DECISION TABLES AND COMPUTER PROGRAMMING

R. WELLAND

DECISION TABLES AND COMPUTER PROGRAMMING

R.WELLAND

Department of Computer Science
Strathclyde University
Glasgow UK

Heyden & Son Ltd., Spectrum House, Hillview Gardens, London NW4 2JQ, UK
Heyden & Son Inc., 247 South 41st Street, Philadelphia, PA 19104, USA
Heyden & Son GmbH, Deveseldergstrasse 6, 4440 Rheine, West Germany

---

British Library Cataloguing in Publication Data

Weiland, R. Decision tables and computer programming.
1. Electronic digital computers – Programming
2. Decision logic tables
I. Title
001.64'25 QA76.6

ISBN 0-85501-708-2

Printed in Great Britain by St Edmundsbury Press, Bury St Edmunds

PREFACE

I was originally asked, several years ago, to write a book about decision tables aimed at scientists and engineers, the object being to show how decision tables might be applied to problems in these fields. One of the difficulties of producing a book to satisfy this original aim has been the lack of existing applications in science and engineering, or more correctly literature describing such applications. Therefore this book reviews the basic theory of decision tables, including methods for translating them into programs, in the hope that scientists and engineers will find the applications!

There are a number of texts on decision tables which deal with the role of decision tables in the commercial data processing environment, these are referenced in the text, where appropriate. These books tend to concentrate on the use of decision tables as a specification tool in systems analysis and design, while this book is aimed at the algorithmic use of decision tables.

The content of the book falls into three major sections: chapters 1-3 form a review of the basic theory of decision tables; chapters 4-6 deal with algorithms for the automatic translation of decision tables into programs; and chapter 7 describes some applications of decision tables.

I would like to thank all the people who have contributed to this book by discussing particular points with me and providing material. These include Professor P.J.H. King and Mrs. J. Mercer of Birkbeck College London, Gordon Davies of University College London and Mr. A. Black of the NCC. I would also like to thank my colleague John Patterson for allowing me to describe his application of decision tables to finite state automata.

Two other colleagues have assisted in the production of this book: Zdrav Podolski, who provided technical support, and John Jeacocke, who spent a lot of time studying the drafts of the book and making constructive comments about the contents and my English! Finally I should like to record my thanks to Mr. L.C. Thomas, the series editor, for his very helpful comments about the general style of the book.

University of Glasgow
December 1980

Ray Welland

CONTENTS

Chapter 1

BASIC PRINCIPLES

A decision table is a symbolic way of representing the logical interdependence between events and is one of a number of such techniques, which include narrative form (reports), flowcharts, Karnaugh maps and boolean algebra. The use of decision tables has been widely discussed and among the advantages claimed for their use have been:-

(i) the logic is stated explicitly and precisely in a compact form,
(ii) a table can be checked for omissions and logical inconsistencies,
(iii) a table can be modified easily to reflect a change of circumstances,
(iv) it is possible to "automate" the process of producing a program from a given decision table,
(v) the table is useful as a documentation aid,
(vi) the format of most tables is comprehensible to the non-computer specialist.

Before detailing the actual structure of a decision table, it is worth considering these claims briefly and relating the use of the tabular format to the two main forms of expression used in computing: the narrative and the flowchart. There are two main weaknesses of the narrative form when expressing complex logic, these are precision and relevance. It is very

difficult to specify the rules for a particular situation precisely, in English, and when reading a report it is often difficult to locate and extract the relevant facts. This type of specification problem arises frequently every day, for example when a government department attempts to describe, for the benefit of the general public, precisely what is meant by a particular piece of legislation!

A flowchart is a much better way of defining problem logic than narrative but there are two ways in which it also fails, namely its serial nature and the inability to check it thoroughly. To understand the meaning of a flowchart it must be followed through step by step (box by box) i.e. serially and for even a small number of logical combinations it can become very unwieldy; the isolation of one logical combination of events may be spread over several pages of flowchart. Another drawback of the flowchart, associated with its serial nature, occurs when it is used as a programming, rather than an analysis or design, tool. It is not purely a representation of problem logic but also implies the structure of the program to be generated, thus an attempt is being made to solve two problems in one step. The second area of failure of the flowchart is the difficulty of systematic checking; in certain problem areas e.g. process control all combinations of events must be considered and it is quite difficult to check a flowchart for "completeness".

The ability to modify a program rapidly can be important in some environments, and a program automatically generated from a decision table can be very flexible in this respect. In commercial applications, particularly, the environment in which a program is used can be changed quite rapidly by external influences e.g. government legislation on taxes, and in this situation ease of modification is an important consideration.

The documentation aspect of using decision tables has, in many cases, been oversold in the past! The use of a decision table to specify the logic of a program does *not* remove the need for other documentation. Similarly while decision tables are a useful tool in problem specification they are unlikely to constitute a complete description of a problem, except at the

lowest levels of refinement. Decision tables are widely recommended as a communications tool for the systems analyst because of their compact and precise nature, and also because of their general comprehensibility. However a great deal of work has been carried out in the last few years on the automatic checking of decision tables and their translation into program segments. It is the aim of this book to survey this area in the hope that an appreciation of the usefulness of decision tables as a programming tool will become more widespread.

1.1 The basic components of a decision table

The general format used for decision tables in this book conforms to the British Standard for Decision Tables[1], with the exception of skeletal tables with formal conditions (of the form C_i) where table names are not given unless required for looping and actions are sometimes excluded.

A decision table is conventionally considered as consisting of four quadrants, as shown in Figure 1.1. The table is divided into two horizontally: the upper part being the conditions to be evaluated and the lower part the consequent actions. The vertical division separates the stub from the entries, and a column of entries is known as a rule. A simple example of a table is shown in Figure 1.2. This particular table is a limited entry table in which condition entries are restricted to 'Y' (yes), 'N' (no) and '-' (don't care), and the action entries are limited to 'X' (perform this action) and '-' (do nothing).

Each vertical column in the entry half of the table is called a rule which consists of a number of condition entries,

Condition Stub	Condition Entries	R U
Action Stub	Action Entries	L E

Fig. 1.1

Zout		R_1	R_2	R_3	R_4
$X_1 = 1$		Y	N	N	-
$X_2 = 1$		-	Y	N	-
$Y = 1$		Y	Y	Y	N
Z := 1		X	X	-	-
Z := 0		-	-	X	X

Fig. 1.2

indicating the truth values of relevant conditions, and action entries, indicating which actions are to be performed. For example R_2 (Rule 2) in Figure 1.2 states:

if $X_1 \neq 1$ and $X_2 = 1$ and Y = 1 then Z := 1

and R_1 states:

if $X_1 = 1$ and Y = 1 then Z := 1 { X_2 is not relevant }

Any particular set of circumstances is known as a transaction thus $X_1 = 1$, $X_2 = 1$, $Y_1 = 1$ corresponds to the transaction {Y,Y,Y} which satisfies the first rule of the table. The process of deciding which actions to perform consists of evaluating the conditions to produce a transaction vector and then comparing this with each of the rules (condition entries only) in turn looking for a match. When the transaction matches the rule in all relevant positions (i.e. ignoring dash entries) the actions relevant to the rule are performed.

A second example of a decision table is given in Figure 1.3, here the notation has been extended by removing the restriction on entries, this is an extended entry table, where the whole condition or action is constructed by concatenating the stub and entry. This simple example deals with the problem of identifying the position of a point (x,y) on a two

Quads		R_1	R_2	R_3	R_4	R_5	R_6	R_7
x		>0	>0	<0	<0	≠0	=0	=0
y		>0	<0	>0	<0	=0	≠0	=0
(x,y)		$quad_1$	$quad_4$	$quad_2$	$quad_3$	$axis_x$	$axis_y$	origin

Fig. 1.3(a)

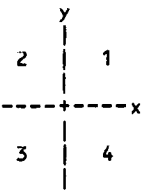

Fig. 1.3(b)

Extended-example	R_1	R_2	R_3	R_4	R_5	R_6	
Wholesale customer ?	Y	Y	Y	N	N	N	
Order > 50 items	Y	Y	N	Y	Y	N	
Delivery distance	<50	>50	-	<75	>75	-	*
Discount =	15%	10%	0	10%	5%	0	*
Send salesman	-	-	X	X	-	-	

Fig. 1.4

dimensional graph, where the quadrants are numbered as shown in Figure 1.3(b). Although fundamentally a very simple problem this sort of logic can prove quite difficult to the beginning programmer, especially when faced with only a simple IF statement which compares two values, such as that used in BASIC.

Figure 1.4 shows a further example illustrating a mixed entry table which is a decision table consisting partly of limited entry rows and partly of extended entry rows (starred). This table is abstracted from a commercial application involving the calculation of discounts which depend upon the type of customer, the quantity of goods ordered and the transport costs involved in delivery.

Mixed or extended entry tables can always be converted to limited entry tables by expanding all extended entry rows to an equivalent set of limited entry rows[2]. The expanded version of the table of Figure 1.4 is shown in Figure 1.5; apart from increasing the size of the table this expansion may cause other problems which will be explored later. The fact that mixed and extended entry decision tables can always be converted to limited entry format means that most of the work carried out on

Extended-to-limited	R_1	R_2	R_3	R_4	R_5	R_6
Wholesale customer ?	Y	Y	Y	N	N	N
Order ≥ 50 items	Y	Y	N	-	-	-
Delivery distance < 50	Y	N	-	-	-	-
Delivery distance < 75	-	-	-	Y	N	-
Discount = 15%	X	-	-	-	-	-
Discount = 10%	-	X	-	X	-	-
Discount = 5%	-	-	-	-	X	-
No discount allowed	-	-	X	-	-	X
Send salesman	-	-	X	X	-	-

Fig. 1.5

the theory of decision tables has been concerned with the manipulation of limited entry tables.

1.2 Fundamental theory

This section is intended to introduce some of the fundamental concepts of rules and tables, with reference to limited entry decision tables. The first distinction to be drawn is between simple rules and complex rules. A rule which contains only "Y" and "N" entries is a simple rule, for example Figure 1.6(a). A rule containing one or more dashes is a complex rule and can be expanded into simple rules e.g. the complex rule of Figure 1.6(b) can be expanded into two simple rules as shown. The combining of two simple rules into a complex rule is known as consolidation and can be performed when two simple rules have only one distinct "Y","N" pair of condition entries and exactly the same actions.

Two rules are said to be disjoint if they do not have any simple rules in common. Therefore the two complex rules shown in Figure 1.6(c) are disjoint because when they are expanded the two sets of simple rules are completely different. When two rules are not disjoint they are said to be overlapping. Overlapping rules give rise to two problems: redundancy and contradiction, sometimes collectively known as ambiguity.

An example of contradiction is shown in Figure 1.7, when

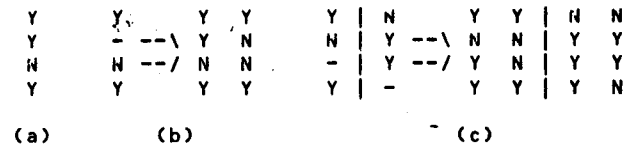

Fig. 1.6

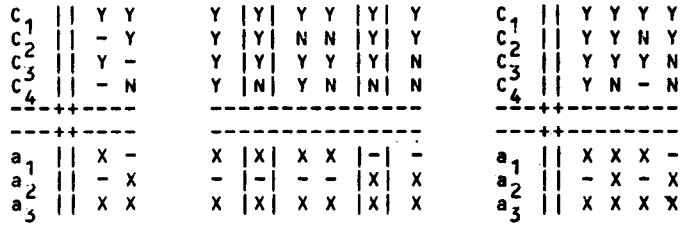

Fig. 1.7

the complex rules are expanded this gives rise to six simple rules, one of which {Y,Y,Y,N} appears twice with contradictory actions, dependent upon which of the complex rules it was derived from. The solution to this problem is to reconsider the meaning of the table and reformulate the table in an unambiguous way, a possible reformulation is shown on the right of Figure 1.7. The other possible problem is redundancy which occurs when overlapping rules specify the same action(s). This is not such a serious problem as contradiction although most of the decision tree algorithms, described later in this book, will fail if there are redundant rules in the decision table being processed. To eliminate redundancy the rules are simply redefined so that they are disjoint, an example illustrating redundancy and its elimination is given in Figure 1.8.

Two more pieces of terminology which are used throughout this book are action sets and rule sets. It is often convenient to think of the actions associated with a particular rule as an action set and many of the examples in this book deal with action sets rather than individual actions. In the table shown in Figure 1.9 the A_i represent action sets, note that action sets are not necessarily disjoint and that more than one rule

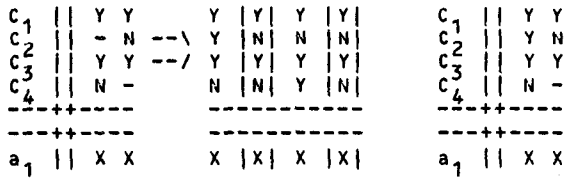

Fig. 1.8

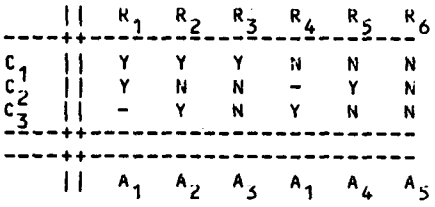

Fig. 1.9

may lead to the same action set. This leads naturally to the concept of a rule set which is the set of rules having the same action set. In the example of Figure 1.9 $\{R_1, R_4\}$ form a two member rule set while all the other rules form single member rule sets.

The final topic which it is appropriate to discuss here is the problem of completeness i.e. whether all possible combinations of conditions have been considered in a particular table. To ascertain whether a decision table is complete two simple formulae are required ;-

(i) A complex rule containing d dashes is equivalent to 2^d simple rules. (look back at the examples of rule expansion)
(ii) A table having c conditions is complete if it is equivalent to 2^c disjoint simple rules.

Applying formula (i) to the table given in Figure 1.10 indicates that it is equivalent to 14 simple rules and that therefore it is not complete since a complete table would contain $2^4 = 16$ disjoint simple rules. If the table is expanded into its 14 simple rules then the missing combinations can be identified as {Y,N,N,N} and {N,N,Y,Y}.

C_1	Y	Y	Y	N	N	N	
C_2	Y	N	N	Y	-	-	
C_3	-	Y	N	Y	Y	N	
C_4	-	-	Y	Y	N	-	
Simple Rules	4	2	1	1	2	4	=> 14

Fig. 1.10

1.3 Extending the table format

In this section a number of extensions to the basic decision table format are illustrated and discussed. The first, and probably oldest, of these ideas is the inclusion of a special rule called the ELSE rule. This is conventionally written as the last (rightmost) rule of an incomplete decision table and defines an action set for all transactions not satisfying the explicit rules of the table. Figure 1.11 gives a simple example of a table which includes an ELSE rule. It is obvious that a table with an ELSE rule is complete since all possible transactions lead to a defined action set. With a table in which all conditions are in limited entry format it is a straightforward process to calculate how many simple rules are covered by the ELSE rule, the procedure can be stated as follows :-

(i) Check that all explicitly stated rules are disjoint i.e. that there are no redundancies or contradictions.
(ii) For each explicit rule calculate the number of equivalent simple rules (using the formula from the previous section). Compute s, the total number of simple rules equivalent to the explicit rules of the table.
(iii) If the table contains c conditions then the ELSE rule is equivalent to $2^c - s$ simple rules.

For the table given in Figure 1.11 it can be seen that :-

(i) All the explicitly stated rules are disjoint.
(ii) $s = 1 + 1 + 4 + 4 = 10$
(iii) The ELSE rule is equivalent to $2^4 - 10 = 6$ simple rules.

Deal-with-customer	R1	R2	R3	R4	
Customer is wholesaler ?	Y	Y	Y	N	E
Order value exceeds £100 ?	Y	Y	-	Y	L
Account balance in credit ?	Y	Y	N	-	S
Delivery within 50 miles ?	Y	N	-	-	E
Despatch goods	X	X	-	X	-
Despatch payment reminder	-	-	X	-	-
Give discount of	10%	5%	-	2%	-
Send sales representative	-	-	-	X	X

Fig. 1.11

For a decision table with conditions in extended entry format it is not really meaningful to define how many simple rules are "missing" (and therefore covered by the ELSE rule) because this involves defining the total number of outcomes for each extended entry question.

Pollack et al[2] recommend that the ELSE rule should be used only as an error rule, not as a "residual rule", while other authors, for example Knight[3], are opponents of the ELSE rule per se. A note in the British Standard[1] states that "the ELSE rule should always be used with caution since it could be accepted in place of a rule omitted by mistake". In the author's opinion the ELSE rule is useful in the limited way suggested by Pollack, especially if there are a large number of highly improbable combinations of conditions which can be dealt with by one error routine.

Johnson[4] suggests two new conventions associated with extended entry tables. The first of these is the use of the special symbol "@" to represent "any value other than those explicitly mentioned elsewhere". The table in Figure 1.12 specifies a code checking algorithm for a nine character code of the form: two alphabetic characters, six numeric characters, terminated by one of the letters A, B, C or D. This illustrates a possible use of the '@' convention, rule 5 separates the special case where the code letter is not equal to A, B, C or D from the more general error conditions.

Johnson's other suggestion for extending the table format