acm

ACM MONOGRAPH SERIES

Decision Table Languages and Systems

John R. Metzner

Bruce H. Barnes

DECISION TABLE LANGUAGES AND SYSTEMS

JOHN R. METZNER

Computer Science Department
University of Missouri-Rolla
Rolla, Missouri

BRUCE H. BARNES

Division of Mathematical and Computer Sciences
National Science Foundation
Washington, D. C.

ACADEMIC PRESS New York San Francisco London 1977
A Subsidiary of Harcourt Brace Jovanovich, Publishers

ACADEMIC PRESS, INC.
111 Fifth Avenue, New York, New York 10003

United Kingdom Edition published by
ACADEMIC PRESS, INC. (LONDON) LTD.
24/28 Oval Road, London NW1

LIBRARY OF CONGRESS CATALOG CARD NUMBER:

Library of Congress Cataloging in Publication Data

Metzner, John R
Decision table languages and systems.

(ACM monograph series)
Bibliography: p.
Includes index.
1. Electronic digital computers–Programming.
2. Decision logic tables. I. Barnes, Bruce H., joint author. II. Title. III. Series: Association for Computing Machinery. ACM monograph series.
QA76.6.M484 001.6'42 77-4011
ISBN 0–12–492050–0

PRINTED IN THE UNITED STATES OF AMERICA

DECISION TABLE LANGUAGES AND SYSTEMS

PREFACE

This monograph addresses the entire topic of decision tables and considers its various aspects at three distinct but nested levels. At the lowest level is the decision table itself, its structure, conventions, and semantics. At the second level is the decision table language, of which decision tables are components. The third, or system, level includes the decision table language, its processor or translator, its users and the application area from which their problems are drawn, and the environment in which the algorithms expressed in the decision table language are to be operant.

The overall purpose of this volume is to widen the use and appreciation of decision tables. The authors' intent is to stimulate interest in decision tables, especially in those who have not made use of them in the past. Of greater moment is our intent to encourage and facilitate further research, innovation, and development of decision table languages and systems. This is not another volume on "how to use decision tables," but rather a treatment of the nature of decision tables as linguistic entities and of ways in which decision table systems can be adapted to suit a wide variety of particular purposes.

Accordingly, our attempt has been to orient the material herein toward innovators in system analysis, designers of languages and their systems, and students of the discipline often called "computer science" whether they be in academic institutions or not. We seek to further the stated goals by appealing to this audience and providing:

(1) a thoroughgoing linguistic examination of decision tables, a survey of the features of existing decision table languages and systems, and an exploration of generalizations thereof which exposes their potential for improvement and adaptation;
(2) a sound conceptual basis for further research on decision tables and a comprehensive bibliography of the relevant literature; and
(3) encouragement to innovative but cogent language and system design in the forms of provocative suggestions, opinions on the utility of various features to particular classes of users and in constrained operating environments, and a framework with which to evaluate the features of candidate designs in their entire system context.

It must be admitted that decision tables often have a natural appeal to those of us who have a strong appreciation of the succinct, and the authors are not exceptions. This has led to a tendency toward brevity in style which we were not wholly successful in counteracting, and for which we must apologize.

CONTENTS

CHAPTER 1

INTRODUCTION

Motivation and Outline

Procedural disciplines in general and computer science in particular have experienced a continuing and urgent need for improved vehicles for the communication of algorithms. Programming languages used for human-to-machine communication must bear an extra burden of utility in human-to-human communication, even if it is only from the author of a program to himself at a later time. Because of this extra burden and to increase efficiency in program synthesis, programming languages should be concise yet easily understood. Within programming languages, these needs for clarity and compactness are especially acute with respect to linguistic elements for conditioning, particularly those linguistic elements used to express the selection procedure by which a processing task is broken down by cases.

The conditioning elements commonly included in higher-order languages supply binary branching based on combinations of arithmetic tests along with many-way branching based on integral values of variables. While quite involved conditioning can be built up using these elements, the resulting programs are often cluttered with housekeeping variables which must be used to arithmetize the decision process and record test results. For a simple example, suppose it is desired to determine the quadrant in which the two-place vector X falls and branch to a correspondingly labeled statement. The code to accomplish this might look like

```
          IF (X(1). GE.0.) GO TO 9
          IF (X(2). LT.0.) GO TO 3
          GO TO 2
    9     IF (X(2). LT.0.) GO TO 4
    1     . . .
```

or

```
      I = 1
      IF (X(1). LT.0.) I = I + 1
      IF (X(2). LT.0.) I = I + 2
      GO TO (1,2,4,3), I
```

or

```
      IF X(1) < 0 & X(2) < 0 THEN GO TO S3; ELSE
      IF X(1) < 0 THEN GO TO S2; ELSE IF X(2) < 0
      THEN GO TO S4;
S1:   . . .
```

It is easy to imagine how obscure such code can become as increasingly involved conditioning is needed in the effort to express increasingly general algorithms.

Moreover, when many-way branching is built from test-and-branch language elements, the flow of control is used to "remember" the outcomes of comparison tests. This use of control flow leads to programs which are difficult for people to comprehend because they force the reader to backtrack in order to trace the paths and deduce the combinations of conditions leading to the various branches. The situation is further confounded by the common (and space-efficient) tendency for conditional statements to be distributed amid other types of code rather than collected into sections which express the many-way conditional branches involved. Flow charts offer only a small measure of relief from the difficulties in comprehending programs containing involved or distributed conditioning but are often laboriously produced to lend their increment of enlightenment.

When decision tables were introduced into the programming process, it was felt that they had great potential for succinctly expressing highly articulated procedures which select from many alternative case treatments. However, they have not enjoyed acceptance in the programming community commensurate with this promise, nor have they shared in the rapid development of software technology experienced by other areas such as operating systems and compilers.

The use of decision tables has been resisted partly because they require a departure from the accustomed intermixing of decisions and actions within algorithms. In other words, the natural opportunities to employ decision tables when synthesizing algorithms are not suggested when bottom-up programming practices are used. As the benefits of top-down methods in problem analysis and programming become more widely appreciated, and the use of top-down methods becomes more prevalent, this barrier to decision table use is expected to diminish considerably.

A more serious impediment to wider decision table use has been caused by that lack of general acceptance itself. Decision table languages and systems have not, in the forms developed so far, been adapted to a very wide range of algorithmic situations, and they therefore have failed to reach the "critical mass" of everyday use necessary to generate impetus toward significant innovation and improvement.

Although this work naturally seeks to stimulate a general interest in decision tables, its major attack is on the latter aspect of the problem. It seeks to accelerate the development of decision table programming by identifying its conceptual basis, generalizing its features, suggesting innovative uses for the generalized features, and providing a method for estimating their efficacy in decision table programming systems. Following the introductory overview of the topic in this chapter, this goal is attacked in four steps which form the subjects of the following chapters.

Decision tables are first examined as linguistic entities by themselves and as elements of decision table languages. Chapter 3 concentrates on features of decision table programming languages and systems and includes a summarization of the mixes of features found in several which have been developed to operational status. The fourth chapter proposes generalizations and extensions of the features identified. This sampling of possible generalizations is intended to be provocative rather than definitive; many are simply carry-overs from other, more richly developed areas of software technology.

The final chapter outlines a procedure for designing decision table programming languages and their systems. To underscore the richness and adaptability of decision tables, the procedure assumes that the language and system are to be tailored for a particular environment. Taking account of the larger context containing people, problems, and processors, environments are considered to include a conceptual portion relating to the subject area in which problems are formulated and algorithms are synthesized. Also considered are the physical components of environments which impose design constraints such as core storage limitations or interpretive execution in conversational use. The last chapter also bears the burdens of tying together the concepts and features presented in the previous three chapters and of indicating some of the many opportunities for further research, experimentation, and development in decision table languages and systems.

Decision Structure Tables

Decision structure tables, as decision tables were initially termed, have been used for many years to organize and document complex decision procedures concisely. Their uses in tax rate tables and in the rate books of insurance salesmen are familiar examples. Early formal usage [157] was for the expression of a set of boolean functions of several variables. Incorporating the interpretations of the variables and functions [78] created useful communication devices which were, for many purposes, more suitable than either flow charts or prose [14, 180]. The development of these devices served somewhat to standardize decision table languages around a core of conventions which ever since has been retained.

A decision table is divided into two main regions, one specifying sets of *conditions* which must be satisfied simultaneously, and the other specifying sets of *actions* to be taken when corresponding condition sets are satisfied. The two areas are placed with the condition portion above the action portion, although some early formulations placed them horizontally adjacent [59]. The vertical orientation will be used in this book because tables in that form are more compact and, in the opinion of the authors, easier to read. A matrix of symbol strings called *entries* is placed in each of the regions to indicate the condition and action specifications. The two matrices have the same number of columns which are aligned across the common boundary. The columns individually describe an "if . . . , then . . ." relationship called a *rule*; if the conditions of the upper portion are all met, then the actions of the lower portion are to be carried out.

Figure 1.1 illustrates a collection of such rules describing a decision on how to spend a spring Saturday afternoon. The horizontal double line separates its two regions, and the vertical lines distinguish its rules. The internal horizontal line between its two rows of condition

Raining	Not raining	Not raining
Windy or calm	Calm or very windy	Breezy
Clean the basement	Spade the garden	Fly kites with children

FIGURE 1.1 Spring Saturday afternoon decision.

phrases represents the "and" conjoining them. This decision contains three rules:

"If it is raining and either windy or calm, then clean the basement";
"If it is not raining and either calm or very windy, then spade the garden";
"If it is breezy and not raining, then fly kites with the children."

In this example, the symbol strings are English phrases which appear in both the textual and tabular forms of decision description. In more formal situations, the strings are restricted to be in a particular specified language which we will abstractly call the *base language*.

Conciseness can be enhanced by "factoring out" common information from the array elements in each row and placing it in the leftmost column (set off by a double vertical line) called the *stub*. For conditions, a truth-valued expression may be placed in the stub, leaving only truth values in the columns specifying tests. For actions, the entire description of the action may be put in the stub, so that entries need only indicate performance or nonperformance of the action. A table for deciding about a trip to Boston is shown in this primitive form in Figure 1.2.

This example shows a table for making one decision. It is called a *complete* table because for every situation there is a rule whose conditions will be satisfied; it cannot fail to specify a set of actions to be taken. The confusing clutter of entry symbols in decision tables of this form suggests the simplification to what is termed *limited entry form* shown in Figure 1.3. Blank entries in condition rows indicate "don't care," and blanks as action entries signify "don't perform." Since there are now only two possible action entries, any nonblank character can be used to carry the connotation "perform."

Limited entry form also offers some help in recovering the textual

Weather is fair	T	F	T	F	T	F	T	F	CONDITIONS
Plane seat reserved	T	T	F	F	T	T	F	F	
Hotel room reserved	T	T	T	T	F	F	F	F	
Take the plane	T	F	F	F	F	F	F	F	ACTIONS
Cancel plane seat	F	F	F	F	T	F	F	F	
Take the train	F	T	T	T	F	F	F	F	
Try again tomorrow	F	F	F	F	T	T	T	T	

STUB ENTRIES

FIGURE 1.2 Trip to Boston decision table—primitive form.

forms of the rules of which the tables are composed. Reading down the rule column, a T entry signals inclusion of the condition from the horizontally corresponding stub into the "if . . ." portion of the textual form, and an F entry calls for including the negation of the expression in the corresponding stub. Blank condition entries require neither; the condition is simply not mentioned in the textual form. Similarly,

Weather is fair	T	F			
Plane seat reserved	T	T	F	T	F
Hotel room reserved	T	T	T	F	F
Take the plane	X				
Cancel plane seat				X	
Take the train		X	X		
Try again tomorrow				X	X

FIGURE 1.3 Trip to Boston decision table—limited entry form.

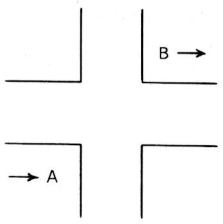
action entries and blanks call for the inclusion and omission, respectively, of their action stubs into the "then . . ." portion of the rule. For example, the fourth rule of Figure 1.3 becomes,

"If a plane seat has been reserved, and a hotel room has not been reserved, then cancel the plane reservation, and try again tomorrow."

This algorithm for rule reconstruction indicates that the rules of a decision table are to be read from top to bottom and thereby places that ordering on conditions and actions. The ordering was not conveyed by the terms "sets of conditions" (*condition sets*) or "sets of actions" (*action sets*) used previously, but ordering is regarded as implicit only for actions in decision tables. Although the "and" conjoining individual conditions in a rule suggests that they are independent of the order in which they appear, they need not be so regarded. An example of order-dependence among conditions is presented later.

It is far more obvious that the actions specified by a rule are to be carried out in their top-to-bottom order of appearance. Figure 1.4 illustrates a rather mundane decision in which action sets differ only in the ordering of their components. The situation is that of a pedestrian (point A in Figure 1.4a) approaching an intersection governed only by a red/yellow/green traffic signal and wishing to proceed in the same direction from the diagonally opposite corner (point B) without undue delay. The color of the signal he is facing initially determines which of the two sequences of crossings and turns he will perform, as shown in the decision table of Figure 1.4b. Since any nonblank symbol can be used in action entries to indicate performance of actions, one could make the tables smaller in some cases by using numerals to indicate the order in which the corresponding actions are to be carried out. Figure 1.4c illustrates the reduction for this example.

Limited entry form for decision tables has several other desirable characteristics. The use of the base language (English in the figures presented so far) is confined to the stub portion, and the semantics of the entry portion are quite simple—so simple, in fact, that the rules of a limited entry decision table can be mechanically checked for logical completeness, consistency, and redundancy [117]. Inconsistency or redundancy can arise when a situation can be constructed in which the "if . . ." portions of two rules can be found to hold. If the actions of the two rules are the same in such a case, the pair is said to be *redundant*. If the actions differ, the two rules (and thus, their table) are termed *inconsistent*. Figure 1.5 illustrates these two conditions. The situation "Condition 1 is true, Conditions 2 and 3 are false" meets the condition



(a)

FACING RED LIGHT	Y	N
CROSS ON GREEN		X
TURN LEFT	X	X
CROSS ON GREEN	X	X
TURN RIGHT	X	X
CROSS ON GREEN	X	

(b)

FACING RED LIGHT	Y	N
TURN LEFT	1	2
CROSS ON GREEN	2	1
TURN RIGHT	3	4
CROSS ON GREEN	4	3

(c)

FIGURE 1.4 Crossing intersection decision: (a) situation, (b) "perform" action entries, (c) ordinal action entries.

specifications for both Rule 1 and Rule 2. Since their action specifications are identical, these two rules are redundant. The situation "Conditions 1 and 2 are true and Condition 3 is false" is said to satisfy the condition sets of both Rule 1 and Rule 3. As these two rules call for the performance of different sets of actions, they are inconsistent. (The term *contradiction* is also used to describe this latter case [12, 81].)

It should be noted that these cases of apparent ambiguity derive from the convention that the ordering of the rules within a decision table carries no significance. Otherwise, the positions of the rules can be used to denote their priorities. This and other related semantic aspects will be explored more fully later.

A slight mutation of the limited entry conventions allows the rightmost rule to contain only "don't care" condition entries. Such a rule is called an *else rule* and is understood to be invoked when the conditions for no other rule are met. Figure 1.6 shows the convenience of using an else rule by contrasting two limited entry tables (defining a logical