# Decision Tables Revisited:
## The DDLA Decision Table Software System

—*by Don Latour*
*Sea Cliff, New York*

EVEN THOUGH DECISION TABLES HAVE BEEN AROUND since the late 1950's, they have been largely ignored in recent years. Nonetheless, they still can serve as a useful tool in building a computer application. This paper discusses decision table format and its practicality in the context of the DDLA Decision Table Software System, a PC-based APL+DOS implementation of a decision table editor, validater, file manager and mainframe (BAL) program generator.

## Introduction

This is a decision table:

| | | | | |
|---|---|---|---|---|
| *GENDER EQ FEMALE* | *Y* | *Y* | *N* | *N* |
| *AGE LE* 65 | *Y* | *N* | *Y* | *N* |
| *PROCESS*1 | *X* | | | |
| *PROCESS*2 | | *X* | *X* | *X* |

Figure 1

It is a format for specifying programming logic. It is also:

1. a tool of thought,
2. a format that laymen can understand, and
3. is a basis for:
   a. verifying specification completeness,
   b. generating a computer program to implement the logic,
   c. performing run-time dynamic optimization,
   d. providing run-time audit trails,
   e. generating test data,
   f. detecting system integrity failure,
   g. organizing parallel processing of program logic and
   h. encouraging forward and back tracking analysis.

The table of Figure 1 will be discussed below as will each item in the preceding list, but first a word about the subject software.

The DDLA Decision Table Software System consists of a decision table editor, a validater, a data definition facility, a file manager and a mainframe assembler program generator. It is a PC-based system written in APL+DOS and, therefore, runs on any platform supporting DOS.

The software is industry independent but was developed to assist in the specification and implementation of complex systems used in the Direct Marketing industry. These systems target multiple geographic, demographic and product-usage groupings from huge files of consumer households (100 million records). Tens of millions of pieces of mail can be generated from a single pass of a file. Note that these files were compiled from data voluntary submitted by the consumer and with the understanding that the data would be used for advertising purposes. In any case, a consumer can have their name removed by contacting the Direct Marketing Association.

Decision tables are an ideal specification tool in this environment because both marketing and technical personnel can communicate in a technically precise manner with the knowledge that the approved specifications will be implemented faithfully and efficiently.

APL is an excellent language for this application because it communicates well with other environments used by mainstream marketers, provides fast application development and is adept at handling strings and tables.

## Decision table basics

In Figure 1, *GENDER* and *AGE* are the names of variables and *FEMALE* and 65 are values of these variables.

*GENDER EQ FEMALE* and *AGE LE* 65 are conditional boolean statements yielding boolean results. The blank line separating them is interpreted as an *AND*.

*PROCESS*1 and *PROCESS*2 are actions available for selection.

The columns to the right of the double vertical bars are rules. They define the action to be selected based upon combinations of the booleans. Thus, rule 1 (i.e. the first column) is read as: If *GENDER EQ FEMALE* is true (*Y*) and *AGE LE* 65 is true (*Y*), then select *PROCESS*1 (the *X* at the bottom of the column selects *PROCESS*1).

Rules 2, 3 and 4 specify that all other combinations of conditions select *PROCESS*2.

Note that rules 3 and 4 are the same except for the *AGE* condition, and that condition specifies *Y* in one case and *N* in the other. These two columns may be combined, replacing the differing truth values with a dash, meaning "don't care". Thus, the table is simplified to:

|  |  |  |  |
|---|---|---|---|
| *GENDER EQ FEMALE* | *Y* | *Y* | *N* |
| *AGE* *LE* 65 | *Y* | *N* | - |
| *PROCESS*1 | X |  |  |
| *PROCESS*2 |  | X | X |

Figure 2

Some variables such as *GENDER* have values that are mutually exclusive and when more than one value is significant the syntax is different:

|  |  |  |  |  |
|---|---|---|---|---|
| *GENDER EQ MALE* | *Y* | * | * | *N* |
| *GENDER EQ FEMALE* | * | *Y* | *Y* | *N* |
| *AGE* *LE* 65 | - | *Y* | *N* | - |
| *PROCESS*4 | X |  |  |  |
| *PROCESS*5 |  | X |  |  |
| *PROCESS*6 |  |  | X | X |

Figure 3

In Figure 3 the lines containing *GENDER* are called a sub-table because they reference the same variable. The * means *N* by virtue of a more important condition, in this case a *Y* in the same column of the sub-table. That is, if *GENDER* is *MALE* then it cannot be either *FEMALE* or any other value. The fourth column of the sub-table contains all *N*'s. This indicates a value other than *MALE* or *FEMALE*.

Numeric variables may be used to specify ranges and this leads to another syntax and the last symbol that is used in the rules.

Figure 4 demonstrates a range sub-table for *AGE*. The new symbol $ means *Y* by virtue of a more important condition. That is, if *AGE is LE to* 17 it must be *LE* 49, 65 and 999. Rule 4 specifies an *AGE* greater than 999. This is most likely an "impossible" condition but must be specified to assure specification completeness (more on this later). It is a requirement that even impossible conditions must be specified.

You will note that the only relational operators allowed are *EQ* and *LE*. This is all that is required, and by not allowing *NE*, *LT*, *GT* and *GE* the preparation and understanding of a table are much easier (e.g., no double negatives).

Returning to Figure 1, note that the order (sequence) of the conditions or the rules does not change the semantics of the table.

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| *GENDER EQ FEMALE* | *Y* | *Y* | *Y* | *Y* | *N* |
| *AGE* *LE* 999 | $ | $ | *Y* | *N* | - |
| *AGE* *LE* 65 | $ | $ | - | * | - |
| *AGE* *LE* 49 | $ | *Y* | *N* | * | - |
| *AGE* *LE* 17 | *Y* | *N* | * | * | - |
| *PROCESS*7 | X |  |  |  |  |
| *PROCESS*8 |  | X |  |  |  |
| *PROCESS*9 |  |  | X |  |  |
| *PROCESSA* |  |  |  | X |  |
| *PROCESSB* |  |  |  |  | X |

Figure 4

## Why decision tables?

Let's go through the list from the Introduction.

### *A tool of thought*

At times in my career I have found myself presented with a programming logic problem so complex that it was literally mind boggling. I would start to analyze the requirements and then become so confused that I would have to start over. Finally, I would think I had it right, program it, start the almost impossible task of testing and then the requirements would change!

Decision tables offer an organized approach. First determine if all the information required for the resolution of the logic is available. If it is not then the problem cannot be resolved. Second, write down all the actions that are relevant. Third, write down all the data elements that are relevant. Fourth, one by one fill in the rules. It's really that simple!

### *A format that laymen can understand*

It is a demonstrated fact that sales and management personnel with no programming experience but, that otherwise understand the underlying requirements, can easily understand decision tables. Thus, responsible user management can directly participate in development and then sign-off on the resulting specifications.

### *A basis for verifying specification completeness*

Specification completeness means that every mathematical combination of conditions is specified without redundancy. The number of combinations is easily computed from the conditions (×/1+*L* where *L* is a vector containing the number of rows in each sub-table). If the rule count does not equal this number then the table is incomplete (remember a dash counts as two rules).

If the two figures agree then it is necessary to verify (ignoring actions) that the rules contain no redundancies. Bearing in mind that a dash will always produce an equal condition, each rule is compared to every other rule. If there are no equal rules then the specification is complete.

Program generation is prohibited unless a table is complete.

## *A basis for generating a computer program to implement the logic*

An understanding of this process is directly related to the specifics of the implementation. The generated programs are S/370 callable assembler sub-routines which presume two primary sources of input data and will return in memory the action(s) designated by the decision table logic.

The PC software has a data definition facility to define a mainframe tape record layout and, separately, fields passed by the CALL statement. These are translated into assembler format and are addressable at run-time on the mainframe. Special condition line prefixes (e.g., MAC., CALL., COPY. or DT.) permit system macros (e.g.,, a random-number generator), calling external programs or nesting decision tables.

Comparand data (to the right of the relational operator) is taken directly from the decision table either as a single value, a list of values or from a PC-resident file. A list example is: *STATE EQ NY,NJ,CT* where the comma is interpreted as an *OR*. PC-resident files are a convenient way to store long lists of values (e.g., ZIP codes).

The generated program compares the input data to the comparand(s) and builds a boolean transaction vector with one bit assigned to each condition. Ignoring blank rows, if there are ten conditions, then there are ten bits in the transaction vector. This results in each condition being evaluated once and only once for each CALL.

The actions may take one of three forms: the name of an external program, a character string or a signed integer. Accordingly, the external program address(es), the character string(s) or the integer(s) are returned in a work area to the calling program.

The process of determining which rule satisfies the input data is called rule masking. It consists of constructing two boolean matrices: a mask matrix (*M*) and a condition matrix (*C*). Both are constructed using the character configuration (*B*) in the upper right hand quadrant of the table (blanks are ignored):

```
M←B∊'YN'    and    C←B='Y'
```

The transaction vector is *AND*ed with a column in the mask matrix and the result compared to the corresponding column in the condition matrix. A rule is satisfied when the latter compare is equal. If *V* is the transaction (column) vector and *Z* is the rule number satisfying the conditions then, in APL the process is:

```
X←(¯1↑⍴B)/V ⋄ M←B∊'YN' ⋄ C←B='Y' ⋄ Z←(∧⌿X∧M)=C)/⍳¯1↑⍴B
```

Having resolved the rule number, simple indexing locates the action(s).

Using the table of Figure 4 by way of an example:

```
B is: YYYYN       M is:  11111        and C is:  11110
      $$YN-              00110                   00100
      $$-⋆-              00000                   00000
      $YN⋆-              01100                   01000
      YN⋆⋆-              11000                   10000
```

If we assume an input record to represent a *FEMALE* of *AGE* 62, then the boolean transaction vector *V* is:

```
  1  (GENDER EQ FEMALE  is true)
  1  (AGE    LE 999     is true)
  1  (AGE    LE  65     is true)
  0  (AGE    LE  49     is false)
  0  (AGE    LE  17     is false)
```

```
X is: 11111     X∧M is: 11111    (X∧M)=C is: 11110
      11111             00110                11101
      11111             00000                11111
      00000             00000                10111
      00000             00000                01111
```

In the last matrix, only column 3 is all ones and, therefore, the input data satisfies rule 3 of the table and the actions associated with rule 3 are selected.

After a program is generated it must be up-loaded to the mainframe, assembled and link edited with the calling program. All mainframe data types except floating point are supported, as well as, multi-valued fields (e.g., a 50-byte *INTEREST* field containing 25 values of two bytes each).

## *A basis for performing run-time dynamic optimization*

Rule masking as implemented on the mainframe is a simple AND and compare loop (NC and CLC instructions) that proceeds column by column through the mask and condition matrices until a compare is equal. By tracking rule numbers, the high probability rules can be placed at the beginning of the two matrices, thus, shortening the search path.

## *A basis for providing run-time audit trails*

The transaction vector is the result of evaluating every condition. The run-time address of this vector is returned to the calling program so that it may be saved for future analysis and audit.

## *A basis for generating test data*

The input data formats and the significant data values are present in a decision table specification and, therefore, combinations of the data can be generated on the PC. Additionally, the transaction vector and the action(s) selected can be prepared and included with the test data to compare with the results of mainframe processing.

### *A basis for detecting system integrity failure*

The rule masking process results in one and only one rule that satisfies the decision table logic. If the mask and condition matrices were to be exhaustively searched then multiple rule resolution (or no resolution) would be indicative that something had gone wrong, probably a memory overwrite by an errant part of the system.

### *A basis for organizing parallel processing of program logic*

Given an appropriate set of processors, the evaluation of the conditions can take place in parallel as can the rule masking column evaluations. This should result in very fast resolution of complex logical systems. Note that the system integrity detection (above) is almost free in this environment.

### *A basis for encouraging forward and back tracking analysis*

Decision tables are intrinsically rule based and run-time software can count the rules that are fired and the actions that are selected. This may be fertile ground for artificial intelligence applications.

## Summary

```
ZIP EQ 02100~02299        ‖ N   Y *   Y Y Y Y Y Y   * * * * * *
ZIP EQ LST.CHICAGO        ‖ N   * Y   * * * * * *   Y Y Y Y Y Y
                          ‖
DT.CHILDREN EQ 1          ‖ -   - -   Y Y Y N N N   Y Y Y N N N
                          ‖
NUMDOGS LE 9              ‖ -   N N   $ $ Y $ $ Y   $ $ Y $ $ Y
NUMDOGS LE 1              ‖ -   * *   $ Y N $ Y N   $ Y N $ Y N
NUMDOGS LE 0              ‖ -   * *   Y N * Y N *   Y N * Y N *
--------------------------╫--------------------------------------
¯1  ERROR                 ‖ .   X X   . . . . . .   . . . . . .
 0  BYPASS RECORD         ‖ X   . .   X . . X . .   X . . X . .
 1  KIDS, 1 DOG           ‖ .   . .   . X . . . .   . X . . . .
 2  KIDS, 2 TO 9 DOGS     ‖ .   . .   . . X . . .   . . X . . .
 3  NO KIDS, 1 DOG        ‖ .   . .   . . . . X .   . . . . X .
 4  NO KIDS, 2 TO 9 DOGS  ‖ .   . .   . . . . . X   . . . . . X
```

Figure 5

Figure 5 a an example of a decision table that selects one of six actions for households in Boston or Chicago based upon the presence of children and one or more dogs. The "." and blank columns in the rule area are for ease of reading the table.

The actions are integers and are described by comments to the right of each value. Presumably, actions 1, 2, 3 and 4 represent advertising materials.

The Boston ZIP codes are specified as `02100~02299`, where the tilde signifies an inclusive range of numbers. The specification for Chicago ZIP codes, *LST.CHICAGO* refers to a PC-resident file of Chicago ZIP codes.

The specification *DT.CHILDREN* refers to a nested decision table that returns a one if a household includes children. Typically, children's data are contained in several fields (age, gender, etc.) requiring multiple comparisons to determine the presence of children. Using a nested table significantly simplifies the table of Figure 5.

Notice how the rule specifications tend to follow patterns. This is a common occurrence that simplifies table construction by using block copies. For example, after Boston is defined, simply block copy Boston and change the first two rows (a *Y* to an ⋆ and an ⋆ to a *Y*) and you have the specification for Chicago.

It is significant that the above table can be turned into an assembler program by a few key strokes and, therefore, the development time, energy and cost may be spent on the specification and not the programming of the specification. Also, If the input file is in ZIP code sequence, as is normally the case with direct marketers, then the impact of dynamic optimization is spectacular.

## Conclusion

Decision tables were in vogue 35 years ago but they fell into disuse and today they seem to be virtually ignored by mainstream data processing. This is unfortunate because, as I have tried to point out, they are a very practical tool for modern systems engineering. As a matter of fact, the more complex the logic, the more decision tables shine as a simple and cost effective methodology. ■

## References

1. U. W. Pooch, Translation of Decision Tables, ACM Computing Surveys, Volume 6, Number 2, June 1974.
2. E. H. Bietz, et al, A Modern Appraisal of Decision Tables, CODASYL Task Group, 1982 Report.

***Don Latour's** data processing career started in 1957 and has included field engineering, systems engineering, programming, project management, MIS management, software development and consulting. He can be reached via: Don Latour, President, D D L Associates, Inc., 172 DuBois Avenue, Sea Cliff, NY 11579; Telephone: 516-676-8850; E-mail: "latour@ibm.net". Also refer to the D D L Associates website at "http://www.ddlainc.com".*