The interpretation of limited entry decision table format and relationships among conditions

By P. J. H. King*

Recent published material suggests that basic decision table format and the dependence/independence of conditions in a table require clarification. These are separate matters but tend to be confused. This paper emphasises the interpretation of basic format. Relationships among conditions are discussed and it is shown that this is a separate matter from basic format. Some formal definitions are proposed.

(Received March 1969)

In King (1967) the decision table shown in **Fig. 1**

C_1	Y	Y	—	N
C_2	Y	—	N	—
C_3	N	Y	N	—
GO TO	R_1	R_2	R_3	R_4

Fig. 1

was given together with several possible sequential testing implementations in flowchart form, one of which is reproduced here as **Fig. 2.**

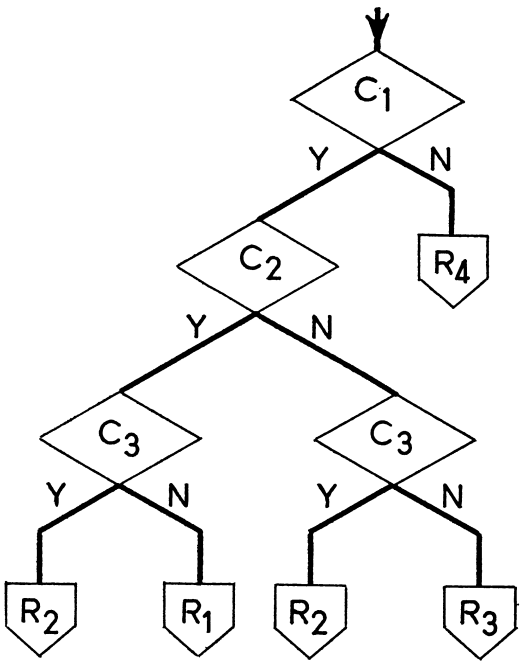

Fig. 2

This flowchart is described as a 'necessarily wrong' conversion by Pollack (1967). He states that the conversion implies that a '—' is a 'Y' with the suggestion that this is somehow wrong. It appears that Pollack regards '—' as a legitimate condition entry only if used in the sense of 'either Y or N'. This interpretation is not implied by limited entry table format but requires in addition the assumption of the independence of conditions in the stub.

A similarly confused interpretation of dash as a condition entry is given by Chapin (1967) who states:

> 'and blank or hyphen indicates that the condition to be met is both "Yes" and "No" and hence is either ignored, or not considered, or irrelevant, depending upon the context'.

This seems to accord with Pollack's view that '—' means 'either Y or N'. If this view is correct then the table in **Fig. 3** cannot be regarded as a proper decision table since the dash in R_1 never actually can 'mean Yes'. This is not a satisfactory point of view.

	R_1	R_2	R_3
Age $<$ 18	Y	N	N
Age $>$ 65	—	Y	N
	A_1	A_2	A_3

Fig. 3

Before discussing basic format and dash as a condition entry, it should be made clear that this writer regards the flowchart of Fig. 2 as a perfectly satisfactory conversion of the decision table of Fig. 1 and Pollack's comment that it is 'necessarily wrong' as mistaken. We note that in its original context (King, 1967) it was observed that the table is ambiguous (i.e. fails to specify a unique action set) should all three conditions give a 'No' outcome on testing and hence there must be relationships among the conditions which make this impossible. If this is not the case then the table has not been properly checked out and conversion to program cannot yet take place. The view taken is that when *implementing* a table it should be assumed that it correctly specifies the problem solution, and any apparent ambiguity will be resolved automatically by relations among the conditions. Implementation is regarded as quite a separate matter from checking for correctness.

Two sets of conditions for which the table of Fig. 1 is satisfactory are shown in **Fig. 4.** For either of these it can be seen that the flowchart in Fig. 2 is a satisfactory implementation.

$x < 5$	Y	Y	—	N
$x > 2$	Y	—	N	—
$y > 0$	N	Y	N	—
	R_1	R_2	R_3	R_4

(*a*)

Age $<$ 60	Y	Y	—	N
Age $>$ 20	Y	—	N	—
Sex = 'M'	N	Y	N	—
	G_1	G_2	G_3	G_4

(*b*)

Fig. 4

* *Computer Unit, University College of Wales, Aberystwyth, Cards.*

The table of Fig. 4 (*a*) divides the points of the cartesian plane into the four mutually exclusive and exhaustive regions shown in **Fig. 5.** The table determines uniquely for any point (x, y) the region to which it belongs.

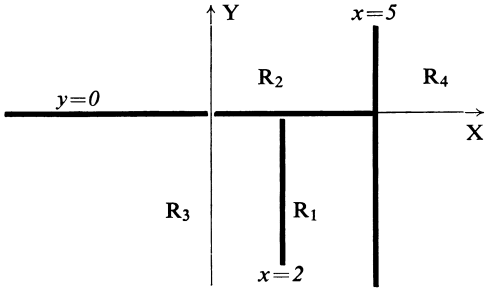

Fig. 5

The table of Fig. 4 (*b*) divides a set of personnel records into four groups with each record being in one and only one group. These are: females between 20 and 60 (G_1), males aged less than 60 (G_2), females aged 20 or under (G_3) and personnel aged 60 or over (G_4).

These two examples show that it is inappropriate for Pollack (1967) to describe Fig. 2 as a 'necessarily wrong' conversion of Fig. 1. The so called 'correct' version given by him and reproduced here as **Fig. 6** is simply not relevant. There is no question of unresolved ambiguity in the examples of Fig. 4, in both of which a 'No' outcome to the first condition decides the case finally and further testing is pointless. The implication of Fig. 6 seems to be that table error checking may be carried out as a by-product of conversion to program. This philosophy has been adopted by many writers of ALGOL, FORTRAN and other compilers but seems unlikely to prove satisfactory for table processors.

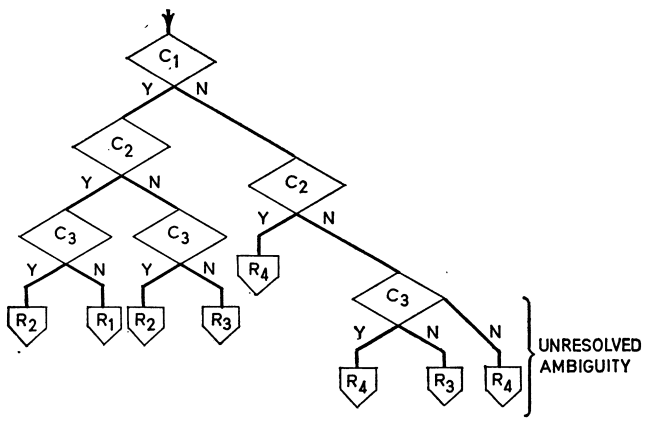

Fig. 6

Basic format for limited entry tables

The basic structure for limited entry tables has been given frequently in the literature, e.g. see Pollack (1963). The format for the condition section of the table is shown in detail in **Fig. 7.** $C_1, C_2, \ldots, C_m$ (the 'conditions'), denote logical propositions about some set of prime or basic variables in the problem.

The **if, and, and, . . ., then** to the left of the table and the rule references $R_1, R_2, \ldots, R_n$ are implicit in the format and are normally not shown explicitly. The action sets

		R_1	R_2	R_3	.....	R_n
if	C_1	e	e	e	.....	e
and	C_2	e	e	e	.....	e
and	C_3	e	e	e	.....	e
.	.	.	.	.		.
.	.	.	.	.		.
.	.	.	.	.		.
.	.	.	.	.		.
and	C_m	e	e	e	.....	e
then		—	—	—		—
		A_1	A_2	A_3	.....	A_n

Fig. 7

corresponding to each rule are shown briefly as $A_1, A_2, \ldots, A_n$. Note that these are not all necessarily distinct. Each 'e' denotes a limited condition entry which may be either 'Y' (yes), 'N' (no) or '—' (dash). Leaving an entry blank is frequently used as an alternative convention to entering a dash.

A 'rule' is specified by a single vertical column to the right of the first vertical line. A rule 'holds' if, when the table is activated in respect of some data, the relevant conditions are found to be satisfied or not satisfied as specified by the condition entries. (Some readers may prefer to regard the condition part of a table as an 'action set selection function' which is 'applied' to data.) The interpretation of the condition entries is:

(*a*) A dash specifies the corresponding condition to be *not relevant* for the rule.

(*b*) A 'Y' or 'N' specifies the corresponding condition *to be relevant* for the rule. This means it must be tested to determine whether the rule holds. Moreover, if the entry is 'Y' then the condition must be found to be true for the rule to hold and if 'N' it must be found to be false.

We take the table of Fig. 4(*b*) as a specific example. The interpretation of the four rules is:

(1) **if** Age $<$ 60 is true **and** Age $>$ 20 is true **and** Sex = 'M' is not true **then** G_1
(2) **if** Age $<$ 60 is true **and** Sex = 'M' is true **then** G_2
(3) **if** Age $>$ 20 is not true **and** Sex = 'M' is not true **then** G_3
(4) **if** Age $<$ 60 is not true **then** G_4

The interpretation of the condition section of a table for a particular rule derives directly from the basic format described above. The interpretation of the dash entry is that the condition is ignored for the particular rule. Thus in the foregoing, the second condition (Age $>$ 20) is ignored in the process to decide whether the second rule holds. Dash thus 'means' either that the condition should not be tested for the rule or, if already tested, the result obtained is not taken into account. In Fig. 3, therefore, the dash in R_1 means that the second condition is not relevant in deciding whether to take action A_1. The decision on whether to take this action depends only on the first condition. If R_1 does hold and the second condition is tested then it will necessarily be found to be false but this is irrelevant.

We discuss five alternative conventions on the overall meaning of a table. These are that the entries in the table must be such that:

(i) one and only one rule will hold. In this case one action set will be specified on each activation of the table,

(ii) either one rule holds or no rules hold. In this case tables are an extension of the concept of the simple **if** statements of ALGOL and COBOL where no action results if the condition is found to be false,

(iii) one and only one action set is specified. In this case if more than one rule holds then these rules must specify the same action set,

(iv) either one action set is specified as in (iii) or no action set is specified,

(v) any number of rules may hold and more than one action set may be specified, namely those corresponding to the rules that hold.

The first four conventions are similar in that they each specify that not more than one action set is performed on each occasion the table is applied to data. In terms of implementation this implies a search for a rule which holds and the performance of the corresponding action set. Once a rule which holds is found no further rules are tested. This is the approach adopted in the implementation of a recent decision table extension to ALGOL by Bjork (1968). The last convention, although not generally used, requires that every rule be tested and the action set corresponding to every rule that holds performed. This convention has been used by Barnard (1969). Sprague (1966) suggests that conventions (iii) and (iv), where the emphasis is on action set selection, are preferable to (i) and (ii) and convention (iii) was assumed by King (1967, 1968). We suggest that either this or perhaps (iv) will prove more satisfactory than (v) when implementation is to be on a computer with a single central processor and systems analysis and design are oriented to this end. It seems likely that the last convention will prove most useful for specifying systems and programs for multi-processor configurations.

It is worth noting that limited entry tables can be implemented simply in languages with the facilities of ALGOL by direct interpretation of the basic format. We illustrate this implementation and the variations corresponding to the semantic conventions (i), . . ., (v) using Fig. 4(*b*). The G_i are now regarded as action sets. An implementation suitable for any of the conventions, (i), (ii), (iii) or (iv) is:

begin boolean C_1, C_2, C_3;
  $C_1 := Age < 60$; $C_2 := Age > 20$; $C_3 := Sex = type$ ('*M*');
  if $C_1 \wedge C_2 \wedge \neg C_3$ **then** G_1 **else**
  if $C_1 \wedge C_3$ **then** G_2 **else**
  if $\neg C_2 \wedge \neg C_3$ **then** G_3 **else**
  if $\neg C_1$ **then** G_4
end

In the case of conventions (i) and (iii) it may be desirable to add after G_4 '**else** *ERROR*'. It is clear that the rules can be tested in any order. The implementation for the fifth convention is:

begin boolean C_1, C_2, C_3;
  $C_1 := Age < 60$; $C_2 := Age > 20$; $C_3 := Sex = type$ ('*M*');
  if $C_1 \wedge C_2 \wedge \neg C_3$ **then** G_1;
  if $C_1 \wedge C_3$ **then** G_2;
  if $\neg C_2 \wedge \neg C_3$ **then** G_3;
  if $\neg C_1$ **then** G_4
end

Whilst the second implementation would suffice for conventions (i) and (ii) considerations of efficiency suggest that testing should cease on finding a rule which holds. Note that this implementation would not be satisfactory for conventions (iii) and (iv) as it may lead to an action set being performed more than once.

Proposals have been made by Wirth (1966) for indicating that statements in ALGOL may be carried out in parallel. His proposal is to introduce a new separator, **and**, to be used instead of ; to mean that the statements either side can be carried out in parallel as opposed to the sequential interpretation of ;. The symbol **with** is used here to represent this separator in preference to **and** as proposed by Wirth to avoid confusion with the earlier use of **and** in this paper to represent $\wedge$. We suggest, and assume here, that the relationship proposed should be transitive even though this is a matter which seems to require further discussion. (It is easy to give examples where S_1 may be carried out in parallel with S_2 and S_2 with S_3 but parallelism between S_1 and S_3 seems likely to lead to difficulty, e.g.

$a := a + b$ **with** $x := y$ **with** $a := a + c$.)

Using this notation for parallelism then the implementation for convention (v) can be written:

begin boolean C_1, C_2, C_3;
  $C_1 := Age < 60$ **with** $C_2 := Age > 20$ **with** $C_3 := Sex = type$ ('*M*');
  if $C_1 \wedge C_2 \wedge \neg C_3$ **then** G_1 **with**
  if $C_1 \wedge C_3$ **then** G_2 **with**
  if $\neg C_2 \wedge \neg C_3$ **then** G_3 **with**
  if $\neg C_1$ **then** G_4
end

This would also be suitable for conventions (i) and (ii) but not for (iii) and (iv). The value of the parallelism naturally depends upon the extent and complexity of the actions denoted by G_i and of the logical propositions in the boolean assignments.

Table checking and implementation

The view taken in this paper is that a clear distinction should be made between checking that a table is a satisfactory specification of requirements and its implementation. We suggest that these two aspects should be treated quite separately and dealt with independently. Checking should be completed satisfactorily prior to implementation so that during conversion to program it is assumed the table is a correct specification of the problem solution. It is not satisfactory to regard checking as a by-product of this conversion.

Methods of checking clearly depend upon the convention adopted for overall table meaning. The discussion in the remainder of this paper relates to the third and fourth conventions of the previous section. It has been argued previously (King, 1968) than when checking a table adhering to one or other of these conventions it is necessary to ensure that it is unambiguous. For the third convention it is also necessary to ensure that the table is complete in the sense that an action is specified for every possible outcome.

It is important to realise that a decision table should not be regarded as a statement of relationships which hold in particular circumstances. When constructing the

condition section of a table, sufficient entries must be made to resolve adequately the decision situation. Entries additional to these will lead to inefficient and cumbersome program. Thus in Fig. 4(*b*) the entry for the first condition in the third rule would be 'Yes' if our aim were to make statements about relationships which hold, since not being older than 20 implies being under 60. Such an entry is undesirable, however, since the group to which a person belongs is uniquely determined by the table as it stands. Referring to the interpretation of basic format we see that such an entry in the third rule would imply that the first condition must be tested to determine whether this rule holds. In fact, testing this condition is not relevant in deciding on this rule.

To prevent unnecessary testing of conditions it is desirable to maximise the number of dashes in the condition section of a table consistent with fully specifying actions to be taken. It is important to realise that a 'Y' or 'N' entry is stating that a condition is *relevant* to a rule and therefore *must* be tested to determine whether it holds.

A useful way of indicating the outcome of conditions which are not relevant in deciding whether a rule holds but for which, when it does hold, the outcomes are known, is to give these outcomes in brackets. These bracketed entries have the same status as comment statements in programming languages, i.e. their purpose is to improve readability and they are ignored by processors. Using this convention the table of Fig. 4(*b*) can be written as shown in **Fig. 8** in which it is noted that personnel in G_3 are aged less than 60 and those in G_4 are aged more than 20.

Age $<$ 60	Y	Y	(Y)	N
Age $>$ 20	Y	—	N	(Y)
Sex = 'M'	N	Y	N	—
	G_1	G_2	G_3	G_4

Fig. 8

The bracketed entries are implied by other unbracketed entries. We describe a table which only contains the entries required to determine the rules as an *algorithmic table* and one that includes all implied entries as a *descriptive table*. The notation suggested above enables these two forms to be combined.

Relationships among conditions of a table

In the introductory discussion of the tables of Fig. 4 the argument depended on the relationship which exists between the first and second conditions. A simultaneous 'No' outcome to these two conditions is impossible and hence the third and fourth rules are never simultaneously satisfied and the table always specifies a unique action set. In this section we attempt a formulation of relationships between the conditions of a table in terms of the first order predicate calculus (e.g. Stoll, 1963) and suggest definitions for 'dependence' and 'independence' of conditions. A previous definition of independence of conditions was given by Pollack (1963, pp. 33–34). The reasoning behind Pollack's definition is difficult to understand. Under it the conditions in the table in **Fig. 9**(*a*) would be 'independent', whereas those of **Fig. 9**(*b*) would be 'dependent'.

a $<$ 18	Y	—	N
a $>$ 60	—	Y	N
	A_1	A_2	A_3

(*a*)

a $\geqslant$ 18	N	—	Y
a $\leqslant$ 60	—	N	Y
	A_1	A_2	A_3

(*b*)

Fig. 9

As these two tables are, for all practical purposes, identical, it seems that any definition of independence of conditions which regards them differently cannot be satisfactory.

Denoting the conditions of Fig. 9(*a*) by $C_1(a)$ and $C_2(a)$ respectively we can express the relationship between them by the tautology (1).

$$\models (\forall a)(C_1(a) \rightarrow \neg C_2(a)) \quad (1)$$

$$\models (\forall a)(C_2(a) \rightarrow \neg C_1(a)) \quad (2)$$

$$\models \neg(\exists a)(C_1(a) \wedge C_2(a)) \quad (3)$$

We note that (2) can be obtained from (1) by replacing the sub-expression to which the quantifier applies by its contrapositive and from either (1) or (2) we can derive (3) by negation. (1), (2) and (3) are thus different ways of expressing the same relationship.

This information can be included in the decision table notation as shown in **Fig. 10** where it has been added to the algorithmic specification given in Fig. 9(*a*).

	R_1	R_2	R_3	R_4
C_1 (a)	Y	(N)	N	(Y)
C_2 (a)	(N)	Y	N	(Y)
	A_1	A_2	A_3	P*

Fig. 10

In R_1 and R_2 the entries implied by (1) and (2) respectively are shown using the notation already introduced. R_4 has been added to the table with the notation P* in the action section denoting the rule as *a priori* excludable. That is, the indicated combination of condition values will never occur and the rule will not appear in the algorithmic form of the table. For consistency of notation the entries in *a priori* excludable rules are also bracketed. We note that the table of Fig. 10 is complete in the sense that it covers all possible distinct rule entries for two conditions. Note that the algorithmic version of the table is obtained by first omitting all bracketed entries and then rules with no entries.

We describe the table of Fig. 9(*a*) as being '*algorithmically complete*' since it can be rendered complete in the sense of covering all possible distinct rule entries by the addition of implied and *a priori* excludable information. An algorithmically complete table is one that satisfies the third of the conventions on overall meaning given above.

A similar type of analysis to determine completeness is now given for a more complex table.

$w_1 > R$	Y	Y	N	N	—	—	—	—	—
$w_1 > 0$	—	—	Y	Y	—	—	N	N	N
$w_1 + w_2 + w_3 > 0$	—	—	—	—	N	N	Y	Y	Y
$A > 3 \times R$	—	—	Y	N	—	—	N	Y	—
$A > 6 \times R$	N	Y	N	—	N	Y	—	N	Y
	a_1	a_2	a_3	a_1	a_4	a_5	a_1	a_6	a_5

Fig. 11

The table shown in **Fig. 11** appeared in King (1968) in less formal notation. The variables w_1, w_2, w_3, R and A which appear in the condition stub are always positive. The action sets are abbreviated as a_1, a_2, . . ., a_6.

Denoting the five conditions in Fig. 11 by $C_1(w_1, R)$ $C_2(w_1)$, $C_3(w_1, w_2, w_3)$, $C_4(A, R)$ and $C_5(A, R)$ respectively we note that there are four relationships among the conditions which can be expressed by the tautologies (4), (5), (6) and (7) in which, for convenience, the condition arguments are not shown.

$$\models (\forall w_1)(\forall R)(C_1 \rightarrow C_2) \quad (4)$$

$$\models (\forall w_1)(\forall w_2)(\forall w_3)(\forall R)(C_1 \rightarrow C_3) \quad (5)$$

$$\models (\forall w_1)(\forall w_2)(\forall w_3)(C_2 \rightarrow C_3) \quad (6)$$

$$\models (\forall A)(\forall R)(\neg C_4 \rightarrow \neg C_5) \quad (7)$$

Four further tautologies can be obtained from these by replacing the *inner* expressions by their contrapositives giving (4a), . . ., (7a).

$$\models (\forall w_1)(\forall R)(\neg C_2 \rightarrow \neg C_1) \quad (4a)$$

$$\models (\forall w_1)(\forall w_2)(\forall w_3)(\forall R)(\neg C_3 \rightarrow \neg C_1) \quad (5a)$$

$$\models (\forall w_1)(\forall w_2)(\forall w_3)(\neg C_3 \rightarrow \neg C_2) \quad (6a)$$

$$\models (\forall A)(\forall R)(C_5 \rightarrow C_4) \quad (7a)$$

The *a priori* excludable outcomes are given by (4b), . . ., (7b) which can be obtained from either (4), . . ., (7) or (4a), . . ., (7a) by negation:

$$\models \neg(\exists w_1)(\exists R)(C_1 \wedge \neg C_2) \quad (4b)$$

$$\models \neg(\exists w_1)(\exists w_2)(\exists w_3)(\exists R)(C_1 \wedge \neg C_3) \quad (5b)$$

$$\models \neg(\exists w_1)(\exists w_2)(\exists w_3)(C_2 \wedge \neg C_3) \quad (6b)$$

$$= \neg(\exists A)(\exists R)(C_5 \wedge \neg C_4) \quad (7b)$$

Fig. 12 shows the algorithmic table of Fig. 11 rewritten to include all entries implied by the tautologies (4), . . ., (7a) and the *a priori* excludable rules specified by (4b), . . ., (7b). The references to the tautologies justifying particular entries are shown below the actions in the same order as the implied entries. Thus (4) is equivalent to the (Y) entry against C_2 in R_1, (5) is equivalent to the (Y) entry against C_3 in R_1, (4b) justifies the *a priori* excludable R_{10}, etc.

We can now assert that the table of Fig. 11 is *algorithmically complete* if the table of Fig. 12 is complete in the sense that all of the 32 (i.e. 2^5) distinct rule entries composed only of Y's and N's (i.e. not —'s) are either explicitly included or implied. If the table is such that for any two rules we can find at least one condition for which one of the rules has a Y entry and the other a N entry then the completeness or otherwise of the table can be determined by a simple counting process. This is immediately apparent when we note that a rule with p dash entries can be satisfied by 2^p distinct outcomes if it is assumed that all Y, N combinations can actually occur, i.e. that the conditions are in some sense 'logically independent'.

We see from Fig. 12 that every rule pair has at least one condition for which one of the rules has a Y entry and the other a N entry except for the rule pairs $\{R_{10}, R_{11}\}$, $\{R_{10}, R_{13}\}$, $\{R_{11}, R_{12}\}$, $\{R_{11}, R_{13}\}$, $\{R_{12}, R_{13}\}$. We note that all of the rules involved have the pseudo action set P*. These pairings indicate that particular Y, N combinations are covered by more than one rule. This multiple inclusion of particular Y, N combinations is eliminated if rules R_{10}, . . ., R_{13} are replaced by the rules R_{10}, . . ., R_{15} of **Fig. 13.** These rules cover all the Y, N combinations that were covered by R_{10}, . . ., R_{13} but each combination is included only once.

In Fig. 13 the number of distinct rule entries composed only of Y's and N's explicitly included or implied by each rule is given below the action-sets. We see that these total only 31 and the table is incomplete. The table of Fig. 11 is not, therefore, algorithmically complete and there is one logically possible outcome which

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}
C_1 (w_1, R)	Y	Y	N	N	(N)	(N)	(N)	(N)	(N)	(Y)	(Y)	—	—
C_2 (w_1)	(Y)	(Y)	Y	Y	(N)	(N)	N	N	N	(N)	—	(Y)	—
C_3 (w_1, w_2, w_3)	(Y)	(Y)	(Y)	(Y)	N	N	Y	Y	Y	—	(N)	(N)	—
C_4 (A, R)	—	(Y)	Y	N	—	(Y)	N	Y	(Y)	—	—	—	(N)
C_5 (A, R)	N	Y	N	(N)	N	Y	(N)	N	Y	—	—	—	(Y)
Tautologies which justify bracketed entries	a_1 4 5	a_2 4 5 7a	a_3 6	a_1 6 7	a_4 5a 6a	a_5 5a 6a 7a	a_1 4a 7	a_6 4a	a_5 4a 7a	P* 4b	P* 5b	P* 6b	P* 7b

Fig. 12

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}
C_1 (w_1, R)	Y	Y	N	N	(N)	(N)	(N)	(N)	(N)	(Y)	(Y)	(N)	(Y)	(N)	(N)
C_2 (w_1)	(Y)	(Y)	Y	Y	(N)	(N)	N	N	N	(N)	(Y)	(Y)	(Y)	(N)	(Y)
C_3 (w_1, w_2, w_3)	(Y)	(Y)	(Y)	(Y)	N	N	Y	Y	Y	—	(N)	(N)	(Y)	—	(Y)
C_4 (A, R)	—	(Y)	Y	N	—	(Y)	N	Y	(Y)	—	—	—	(N)	(N)	(N)
C_5 (A, R)	N	Y	N	(N)	N	Y	(N)	N	Y	—	—	—	(Y)	(Y)	(Y)
No. of distinct rule entries	a_1 2	a_2 1	a_3 1	a_1 1	a_4 2	a_5 1	a_1 1	a_6 1	a_5 1	P* 8	P* 4	P* 4	P* 1	P* 2	P* 1

Fig. 13

it does not cover. We note that this is the outcome shown in **Fig. 14** which must be added to the rules of Fig. 11 (with dashes replacing the (Y)'s) and an action set specified if we require the table to be algorithmically complete, i.e. to adhere to the third of the conventions discussed previously.

C_1 (w_1, R)	N
C_2 (w_1)	Y
C_3 (w_1, w_2, w_3)	(Y)
C_4 (A, R)	(Y)
C_5 (A, R)	Y

Fig. 14

The foregoing discussion has illustrated by example in an informal way how relationships among the conditions of a table may be taken into account in table checking. There has been no attempt to enumerate formal rules for carrying out such checking manually since it is the author's view that tables with five or more conditions should normally be checked out by computer and that this is desirable even with only three or four conditions. The manual checking of tables is tedious and error prone and suitable algorithms are available upon which satisfactory checking programs can be based. As some of these involve the enumeration of 'Y, N' combinations and pairwise comparisons of rules they are not suitable for manual use. The determination of rules $R_{10}, \ldots, R_{15}$ of Fig. 13, as a non-redundant equivalent of rules $R_{10}, \ldots, R_{13}$ of Fig. 12, is best left to computer program, as is the detection of the rule shown in Fig. 14 as the one not covered by Fig. 13 and hence omitted from Fig. 11 and the tautologies (4), (5), . . ., (7b).

We now attempt formal definitions of '*logical dependence*' and '*logical independence*' for a pair of conditions. We denote by x the prime or basic variables in the problem about which $C_a(x)$ and $C_b(x)$ are logical propositions, i.e. entries in the condition stub of a limited entry decision table. We say that $C_a(x)$ and $C_b(x)$ are *logically independent* conditions if *all* of the four statements (8), (9), (10) and (11) are true. If one or more is false then we say the conditions are *logically dependent*.

$$(\exists x)(C_a(x) \wedge C_b(x)) \quad (8)$$

$$(\exists x)(C_a(x) \wedge \neg C_b(x)) \quad (9)$$

$$(\exists x)(\neg C_a(x) \wedge C_b(x)) \quad (10)$$

$$(\exists x)(\neg C_a(x) \wedge \neg C_b(x)) \quad (11)$$

We have seen in the previous discussion that if a proposition of the form of (8), . . ., (11) is false then it gives rise to a group of *a priori* excludable 'Y, N' combinations. Moreover, if false, it is also equivalent to two tautologies each of which gives rise to an implied entry in one or more rules, e.g. if (8) is false then we have (12) and (13)

$$\models (\forall x)(C_a(x) \rightarrow \neg C_b(x)) \quad (12)$$

$$\models \forall x(C_b(x) \rightarrow \neg C_a(x)) \quad (13)$$

We note that if two of the statements (8), . . ., (11) are false, then either one or other of $C_a(x)$ and $C_b(x)$ is a logical constant or they are propositions which either always have the same value or always have opposite values. If three of the statements (8), . . ., (11) are false then both propositions are logical constants. The formulation is such that at least one of the statements (8), . . ., (11) must always be true. If the conditions in a table have been properly selected, therefore, only one of the statements (8), . . ., (11) will be false when a pair of conditions is logically dependent.

If a table is such that no pair of conditions is logically dependent then there are no implied condition entries nor any *a priori* excludable 'Y, N' combinations. In this, case the table must contain or imply 2^m distinct, 'Y, N' combinations if it is to be complete, where m denotes the number of conditions in the table. If it is to be unambiguous then, if a particular 'Y, N' combination is included in more than one rule, these rules must specify the same action set. If a table has one or more pairs of rules which are logically dependent then the problem specification should include statements of this logical dependence which can be used in table checking as indicated in the foregoing discussion.

It is important to distinguish logical dependence and independence as defined above from dependence in a probabalistic sense (stochastic dependence). With this type of dependence, knowing the outcome of $C_a(x)$ provides information on the likelihood of the outcome of $C_b(x)$. Thus a Yes outcome to 'height greater than 6 ft' means that we are more likely to get Yes than No to 'weight greater than 140 lbs'. Whilst information on stochastic dependence may be taken into account at the implementation stage, to obtain efficiency it is not relevant at the checking stage when we are concerned only that there is correct provision for every outcome which may occur, however infrequently.

Conclusions

It is important to distinguish between decision table checking to establish that the problem solution is adequately defined and the translation of a table to a lower program level. Decision tables require checking procedures which should be quite distinct from procedures to convert to other program forms. Recent published material indicates that this is not fully appreciated and that basic decision table format is not properly understood.

If algorithms are to be efficiently specified by decision tables then it seems necessary to take account of relationships between conditions in the checking process. This is particularly true of application in the area of commercial data processing where the conditions are frequently highly related. It seems clear that table checking should be a computer function and that statements of relations between conditions should be supplied by the problem analyst in addition to the table itself. It is not suggested that the notation used in this paper is a suitable vehicle for supplying this information in the d.p. field but it might help to give insight in theoretical investigations. A simple more widely understood notation is easily specified for practical d.p. work.

Translation from the usual type of branching specification to decision tables, followed by a checking of the resulting tables, has been used as a method of checking branching-type specifications (Harris, 1967). If this method is used as a basis for producing warning diagnostics, it would seem desirable for programming languages to include facilities for making general statements about a program in a notation equivalent to that of the first order predicate calculus.

References

BARNARD, T. J. (1969). A new rule mask technique for interpreting decision tables, *Comp. Bull.* **13**, pp. 153–154.
BJORK, H. (1968). Decision Tables in ALGOL 60, *BIT*, **8**, pp. 147–153.
CHAPIN, N. (1967). An Introduction to Decision Tables, *DPMA Quarterly*, April 1967, pp. 2–23.
HARRIS, F. T. C. (1967). The partial automation of systems analysis, contribution to Datafair '67.
KING, P. J. H. (1967). Decision Tables, *Comp. J.*, **10**, pp. 135–142.
KING, P. J. H. (1968). Ambiguity in Limited Entry Decision Tables, *CACM*, **11**, pp. 680–684.
POLLACK, S. L. (1967). CR 12948, *Comp. Revs.*, **8**, pp. 501–502.
POLLACK, S. L. (1963). Analysis of the Decision Rules in Decision Tables, Memo RM-3669, Rand Corp., Santa Monica, California.
SPRAGUE, V. G. (1966). Letter to Editor (On Storage Space of Decision Tables), *CACM*, **9**, p. 319.
STOLL, R. R. (1963). *Introduction to Set Theory and Logic* (Chapter 4), W. H. Freeman and Co., San Francisco and London.
WIRTH, N. (1966). A note on Program Structures for Parallel Processing, *CACM*, **9**, pp. 320–321.

Book Review

Automatic Information Organization and Retrieval, by Gerard Salton. 1968. 514 pages. (*McGraw-Hill*, 126s.)

The main theme here is reference retrieval. Straight-forward mechanical aids to essentially manual methods are not really considered, however—the emphasis is on the sophisticated techniques needed to carry out the whole operation by computer, both the initial analysis of each incoming document leading to storage of a summary, and the subsequent retrieval of references to those documents relevant to a particular request. The main problem is to obtain an accurate picture of the meaning or content of each document and request; each is summarised as 'a vector of content identifiers', which vectors can then be matched. Thus the techniques involved include the construction and use of synonym dictionaries, syntactic analysis of English text, cluster analysis, statistical phrase analysis and structure matching techniques. Further, most importantly, ways of measuring the effectiveness of the various techniques are derived.

All these topics are explained, starting from a level to make them comprehensible to a reader with a general understanding of computer processing, but without specialist knowledge: representation of tree structures and the binary search technique, for example, are clearly explained. Rather more mathematical background is necessary: matrix manipulations appear without prior explanation, though the section on mathematical retrieval models is prefaced by an introduction to the necessary set theory.

Professor Salton being one of the pioneers in this research area, these are topics on which he is well qualified to write. The treatment is comprehensive, the material very well organised and the explanations clear. Many of the results tabulated in support of the theories have been obtained on the Smart experimental automatic retrieval system built up in the last few years by Professor Salton and co-workers. This system is indeed described, in a 65-page appendix, in the detail needed for a user to punch the cards specifying the options to be exercised on a particular computer run; do such details really belong in a text of this kind?

Results quoted all seem to relate to quite small collections of documents, which are adequate for comparison of various automatic techniques But little attention is paid to a comparison of either the effectiveness or the economics of these automatic methods with manual systems. In addition to the principal theme of reference retrieval, a chapter is devoted to systems for storing and retrieving facts themselves, rather than references. A number of auxiliary operations—text editing, indexing, selective dissemination of information—are touched upon in a further chapter. It is unfortunate that the sample of output from a syntactic analyser program on page 167 should show an incorrect analysis of the sentence chosen, but this is an isolated flaw in a book where a great mass of detail has been carefully assembled.

This book, then, gathers together the main theories and results of recent work in automatic I.R.; over 300 references are given, as well as a selective bibliography of another 300 items. In addition to those directly involved with I.R., programmers in other applications areas may well find some parts of the text relevant, for many of the techniques explained have wider application.

A. J. MITCHELL (Leeds)