installment buying where payments are made in cash on a weekly basis, the action taken when an account goes into arrears is a crucial aspect of the operation. This is an area where decision tables have been used with considerable success.

The table of Figure 7 illustrates a simplified arrears procedure. It is seen that the first three conditions are directly related. Thus a No outcome for the third condition implies No outcomes to the first two conditions. The last two conditions are also directly related. It is worth noting that there is also a probabilistic relationship between some of the conditions. Consider the third and fourth conditions: a customer who has made no payments for the last three weeks is probably (but not necessarily) more than three weeks in arrears. (This type of relationship is, however, outside the scope of this paper which is concerned only with *logical* relationships.)

Figure 7 is the processor output for this table, the table itself being given initially, followed by a tabulation of apparent ambiguities with the condition stub repeated for convenience. We see that the first possible ambiguity listed is between R_1 and R_5, and the outcome giving rise to it is Yes to the first condition and No to the third and fifth conditions, the second and fourth conditions being nonpertinent for both R_1 and R_5. We check that this outcome is logically impossible since if there has been no cash payment made for the last three weeks then this week's payment (which is therefore zero) cannot be greater than the standard weekly rate of payment. Similarly the outcome listed for R_1 and R_8 is logically impossible because this week's cash cannot be greater than the weekly rate unless it is greater than zero. The analyst must check each of the apparent ambiguities listed. If he finds one that is *not* logically impossible then there is an error in the table which has been pinpointed, and correction and reprocessing are required.

```
                    DECISION TABLE AMBIGUITY ANALYSIS

TABLE HAS  5 CONDITIONS,  9 RULES,  6 ACTIONS.

                                    RULE    1  2  3  4  5  6  7  8  9
THIS WEEKS CASH GREATER WEEKLY RATE         Y  Y  N  N  -  -  -  -  -
THIS WEEKS CASH GREATER ZERO                -  -  Y  Y  -  -  N  N  N
ANY CASH DURING LAST THREE WEEKS            -  -  -  -  N  N  Y  Y  Y
ARREARS GREATER 3 X WEEKLY RATE             -  -  Y  N  -  -  N  Y  -
ARREARS GREATER 6 X WEEKLY RATE             N  Y  N  -  N  Y  -  N  Y

SEND ARREARS LETTER A                       -  X  -  -  -  -  -  -  -
SEND ARREARS LETTER B                       -  -  X  -  -  -  -  -  -
SEND ARREARS LETTER C                       -  -  -  -  X  -  -  -  -
SEND ARREARS LETTER D                       -  -  -  -  -  -  -  X  -
NOTE ACCOUNT                                X  -  -  X  -  -  X  -  -
TAKE SPECIAL ARREARS ACTION                 -  -  -  -  -  X  -  -  X


                APPARENT AMBIGUITY FOR RULE 1  1  2  2  2  3  4  4  7
                                   AND RULE 5  8  6  7  9  5  5  6  9

THIS WEEKS CASH GREATER WEEKLY RATE         Y  Y  Y  Y  Y  N  N  N  -
THIS WEEKS CASH GREATER ZERO                -  N  -  N  N  Y  Y  Y  N
ANY CASH DURING LAST THREE WEEKS            N  Y  N  Y  Y  N  N  N  Y
ARREARS GREATER 3 X WEEKLY RATE             -  Y  -  N  -  Y  N  N  N
ARREARS GREATER 6 X WEEKLY RATE             N  N  Y  Y  Y  N  N  Y  Y
```

FIG. 7

```
                    DECISION TABLE AMBIGUITY ANALYSIS

6 CONDITIONS,  1Ø RULES,  11 ACTIONS.

                                    RULE    1  2  3  4  5  6  7  8  9  1Ø
                              LRN EQ 3      Y  Y  Y  -  -  -  -  -  -  -
                              LRN EQ 2      -  -  -  Y  Y  -  -  -  -  -
                              NN8 EQ Ø      -  -  -  -  -  N  -  -  -  -
                              KN  EQ Ø      -  -  -  -  -  -  N  -  -  -
                              LN  EQ Ø      N  Y  Y  -  -  -  -  N  -  -
                              NSTRT EQ NM   -  Y  N  Y  N  -  -  -  Y  N

                              ISWI=8        -  -  -  -  -  X  -  -  -  -
                              J=NSTRT       X  -  -  -  -  X  X  X  -  -
                              GO TO 2Ø6     -  -  -  -  -  X  -  -  -  -
                              ISWI=KN       -  -  -  -  -  -  X  -  -  -
                              KK=KN+1       -  -  -  -  -  -  X  -  -  -
                              GO TO 226     -  -  -  -  -  -  X  -  -  -
                              ISWI=LN       X  -  -  -  -  -  -  X  -  -
                              KK=LN+1       X  -  -  -  -  -  -  X  -  -
                              GO TO 236     X  -  -  -  -  -  -  X  -  -
                              ERFLAG=2.Ø    -  -  X  -  X  -  -  -  -  X
                              RETURN        -  X  X  X  X  -  -  -  X  X

APPARENT AMBIGUITY FOR RULE   :  1  1  1  1  1  2  2  3  3  4  4  4
                     AND RULE 4  5  6  7  9  1Ø 6  7  6  7  6  7  8

                              LRN EQ 3      Y  Y  Y  Y  Y  Y  Y  Y  Y  -  -  -
                              LRN EQ 2      Y  Y  -  -  -  -  -  -  -  Y  Y  Y
                              NN8 EQ Ø      -  -  N  -  -  -  N  -  N  -  N  -  -
                              KN  EQ Ø      -  -  -  N  -  -  -  N  -  N  -  N  -
                              LN  EQ Ø      N  N  N  N  N  N  Y  Y  Y  Y  -  -  N
                              NSTRT EQ NM   Y  N  -  -  Y  N  Y  Y  N  N  Y  Y  Y

APPARENT AMBIGUITY FOR RULE   5  5  5  6  6  6  6  7  7  7  8  8
                     AND RULE 6  7  8  7  8  9  1Ø 8  9  1Ø 9  1Ø

                              LRN EQ 3      -  -  -  -  -  -  -  -  -  -  -  -
                              LRN EQ 2      Y  Y  Y  -  -  -  -  -  -  -  -  -
                              NN8 EQ Ø      N  -  -  N  N  N  N  -  -  -  -  -
                              KN  EQ Ø      -  N  -  N  -  -  -  N  N  N  -  -
                              LN  EQ Ø      -  -  N  -  N  -  -  N  -  -  N  N
                              NSTRT EQ NM   N  N  N  -  -  Y  N  -  Y  N  Y  N
```

FIG. 8

The second example is from the field of numerical analysis. In [1] an integration subroutine is given in an appendix. For comparison this is written in FORTRAN and in FORTAB. The latter version contains the table in Figure 8.

The rules regarding contradiction and redundancy for DETAB-65 tables do not apply in the case of FORTAB. Here the philosophy is that "anything goes," and if there is ambiguity then the rules will be decided on a "left-to-right" basis or in an order which the user can specify. From the point of view of the user, of course, it is still necessary to check the tables for error, and therefore any ambiguity in the final table must be only apparent. From Figure 8 we see that the table contains a large number of apparent ambiguities. Many of these are straightforward; for example, that between rules 1 and 4 is clearly only apparent since LRN cannot be equal to both 2 and 3. The table does not, however, contain sufficient information to decide the logical impossibility of, say, rules 7 and 10. Greater knowledge of the problem than the table provides is required to decide whether it is possible for KN not to be zero at the same time as NSTRT is not equal to NM.

5. Conclusion

As already indicated, it is the author's view that the present position with regard to ambiguity in limited entry tables is unsatisfactory. On the one hand, we have the philosophy of DETAB-65 which is essentially that of assuming the conditions to be unrelated and hence that all combinations of Y's and N's occur in practice. This is unrealistic, since it is in situations with relationships between conditions that decision tables have been used most successfully and where their potential is probably the greatest. The practical consequence is to impose on the analyst checking procedures dictated by computer processing convenience, which, in terms of the logic of the situation, may seem obscure and unnecessary. The alternative approach of FORTAB, where from the point of view of the processor "anything goes," is equally unsatisfactory. Here the user must decide and must invent his own checking procedures to locate errors in tables. The considerable value of error diagnostic comments in developing programs is thus ignored. These are likely to be very valuable in checking out large decision tables. The method outlined in this paper has proved useful, and it is suggested that this represents an advance.

Acknowledgment. I am grateful to Messrs. W. Clark and F. J. J. Johnston of General Information and Control Systems Ltd. for access to information on which Figure 7 is based.

RECEIVED SEPTEMBER, 1967; REVISED APRIL, 1968

REFERENCES

1. ARMERDING, G. W. FORTAB: A decision table language for scientific computing applications. Memo RM-3306-PR, Rand Corp., Santa Monica, Calif., Sept. 1962.
2. CALLAHAN, M. D., AND CHAPMAN, A. E. Description of basic algorithm in DETAB/65 preprocessor. *Comm. ACM 10*, 7 (July 1967), 441–446.
3. CHAPMAN, A. E. DETAB-65 preprocessor. SHARE program library package, SDA 3396, 1966.
4. KING, P. J. H. Conversion of decision tables to computer programs by rule mask techniques. *Comm. ACM 9*, 11 (Nov. 1966), 796–801.
5. ——. Decision tables. *Comput. J. 10* (Aug. 1967), 135–142.
6. KIRK, H. W. Use of decision tables in computer programming. *Comm. ACM 8*, 1 (Jan. 1965), 41–43.
7. POLLACK, S. L. Analysis of decision rules in decision tables. Memo RM-3669-PR, Rand Corp., Santa Monica, Calif., 1963.
8. REINWALD, L. T., AND SOLAND, R. M. Conversion of limited-entry decision tables to optimal computer programs I: minimum average processing time. *J. ACM 13*, 3 (July 1966), 339–358.
9. ——. Conversion of limited-entry decision tables to optimal computer programs II: minimum storage requirement. *J. ACM 14*, 4 (Oct. 1967), 742–756.