The conversion of decision tables to sequential testing procedures

P. J. H. King and R. G. Johnson*

Department of Computer Science, Birkbeck College, University of London, Malet Street, London WC1E 7HX

This paper presents a general method for the conversion of a decision table to a sequential testing procedure represented as a tree. The method resolves difficulties which have been discussed previously in the literature but leaves open the question of the production of optimum or near optimum trees related to particular criteria. It provides, however, a framework within which such optimising methods may be developed. Suggestions are also made for improving the concepts and notations used in extended entry tables and for their use in translation.

(Received March 1974)

A number of papers including particularly Press (1965), Pollack (1965), Reinwald and Soland (1966, 1967), and Callahan and Chapman (1967), have considered the construction of sequential testing procedures or decision trees from a given limited entry decision table. In general, for limited entry tables, such methods proceed by the selection of a condition from the table and the derivation of two subtables not containing the selected condition, one as the successor of the true or Yes branch from this condition, and the other as the successor of the false or No branch. The two subtables are obtained by deleting the row for the selected condition from the table and, in the case of the subtable for successor to the Yes branch, all of the rules for which the condition entry was 'N', while for the successor table to the No branch, all those rules for which the condition entry was 'Y'.

The approach is illustrated using the table of **Fig. 1** with the further information that the three conditions are logically independent. A discussion of logical dependence and a definition of logical independence for the conditions of a limited entry table is given by King (1969). The sets of actions to be taken for each of the five rules are denoted by $A_1, A_2, \ldots A_5$.

C_1	Y	Y	—	N	N
C_2	Y	N	N	Y	N
C_3	—	Y	N	—	Y
	A_1	A_2	A_3	A_4	A_5

Fig. 1

If the first condition is selected for initial evaluation then the two subtables obtained are shown in **Fig. 2.**

If the subtables are now treated similarly and, for the sake of illustration, we select C_2 for next evaluation on the left branch and C_3 on the right branch then we obtain the tree shown in **Fig. 3** with single condition tables remaining to be developed. On developing the single condition tables the tree shown in **Fig. 4** is obtained.

In the process which we have illustrated informally three distinct activities can be identified. These activities are:

(*a*) the selection of conditions

(*b*) the editing of a table to remove a selected condition and produce successor subtables

(*c*) the interpretation of final tables as specifying that one or more of the action sets are to be carried out.

Of these three activities the selection of conditions for testing is the critical aspect from the point of view of the derivation of optimal sequential testing procedures but this aspect can only be considered when general and well defined methods exist for the mechanics of the other basic activities. In practice one would expect to have general purpose processors to carry out the conversion to a sequential testing procedure which could be particularised to different methods of condition selection by the addition of a condition selection procedure. An example of this approach is given by Dial (1970).

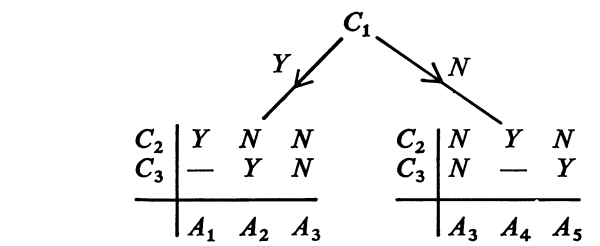


Fig. 2

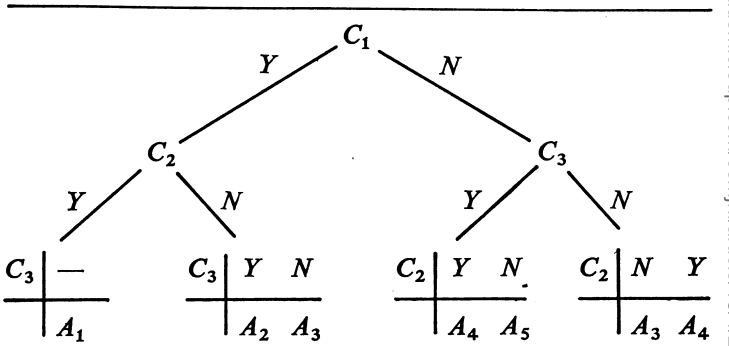


Fig. 3

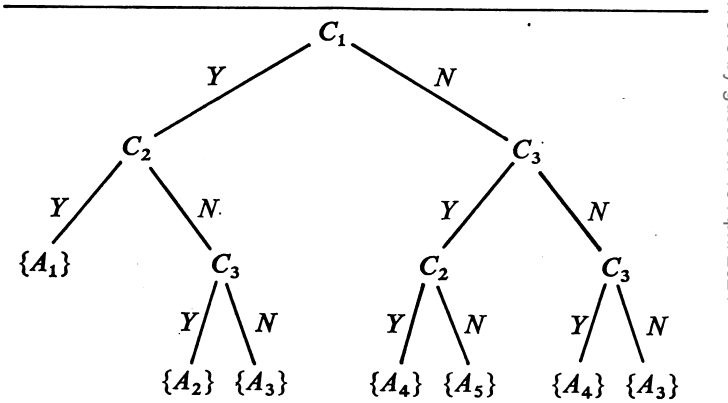


Fig. 4

Normally the objective of an optimising condition selection procedure is to minimise run-time. Whilst it is known that a

*now at: Computer Analysts and Programmers Ltd., 14/15 Great James Street, London, WC1N 3DY.

procedure to produce the sequential testing procedure which is the absolute optimum with respect to run-time will be complex (see Reinwald and Soland, 1966), a number of condition selection procedures have been discussed in the literature which are inherently more simple and have been regarded by their authors as being sufficiently near optimal for many practical purposes. Examples of such procedures are those of Schwayder (1971), Ganapathy and Rajaraman (1973), and Verhelst (1972). The latter, which was originally thought by its author to produce the absolute optimum, is discussed further by King and Johnson (1974). An elementary and incomplete but nonetheless readable and interesting discussion of the problem of deriving optimum sequential testing procedures from decision tables is to be found in Humby (1973) and a discussion of the basic information required for its solution is to be found in Inglis and King (1968). A brief discussion can also be found in Pollack *et al.* (1971).

Relationships among conditions

The table discussed in the preceding section was stated to have logically independent conditions. A table may, however, have specified an associated group of logical relations which always hold among its conditions and, in general, when we use the term 'table' this term should be taken as including the statements of the logical relationships among its conditions. Consider the table shown in **Fig. 5** which has been used for illustration previously by King (1967) and was subsequently discussed by Pollack (1967), and King (1969).

(C_1) AGE < 60	Y	Y	—	N
(C_2) AGE > 20	Y	—	N	—
(C_3) SEX = 'M'	N	Y	N	—
	A_1	A_2	A_3	A_4

(C_1, N) AND (C_2, N) IMPOSS.

Fig. 5

The logical relationship shown in Fig. 5 states that it is not possible to obtain a No outcome to both C_1 and C_2 for a particular transaction. It should be noted, see King (1969), that the logical relationship can also be stated in either of the two forms

(C_1, N) IMPLIES (C_2, Y) ,

and (C_2, N) IMPLIES (C_1, Y) ,

and that from any of these three statements the other two may be derived. The logical relationships may give rise to implied entries which can be shown explicitly in the table as in **Fig. 6.**

C_1	Y	Y	(Y)	N
C_2	Y	—	N	(Y)
C_3	N	Y	N	—
	A_1	A_2	A_3	A_4

(C_1, N) AND (C_2, N) IMPOSS.

Fig. 6

The development of a tree from the table of Fig. 6 is illustrated in **Fig. 7** where it is seen that consideration of the implied entries has ensured that only logically possible paths are included in the tree. Note that the successor subtables shown in Fig. 7(*a*) result from the editing process described previously but that further editing has been carried out on the No subtable, both condition rows having been deleted, to give the primitive table shown down the No branch from C_1 in Fig. 7(*b*). This deletion of condition rows may be made since the subtable

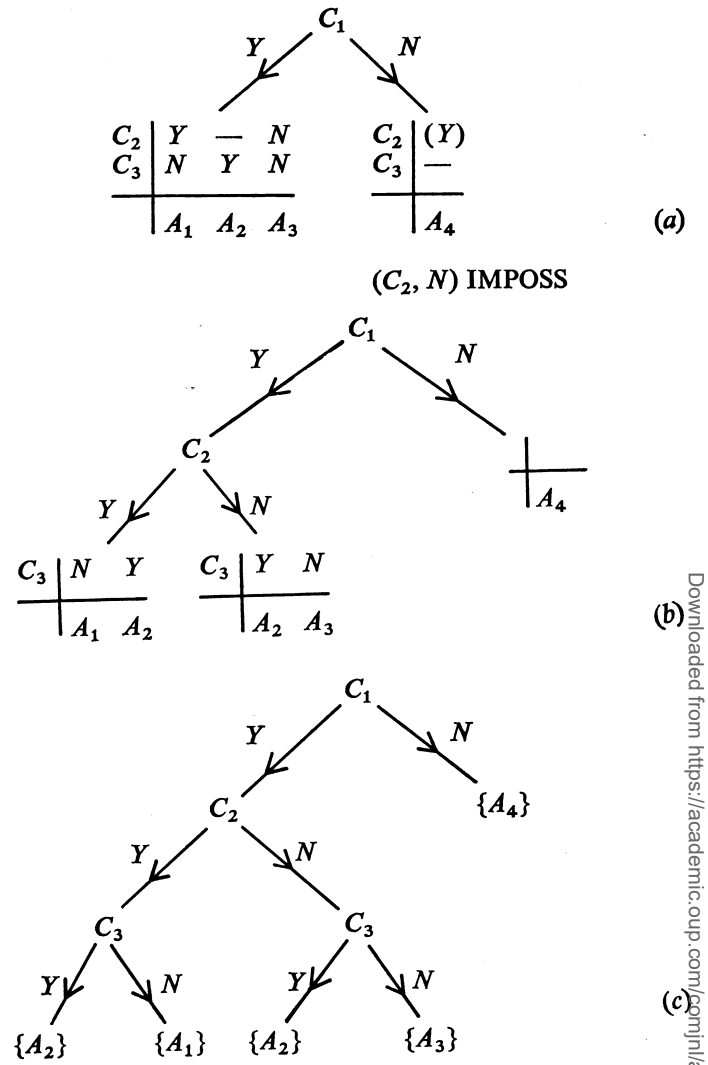


Fig. 7

does not specify that any testing of C_2 and C_3 is required.

In addition to editing the table to produce successor subtables for both the Yes and No branches from the condition selected, it is also necessary to edit the statements of logical relationships among conditions associated with the table to produce statements of logical relationship to be associated with the successor subtables. This aspect is discussed in more detail below. In the example considered in Fig. 7, the statement of logical relationship shown in Fig. 6 provides no information when C_1 is true so that there are no logical relationships to be considered with the Yes branch subtable shown in Fig. 7(*a*). When C_1 is false then the statement of Fig. 6 reduces to a statement that C_2 is necessarily true and this fact is indicated by the bracketed entry in the No successor subtable in Fig. 7(*a*). Note that this situation involves a slight widening of the interpretation of bracketed entries from that shown in Fig. 6 since they now also indicate outcomes implied by tests which have taken place in the course of reaching this particular point on the partial tree. Condition rows which contain only implied entries and dashes may be immediately deleted in the editing process as occurred in Fig. 7.

Overall table meaning

In the first proposals for the use of decision tables in computing the convention was adopted that one and only one rule of a decision table should be satisfied for each transaction and methods for converting tables to sequential testing procedures have hitherto been based on this assumption. It now appears that in certain circumstances and applications other conven-

tions may be used with advantage. In particular the multi-rule convention proposed by Barnard (1969) has been used most successfully in the FILETAB system (1972).

The different possible forms of overall table meaning were discussed by King (1969), are further discussed by King and Johnson (1973), and commented upon by Harrison (1973). Various authors, e.g. Press (1965), have suggested that a drawback of the sequential testing procedure method of translation is that it cannot handle ambiguous tables satisfactorily. Press concluded that for this reason binary mask methods of the type introduced by Kirk (1965) are to be preferred with apparently ambiguous tables. It now seems clear that suitably designed tree methods can satisfactorily handle tables conforming to any of the conventions on overall table meaning. When a table is required to conform to the one and only one rule convention the method described in this paper will detect errors by identifying those circumstances in which more than one rule can be satisfied.

The general method proposed in this paper is further illustrated using the table shown in **Fig. 8** which specifies that a distinct group of actions, A_1, is required if 'seat revenue' exceeds £7,500, actions A_2 are required if 'sales revenue' exceeds £2,500, actions A_3 if total revenue exceeds £10,000, and actions A_4 if none of these three conditions hold. Note that total revenue = seat revenue + sales revenue + sundry revenue so that the third condition may hold without either of the first two conditions holding. If the first *and* second conditions hold, however, then the third condition will necessarily hold and all three action sets, A_1, A_2 and A_3 are required. It should be noted that in the initial table the logical relationship among the conditions does not give rise to implied entries of the form (Y) or (N).

(C_1) SEAT REVENUE > 7,500	Y	—	—	N
(C_2) SALES REVENUE > 2,500	—	Y	—	N
(C_3) TOTAL REVENUE > 10,000	—	—	Y	N
	A_1	A_2	A_3	A_4

(C_1, Y) AND (C_2, Y) IMPLIES (C_3, Y)

Fig. 8

The logical relationship stated in Fig. 8 may also be stated in, among others, the following three alternative ways:

(C_1, Y) AND (C_2, Y) AND (C_3, N) IMPOSS.

(C_1, Y) AND (C_3, N) IMPLIES (C_2, N).

(C_2, Y) AND (C_3, N) IMPLIES (C_1, N) .

For convenience a briefer notation is used in the following discussion where we write C_i in place of (C_i, Y), $\neg C_i$ in place of (C_i, N), and use the symbols $\wedge$ in place of AND and $\rightarrow$ which stands for IMPLIES when there are symbols to the right and IMPOSS when there are no symbols to the right. (See Appendix).

In **Fig. 9** we show the stages in the development of a sequential testing procedure from the table illustrated in Fig. 8. Fig. 9(*a*) shows the table and its associated condition relationship in the briefer notation we are now adopting. In Fig. 9(*b*) we see that the Yes and No branch successor subtables include the results of editing the logical relationship statement shown in Fig. 9(*a*). This process involved substituting for the condition tested, the value which obtains down the particular branch under consideration, simplifying the resulting logical statement, and associating the result with the particular subtable if this result is not always satisfied by the possible combinations of values of the conditions remaining to be tested.

For the Yes branch the substitution yields

$$\textbf{true} \wedge C_2 \wedge \neg C_3 \rightarrow$$

which simplifies to

$$C_2 \wedge \neg C_3 \rightarrow$$

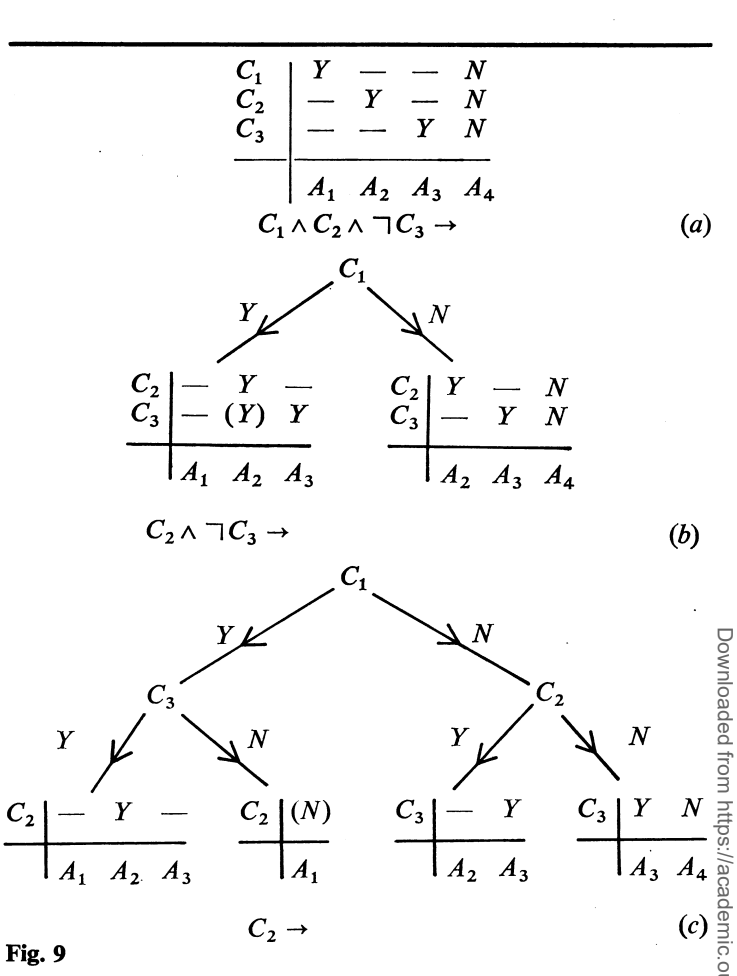


Fig. 9

which has the alternative forms $C_2 \rightarrow C_3$ and $\neg C_3 \rightarrow \neg C_2$ and is associated with the Yes branch subtable. It gives rise to the implied entry (Y) shown in that subtable.

For the No branch the substitution yields

$$\textbf{false} \wedge C_2 \wedge \neg C_3 \rightarrow$$

which is satisfied for all four possible combinations of values of C_2 and C_3. There is, therefore, no constraint on the values of the conditions that may occur down the No branch and so no statements of logical relationships are associated with that subtable. The reader may care to verify that substituting in the three alternative forms of the relationship

$$C_1 \wedge C_2 \rightarrow C_3 ,$$

$$C_1 \wedge \neg C_3 \rightarrow \neg C_2 ,$$

$$\text{and } C_2 \wedge \neg C_3 \rightarrow \neg C_1$$

yields the alternative forms

$$C_2 \rightarrow C_3$$

$$\neg C_3 \rightarrow \neg C_2$$

for the Yes branch and no constraints for the No branch.

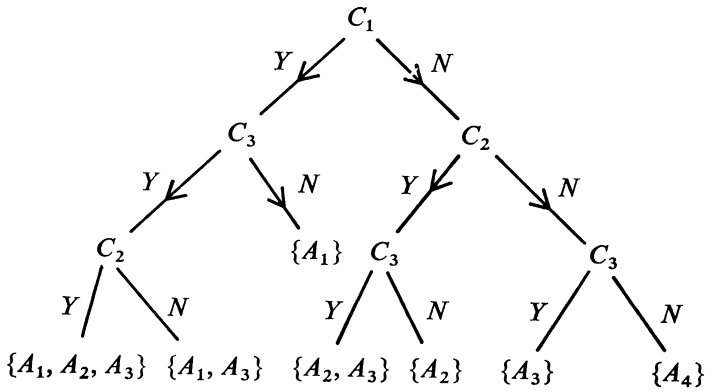


Fig. 10

In Fig. 9(*c*) we show the further development of the subtables shown in Fig. 9(*b*). The development of the No branch subtable is straightforward. In the left branch subtable the values of the condition C_3 are substituted in the logical relationship to yield the statements

$$C_2 \wedge \neg \textbf{true} \rightarrow$$

$$\text{and } C_2 \wedge \neg \textbf{false} \rightarrow$$

to be associated with the Yes branch and No branch subtables respectively. The first of these statements is always satisfied and does not, therefore, constitute a constraint whereas the second states that C_2 will necessarily be found to be false down the No path and the statement gives rise to the (*N*) entry shown in the subtable.

In **Fig. 10** we show the development of the remaining subtables of Fig. 9(*c*). It will be seen that this flowchart shows that if all three conditions are true, which is a logically possible outcome, then the first three rules of the table are all satisfied by the transaction. Again if the first condition is false and the remaining two are true then the transaction satisfies both the second and third rules.

Completeness and ambiguity for decision tables

Particularly in the earlier literature on decision tables much emphasis was placed on the checking of tables to ensure that all possible combinations of condition outcomes were covered and that for each such outcome the actions required were clearly specified.

For the purpose of formalising these checking procedures a decision table is said to be *complete* if it specifies an action set for every logically possible transaction. A table which is not complete is said to be *incomplete*. A table is said to be *ambiguous* if more than one action set is specified for any logically possible transaction. In many of the classical uses of decision tables a requirement has been that a table should be complete and not ambiguous, i.e. one and only one action set is specified for every logically possible transaction. A table which is not ambiguous is said to be *unambiguous*.

Methods for translating tables to sequential testing procedures such as those of Pollack (1965), and Callahan and Chapman (1967), have been applicable only to tables which are unambiguous and complete or have been made complete by the inclusion of an 'else' rule.

In the process for the derivation of sequential testing procedures described in this paper there is no restriction on the properties of the table from which the testing procedure is derived. In the table shown in Fig. 8 it is possible for a transaction to satisfy all of the first three rules and so the three action sets, A_1, A_2 and A_3 may all be specified. It is seen that these three action sets are specified for the path corresponding to all three conditions being true in the testing procedure shown in Fig. 10 where it will also be noted that there is a path which specifies both A_1 and A_3, and also one which specifies both A_2 and A_3.

Similarly if the table is not complete, i.e. there are logically possible transactions which satisfy none of the rules, then there will be one or more paths in the testing procedure which result in the selection of no rules at all. We illustrate this aspect in **Fig. 11**(*b*) where a development of the table of **Fig. 11**(*a*) is shown.

It is seen, therefore, that the method described in this paper is applicable to the derivation of a sequential testing procedure from a decision table whatever the properties of the table in regard to ambiguity and completeness. These matters relate to the interpretation and use of the sequential testing procedure obtained but do not in any way inhibit the mechanics of its derivation.

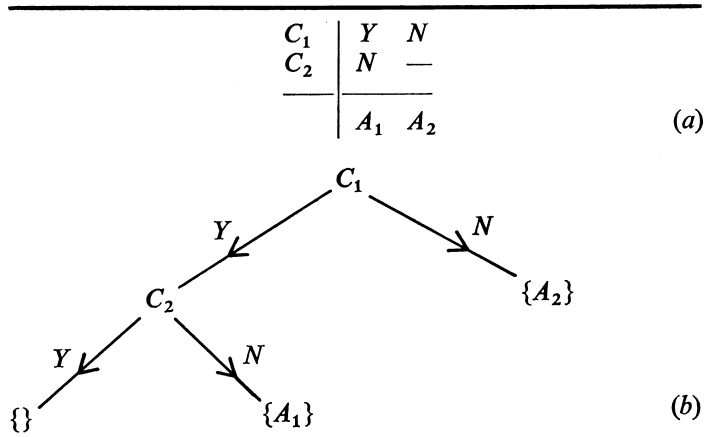


Fig. 11

Primitive final tables with no conditions

In the process we have informally described, the number of conditions in a successor table is at least one less than the number of conditions in the table at the immediately preceding higher level from which it was derived. In the next section we formalise the algorithm for the process of constructing a sequential testing procedure from a decision table. In this formalisation the splitting or 'bifurcation' of a table on a condition to form its two successor subtables is applied iteratively until only tables with no conditions at all remain. We describe such tables with no conditions as 'primitive final tables'.

It can be readily seen that a primitive final table will always be one of the three types shown in **Fig. 12.**

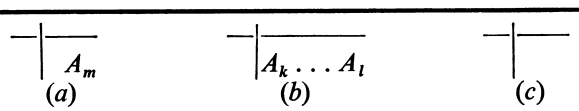


Fig. 12

In Fig. 12(*a*) we have a table with a single rule specifying the action set A_m, Fig. 12(*b*) shows a table with more than one rule each having one of the distinct action sets $A_k, \ldots, A_l$, and in Fig. 12(*c*) we have a null table with no conditions and no rules.

For each transaction satisfying the path through the sequential testing procedure terminated by a final table with a single rule of the type shown in Fig. 12(*a*), the single action set, A_m, is specified. A single rule final table is thus complete and unambiguous in the sense that for every transaction satisfying the path leading to it, one and only one action set is specified.

For each transaction satisfying a path terminated by a multi-rule final table of the type shown in Fig. 12(*b*), the action sets, $A_k, \ldots, A_l$, are all specified. A final table with more than one rule is thus complete in the sense that all transactions satisfying the path leading to it cause the specification of at least one action set, and ambiguous in the sense that it does not specify a unique action set for each transaction.

A null final table as shown in Fig. 12(*c*) is unambiguous since for a transaction satisfying the path leading to it we do not have more than one action set specified. It is not complete since it does not specify an action set for these transactions.

It has been shown by Johnson (1974) that if a decision table is complete then all of the final tables in any sequential testing procedure derived from it will also be complete, i.e. of the form shown in Figs. 12(*a*) and 12(*b*), and conversely that if, in a sequential testing procedure developed from a decision table, all of the final tables are complete, then the original table is complete. Johnson has also shown that a similar relationship holds for the property of ambiguity. In this case if any of the final tables is of the form of Fig. 12(*b*) then the original table is ambiguous, and conversely, in a sequential testing procedure derived from an ambiguous decision table at least one of the

final tables will be of the form of Fig. 12(*b*).

From the above results we see that if a table is required to conform to any particular convention in regard to completeness and ambiguity then this can be determined by an examination of the final tables in any arbitrary sequential testing procedure derived from the table. Thus if all of the final tables are of the form of Fig. 12(*a*) then the original table is complete and unambiguous.

A formalisation of the algorithm

The process of deriving a sequential testing procedure from a decision table which has been described informally is now specified more precisely. As stated previously we require the set of logical relationships, if any, which exist among its conditions, to be associated with the table.

The process of bifurcation of a table on an arbitrarily selected condition to form the true (Yes branch) and false (No branch) successor table is first defined. The steps in this bifurcation process are:

1. Using the statements of logical relationships among the conditions, make explicit in the table all possible implied entries.
2. Select a condition from among those present in the table and construct the two successor subtables. Include in these tables all of the condition lines in the original table except the one selected. Assign to the Yes branch successor table all rules with a Yes, implied Yes, or dash entry, and to the No branch successor table all those rules with a No, implied No, or dash entry. Rules with a dash entry for the selected condition thus appear in both successor tables.
3. The logical relations among the conditions of the two successor tables are obtained by substituting, in the case of the Yes branch table, the value **true** for the condition selected and for the No branch table, the value **false.** The logical relationships should then be simplified.
4. If a successor table is produced containing one or more conditions, is complete, and has the same action set specified for every rule, then it should be replaced by a complete and unambiguous primitive final table with that action set, i.e. a table of the type shown in Fig. 12(*a*).

The process of constructing a complete sequential testing procedure from a decision table proceeds by the selection of a condition from the original table and its bifurcation using this condition, the selection of conditions in the successor tables and their bifurcation *et seq*, until a tree is formed in which each path terminates in one or other of the types of primitive final table. The primitive final tables are then each interpreted as specifying either no action set, one action set, or several action sets.

The recognition of common subtables

The method described will generate a sequential testing procedure in the form of a binary tree. It has been suggested by Sprague (1966) that a sophisticated procedure might recognise common subtables when they occur and so represent the sequential testing procedure in a more compact form as a directed graph. We illustrate this process using the table shown in **Fig. 13,** where the first and third rules specify the same actions as do the second and fourth. The conditions in the table are

C_1	Y	Y	N	N	N
C_2	—	—	Y	Y	N
C_3	Y	N	Y	N	—
	A_1	A_2	A_1	A_2	A_3

Fig. 13

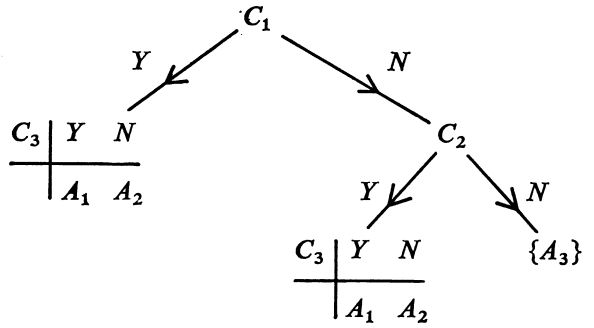


Fig. 14

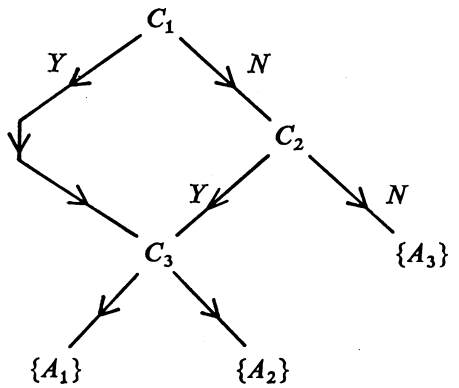


Fig. 15

logically independent so there are no associated logical relations.

If for the initial bifurcation C_1 is chosen and for bifurcation of the resulting No branch subtable C_2 is chosen we obtain the partially developed procedure shown in **Fig. 14.**

The remaining two tables to be developed are identical and therefore the branches leading to them may be linked. If this is done and the table then bifurcated on the remaining condition we obtain the representation of the sequential testing procedure shown in **Fig. 15.**

It should be noted that this recognition of common subtables results only in a more concise representation of the sequential testing procedure and hence some economy in store utilisation. It does not improve the run-time of the procedure in any way and, in some circumstances, it can frustrate the production of an optimum procedure with respect to run-time since the optimum procedure may require the two identical but distinct subtables to be further developed in different ways.

In the example used for illustration the conditions were logically independent so that there was no associated information on logical dependence with the various subtables. However in general there may be such information and it is important that the process which recognises common subtables should also ensure that a common pair also have identical statements of logical relationship among their conditions. If the statements of logical relationship for the two tables are different then they cannot be regarded as a common subtable.

The testability of conditions

In the preceding sections of this paper and in all previously

MONTH-NO GE 1	N	—	—	Y
MONTH-NO LE 12	—	N	—	Y
DAY-NO GE 1	—	—	N	—
DAY-NO LE DAYS-IN (MONTH-NO)	—	—	—	N
REPORT ERROR NUMBER	1	2	3	4

Fig. 16

published work on decision tables it has been assumed implicitly that, when selecting a condition from a table for bifurcation, any condition may be selected and its truth or falsity established from the current transaction whether or not any of the other conditions in the table have been tested. In some circumstances, however, it may only be possible to test a condition if certain other conditions have already been tested and found to have particular values.

Consider the table shown in **Fig. 16** which appeared previously in King and Johnson (1973). In this table C_4 cannot be tested unless C_1 and C_2 are both true. If a program does not take this fact into account and C_4 is tested first then an erroneous transaction may cause an array bounds error.

In the following discussion we represent by tC_4 the 'testability of C_4'. Thus $\neg tC_4$ is stating that C_4 cannot be tested. With this formalisation we can deal with the concept of the testability of conditions within the general framework already established for handling logical dependencies among conditions.

In the example under discussion we have C_4 testable if and only if both C_1 and C_2 are true. Hence if either C_1 or C_2 is false then C_4 cannot be tested. Symbolically we have:

$$C_1 \wedge C_2 \rightarrow tC_4$$

and

$$tC_4 \rightarrow C_1 \wedge C_2 \ .$$

Using the symbol $\leftrightarrow$ to denote implication in both directions these two statements may be written

$$C_1 \wedge C_2 \leftrightarrow tC_4 \ .$$

We also have

$$\neg C_1 \vee \neg C_2 \leftrightarrow \neg tC_4$$

which is derivable from the previous statement by negating both sides. This form states that if either C_1 is false or C_2 is false then C_4 is not testable. In the table of Fig. 16 there is also a logical relationship between the first two conditions of the table. The initial table to which the algorithm is applied is, therefore, as shown in **Fig. 17.**

The algorithm for the selection of a condition for the bifurcation of a table must restrict its choice to those conditions for which there are no testability constraints. Thus, for the initial bifurcation of the table of Fig. 17, C_4 may not be selected. If C_2 is selected for initial bifurcation the first step produces the

C_1	N	(Y)	—	Y
C_2	(Y)	N	—	Y
C_3	—	—	N	—
C_4	—	—	—	N
	A_1	A_2	A_3	A_4

$C_1 \wedge C_2 \leftrightarrow tC_4$
$\neg C_1 \wedge \neg C_2 \rightarrow$

Fig. 17

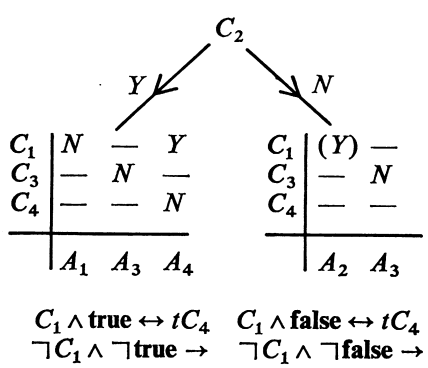


Fig. 18

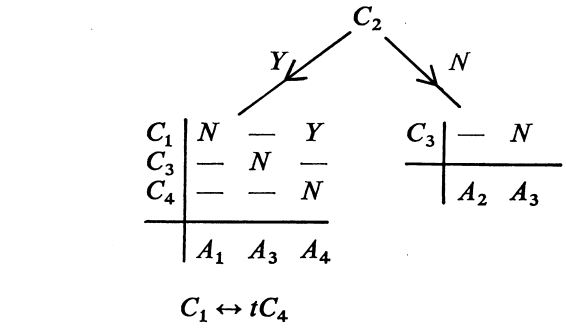


Fig. 19

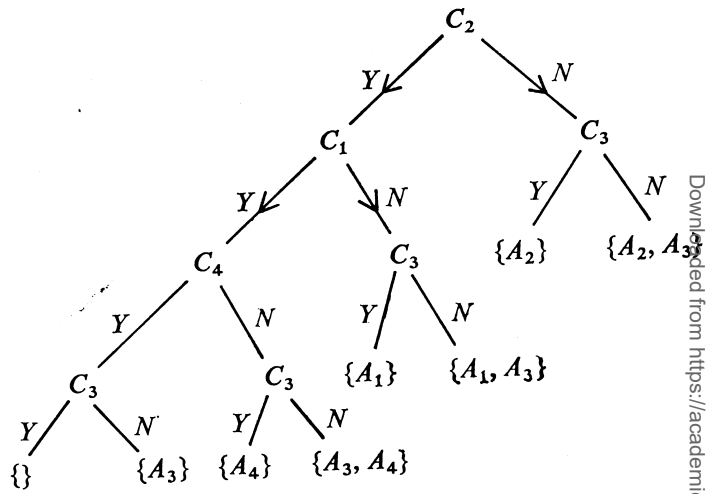


Fig. 20

situation as shown in **Fig. 18** where there has been a straight substitution of the values **true** and **false** for C_2 in the associated logical and testability constraints.

The first of the constraints associated with the Yes branch successor table simplifies to $C_1 \leftrightarrow tC_4$, and states that C_4 can be tested if and only if C_1 is true. C_4 cannot, therefore, be selected as the condition for the bifurcation of this table. The second relationship simplifies to **false** $\rightarrow$ which is always satisfied (see Appendix) and so does not constitute a constraint.

The first of the constraints on the No branch successor table simplifies to **false** $\leftrightarrow tC_4$ which states that C_4 may not be tested down this branch. It should therefore be verified that the row for this condition contains only dash entries and the condition deleted from the table. If any of the entries for an untestable condition are other than dash then there is a contradiction between the testability constraints and the entries in the table which indicates an error which requires to be reported. The second constraint down the No branch simplifies to $\neg C_1 \rightarrow$ which states that C_1 is always true down this branch and so the condition may be deleted from the table.

The result of these simplification and editing processes produces the situation shown in **Fig. 19.**

If the Yes branch table in Fig. 19 is now bifurcated on C_1 then C_4 will be found to be testable down the Yes branch from C_1, but untestable down the No branch. With the subsequent selection of conditions for bifurcation as indicated, the sequential testing procedure shown in **Fig. 20** is obtained.

The conversion of extended entry tables

An extended entry table can always be converted into an equivalent limited entry table. The resulting limited entry table can then be converted to a sequential testing procedure in the way described previously in this paper. This approach was discussed by Press (1965). An alternative method is to create a sequential testing procedure directly from the extended entry

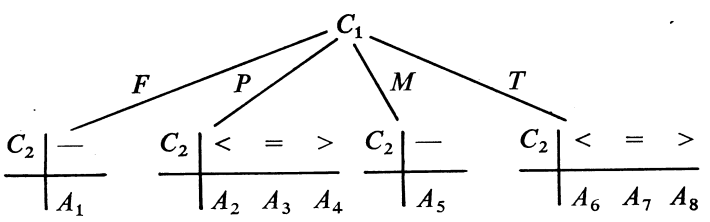
Type = Assets: Mkt. Value	Foods –	Property <	Property =	Property >	Metals –	Trusts <	Trusts =	Trusts >
	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8

Fig. 21

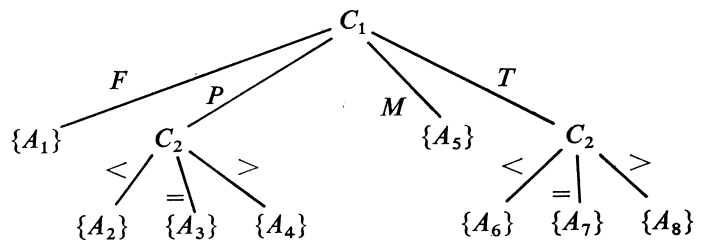
table. The initial form of the procedure is represented as a tree in which, as previously, a node corresponds to a condition, but now there will be as many paths leading from a node as there are distinct states for the condition.

Consider the extended entry table in **Fig. 21** where the entries for the first condition indicate four mutually exclusive states and those for the second condition three mutually exclusive states.

If the condition is selected for initial bifurcation we obtain the partially developed tree shown in **Fig. 22**(*a*) which, when the remaining subtables are developed, gives the sequential testing procedure shown in Fig. 22(*b*). From Fig. 22 we see that an algorithm, similar to that already given for limited entry tables, can be formalised to deal with extended entry tables.



(*a*)



(*b*)

Fig. 22

Extended entry conditions with mutually exclusive states

In the extended entry table given in Fig. 21 the second condition has three states which were mutually exclusive and exhaustive. The first condition is shown by the table to have four mutually exclusive states which, in the development of the sequential testing procedure shown in Fig. 22, have been assumed to be exhaustive. Thus the procedure would not handle a situation in which the value of type was other than one of the four values stated in the table which must, therefore, be precluded by the context in which the table appears. It is necessary that such information be supplied to the processor with the table since if values of type other than the four which appear explicitly can occur then there must also be an 'other values' path leading from C_1 in addition to the four shown. In the second condition, however, we see that the three mutually exclusive states are necessarily exhaustive and no other state is possible.

A basic assumption required by the process we are describing is that all possible states of the condition can be resolved into mutually exclusive sets of values or states. Suppose we have the first condition of Fig. 21 but without the restriction that the four values shown explicitly are the only values which can occur. In these circumstances a useful notational device is to use a special symbol, which we show here as @, to denote all of those states which are logically possible but do not appear as explicit entries elsewhere in the condition row. The use of @ to denote 'other possible states' of a condition should not be confused with the 'error' or 'else' rule which frequently appears in earlier published work on decision tables but which, in our view, is a rather unsatisfactory concept.

Type = Divs. Paid	Foods Y	Property —	Metals Y	Trusts Y	@ Y	– N
	A_1	A_2	A_3	A_4	A_5	A_6

Fig. 23

In the table shown in **Fig. 23** we illustrate the use of the @ symbol and distinguish it from the use of the ignore symbol If we consider a transaction for which the type has the value 'Mines' and for which dividends are paid then this transaction will satisfy the rule with action set A_5 since the type is not one of the values explicitly mentioned. A transaction with value 'Metals' and the divs. paid marker set specifies only A_3, it does not specify A_5. Note that the table is complete but ambiguous since a transaction can satisfy both the second and last rules. The first stage in the derivation of a sequential testing procedure from this table is illustrated in **Fig. 24,** the subsequent development being straightforward.

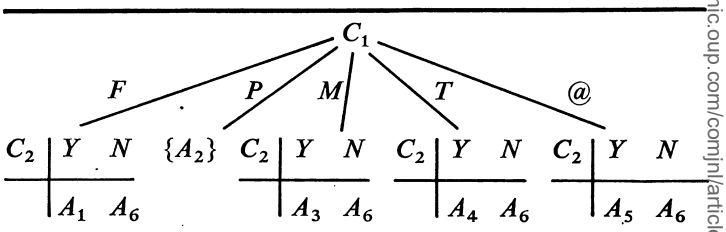


Fig. 24

In Fig. 24 it is seen that the value @ for the first condition is treated in exactly the same way as the other states of this condition. By its definition it becomes simply another of the mutually exclusive states.

Extended entry conditions with overlapping states

An illustration of a table with an extended entry condition with overlapping states is given by Fisher (1966), and reproduced here as **Fig. 25** with the action sets shown in abbreviated form. The context of the table ensures that CLASS NR can only be one of six distinct values (2, 3, 4, 5, 6, or 7) but that the states

TR-CODE EQ CLASS NR	2	3,4,5,6	3,6	3,6	4,5	7
RCD FIELD EQ ZEROS	—	*N*	*Y*	*Y*	*Y*	—
TR FIELD EQ RANGE	—	—	*Y*	*N*	—	—
	A_1	A_2	A_3	A_4	A_5	A_6

Fig. 25

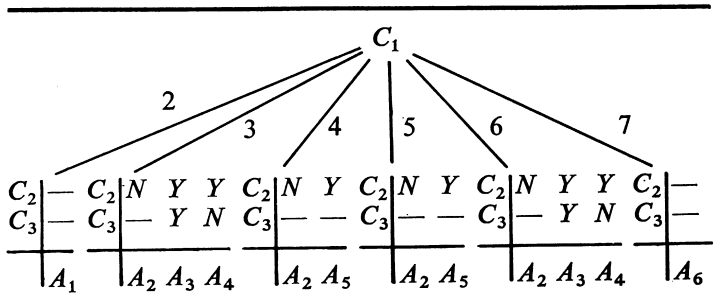


Fig. 26

which require to be recognised for satisfaction of four of the rules are combinations of these six distinct states.

With initial bifurcation on the first condition the partially developed sequential testing procedure shown in **Fig. 26** is obtained. We see that from the initial node there is a branch corresponding to each of the recognisable mutually exclusive states and that, in the table editing process to obtain the successor subtables, a rule is only deleted for a particular branch if the primitive state, which that branch represents, is not included among the primitive states which make up the state required for the rule to be satisfied. Thus we see that the second rule in the table of Fig. 25 remains in four of the subtables and is only deleted for the branches corresponding to the values 2 and 7. It is seen that the successor subtables for the 4 and 5 branches are identical and these two branches could be recognised as common and linked in the way previously described.

A particular type of condition which gives rise to overlapping

Age <	18	60	65	—
Age ⩾	—	18	60	65
	A_1	A_2	A_3	A_4

Fig. 27

states and is widely used in practice, involves the use of the numeric relational operators $>$, $\geqslant$, $<$, and $\leqslant$. These type of conditions often occur in pairs and we give a simple illustration of such a case in **Fig. 27.**

If we consider the first condition in isolation we can re-express it as illustrated in **Fig. 28** where we see that we now have four mutually exclusive states. The condition now has the same characteristics as the first condition of the table shown in Fig. 25 and can therefore be treated similarly in the derivation of sequential testing procedures.

state =	s_1	s_1, s_2	s_1, s_2, s_3	—

state	*value*
s_1	$A < 18$
s_2	$18 \leqslant A < 60$
s_3	$60 \leqslant A < 65$
s_4	$65 \leqslant A$

Fig. 28

A single condition involving a relational operator can always be treated in this manner. If we treat the second condition shown in **Fig. 27** similarly we see that the states already listed suffice and the table shown in **Fig. 29**(*a*) is obtained which can be readily simplified to the table in Fig. 29(*b*).

state =	s_1	s_1, s_2	s_1, s_2, s_3	—
state =	—	s_2, s_3, s_4	s_3, s_4	s_4
	A_1	A_2	A_3	A_4

(*a*)

state =	s_1	s_2	s_3	s_4
	A_1	A_2	A_3	A_4

(*b*)

Fig. 29

It would seem from the example discussed above that the present commonly used conventions for expressing conditions in decision tables involving relational operators could be considerably improved. Probably because of the frequently adopted approach of implementing decision table software in the form of a pre-processor to one of the commonly used language compilers the expressions permitted in the condition section of a table have been constrained by the logical and conditional expressions permitted in the programming languages concerned. A condition of the form $18 \leqslant \text{Age} < 60$ frequently arises in the real world problems under consideration and it seems a defect of the currently used programming languages that it is required to be rendered in a form such as $18 \leqslant$ Age AND Age < 60. We suggest here that this constraint should not continue to apply to decision table conventions and would recommend that conditions of the form $18 \leqslant \text{Age} < 60$ should be freely adopted in limited entry tables and a corresponding form of notation be developed for extended entry tables. We suggest a form the notation might take in **Fig. 30** where the table of Fig. 27 is re-expressed.

$\leqslant$ Age <	:18	18:60	60:65	65:
	A_1	A_2	A_3	A_4

Fig. 30

If such a notation is adopted the states expressed will frequently become mutually exclusive. Where, however, the notation is used to specify overlapping intervals then decision table software can readily be designed to resolve the intervals expressed into a set of non-overlapping states and the table treated in the way illustrated for the table of Fig. 25.

Logical relations among extended entry conditions

Logical relations among extended entry conditions can, in general, be treated in a similar way to those for limited entry conditions. We assume that the states for a particular extended entry condition have been resolved into a primitive set of mutually exclusive states and that the logical relations are expressed in terms of these primitive states. A simple illustration is given in **Fig. 31** where, by virtue of the logical relations expressed between the conditions, the table is seen to be complete.

TYPE =	*A*	*A*	*B*	*B*	*C*
< WT $\leqslant$	:5	5:10	5:10	10:	(10:)
	A_1	A_2	A_3	A_4	A_5

(TYPE = A) IMPLIES (WT $\leqslant$ 5) OR (5 < WT $\leqslant$ 10)
(TYPE = B) IMPLIES (5 < WT $\leqslant$ 10) OR (10 < WT)
(TYPE = C) IMPLIES (10 < WT) .

Fig. 31

Logical relations expressed in the form of an implication can have the right hand side expressed in either a positive way as shown in Fig. 31 or alternatively in a negative sense. Thus the first of the conditions could alternatively be expressed in the form:

(TYPE = A) IMPLIES NOT (10 < WT) .

In the fifth rule of Fig. 31 we see that the entry for the second condition is bracketed to show that the entry is implied by, in this case, the other entry for this rule.

Conclusions

Hitherto methods for the derivation of sequential testing procedures from decision tables have depended upon the tables satisfying completeness and non-ambiguity constraints. These methods could not be used for tables conforming to a multi-rule convention and with no requirement that they should be complete. Harrison (1973) has suggested that more general methods than those presently available present difficulties. In this paper a method has been described in detail which resolves such difficulties and can be readily implemented.

Suggestions have also been made for improving the concepts and notations for extended entry tables. Using this approach to extended entry tables the methods for decision table processor construction described for limited entry tables can be readily

developed to deal also with extended entry tables and the general lines of this development have been indicated. It is thought that the methods described provide the basic framework within which the various optimising methods described in the literature can be further improved and developed.

Appendix

A note on the use of propositional calculus notation for expressing the relationships among conditions

The truth or falsity of the implication $p \rightarrow q$ (to be read as 'p implies q') is given by the truth table:

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

When an implication is associated with a decision table it is being implicitly stated that the implication is necessarily always true. The first, third, and fourth lines only of the above truth table are, therefore, relevant to our discussion. In general p and q will be statements of values which one or more of the conditions can take, e.g. $C_1 \wedge \neg C_2$ or equivalently, in a notation more suitable for use in a Cobol context, (C_1, Y) AND (C_2, N).

We see from the truth table that, if the left hand side of the implication

$$p \rightarrow q$$

is satisfied then so necessarily is the right hand side. If the left hand side is not satisfied then we see that we have no information as to whether the right hand side will be satisfied or not. Merely as a matter of notation it has been found clearer in a Cobol context to use the word IMPLIES for the symbol $\rightarrow$ (or any of the other variants of this symbol found in the literature). If we wish to state within this framework that some combination of values can never occur then it can be seen from the fourth line of the above truth table that it can be stated in the form

$$p \rightarrow \textbf{false} \ .$$

As a matter of notation the value **false** has not, in general, been shown explicitly to the right of the implies sign. In a Cobol context it has been found useful to use the notation IMPOSS (to be read 'impossible') to stand for $\rightarrow$ **false.**

The construction and programming of algorithms for the manipulation of expressions in the propositional calculus is a well explored and developed field. Based on this work, therefore, suitable program modules can be readily constructed for incorporation in decision table software to carry out the various manipulations of the logical expressions required by the approach to decision table translation described in this paper.

References

BARNARD, T. J. (1969). A new rule mask technique for interpreting decision tables, *The Computer Bulletin*, Vol. 13, pp. 153-154.

CALLAHAN, M. D., and CHAPMAN, A. E. (1967). Description of basic algorithm in DETAB/65 preprocessor, *CACM*, Vol. 10, pp. 441-446.

DIAL, R. B. (1970). Algorithm 394, decision table translation, *CACM*, Vol. 13, pp. 571-572.

FILETAB USER MANUAL (1972). The National Computer Centre Ltd., Manchester, UK.

FISHER, D. L. (1966). Data, documentation and decision tables, *CACM*, Vol. 9, pp. 26-31.

GANAPATHY, S., and RAJARAMAN, V. (1973). Information theory applied to the conversion of decision tables to computer programs, *CACM*, Vol. 16, pp. 532-539.

HARRISON, W. J. (1973). CR 26028: Review of King and Johnson (1973), *Comp. Revs*, Vol. 14, p. 541.

HUMBY, E. (1973). *Programs from decision tables*, *Computer Monograph* 19, Macdonald, London and American Elsevier, New York, p. 91.

INGLIS, J., and KING, P. J. H. (1968). Flowcharts and Decision Tables, *The Computer Journal*, Vol. 11, pp. 117-118.

JOHNSON, R. G. (1974). *Logical Relations between Conditions in Decision Tables*, Ph.D. thesis, University of London.

KING, P. J. H. (1967). Decision tables, *The Computer Journal*, Vol. 10, pp. 135-142.

KING, P. J. H. (1969). The interpretation of limited entry decision table format and relationships among conditions, *The Computer Journal*, Vol. 12, pp. 320-326.

KING, P. J. H., and JOHNSON, R. G. (1973). Some comments on the use of ambiguous decision tables and their conversion to computer programs, *CACM*, Vol. 16, pp. 287-290.

KING, P. J. H., and JOHNSON, R. G. (1974). Comments on the algorithm of Verhelst for the conversion of limited entry decision tables to flowcharts, *CACM*, Vol. 17, pp. 43-45.

KIRK, H. W. (1965). Use of decision tables in computer programming, *CACM*, Vol. 8, pp. 41-43.

POLLACK, S. L., HICKS, H. T., and HARRISON, W. J. (1971). *Decision Tables*: *Theory and Practice*, John Wiley and Sons, Inc., New York and London.

POLLACK, S. L. (1965). Conversion of limited entry decision tables to computer programs, *CACM*, Vol. 8, pp. 677-682.

POLLACK, S. L. (1967). CR 12948: Review of King (1967), *Comp. Revs*, Vol. 8, pp. 501-502.

PRESS, L. I. (1965). Conversion of decision tables to computer programs, *CACM*, Vol. 8, pp. 385-390.

REINWALD, L. T., and SOLAND, R. M. (1966). Conversion of limited entry decision tables to optimal computer programs I: minimum average processing time, *JACM*, Vol. 13, pp. 339-358.

REINWALD, L. T., and SOLAND, R. M. (1967). Conversion of limited entry decision tables to optimal computer programs II: minimum storage requirement, *JACM*, Vol. 14, pp. 742-756.

SHWAYDER, K. (1971). Conversion of limited entry decision tables to computer programs—a proposed modification to Pollack's algorithm, *CACM*, Vol. 14, pp. 69-73.

SPRAGUE, V. G. (1966). On storage space of decision tables, *CACM*, Vol. 9, pp. 319-320.

VERHELST, M. (1972). The conversion of limited entry decision tables to optimal and near optimal flowcharts: two new algorithms, *CACM*, Vol. 15, pp. 974-980.