

Jan Vanthienen and Elke Dries

Decision Tables

Refining the Concept

and a Proposed Standard

Although at first sight the decision table looks almost the same as the decision table that arose years ago in data processing applications, some fundamental changes in content as well as in applications can be noted. The application area has not only been enlarged towards other domains involving procedural decision situations, but the power of the decision table to represent complex decision situations in a compact way, easy to check for completeness and consistency, became more and more emphasized. In particular, a renewed interest in the technique is observed in areas as knowledge acquisition, knowledge representation, verification and validation (V&V).

This generalized practice of the decision table technique, however, has led to a variety of extensions and interpretations of the decision table formalism. The proposed extra features have caused a loss of simplicity and uniformity. The effect has been that the power and applicability of the technique were reduced rather than enlarged. A back to basics approach seems recommended here. Based on numerous experiences with the technique in a variety of application areas, this paper contains a detailed discussion about the requirements and proposed standards to be respected when working with decision tables. More specifically, the following items are elaborated:

- different kinds of tables
- evolution of the decision table technique
- application areas
- a proposed standardization
- tool support.

Intuitive Definition

A decision table is a tabular representation used to describe and analyze procedural decision situations, where the state of a number of conditions determines the execution of a set of actions. Not just any representation, however, but one in which all distinct situations are shown as columns in a table, such that every possible case is included in one and only one column (completeness and exclusivity).

"A decision table is a table, representing the exhaustive set of mutually exclusive conditional expressions, within a predefined problem area."
(Verhelst [11])

The tabular representation of the decision situation is characterized by the separation between conditions and actions, on one hand, and between subjects and entries (condition states or action values), on the other. The decision table thus consists of four quadrants: *condition stub*, *action stub*, *condition entries* and *action entries*. Every table column (decision column) indicates which actions should (or should not) be executed for a specific combination of condition states (possibly containing irrelevant ("-")) if contraction of states occurs). A simple example of a decision table is shown in figure 1.

If each column only contains simple states, the table is called an *expanded* decision table (*canonical form*), in the other case the table is called a *contracted* decision table (*consolidated form*). Condition or action names in the stub can refer to

other tables (subtables). *Factoring* is defined as the process of division into subtables.

When are decision tables useful?

The condition oriented approach of the decision table makes it very useful to represent procedural knowledge, with such advantages as: overview, readability, conciseness, easy checking for correctness, consistency and completeness (Verhelst [11], CODASYL [2], Vanthienen and Dries [9]). However, the advantages of the technique are not always present or recognized to the same extent (see e.g. Vessey and Weber [12] and Subramanian, Nosek, Raghunathan and Kanitkar [8]). Nor is it true that no disadvantages can be found. In this respect, the following considerations are important (figure 1):

- The decision table has to represent a **selection**. Earlier attempts to fit, for instance, iterations in the decision table format (by using several "entry points") did not result in more convenient arrangements than conventional program structures. The decision table must be thought of as a structured one-entry-one-exit component, that can, however, be part of an iteration, selection or sequence on a higher level (nesting).
- The **separation between conditions and actions** has to be clear (at least on the level of the specification, thus before optimization) or easy to reach (for instance through the use of subtables or "bound actions"). When conditions and actions are heavily merged, it will be hard to construct a well-organized decision table, so that other notations (for instance the decision tree or the nested IF-THEN-ELSE structure) can be more appropriate.

When to use decision tables?

1. Problem domain is selection	yes				no
	simple		complex		-
3. Common actions	yes	no	yes	no	-
1. Decision table recommended	x	-	-	-	-
2. Decision table considered	-	x	x	-	-
3. Decision table unsuited	-	-	-	x	x

Figure 1.
An example decision table

- The concise representation of the decision table appears to full advantage when various paths (columns in the contracted table) indicate **common actions**. In that case, the application area of the resulting actions is represented in a clearly structured and compact way, making the decision table a better alternative than the flowchart and the decision tree.

The appropriateness of the decision table technique depends on the fact whether these three conditions are satisfied or not (see e.g. the decision table in figure 1, where all conditions are satisfied). However, as can be seen from the description of the historical evolution of decision tables, the existence of these three requirements does not imply that the application field is small or unimportant.

Evolution of Decision Table Research and Practice

Though the decision table still looks almost the same as in the days of its first developments, some profound changes can be noted (see also earlier descriptions of the stages in the history of decision tables, e.g. CODASYL [2]):

- The **application area** has extended from computer programming towards various other domains with logical complexity. This extension has directed research efforts from the efficient conversion of the table into program code towards the construction process of the table. In recent developments, this enlargement of the application field is illustrated through the use of decision tables in knowledge engineering and validation (cf. infra).

- The **objective** has changed: the emphasis has moved towards the power of the decision table to represent complex decision situations in a simple manner, easy to check for consistency, completeness and correctness.

Figure 2 shows the main points of attention in the evolution of decision table research and practice.

Different Kinds of Tables

Many variations of the decision table concept exist which look similar at first sight. In practice one has to distinguish between some major kinds of tables, with the decision grid chart at one end of the spectrum and the real decision table at the other end.

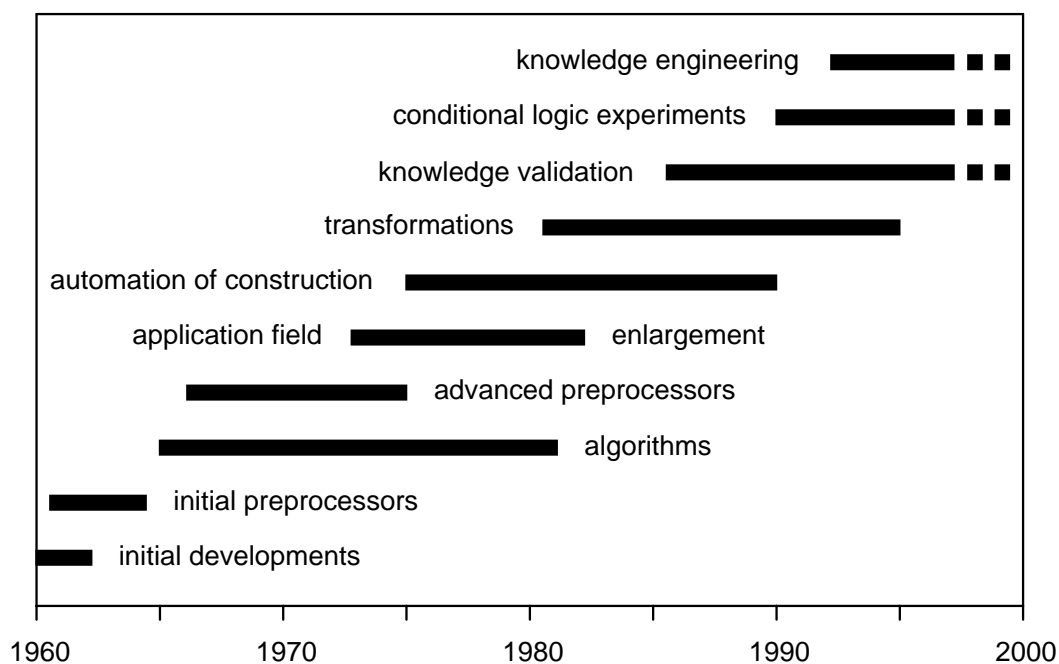


Figure 2. Evolution of the decision table techniques

The most important criterion when distinguishing tables, is the question whether all columns are mutually exclusive (*single hit* versus *multiple hit*). In a single hit table each possible combination of conditions matches exactly one (one and only one) column. This makes an unambiguous use of the table possible.

If the columns are not exclusive, some combination of conditions matches more than one column, which may lead to ambiguity or inconsistency. When consulting the table, the first hit rule will often be used. This rule states that the *first hit* (when scanning the table from left to right) will determine which set of actions has to be executed, thus preventing contradictions. Another possibility is that *all hits* are used to determine the set of actions to be executed. In this case, each hit from left to right can add actions (not mentioned by previous columns) or delete actions (overwriting previous columns) to the set. An interesting concept of this latter form is the so called decision grid chart, a tabular representation of a set of (action oriented) decision rules.

In both multiple hit cases (first hit versus all hits) the same combination of conditions can occur in different columns. As a result the overview over the columns is lost, and with it, the simplicity of inspection. For these reasons we do not consider these tables to be real decision tables, even though they are frequently mentioned in practice.

Based on these considerations, the subdivision in figure 3 is put forward. This grouping, however, does not mean that one form is always better than the others. The decision grid chart (rule base) primarily has a specification function while the expanded single hit decision table has a verification function. When constructing decision tables, we will use both forms in this order, as steps in the process leading to the final contracted table. In this context Maes and Van Dijk [5] discuss the *life cycle* of the decision table, distinguishing between 'construction time', 'test time' and 'interpretation time' decision tables (corresponding to our types MH/A, SH/X and SH/C respectively).

Holidays (multiple hit, all hits)

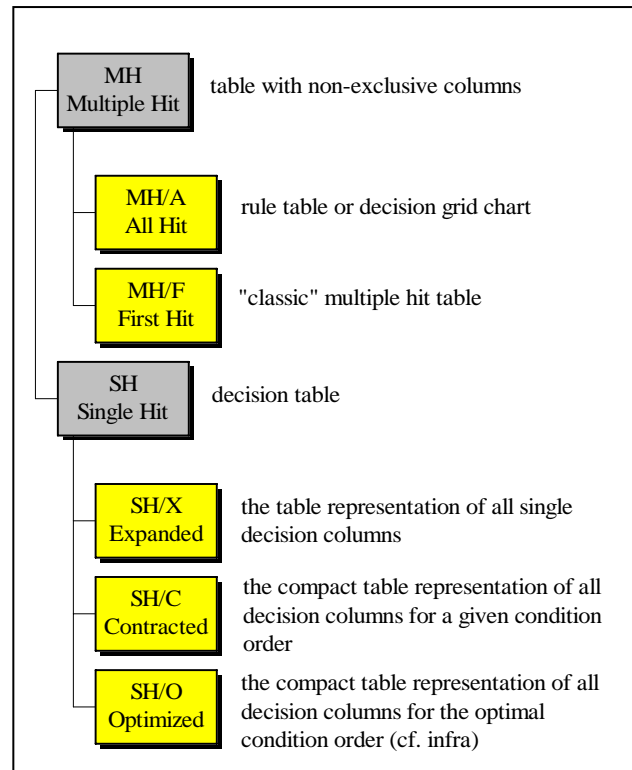


Figure 3. Kinds of tables

An Example: Holidays

To illustrate the different kinds of tables in detail, an example is given with regard to the following small decision problem:

The number of holidays depends on age and years of service. Every employee receives at least 22 days. Additional days are provided according to the following criteria:

- Only employees younger than 18 or at least 60 years, or employees with at least 30 years of service will receive 5 extra days.
- If the employee has at least 15 but less than 30 years of service, 2 extra days are given. These 2 days are also provided for employees of age 45 or more. The 2 extra days can not be combined with the 5 extra days.
- Employees with at least 30 years of service and also employees of age 60 or more, receive 3 extra days, on top of possible additional days already supplied.

(Note the implicit assumption that it is impossible for employees younger than 18 to have 15 or more years of service.)

1. Age	-	<18	>=60	-	18-<60	45-<60	-	>=60
2. Service	-	-	-	>=30	15-<30	<30	>=30	-
1. Assign 22 days	x	-	-	-	-	-	-	-
2. 5 extra days	-	x	x	x	-	-	-	-
3. 2 extra days	-	-	-	-	x	x	-	-
4. 3 extra days	-	-	-	-	-	-	x	x

Holidays (multiple hit, first hit)

1. Age	<18	>=60	-	-	>=45	-
2. Service	-	-	>=30	15-<30	-	-
1. Assign 22 days	x	x	x	x	x	x
2. 5 extra days	x	x	x	-	-	-
3. 2 extra days	-	-	-	x	x	-
4. 3 extra days	-	x	x	-	-	-

Holidays (single hit contracted)

1. Age	<18	18-<45		45-<60		>=60	
2. Service	-	<15	15-<30	>=30	<30	>=30	-
1. Assign 22 days	x	x	x	x	x	x	x
2. 5 extra days	x	-	-	x	-	x	x
3. 2 extra days	-	-	x	-	x	-	-
4. 3 extra days	-	-	-	x	-	x	x

Figure 4. Illustration of different kinds of tables

A Proposed Standardization of Decision Tables

In the past, many variations of the decision table concept have been used, without making a proper distinction between them, thus leading to confusion and reduction of the applicability of the formalism. In order to deal with these problems, a number of constraints to the decision table formalism are proposed in the following paragraphs. These constraints deal with the form as well as with the contents of the table and emanate from the need to use the decision table as a well structured tool across application areas. The purposes served by this proposed standard have been validated by extensive use of the technique in a large number of environments and applications.

Ten commandments on decision table usage are proposed (see figure 5):

Content

- Multi-valued states (extended entry conditions)
- Exclusivity and completeness of the states (domain partitioning)
- Exclusivity and completeness of the columns (single hit tables)
- Predefined actions (refined action entries)

Form

- Optimization (group oriented contraction and row order optimization)
- Tree structures (top-down readability)
- Concise representation (block-oriented notation)
- Indication of impossibilities (contracted impossibilities)

Purpose

- Selection structure (no initialization or repeat actions)
- Subtables (closed subtables)

Figure 5. Commandments on decision table usage

Requirement 1: Multi-valued states (extended entry conditions)

Meaning: The distinction between limited, mixed or extended entry tables is outdated. Whether a condition happens to take several, mutually exclusive states (extended entry) or only two (limited entry) is irrelevant.

Motive: The well known fact that all extended entry tables can be converted to limited entry tables (CODASYL [2]) does not imply that extended entry tables do not have to be considered. First the implementation with limited entries will always result in numerous impossibilities, since the different states should be mutually exclusive. Second this approach neglects the modeling advantages that come with the higher level of the extended entry table (conciseness, manageability, abstraction power, overview).

Requirement 2: Exclusivity and completeness of the states (domain partitioning)

Meaning: The conditions have to obey the requirement of completeness and exclusivity with respect to the states. This means that each possible value of a condition has to be included in one and only one state (Verhelst [11]). To obtain completeness, a final state OTHER may be defined for a condition. This state (called OTHER to prevent confusion with the ELSE column which contains combinations of states for multiple conditions) then embodies all values of the condition that were not mentioned before.

Motive: This requirement is necessary to obtain a complete and exclusive decision table (see requirement 3).

Requirement 3: Exclusivity and completeness of the columns (single hit tables)

Meaning: The decision table has to meet the demand of completeness and exclusivity with respect to the columns. Therefore, each combination of conditions has to be included in one and only one column.

Motive: This requirement, together with requirement 2, leads to exhaustive and exclusive decision tables,

in which each possible case of the problem domain is found in exactly one decision column (expanded or contracted). Only this kind of table allows easy validation and adaptation and unambiguous decision making. The requirement excludes the use of multiple hit tables (with partially overlapping columns), where additional agreements on interpretation (first hit or all hit) inhibit consistency checking and neglect completeness checking (for instance by using the ELSE column).

Requirement 4: Optimization (group oriented contraction and row order optimization)

Meaning: The layout of the decision table can be optimized at several levels, in increasing order of complexity (see also Vanthienen and Wets [10]):

- *Table contraction:* columns or groups of columns that only differ in the state value for one condition and that contain the same actions, are joined as much as possible. This is not only the case if all states of a condition lead to the same action configuration, which renders the condition entry irrelevant ("-"), but also if (groups of) consecutive states with the same action configuration occur. The states may simply be joined by a connector ("OR"). Contraction minimizes the number of columns for a given condition order.
- *Row order optimization:* this determines the condition order which results in the minimum number of (contracted) columns. The condition order is the same for all columns of the decision table. For a table with n conditions, this implies a choice between n! alternative condition orders, some of which might be infeasible because of precedence constraints.
- *Execution time optimization:* this determines the optimal test sequences, taking into account condition test times and column frequencies (if available). If condition test times or column frequencies are not supplied, they are assumed equal for all conditions or columns respectively and the average number of tests is minimized. In the resulting execution tree, conditions are not always tested in the same order anymore.

Motive: Depending on the purpose of the table, layout and execution time optimization will be significant. Optimizing the layout enhances overview and compactness. As far as the decision table itself is concerned, row order optimization finishes the representation. Execution time optimization is an important implementation issue, but leaves the table representation.

In this context of optimization, the link with a related research area, viz. ordered binary-decision diagrams, should be mentioned. A binary-decision diagram (BDD) represents a boolean function as a rooted, directed acyclic graph. For an ordered binary-decision diagram (OBDD), a total ordering over the set of variables is imposed. In essence, an OBDD is a decision table with limited-entry conditions and one action. From the representational point of view, an OBDD does not equal a decision table (restriction to limited entry conditions, restriction to one action, violation of the tree notation, ...). The power of OBDDs, on the other hand, comes from the ability of binary values and boolean operations to represent and implement a wide range of different mathematical domains. OBDDs have extended the range of these problems that can be solved practically. In spite of the fact that decision tables and OBDDs stress different aspects of a problem situation, there are interesting similarities in the optimization issues (e. g. finding the optimal condition order versus the effect of the variable ordering on the form and size of the OBDD). The interested reader is referred to Bryant [1].

Requirement 5: Tree structures (top-down readability)

Meaning: A tree structured table is a table that can be evaluated top-down (stepwise refinement) by continuously choosing from the relevant condition states until a specific column (traditionally called "rule") is reached. In this case the decision table is a straightforward representation of the *decision tree* with all conditions tested in the same order. The tree structure also implies that the condition combinations occur from left to right in

lexicographical order, in other words that the states of the lowest conditions vary first.

Motive: Besides the fact that the tree structure principle eliminates "rule ambiguity" (more than 1 column satisfied), it provides an easy way to consult the decision table and to check for completeness. This principle excludes the use of multiple-hit tables (with overlapping columns), that have to be evaluated from left to right, as well as the use of the ELSE-column.

Requirement 6: Concise representation (block-oriented notation)

Meaning: In order to obtain a maximal resemblance to the decision tree, it is advised that each node (condition entry) of a path is represented only once in the table. Therefore, consecutive equal condition states (repeating themselves on the same row) with the same value for the higher conditions are only displayed once. The following notation is therefore preferred:

>20	<i>instead of</i>	>20 >20
Y N		Y N

Furthermore, the visual interpretation of the tree principle means that a condition entry in the table (a "block") can never be enlarged by a following condition, but can only be split into other condition entries, depending on the relevant states of this last condition (if there are any). Therefore each vertical line, once started, has to continue to the bottom of the table without interruption.

Y	N	<i>instead of</i>	Y	N
Y	N		Y	N
-	-		-	-
-	Y	N	Y	N

Motive: The proposed concise block-oriented representation makes the table easier to read and understand. In addition, by eliminating redundancy, it enhances compactness which is an important objective when using decision tables. Although it may sometimes save place and time to violate the tree-notation (if this causes no ambiguities) by

combining adjacent equal entries *with different parents* starting from some condition (exactly as adjacent branches in a decision tree can be brought together again), this should never be done because of the unfavorable effect on readability.

Requirement 7: Predefined actions (refined action entries)

Meaning: Actions can only take a specific set of predefined values: execute (x), do not execute (-), unknown (.), contradiction (?). After finishing and validating the table (and only then) actions referring to the same subject should be combined.

Motive: This restriction is derived from the need for a simple manipulation and verification of decision tables.

Requirement 8: Subtables (closed subtables)

Meaning: Two types of (possibly recursive) subtables are possible: the *action subtable*, i.e. a further specification of a certain action (comparable to a procedure call in programming), and the *condition subtable*, determining the value of a condition (comparable to a function in programming). All subtables are of the *closed* type (PERFORM binding), which means that after ending a subtable, the calling table regains control.

Motive: Subtables are used to deal with complexity. The PERFORM binding, in contrast to the GOTO binding (where each subtable explicitly has to indicate the paths to follow), is essential to the use of the decision table as a one-entry-one-exit structure element.

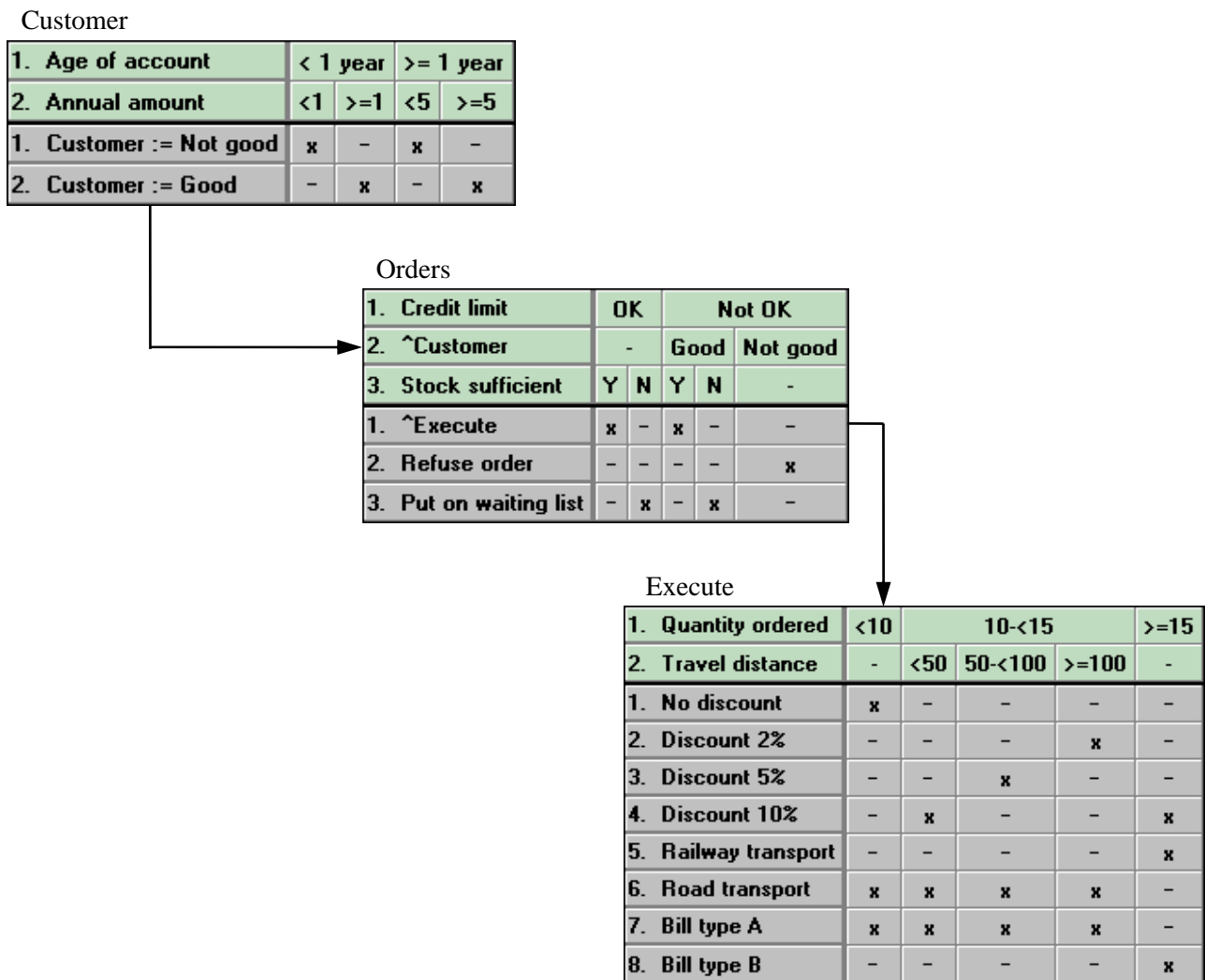


Figure 6. Condition and action subtable

Requirement 9: Selection structure (no initialization or repeat actions)

Meaning: A decision table is the representation of a complex multiple selection. It should therefore not include initialization or iteration facilities in the table itself. These features can be realized using the proper surrounding constructs, procedures, functions or table references. Moreover, as action and condition subtables are considered equivalent to procedures and functions, a table referring to itself is regarded as a recursive table.

Motive: Several exotic features to deal with other than selection structures have caused a loss of simplicity and uniformity of the decision table technique.

Requirement 10: Indication of impossibilities (contracted impossibilities)

Meaning: Some combinations of condition states are impossible because of the nature of the problem, such that the state value of a condition may be implied by state values of higher conditions. In the following, different ways to indicate the existence of impossibilities are discussed, together with their advantages and disadvantages.

- Assign to a *supplementary action*: "impossible". This may be practical for the construction phase, but the impossible combinations result in a final table crowded with columns that do not occur anyway.
- *Deletion* of the impossible columns. This frees the table of the redundant columns, but puts (at least with manual construction) a heavy burden on the demand of completeness, because there is no indication where something was deleted. Moreover, this deletion causes contraction problems because no minimal table width is obtained.
- *Deletion* of the impossible columns *with indication* of the possible (implied) states with !, * or (). This is historically the most common method in literature, but this approach is only suited for "limited entry" conditions. In this case there is an indication that something is impossible, but for more than two states it cannot be indicated

unequivocally what exactly is implied. Moreover, the same contraction problems occur as in the previous option.

- *Contraction* of the impossibilities. In this case, impossibilities are contracted with the neighboring (possibly all other) states. This results in tables of minimal width. Only the naming of the resulting states may sometimes be misleading: it is possible that an irrelevancy is the indication of one or more implied states (cf. the meaning of "-" as "does not have to be tested" versus "should not be tested").
- *Contraction* of the impossibilities *with indication* of the impossible states. The impossibilities are then contracted, but it is indicated which of the states are impossible. However, this way of indicating causes notational overweight, and so the previous option is preferred.

A more formal description

The following formal characterization of a decision table summarizes the above requirements. We assume the existence of a set **C** of conditions and a set **A** of actions.

Let $cnum$ be the number of conditions, then $C = \{C_i, i = 1 \dots cnum\}$. Each condition C_i consists of a condition subject CS_i , a condition domain CD_i and a set of condition states CT_i . CS_i is the name of C_i . CD_i is the set of all possible values condition C_i can take. $CT_i = \{S_{ik}, k = 1 \dots n_i\}$ is an ordered set of n_i condition states S_{ik} ($n_i \geq 2$). A condition state S_{ik} of a condition C_i is a logical expression concerning the elements of CD_i , that determines a subset S_{ik}^* of CD_i such that $CT_i^* = \{S_{ik}^*, k = 1 \dots n_i\}$, the set of all these subsets, constitutes a partition of CD_i . The condition space **CR** is defined as the Cartesian product of the condition state sets: $CR = CT_1 \times CT_2 \times \dots \times CT_{cnum}$. An element of **CR** is called a *condition entry* or a *condition combination*.

Let $anum$ be the number of actions, then $A = \{A_j, j = 1 \dots anum\}$. Each action A_j consists of an action subject AS_j and a set of action values AV_j . AS_j is the name of A_j . Each $AV_j = \{AV_{jl}, l = 1 \dots m_j\}$ is a set of m_j action values. In this text, it is assumed that $\forall j \in \{1 \dots anum\}: AV_j = \{., x, -\}$, with \cdot : *unknown*, x : *execute* and $-$: *do not execute*.

The action space AR is defined as the Cartesian product of the action value sets:

$AR = AV_1 \times AV_2 \times \dots \times AV_{anum}$. An element of AR is called an *action entry* or an *action configuration*.

A decision table is a *function* from the condition space to the action space, by which every condition combination $x \in CR$ is mapped into one (completeness) and only one (exclusivity) action configuration $z \in AR$:

$$DT: CR \rightarrow AR: x \mapsto z = DT(x)$$

Application Fields

Decision tables were originally used as a technique in computer programming. Due to its representational capabilities, its application area has extended later on to several other domains with logical complexity. In this section, three important current application areas are shortly elucidated:

- *software engineering*: e.g. information systems analysis and description of systems requirements; design and programming; test case generation;
- *complex procedural decision situations in general*: e.g. management procedures; checking and visualizing technical, medical and legal specifications;
- *specification and validation of knowledge based systems*: e.g. legal knowledge based systems; help desk applications; verification and validation.

Decision Tables and Software Engineering

Ever since the birth of the decision table, its use in computer programs has been an important point of attention. In the past, numerous compilers, translators and interpreters were developed, allowing this use of decision tables in programs. However, two important aspects were overlooked:

- Almost all the attention went to an efficient processing of the table and few or none to its development.
- It was always assumed that the decision table has to be specified as a table, leading to a variety of syntactical problems (aligning columns, continuation indicators, etc.), especially if extended entries are used.

Automating the construction process of the decision table remedies these difficulties, because then a distinction can be made between the *contents* of the table (the decision logic) and the *representation* (which is not relevant to the computer). It then suffices to enclose the decision logic, e.g. expressed in decision rules, in order to use decision tables in programs, because the construction process of the table can be left to an intelligent editor and the translation into an efficient condition oriented selection is the task of the (pre)compiler. Automation of the decision table construction process thus offers new perspectives for developing intelligent program-editors.

Conditional Logic Representation

The ability of the DT to represent logically complex decision problems in a readable manner is not limited to the world of programming. Examples of other application areas include: medical diagnosis, laws and regulations, management procedures, etc.

Decision tables act as an *intermediate between the specification of a decision process and the real decision making act*, allowing overview and verification throughout the life cycle.

The decision table is **condition oriented** and therefore effective in representing procedural knowledge, with such advantages as: overview, readability, consistency, completeness and correctness. This structured enumeration of decision columns, however, is not the way in which procedural knowledge is acquired or specified. Most texts, procedures, laws, etc. are described in **action oriented**, partial or modular decision specifications, not suited for fast and correct decision making or easy verification. To this end, the decision table construction process allows to transform the action oriented *specification* into a condition oriented *representation*. The decision table being a representation mechanism, (execution) *optimization* is not the main concern. But it may be, once the table has to be converted into an operational system or a manual decision making procedure, e.g. in order to minimize the total test time or the number of questions. The advantage

of the decision table approach, however, is that implementation aspects can be separated from representation aspects through transformation.

The decision table approach, therefore, unifies these three complementary aspects of a decision situation (figure 7): *specification, representation and implementation.*

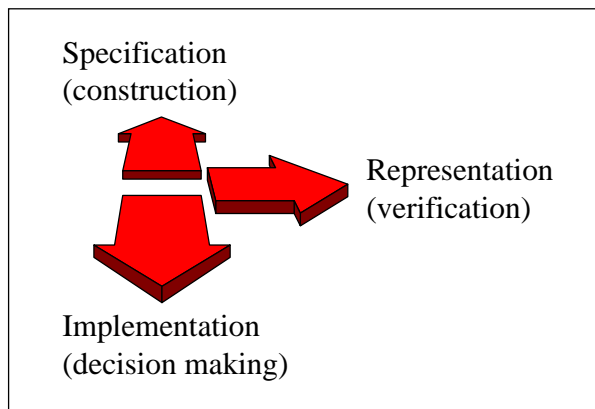


Figure 7. Three aspects of a decision situation

In the past, different experiments have been conducted in order to compare the effectiveness of the decision table as a conditional logic representation tool with other formalisms. Details on these experiments can be found in e.g. Subramanian et al. [8] and Vessey and Weber [12].

Decision Tables and Knowledge Based Systems

In order to develop high quality knowledge based systems, methods and techniques are needed to support different stages in the development life cycle. Three important stages can be distinguished: knowledge acquisition, validation & verification and implementation. It has been recognized in the literature that decision tables can play an important role in each of these stages.

- Verification and Validation

Gathering the correct and complete knowledge is one of the main problems in building knowledge based systems, and usually a lot of contradictions and insufficiencies remain that have to be detected and solved. Also, maintaining the knowledge

base is not a trivial task and often introduces unnoticed inconsistencies or contradictions. Verification and validation of knowledge based systems are receiving increased attention (O'Keefe and O'Leary [6]). As has been reported earlier (e.g. Cragun and Steudel [4]) most of these V&V problems can be solved by the (decision) table technique.

- Implementation

Along with the representational properties of the decision table, its ability to be executed very quickly can be applied to increase the performance of expert systems. In Colomb and Chung [3] a formal equivalence between propositional expert systems and decision tables is proved, and a procedure is given to perform the transformation to decision tables, thereby substantially increasing execution speed.

- Knowledge Acquisition

Instead of building or generating decision tables only during the validation and verification process or as fast way to execute an expert system, they can also be used with significant advantage in the knowledge acquisition phase itself when the domain knowledge is being modeled. The use of decision tables in knowledge acquisition was mentioned in Santos-Gomez and Darnell [7] and Vanthienen and Wets [10].

An integrated framework for these stages

The above mentioned research indicates the occasional and isolated use of schemes, tables or similar techniques in the context of knowledge based systems, however limited to a specific stage of the life cycle and isolated from other stages. The role of the decision table formalism can however be extended, starting from the early stages of knowledge acquisition and structuring, up to and including the final transformation of decision tables into existing knowledge base tools and products, thus covering the full trajectory of the development life cycle. For more information and experiences, see e.g. Vanthienen and Dries [9] and Vanthienen and Wets [10].

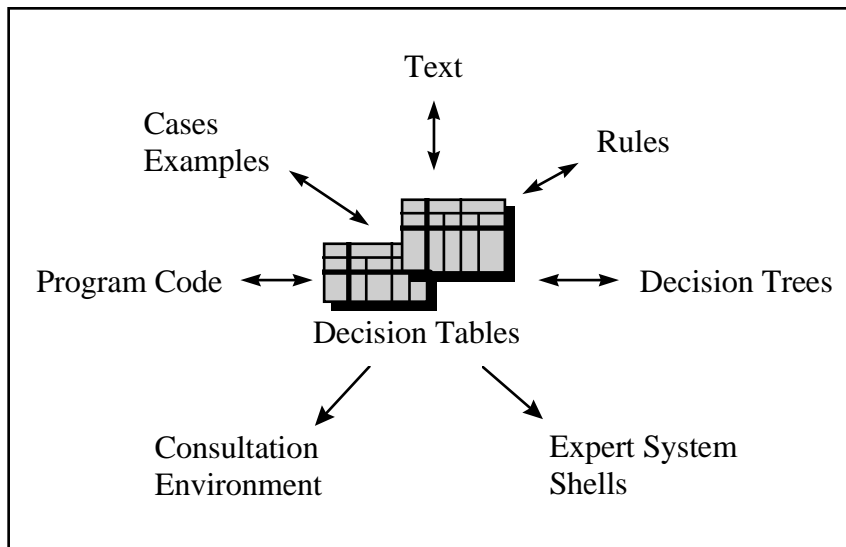


Figure 8. Interfacing features of a decision table workbench

Decision Table Automation and Interfacing

Because manual decision table construction is a cumbersome process, a design tool for computer-supported construction, manipulation, validation and optimization of decision tables proves valuable. Experiences with such a tool, called PROLOGA (PROcedural LOGic Analyzer) (Vanthienen and Dries [9]) indicate its merits in the above mentioned application areas.

Also, the decision table formalism is not an isolated technique and shows a lot of *interfaces* to other representation formalisms such as program code, trees, rules, etc. Making good use of these connections, however, is only possible through flexible computer support. Therefore a variety of bridges have been built between the decision table workbench and other representations, resulting in a large application domain for decision table modeling. Figure 8 just gives an overview of these bridges. Note that most of the bridges are bi-directional.

Conclusion

Over the years there has been a major change in research and application areas of decision tables.

Current decision table applications mainly benefit from the representational advantages of the technique. In the past, different kinds of tables have been used, however, and not all of them offer these representational capabilities. Based on numerous experiences with decision tables, a refined decision table concept and a number of standards have been proposed, in order to allow a more powerful use of the technique.

Acknowledgement

Ideas and viewpoints expressed in this paper should not be attributed exclusively to the authors. They reflect years of working with decision tables at K.U.Leuven and proper credit should therefore be given to current and former research participants. The invaluable contributions of M. Verhelst and R. Maes are highly appreciated. Also F. Robben should not go unrecorded, but naming all other important participants is near to impossible. Any errors or shortcomings, of course, remain the entire responsibility of the authors.

This research was conducted in the context of LIRIS (Leuven Institute for Research on Information Systems) and has been supported in part by the International Business Machines Corporation.

References

1. Bryant, R. E., Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams, *ACM Computing Surveys*, 24(3), 1992, 293-318.
2. CODASYL, A Modern Appraisal of Decision Tables, *Report of the Decision Table Task Group*, ACM, New York, 1982.
3. Colomb, R., Chung, C., Strategies for Building Propositional Expert Systems, *International Journal of Intelligent Systems*, 10, 1995, 295-328.
4. Cragun, B., Steudel, H., A Decision-Table Based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems, *International Journal of Man-Machine Studies*, 1987, 633-648.
5. Maes, R., Van Dijk, J. E. M., On the Role of Ambiguity and Incompleteness in the Design of Decision Tables and Rule-Based Systems, *The Computer Journal*, 31(6), 1988, 481-489.
6. O'Keefe, R. M., O'Leary, D. E., Expert System Verification and Validation: a Survey and Tutorial, *Artificial Intelligence Review*, 7, 1993, 3-42.
7. Santos-Gomez, L., Darnell, M., Empirical Evaluation of Decision Tables for Constructing and Comprehending Expert System Rules, *Knowledge Acquisition*, 4, 1992, 427-444.
8. Subramanian, G. H., Nosek, J., Raghunathan, S. P., Kanitkar, S. S., A Comparison of the Decision Table and Tree, *Communications of the ACM*, 35(1), 1992, 89-94.
9. Vanthienen, J., Dries, E., Illustration of a Decision Table Tool for Specifying and Implementing Knowledge Based Systems, *International Journal on Artificial Intelligence Tools*, 3(2), 1994, 267-288.
10. Vanthienen, J., Wets, G., From Decision Tables to Expert System Shells, *Data & Knowledge Engineering*, 13, 1994, 265-282.
11. Verhelst, M., *De Praktijk van Beslissingstabellen*, Kluwer, Deventer/Antwerpen, 1980, 175 pp.
12. Vessey, I., Weber, R., Structured Tools and Conditional Logic: An Empirical Investigation, *Communications of the ACM*, 29(1), 1986, 48-57.

JAN VANTHIENEN is associate professor of information systems at K.U.Leuven, Department of Applied Economic Sciences. His main research areas are knowledge based systems and expert systems, decision tables, information systems analysis and design. He is a founding member of: Leuven Institute for Research in Information Systems (LIRIS), and Interdisciplinary Center for Law and Informatics (ICRI), both of K.U.Leuven.

Author's Present Address: K.U.Leuven, Dept. of Applied Economic Sciences, Naamsestraat 69, B-3000 Leuven, Belgium.

Email: Jan.Vanthienen@econ.kuleuven.ac.be

ELKE DRIES is research assistant at K.U.Leuven, Department of Applied Economic Sciences, focusing on the area of decision tables and rule based systems.

Author's Present Address: K.U.Leuven, Dept. of Applied Economic Sciences, Naamsestraat 69, B-3000 Leuven, Belgium.

Email: Elke.Dries@econ.kuleuven.ac.be
