IJS delovno poročilo
DP-14761
2024

Marko Bohanec

Transformations of Decision Tables in Qualitative Multi-Criteria Decision Modeling Method DEX

Abstract

We investigate transformations that change the contents of decision tables in multi-criteria decision modeling method DEX (Decision EXpert). A DEX model typically contains a multitude of decision tables whose dimensions (columns and rows) are bound to the context defined by multiple input variables (attributes) and a single output attribute, each of them associated with some discrete (qualitative) value scale. Whenever any of these elements changes while editing a DEX model, the dimensions of the corresponding decision table change, too. Furthermore, the decision table contents, represented in terms of elementary decision rules, might also be affected. The main purpose of the transformations studied here is to *preserve the information content of decision tables* as much as possible. There are three main categories of transformations related to the three main contextual elements: the set of input attributes, the scales of input attributes and the scale of an output attribute. For each category, typical transformations include adding, deleting, duplicating, copying and moving some element, and exchanging and merging two elements.

Keywords

Multi-criteria model, method DEX, decision table, decision rule, decision table transformation.

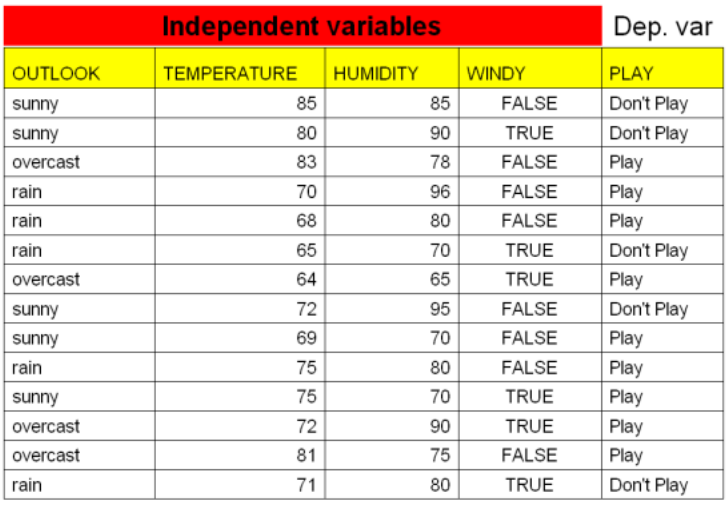
Contents

1 Introduction

A *decision table* is a concise tabular representation for specifying *outcomes* (such as actions, decisions, evaluation results, etc.) depending on given *conditions*. Decision tables are often used to describe and analyze the logic of a complex decision by specifying different conditions and the corresponding decisions or actions to be taken. They are particularly useful in software engineering, business rule management, and other fields where clear and logical decision-making is critical. Decision tables consisting of binary (Boolean) variables are ubiquitous in computer science to represent logical relations in terms of *truth tables*.

Generally, a decision table consists of (see an example in Figure 1):

1. ***Conditions***: Various conditions that can occur in the considered context. Each condition is typically represented by values assigned to one or more *independent variables* (*inputs*).
2. ***Outcomes*** or ***actions***: These are the possible states or values that each condition can take. They are represented by values assigned to one or more *dependent variables* (*outputs*).



Independent variables				Dep. var
OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	FALSE	Don't Play
sunny	80	90	TRUE	Don't Play
overcast	83	78	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	Don't Play

Figure 1: Example: Play Golf *decision table*
(Source: https://en.wikipedia.org/wiki/Decision_table#/media/File:Golf_dataset.png)

Each row represents a ***decision rule*** that assigns the designated output(s) to the situations that fulfil the corresponding conditions. Collectively, a ***decision table*** represents a (possibly partial) *mapping* from inputs to outputs:

$$DT: C_1 \times C_2 \times \cdots \times C_n \to D_1 \times D_2 \times \cdots \times D_d$$

where $C_1 \times C_2 \times \cdots \times C_n$ and $D_1 \times D_2 \times \cdots \times D_d$ are value sets of input (condition) and output (decision) variables, respectively.

In this report we consider a particular type of decision tables that are used in the context of *DEX (Decision EXpert)*. DEX is a qualitative rule based multi criteria decision modelling method (Bohanec, 2022; 2024), aimed at supporting decision makers in complex decision-making tasks. Essentially, the approach consists of developing a hierarchically-structured multi-criteria decision model and using this model to evaluate and analyze decision alternatives. On this basis, the decision maker can assess the decision situation and ultimately rank the alternatives and/or choose the best one. The evaluation (aggregation) of values in a DEX model is governed by user-defined decision tables.

While developing a DEX model, its decision tables are “alive” and subject to various operations that affect their contents. While the *creation* of DEX decision tables from the scratch has been covered extensively and with many examples in the literature (see, for instance, Bohanec (2020; 2022; 2024)), nothing has been published yet about the *transformations* of decision tables that occur due to internal changes of decision model structure: adding, deleting and transforming model variables and modifying value scales of input and output variables. These operations occur frequently while editing the model and may severely damage existing decision tables if not handled correctly. The “correct” implementation should be as helpful and as smooth for the user. At the very least, it should adhere to the following principle as closely as possible:

When transforming a decision table, preserve as much information already contained in the table.

In what follows, the concept of DEX decision tables is defined in more detail in section 2. The process of decision table creation is briefly addressed in section 3, mainly to introduce and illustrate concepts relevant for decision table transformations. Section 4 is the central part of the report, describing transformation methods due to change of the set of input variables (subsection 4.1), change of input variables’ value scales (4.2) and change of the output variable value scale (4.3).

2 Decision tables in DEX

A *DEX model* (Bohanec, 2022) is composed of variables, called *attributes*, and *decision tables*. *Attributes* are structured into a *hierarchy*, that is, a tree or directed acyclic graph. Attributes are “qualitative”: each attribute is associated with a discrete *value scale*, which is typically composed of words, such as ‘low’, ‘medium’, ‘unacceptable’ or ‘good’.

Figure 2 shows a simple example of a DEX model structure aimed at evaluating cars. The involved attributes are called CAR, PRICE, TECH.CHAR., BUY.PRICE, MAINT.PRICE, COMFORT and SAFETY. The structure reflects the decomposition of the problem into sub-problems, for instance, the assessment of CAR to the assessments of PRICE and TECH.CHAR. The evaluation of decision alternatives (cars) is then carried out recursively in the bottom-up direction from terminal attributes (BUY.PRICE, BUY.PRICE, MAINT.PRICE, COMFORT) through internal attributes (PRICE and TECH.CHAR) towards the overall result (CAR).

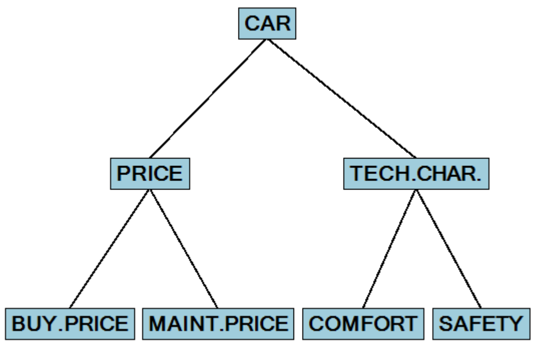


Figure 2: A simple DEX model structure for evaluating cars (Source: https://dex.ijs.si/)

In the remaining part of this report we shall focus only on *individual decision tables* and ignore other aspects of DEX modelling: defining attributes and their scales, structuring models, and representing, evaluating and analyzing decision alternatives – see Bohanec (2020; 2022; 2024) for more information on these topics.

An individual DEX decision table is defined in the context, called *decision space*, which is defined by a single *output attribute* and one or more *input attributes*. The output attribute can be any internal node in the DEX model structure (i.e., CAR, PRICE and TECH.CHAR in Figure 2). For some output attribute (say, CAR) the corresponding input attributes are then its immediate descendants in the model structure (i.e., PRICE and TECH.CHAR.). Thus, the decision space for individual decision table CAR (Figure 3) is defined by the output attribute CAR and input attributes PRICE and TECH.CHAR together with the corresponding value scales.

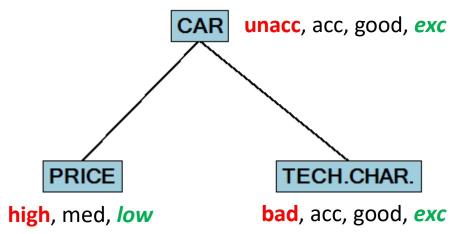


Figure 3: Decision space for the CAR decision table, consisting of an output attribute (CAR), input attributes (PRICE and TECH.CHAR.) and associated value scales.

Figure 3 also shows value scales associated with the three attributes:

value scale of CAR = {**unacc**, acc, good, ***exc***}
value scale of PRICE = {**high**, med, ***low***}
value scale of TECH.CHAR. = {**bad**, acc, good, ***exc***}

As we can see, all scales consist of a small number of discrete values, generally represented by words. Scales can be preferentially ordered or not. All the above scales are ordered in an ascending (increasing) order, so that consecutive values from left to right denote more and more preferred values. By convention in DEX, particularly bad and good values are printed in **bold-red** and ***bold-italic-green***, respectively.

Having defined the example decision space in full, the stage is now set for showing a first example of a DEX decision table. Table 1 displays a decision table defined in the context of decision space from Figure 1. The table consists of four columns and 12 rows. The columns PRICE and TECH.CHAR. correspond to the two input (independent) variables, and the column CAR corresponds to the single output (dependent) variable. Rows in the table are composed of a *conditional part* (corresponding to input attributes) and an *outcome* (corresponding to the output attribute).

Table 1: Decision table CAR, defined in the context of decision space from Figure 3.

	PRICE	TECH.CHAR.	CAR
1	high	bad	unacc
2	high	acc	unacc
3	high	good	unacc
4	high	*exc*	unacc
5	medium	bad	unacc
6	medium	acc	acc
7	medium	good	good
8	medium	*exc*	*exc*
9	*low*	bad	unacc
10	*low*	acc	good
11	*low*	good	*exc*
12	*low*	*exc*	*exc*

In comparison with general decision tables, DEX decision tables are always *bound to the given decision space*. This means that:

- The conditional part of the table *contains all possible combinations* of the values of input attributes. In the above example, since PRICE and TECH.CHAR. can take 3 and 4 values, respectively, there are $3 \times 4 = 12$ rows in the table.
- Consequently, there are *no missing conditions* in a DEX decision table and *all conditions are distinct*.
- There is *only one outcome column* that can hold values from the scale of the output attribute.

While values in the conditional part of the table are always single qualitative values, this is not necessarily so with the outcome values. Normally, as shown in Table 1, they can be single qualitative values, but there are other possibilities: an *unknown* value, an interval or set of values, or a value distribution. These cases are defined in the next section.

Let us take the example in Table 1 to introduce some more terms and concepts associated with DEX decision tables:

- A single table row is often referred to as *an (elementary) decision rule*. Indeed, each row can be read in terms of a simple "**if** <condition> **then** <outcome>" rule. For instance, rule 6 in Table 1 can be interpreted as:

  if PRICE = medium **and** TECH.CHAR. = acc **then** CAR = acc.

- In the Introduction above we mentioned that a decision table represents a mapping from input to output variables. In DEX, this mapping is actually a *function* that maps input attributes to the output one. Furthermore, it resembles many properties of *aggregation functions* (Grabisch, et al., 2009) and *utility functions* as used in the area of Multi-Criteria Decision Analysis (Greco, et al., 2016). Consequently, input attributes are often referred to as *function arguments*.

- Taking the functional interpretation, decision rules then represent *discrete points* in the decision space, as shown in Figure 4. Notice that the function is discrete and defined only in the points shown. Lines drawn between those points serve only as visual aids.

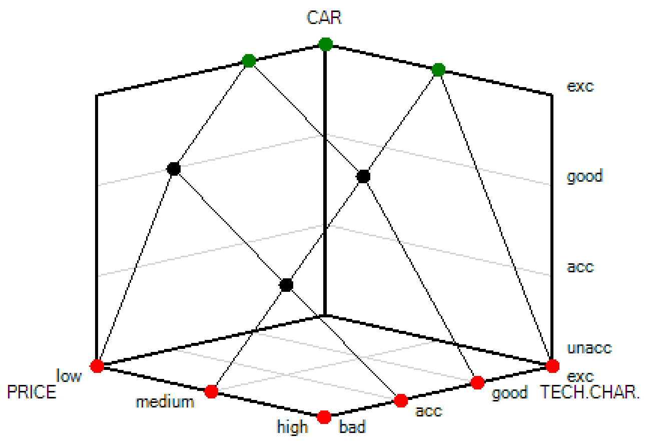


Figure 4: A three-dimensional graphic representation of the CAR decision table. Dots represent individual decision rules.

After getting reasonably familiar with DEX decision tables, let us introduce some formal notation that we shall use hereafter:

- $\boldsymbol{X} = \{x_i, i = 1, \dots, n\}$ denotes the set of *input attributes* of a given decision space. There are n input attributes.
- y denotes the single *output attribute*.

- A *value scale* associated with each input attribute x_i is denoted S_i and composed of $k_i > 0$ discrete values: $S_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,k_i}\}$.
- Similarly, the value scale associated with y is $S_y = \{v_{y,1}, v_{y,2}, \dots, v_{y,k_y}\}, k_y > 0$.
- The size of some scale S is denoted $|S|$. Consequently, $|S_i| = k_i, i = 1, \dots, n$; $|S_y| = k_y$.

A DEX scale can be preferentially ordered or not. We use the notation $\mathrm{ordered}(S)$ to indicate whether scale S is ordered ($\mathrm{ordered}(S) = true$) or not ($\mathrm{ordered}(S) = false$). Actually, the situation in DEX is, due to practical reasons, somewhat more complicated: DEX distinguishes among unordered, increasing (ascending) and decreasing (descending) scales (the latter two interpreted in the left-to-right order). As the consideration of all three possibilities substantially complicates the notation and underlying algorithms, we can take advantage of the fact that the polarity of an ordered scale can be easily changed by reversing the order of scale values. Consequently, we shall restrict further consideration to only two cases, unordered and ascending scales, without loss of generality.

If a scale $S = \{v_1, v_2, \dots, v_k\}$ is ordered, then there is a preference relation '$\succcurlyeq$' (meaning "better than or equally preferred" or "at least as good") defined between consecutive pairs of values: $\forall v_i, v_j \in S, j > i: v_j \succcurlyeq v_i$. No such relation can be assumed with unordered scales; values defined over an unordered scale can only be checked for equality (using the relation '=').

A *decision space* DS, which defines the context of a decision table, is a quadruple, composed of input attributes, an output attribute and associated scales: $DS = \langle \boldsymbol{X}, y, \boldsymbol{S}, S_y \rangle$, where $\boldsymbol{S} = S_1 \times S_2 \times \cdots \times S_n$.

Moving to decision tables, we use the following notation:

- The input space (or domain of the function) is $\boldsymbol{S}$.
- Consequently, input space size is $|\boldsymbol{S}| = k_1 k_2 \dots k_n$.
- For the moment, let us denote the *value space over* y as $V(S_y)$. This value space determines what types of values are admissible as outcomes. There are several possibilities for $V(S_y)$, which are further discussed in section 3.

Considering these definitions, a *decision table* represents the function $F: \boldsymbol{S} \to V(S_y)$. The representation consists of elementary *decision rules* of the form $r_{\boldsymbol{v}} = \boldsymbol{v} \to c, \boldsymbol{v} \in \boldsymbol{S}, c \in V(S_y)$. A DEX decision table R is then composed of rules that correspond to all possible conditions from $\boldsymbol{S}$:

$R = \{r_{\boldsymbol{v}} | \forall \boldsymbol{v} \in \boldsymbol{S}\}$.

The notation $\boldsymbol{v}(r)$ and $y(r)$ is used to denote the constituent parts of some rule $r = \boldsymbol{v} \to c$, so that $\boldsymbol{v}(r) = \boldsymbol{v} \in \boldsymbol{S}$ and $y(r) = c \in V(S_y)$.

An additional bit of information is associated with each decision rule r, denoted $\mathrm{entered}(r)$, which can be either $true$ or $false$. This is because values assigned by the decision maker to outcome cells of a decision table are considered "untouchable" in DEX: they *must not be changed* by any method or algorithm, even if they are evidently wrong (issuing warnings in such cases is admissible, though). On the other hand, rules whose outcomes have not been assigned by the user, are considered "free" and can be modified by DEX, usually according to some general instructions formulated by the user, as discussed in section 3. In displays, such as the example in Table 1, entered rules are denoted by **bold** rule numbers displayed in the leftmost column. Normal typeface is used for rules that have not been entered by the decision maker.

The following notation is used to denote parts of conditional vector space and subset of rules, respectively, that satisfy some Boolean condition $\text{cond}(\boldsymbol{v}), \boldsymbol{v} \in \boldsymbol{S}$ or $\text{cond}(r_v), r_v \in R$:

$$T_{\text{cond}} = \{\boldsymbol{v} \in \boldsymbol{S} | \text{cond}(\boldsymbol{v})\}$$
$$R_{\text{cond}} = \{r_v \in R | \text{cond}(r_v)\}$$

Specifically, we use $E(R)$ (or shortly E when evident from the context) to denote all entered rules in R:

$$E(R) = R_{\text{entered}(r)}$$

Finally, in relation with value scales, we often refer to *ordinal numbers* of scale values rather than (textual) words themselves. Given some scale $S = \{v_1, v_2, \dots, v_k\}$ and considering that $\text{ord}(v_i) = i, i = 1,2, \dots, k$, we represent the scale as $Z = \{\text{ord}(v_1), \text{ord}(v_2), \dots, \text{ord}(v_k)\} = \{1, 2, \dots, k\}$. Such ordinal numbers are interpreted only as labels and are not involved in any numerical computation. With ordered scales we may assume that $j \succcurlyeq i$ for all ordinal values $j > i$. For unordered scales, any comparison of ordinal values other than equality is not admissible.

3 Creation of DEX decision tables

Once some decision space DS within a DEX model has been fully defined, it is possible to create a corresponding decision table R. In DEX, this is typically done by the supporting software that generates all possible combinations of input attributes' values and makes a table consisting of $|S|$ "empty" rules. The decision-maker is then expected to assign output values to "empty" cells positioned in the rightmost column of the table. Usually this is done by assigning a single value $v \in S_y$ to each cell. Applying this principle to the whole table would mean that the value space over y, $V(S_y)$, would be identical to S_y: $V(S_y) \equiv S_y$ However, this is not practical for three reasons: it restricts the handling of not-yet-entered decision rules, it restricts the set of possible outcomes in each cell to a single value and is unsuitable for expressing imprecise and uncertain preferences. Therefore, DEX extends the notion of $V(S_y)$ to other data types, as presented below.

First, let us ask what is an "empty" decision rule? In principle, this is a rule that has not been defined by the user and whose outcome is unknown. It would be inappropriate to assign any value from S_y to such a rule. Therefore, when creating an initial "empty" table, there are two possibilities (Table 2): to mark all yet undefined entries as *undefined*, or to assume that each rule can take any value from S_y. In DEX, "any value" is traditionally denoted by the asterisk '*', meaning either the full interval or full set of values from S_y. The '*' can be interpreted as *unknown*.

Table 2: Two possibilities for initial creation of an "empty" table: using <undefined> (left) or the set of all values '' (right).*

	PRICE	TECH.CHAR.	CAR
1	high	bad	<undefined>
2	high	acc	<undefined>
3	high	good	<undefined>
4	high	*exc*	<undefined>
5	medium	bad	<undefined>
6	medium	acc	<undefined>
7	medium	good	<undefined>
8	medium	*exc*	<undefined>
9	*low*	bad	<undefined>
10	*low*	acc	<undefined>
11	*low*	good	<undefined>
12	*low*	*exc*	<undefined>

	PRICE	TECH.CHAR.	CAR
1	high	bad	*
2	high	acc	*
3	high	good	*
4	high	*exc*	*
5	medium	bad	*
6	medium	acc	*
7	medium	good	*
8	medium	*exc*	*
9	*low*	bad	*
10	*low*	acc	*
11	*low*	good	*
12	*low*	*exc*	*

The above example already indicates three possible candidates for the extension of $V(S_y)$: to value scales that include *undefined*, to *intervals* over S_y and to *value sets* over S_y. Further extensions to *value distributions* over S_y are motivated by the need to represent imprecise and/or uncertain preferences.

Therefore in DEX, the value space $V(S)$ over some scale S can be one of the following (normally, $S = S_y$, but we drop the subscript y to simplify notation):

- *Single value*: $V(S) \equiv S$: Only of theoretical importance, unsuitable and too limiting for practice.
- *Value interval*: $V(S) \equiv I(S) = [v_{\text{low}}, v_{\text{high}}], v_{\text{low}}, v_{\text{high}} \in S, v_{\text{low}} \preccurlyeq v_{\text{high}}$: Represents all values between and including v_{low} and v_{high}. Meaningful only with ordered scales.
- *Value set*: $V(S) \equiv 2^S = \{v | v \in S\}$: Represents some subset of values from S. Meaningful for all discrete scales.
- *Value distribution*: $V(S) \equiv D(S) = \begin{pmatrix} v_1 & v_2 & \cdots & v_k \\ p_1 & p_2 & \cdots & p_k \end{pmatrix}, p_i \in \mathbb{R}^+, i = 1,2, \dots, k$: Associating each value from S with a number $p \in \mathbb{R}^+$.
- *Probability distribution*: $V(S) \equiv P(S) = D(S)$ where $\sum_{i=1}^{k} p_i = 1$: A special case of value distribution where the numbers p_i are interpreted as probabilities.
- *Fuzzy distribution*: $V(S) \equiv F(S) = D(S)$ where $p_i \in [0, 1], i = 1,2, \dots, k$: A special case of value distribution where the numbers p_i are interpreted as fuzzy memberships (possibilities). A *normalized fuzzy distribution* additionally restricts p_i's to $\max_{i=1}^{k} p_i = 1$.
- Value space extended with *undefined*: $V(S) \equiv XW(S) = W(S) \cup \{\text{undefined}\}$, where W denotes one of the above representations.

The support for these representations varies between versions of DEX software (https://dex.ijs.si/). All versions support value intervals. All versions of recent software called *DEXi Suite* (https://dex.ijs.si/dexisuite/dexisuite.html) support *undefined*. *DEXi Suite* software also supports value sets and value distributions, however not in decision tables, but only while evaluating decision alternatives. Full support for value sets and value distributions is foreseen in future implementations. In the remaining part of this report, we shall consider all the mentioned representations and point out the differences between them whenever relevant.

Let us also mention that explicitly considering *undefined* substantially complicates the formal notation. Therefore, we shall exclude *undefined* from further consideration. This can be done without loss of generality considering these simple principles:

- The single outcome *undefined* can be assigned to any decision rule.
- *Undefined* is a special value that cannot be a member of any value interval, set or distribution.
- Whenever a sequence of values is combined (aggregated) together (for instance, with *join* in section 4.1.3), and at least one of them is *undefined*, the final result is *undefined*, too.

After some "empty" decision table R has been created, it is usually populated by assigning (entering) values to decision rules. This is typically done by one or more decision makers that interactively use the DEX software so as to best represent their knowledge and preferences in the given context. The assignments can be done in any order. While doing this, DEX software monitors the decision table for (Bohanec, 2022):

- *Consistency*: Do entered rules violate the *principle of dominance*? Informally, the principle of dominance requires that any rule whose conditions are better than or equal to those of some other rule, should specify an outcome that is better or at least equal to that of the latter rule.

- *Decision table definition/completeness:* To which extent have been the decision table contents defined? Normally, the goal is to enter all rules in the table and, if possible, assign to each of them only a single value from S_y.

DEX software may also modify, in the background, output values of rules that have not been entered yet, according to instructions given by the decision maker. There are two types of instructions (or rule definition “strategies”, as referred to in DEX software documentation):

- *Use scale orders*: Enforcing the principle of dominance on non-entered rules.
- *Use weights*: Determining values of non-entered rules using user-specified weights of attributes and employing a linear approximation function.

	PRICE	TECH.CHAR.	CAR
1	high	bad	unacc
2	high	acc	unacc
3	high	good	unacc
4	high	*exc*	unacc
5	medium	bad	unacc
6	medium	acc	<=good
7	medium	good	good
8	medium	*exc*	>=good
9	*low*	bad	unacc
10	*low*	acc	*
11	*low*	good	*exc*
12	*low*	*exc*	*exc*

	PRICE	TECH.CHAR.	CAR
1	high	bad	unacc
2	high	acc	unacc
3	high	good	unacc
4	high	*exc*	unacc
5	medium	bad	unacc
6	medium	acc	acc
7	medium	good	good
8	medium	*exc*	good
9	*low*	bad	unacc
10	*low*	acc	acc
11	*low*	good	*exc*
12	*low*	*exc*	*exc*

Figure 5: A partially defined decision table with algorithmically assigned values using scale orders (left) and weights (right).

Figure 5 illustrates the two strategies in action. The decision table is only partially defined: the user has defined the outcomes of rules 1, 4, 5, 7, 9, 11 and 12. Employing the principle of dominance (Figure 5, left) generally assigns value intervals to non-entered rules. For example, the interval “<=good” (equivalent to [**unacc**:good]) has been assigned to rule 6 due to two other entered rules: rule 5 whose conditions are worse than 6’s and its outcome is **unacc**, and rule 7 whose conditions are better and the outcome is good. On this basis, we can restrict the value of rule 6 into the [**unacc**:good] interval. Notice that sometimes such intervals are reduced to a single value (rules 2 and 3), while in some other cases (rule 10) the interval cannot be narrowed down to less than ‘*’.

Analogously, Figure 5 (right) shows the results of linear approximation based on weights. In this case, the user specified the weights 40% and 60% for PRICE and TECH.CHAR., respectively, and directed DEX to calculate the non-entered values proportionally to $40 \times \mathrm{ord}(\mathrm{PRICE}) + 60 \times \mathrm{ord}(\mathrm{TECH.CHAR})$. In this way, single values from S_y were assigned to rules 2, 3, 6, 8 and 10.

The principles of decision table creation and decision rules acquisition are covered in more detail in other DEX literature, particularly Bohanec (2020, 2022; 2024). We mentioned them here to explain that the assignment of values to non-entered decision rules happens in DEX in the background while editing the decision table, considering rules actually entered by the user. Furthermore, the values of non-entered rules depend on the used strategy. Therefore, in decision table transformations, presented in the next section, it is important to consider only rules *explicitly defined by the decision maker*. This is the *essential information* that has to be *preserved* in the transformations. Once some table has been transformed, it may be subjected *post-festum* to the same value-defining strategies as illustrated above.

4 Transformations of DEX decision tables

DEX decision tables are typically created, as discussed in the previous section, interactively by using DEX software: the decision maker observes the contents of the table, enters rule values and observes effects of any changes. Furthermore, DEX software monitors the progress and issues messages/warnings interactively whenever necessary. In contrast, the *transformations* discussed in this section occur in the background while the user is making changes to a DEX model, without looking at any table. There are three types of changes that affect components of some decision space $DS = \langle \boldsymbol{X}, y, \boldsymbol{S}, S_y \rangle$ and require transforming one or more underlying decision tables:

1. *Changing the set of input attributes* $\boldsymbol{X}$: deleting an attribute from $\boldsymbol{X}$ (or whole subtree in the model), adding a new attribute to $\boldsymbol{X}$, changing the order of input attributes.
2. *Changing some input attribute's scale* $S \in \boldsymbol{S}$: adding or deleting some value to/from the scale, duplicating or copying a value, merging two values into one, changing the order of values.
3. *Changing the output attribute's scale* S_y: the same operations as above, but considered separately as they cause different effects.

To recap, the main requirement for these transformations is that they *preserve as much existing information defined in the table in terms of already entered decision rules*.

In what follows, we shall use the prime symbol to denote some component after transformation. For instance, $\boldsymbol{X}$ and $\boldsymbol{X}'$ denote the set of input attributes before and after the transformation, respectively. Additionally, we assume that $E \neq \emptyset$. Namely, an empty set of entered rules would indicate that the table does not contain any useful information to preserve. With $E = \emptyset$ we can simply create a new "empty" decision table in the changed decision space.

Table 3: A fully defined decision table with three input attributes used to illustrate decision table transformations.

	A	B	C	Y
1	low	low	low	bad
2	low	low	med	bad
3	low	low	*high*	bad
4	low	med	low	bad
5	low	med	med	acc
6	low	med	*high*	acc
7	low	*high*	low	bad
8	low	*high*	med	acc
9	low	*high*	*high*	acc
10	med	low	low	bad
11	med	low	med	acc
12	med	low	*high*	acc
13	med	med	low	acc
14	med	med	med	acc
15	med	med	*high*	acc
16	med	*high*	low	acc
17	med	*high*	med	acc
18	med	*high*	*high*	*good*
19	*high*	low	low	bad
20	*high*	low	med	acc
21	*high*	low	*high*	acc
22	*high*	med	low	acc
23	*high*	med	med	acc
24	*high*	med	*high*	*good*
25	*high*	*high*	low	acc
26	*high*	*high*	med	*good*
27	*high*	*high*	*high*	*good*

To illustrate the transformations, we shall use an example decision table shown in Table 3. The table is defined upon a decision space consisting of three input attributes A, B and C, and output attribute

Y. All the three inputs have the same value scale {**low**, med, ***high***}, and the output can take the values {**bad**, acc, ***good***}. All value scales are ordered and all decision rules in the table have been entered.

4.1 Transformations due to changes of input attributes

These changes are triggered by changing the set of input attributes $\boldsymbol{X}$, which effectively change the decision space "dimensions" and affect the conditional columns of the decision table. Notice that any change of $\boldsymbol{X}$ is associated with a change of $\boldsymbol{S}$. The decision space changes to $DS' = \langle \boldsymbol{X}', y, \boldsymbol{S}', S_y \rangle$.

4.1.1 Add/Insert input

Given $DS = \langle \boldsymbol{X}, y, \boldsymbol{S}, S_y \rangle$, this change introduces a new attribute x_{n+1}, associated with scale S_{n+1}. Since the order of attributes and scales matter (at least when displaying decision tables) we distinguish between *inserting* a new attribute to some position $q \in [1, n]$ or *adding* it to the end (i.e., effectively an insertion to $q = n + 1$). In the following, we shall only consider adding due to simpler notation. Thus, the decision space changes to $DS' = \langle \boldsymbol{X}', y, \boldsymbol{S}', S_y \rangle$, where

$$\boldsymbol{X}' = \{x_1, x_2, \dots, x_n, x_{n+1}\}$$

$$\boldsymbol{S}' = S_1 \times S_2 \times \dots \times S_n \times S_{n+1}$$

This means that the table size is multiplied by $|S_{n+1}|$. Each rule $r_{\boldsymbol{v}} \in R$ is now reproduced k_{n+1} times, one time for each value from S_{n+1}. Each condition $\boldsymbol{v} \in \boldsymbol{S}$ turns into k_{n+1} conditions of the form $\boldsymbol{v}' = \boldsymbol{v}^{\circ} \circ v_{n+1,i} \in \boldsymbol{S}'$, for $i = 1, \dots, k_{n+1}$. Here, '$\circ$' represents the concatenation of a vector and a scale value.

Now, the key question is how to determine the outcomes in the new decision table? In order to preserve the contents of previously entered rules $r_{\boldsymbol{v}}: \boldsymbol{v} \to c, \forall \boldsymbol{v} \in \boldsymbol{S}, \mathrm{entered}(r_{\boldsymbol{v}})$, the new rules should map to c in all cases that correspond to conditions $\boldsymbol{v}$. Thus, for each $\boldsymbol{v} \ \in \boldsymbol{S}$ and each $v' \in S_{n+1}$, which make the new conditions denoted $\boldsymbol{v}' = \boldsymbol{v} \circ v'$, the new set of entered rules is constructed so that:

$$R' = \{r'_{\boldsymbol{v}'}: \boldsymbol{v}' \to c | r_{\boldsymbol{v}} \in E(R), c(r_{\boldsymbol{v}}) = c, \forall v' \in S_{n+1}, \boldsymbol{v}' = \boldsymbol{v} \circ v'\}$$

Table 4: An example of adding a new attribute D to decision table from Table 3.

	A	B	C	D	AddInput
1	low	low	low	bad	bad
2	low	low	low	acc	bad
3	low	low	low	*good*	bad
4	low	low	med	bad	bad
5	low	low	med	acc	bad
6	low	low	med	*good*	bad
7	low	low	*high*	bad	bad
8	low	low	*high*	acc	bad
9	low	low	*high*	*good*	bad
	=====	=====	=====	=====	=====
73	*high*	*high*	low	bad	acc
74	*high*	*high*	low	acc	acc
75	*high*	*high*	low	*good*	acc
76	*high*	*high*	med	bad	*good*
77	*high*	*high*	med	acc	*good*
78	*high*	*high*	med	*good*	*good*
79	*high*	*high*	*high*	bad	*good*
80	*high*	*high*	*high*	acc	*good*
81	*high*	*high*	*high*	*good*	*good*

Table 4 shows the results of adding a new three-valued attribute D to Table 3 (central rows are removed for brevity). It is easy to observe that each original rule (such as rule 1: <low, low, low> → bad) is reproduced three times, once for each value of D, keeping the same output value.

While this transformation is quite obvious, it is somewhat limited in practice. It effectively adds a new attribute, which completes the table, but has no effect whatsoever on the outcomes. To make it effective, rules must be reviewed and changed explicitly by the user.

4.1.2 Duplicate input

Considering the disadvantages of the above insert/add transformation, there is another way to extend decision tables by one input attribute: duplicating an input attribute. This means taking some attribute $x_s \in \boldsymbol{X}, s \in [1, n]$, making its copy $x' = x_s$ (with $S' = S_s$) and inserting it at some position $q \in [1, n+1], q \neq s$ into $\boldsymbol{X}$. We shall again assume that $q = n + 1$ without loss of generality. In this way, we get the new decision space $DS' = \langle \boldsymbol{X}', y, \boldsymbol{S}', S_y \rangle$, where

$$\boldsymbol{X}' = \{x_1, x_2, \dots, x_n, x'\}$$

$$\boldsymbol{S}' = S_1 \times S_2 \times \dots \times S_n \times S'$$

The only difference in comparison with adding/inserting a new attribute is that S' is now predetermined and equal to S_s; this allows for making different assumptions about the new decision rules.

Again, the table size is multiplied by $|S'|$. Each rule $r_v \in R$ is now reproduced $k' = |S'|$ times, one time for each value from S'. Each condition $\boldsymbol{v} \in \boldsymbol{S}$ turns into k' conditions of the form $\boldsymbol{v}' = \boldsymbol{v} \circ v' \in \boldsymbol{S}'$, for each $v' \in S'$.

Let us consider some condition $\boldsymbol{v} \in \boldsymbol{S}$ and assume it has an entered associated rule $r_v: \boldsymbol{v} \rightarrow c$. Also assume that the s-th element of $\boldsymbol{v}$ is $\boldsymbol{v}_s = w \in S_s$. In order to preserve this information in the new decision table, we require that the new rule $r'_{v'}$ is assigned the same outcome c whenever the new input conditions at positions s and q are equal to w. When they are not, we cannot make any assumptions and leave the corresponding rules non-entered. Formally, the transformed set of entered rules is:

$$R' = \{r'_{v'}: \boldsymbol{v}' \rightarrow c | r_v \in E(R), c(r_v) = c, w = \boldsymbol{v}_s, \boldsymbol{v}' = \boldsymbol{v} \circ w\}$$

Table 5: An example of duplicating attribute C.

	A	B	C	C'	DuplicateInput
1	low	low	low	low	bad
2	low	low	low	med	*
3	low	low	low	*high*	*
4	low	low	med	low	*
5	low	low	med	med	bad
6	low	low	med	*high*	*
7	low	low	*high*	low	*
8	low	low	*high*	med	*
9	low	low	*high*	*high*	bad
	=====================				
73	*high*	*high*	low	low	acc
74	*high*	*high*	low	med	*
75	*high*	*high*	low	*high*	*
76	*high*	*high*	med	low	*
77	*high*	*high*	med	med	*good*
78	*high*	*high*	med	*high*	*
79	*high*	*high*	*high*	low	*
80	*high*	*high*	*high*	med	*
81	*high*	*high*	*high*	*high*	*good*

Table 5 shows results of duplicating attribute C in Table 3. These results clearly differ from Table 4: only rules that have equal values of C and C' are now entered and assigned the same outcome as rules with the same input value in the original table. Combinations of different values of C and C' are not entered in the new table, clearly signaling the user which rules need further attention. This is the main reason why duplicating input attributes is generally preferred to adding/insertion transformations in practice.

4.1.3 Delete input

Deleting and input attribute means removing some attribute $x_q \in \boldsymbol{X}, q \in [1, n]$ from $\boldsymbol{X}$. Consequently, the new decision space becomes $DS' = \langle \boldsymbol{X}', y, \boldsymbol{S}', S_y \rangle$, where

$$\boldsymbol{X}' = \{x_1, \dots, x_{q-1}, x_{q+1} \dots, x_n\}$$

$$\boldsymbol{S}' = S_1 \times \cdots \times S_{q-1} \times S_{q+1} \times S_n$$

In other words, the decision table collapses so that $|\boldsymbol{S}'| = |\boldsymbol{S}|/k_q$. In general, multiple rules from the original table, which were possibly entered and have assigned outcomes, are now represented by single rules in the new table. Specifically, all conditional vectors $\boldsymbol{v} \in \boldsymbol{S}$ turn to $\boldsymbol{v} \setminus q \in \boldsymbol{S}'$, where '$\setminus q$' denotes removal of the q-th vector element. In general, multiple $\boldsymbol{v}$'s map to a single $\boldsymbol{v} \setminus q$.

In order to preserve tabular information, we need to consider all $\boldsymbol{v} \setminus q \in \boldsymbol{S}'$ and find out to which outcomes the corresponding $\boldsymbol{v}$'s map in the original table. In general, these outcomes are multiple and different, and for each $\boldsymbol{v}$ consist of a set of outcomes

$$C(\boldsymbol{v}, q) = \{c(r_{\boldsymbol{v}}) | r_{\boldsymbol{v}} \in E, v_q \in S_q\}$$

Here, v_q denotes the q-th element of $\boldsymbol{v}$.

In order to assign an outcome to a new rule $r'_{\boldsymbol{v} \setminus q}$, the outcomes from $C(\boldsymbol{v}, q)$ need to be combined together, giving a new outcome $c' \in V(S_y)$. Hereafter, we call this operation $join$: given a set of outcomes found in the original table, make a single outcome for the new table:

$$join: \{c \in V(S_y)\} \to V(S_y)$$

Now, it becomes necessary to consider the actual representation of the value space $V(S_y)$; *join* generally depends on it and is different for different representations. Hereafter, we shall consider two cases: using value sets $V(S_y) \equiv 2^{S_y}$ and value distributions $V(S_y) \equiv D(S_y)$. Value-sets results can be approximately converted to value intervals (by taking the worst and best value from 2^{S_y}). Probability and fuzzy distributions are easily derived from a general distribution by normalization of membership numbers $\boldsymbol{p}$ to the sum of 1 (probability) or maximum of 1 (normalized fuzzy), or just mapping them proportionally to the $[0, 1]$ interval (fuzzy).

When $V(S_y)$ represents a set of values from S_y, then *join* makes a union of original assignments, which are also represented by sets in general. For each $\boldsymbol{v} \in \boldsymbol{S}$ and given q:

$$join(C(\boldsymbol{v}, q)) = \bigcup_{c \in C(\boldsymbol{v}, q)} c$$

Notice that whenever the argument of *join* is an empty set, the result is also an empty set. In this case, no assignments are made to the new table and the corresponding rules remain non-entered.

Table 6 shows the results of deleting attribute C from Table 3, considering outcomes as sets. Actually, both set and interval notations are shown in Table 6. Since all three rules in the original table where A=**low** and B=**low** (rules 1–3) map to Y=**bad**, the outcome assigned to rule 1 in the new table is {**bad**}. The next sequence of three original rules 4–6 contain one **bad** and two acc assignments; consequently, the outcome of new rule 2 is {**bad**, acc}. The remaining sets are determined in a similar way.

Table 6: Example of deleting attribute C and considering outcomes as sets. Interval notation is shown in parentheses.

	A	B	DeleteInput
1	low	low	bad
2	low	med	bad,acc (<=acc)
3	low	*high*	bad,acc (<=acc)
4	med	low	bad,acc (<=acc)
5	med	med	acc
6	med	*high*	acc,*good* (>=acc)
7	*high*	low	acc,*good* (>=acc)
8	*high*	med	acc,*good* (>=acc)
9	*high*	*high*	acc.*good* (>=acc)

The second case, where $V(S_y)$ is represents a distribution of values over S_y, differs from the previous one in that it counts the number of times some scale value appears in $C(\boldsymbol{v},q)$. More precisely, it considers real numbers p_i associated with each value $v_i \in S_i$ in the distribution. For single values and each member of a set, we assume $p_i = 1$. Then:

$$join(C(\boldsymbol{v},q)) = \begin{pmatrix} v_{y,1} & v_{y,2} & \dots & v_{y,k} \\ p_1 & p_2 & & p_k \end{pmatrix}$$

so that each $p_i, i = 1,2,\dots k$ is a sum of p_i-s that are in each element of $C(\boldsymbol{v},q)$ associated with value $v_{y,i}$.

The example in Table 7 illustrates the principle. Again, all three original rules 1–3 map to **bad**, so the resulting distribution for the new rule 1 is:

$$\begin{pmatrix} \textbf{bad} & \text{acc} & \textit{good} \\ 3 & 0 & 0 \end{pmatrix}$$

This is shortened to **bad**:3 in Table 7. The next three original rules 4–7 map once to **bad** and twice to acc, yielding **bad**:1,acc:2. The remaining outcomes are obtained in a similar way. Each such value distribution (e.g., **bad**:1,acc:2) can be converted to a probability distribution by normalizing the sum of associated numbers to 1 (**bad**:1/3,acc:2/3) or a normalized fuzzy distribution by changing the numbers proportionally so that their maximum equals 1 (**bad**:1/2,acc:1).

Table 7: Example of deleting attribute C and considering outcomes as value distributions.

	A	B	DeleteInput
1	low	low	bad:3
2	low	med	bad:1,acc:2
3	low	*high*	bad:1,acc:2
4	med	low	bad:1,acc:2
5	med	med	acc:3
6	med	*high*	acc:2,*good*:1
7	*high*	low	bad:1,acc:2
8	*high*	med	acc:2,*good*:1
9	*high*	*high*	acc:1,*good*:2

4.1.4 Exchange and move inputs

The remaining two transformations of inputs are primarily of practical importance. Actually, they do not change any decision rules, but just reshuffle the order of input attributes, affecting the order of columns and rules in the table. *Exchanging* means exchanging positions of two attributes in $\boldsymbol{X}$, and *moving* means repositioning some attribute $x_s \in \boldsymbol{X}$ to a new position $q \in [1,n], q \neq s$ in $\boldsymbol{X}$, shifting positions of other attributes if necessary.

4.2 Transformations due to changes of an input attribute scale

The next set of transformations refers to changing some value scale $S_s \in \boldsymbol{S}, s \in [1, n]$. This includes:

- adding some value to or deleting some value from S_s,
- changing the order of values (copy, move, exchange values), and
- other operations that are similar to adding (duplicating) and deleting (merging), but have different semantics and generally yield different results.

In all cases, the original set of attributes $\boldsymbol{X}$ stays intact. However, the associated scale S_i changes, which means that the structure and contents of the new decision table have to adapt to these changes. The new decision space is $DS' = \langle \boldsymbol{X}, y, \boldsymbol{S}', S_y \rangle$, where

$$\boldsymbol{S}' = S_1 \times S_2 \times \cdots \times S'_s \times \cdots \times S_n$$

4.2.1 Add/Insert input value

In this case, an additional value, say v_{s,k_s+1} is added or inserted at some position in S_s. This gives the new scale S'_s of size $k_s = |S_s| + 1$. Considering the underlying decision table, new entries with the condition $x_s = v_{s,k_s+1}$ have to be added for all existing combinations of other attributes' values. Effectively, this extends the table size from $|\boldsymbol{S}|$ to $|\boldsymbol{S}|\frac{|S_s|+1}{|S_s|}$. All the old decision rules are preserved, and the new rules involving the new value of S_s are left non-entered (Table 8).

Table 8: Example of inserting a new value mh *as the third element of the scale of C.*

	A	B	C	InsertInpValue
1	low	low	low	bad
2	low	low	med	bad
3	low	low	mh	*
4	low	low	*high*	bad
5	low	med	low	bad
6	low	med	med	acc
7	low	med	mh	*
8	low	med	*high*	acc
=================				
29	*high*	med	low	acc
30	*high*	med	med	acc
31	*high*	med	mh	*
32	*high*	med	*high*	*good*
33	*high*	*high*	low	acc
34	*high*	*high*	med	*good*
35	*high*	*high*	mh	*
36	*high*	*high*	*high*	*good*

4.2.2 Duplicate input value

Duplicating an input value, say v, in some scale S_s, is similar to adding a new value, but leads to different transformation rules. Namely, by intending to duplicate some value we effectively require that the resulting decision rules stay the same for both the original input value v and its duplicate v'. Again, the table size is extended from $|\boldsymbol{S}|$ to $|\boldsymbol{S}|\frac{|S_s|+1}{|S_s|}$ and all the old decision rules are preserved. Differently from section 4.2.1, all the new rules involving the new value $v' \in S'_s$ are assigned the same outcome as old rules involving $v \in S_s$. An example with central rules omitted is shown in Table 9.

Table 9: Example of duplicating the value med *of attribute C (calling the duplicate* med2*).*

	A	B	C	DuplicateInpValue

1	low	low	low	bad
2	low	low	med	bad
3	low	low	med2	bad
4	low	low	*high*	bad
5	low	med	low	bad
6	low	med	med	acc
7	low	med	med2	acc
8	low	med	*high*	acc
	=====	=====	=====	=====
29	*high*	med	low	acc
30	*high*	med	med	acc
31	*high*	med	med2	acc
32	*high*	med	*high*	*good*
33	*high*	*high*	low	acc
34	*high*	*high*	med	*good*
35	*high*	*high*	med2	*good*
36	*high*	*high*	*high*	*good*

4.2.3 Delete input value

Deleting an input value $v \in S_s$, where $S_s \in \boldsymbol{S}$ and $|S_s| > 1$, is a straightforward transformation in which all decision rules for which $x_s = v$ are removed from the table. The table size is reduced from $|\boldsymbol{S}|$ to $|\boldsymbol{S}|\frac{|S_s|-1}{|S_s|}$. Formally, $R' = R_{x_s \neq v}$. See the example in Table 10. Let us also remark that deleting a single element from a single-element scale is not admissible, as it violates assumptions about decision spaces (|S|>0 for each scale). In practice, such operations may still be supported in software at the cost of substantial conceptual difficulties and are not addressed in this report.

Table 10: Example of deleting the value med *from the scale of C.*

	A	B	C	DeleteInpValue
1	low	low	low	bad
2	low	low	*high*	bad
3	low	med	low	bad
4	low	med	*high*	acc
5	low	*high*	low	bad
6	low	*high*	*high*	acc
7	med	low	low	bad
8	med	low	*high*	acc
9	med	med	low	acc
10	med	med	*high*	acc
11	med	*high*	low	acc
12	med	*high*	*high*	*good*
13	*high*	low	low	bad
14	*high*	low	*high*	acc
15	*high*	med	low	acc
16	*high*	med	*high*	*good*
17	*high*	*high*	low	acc
18	*high*	*high*	*high*	*good*

4.2.4 Merge input values

Merging is a transformation that turns two values $v_a, v_b \in S_s$ into a new single value v_{ab} of the transformed scale S'_s. With ordered scales, v_a, v_b are usually adjacent values, but this is not required (although may violate the principle of dominance). This transformation is similar to deletion (as we "lose" one scale value along the way), however it makes different requirements to the newly generated decision rules: each new rule referring to the value v_{ab} must combine the outcomes of two original rules that refer to v_a and v_b, respectively. Similarly as in section 4.1.3, we have to *join* a set consisting of at most two outcomes (there may be less due to non-entered rules referring to v_a and v_b). The same type of *join* operations as in section 4.1.3 are applicable. Table 14 shows the example of merging the values med and ***high*** and displays outcomes in terms of general value distributions. These are all easily transferable to intervals, sets, and probability and fuzzy distributions.

Table 11: Example of merging values med *and* ***high*** *of C to a new value* med+high*, showing value distributions.*

	A	B	C	MergeInpValue
1	low	low	low	bad:1
2	low	low	med+high	bad:2
3	low	med	low	bad:1
4	low	med	med+high	acc:2
5	low	*high*	low	bad:1
6	low	*high*	med+high	acc:2
7	med	low	low	bad:1
8	med	low	med+high	acc:2
9	med	med	low	acc:1
10	med	med	med+high	acc:2
11	med	*high*	low	acc:1
12	med	*high*	med+high	acc:1,*good*:1
13	*high*	low	low	bad:1
14	*high*	low	med+high	acc:2
15	*high*	med	low	acc:1
16	*high*	med	med+high	acc:1,*good*:1
17	*high*	*high*	low	acc:1
18	*high*	*high*	med+high	*good*:2

4.2.5 Copy input value

Copying an input value $v \in S_s$ to some position $q \in [1, k_s]$ effectively means the following: make decision table behave the same for rules $R_{v_s=v}$ and $R_{v_q=v}$. That is, to each rule whose condition contains $v_q = v$, assign the same outcome as the outcome of original rule where $v_v = v$, while all the other conditions are the same. Also, leave all the remaining rules intact. In this case, the size of decision table does not change. Affected are only rules containing the condition $v_q = v$, as shown in the example in Table 12. There, the most revealing contents is around the rules 22–24, particularly rule 24. The outcomes of rules 22 and 23 (acc) are intact with respect to Table 3. However, the original outcome of original rule 24 (***good***) was replaced by acc as a result of copying the outcome from rule 23.

Table 12: Example of copying the value med *of C to the third position (previously* ***high****, now* ***med****).*

	A	B	C	CopyInpValue
1	low	low	low	bad
2	low	low	med	bad
3	low	low	*med*	bad
4	low	med	low	bad
5	low	med	med	acc
6	low	med	*med*	acc
	================			
22	*high*	med	low	acc
23	*high*	med	med	acc
24	*high*	med	*med*	acc
25	*high*	*high*	low	acc
26	*high*	*high*	med	*good*
27	*high*	*high*	*med*	*good*

4.2.6 Move input value

Moving an input value is another transformation that does not change the dimensions of decision table, but just reshuffles decision rules within the table. The transformation occurs due to moving some value $v \in S_s$ from its original position in the scale to some other position, shifting other values if necessary. This repositioning is also expected to affect the underlying decision table so that outcomes that were associated with the original v's position move together with v moving to a new position.

Again, let us consider rows 22–24 in Table 13. Moving the value med from the third to second place in the scale of C effectively exchanged the outcomes of rules 24 as 23. In this way, the original value of rule 24 ***good*** was assigned to rule 23, and the original value acc of rule 23 was assigned to rule 24. Actually, a similar transformation was applied to other pairs, such as 2 and 3, 5 and 6, 26 and 27 and others, not shown in Table 13.

Table 13: Example of moving the value med of C from the second to third position.

	A	B	C	MoveInpValue
1	low	low	low	bad
2	low	low	*high*	bad
3	low	low	*med*	bad
4	low	med	low	bad
5	low	med	*high*	acc
6	low	med	*med*	acc
	=================			
22	*high*	med	low	acc
23	*high*	med	*high*	*good*
24	*high*	med	*med*	acc
25	*high*	*high*	low	acc
26	*high*	*high*	*high*	*good*
27	*high*	*high*	*med*	*good*

Here, we should remark that moving values of ordered value scales is "tricky" and generally not advised. In most cases, it violates the principle of dominance and requires additional recovery editing from the user.

4.2.7 Exchange input values

This is another "tricky" operation, similar to moving an input value. Here, two values of the same scale S_s exchange positions and "drag" the original outcome assignments with them. The transformation is evident from the example in Table 14. Again, this transformation often violates the principle of dominance.

Table 14: Example of exchanging the values low *and* high *of attribute C.*

	A	B	C	ExchangeInpValues
1	low	low	high	bad
2	low	low	med	bad
3	low	low	*low*	bad
4	low	med	high	acc
5	low	med	med	acc
6	low	med	*low*	bad
	=================			
22	*high*	med	high	*good*
23	*high*	med	med	acc
24	*high*	med	*low*	acc
25	*high*	*high*	high	*good*
26	*high*	*high*	med	*good*
27	*high*	*high*	*low*	acc

4.3 Transformations due to changes of output attribute scale

The final set of transformations addresses changes of the output attribute y and the associated scale S_y. They are generally similar to those of input attribute scales in section 4.2 (adding, inserting, duplication, etc.), however they affect the underlying decision tables differently. The new decision space is $DS' = \langle \boldsymbol{X}, y', \boldsymbol{S}, S'_y \rangle$. This indicates that the basic dimensions, consisting of attributes $\boldsymbol{X}$ and their scales $\boldsymbol{S}$ do not change. Consequently, the original and transformed decision tables have exactly the same number of rules and only their outcomes are generally changed.

4.3.1 Add/Insert output value

Adding (to the end) or inserting (at some position) a new output value $v' \in S'_y$ does not change the decision table size or structure. Furthermore, all the rules can remain intact, mapping only to old values from S_y. The only disadvantage of this transformation is that the new decision rules never map to v' and require manual post-transformation editing to achieve that.

4.3.2 Delete output value

Deleting an output value $v \in S_y$ means that v has to be removed from all outcomes already existing in R. This is particularly tricky with rules that map to exactly that value: $r: \boldsymbol{v} \rightarrow v$. In this case, it is not possible to preserve the source information; the new rule is assigned the *unknown* or '*' outcome and marked as non-entered. In all other cases it is possible to remove v from the outcome and assign a stripped-down outcome to the new rule.

Let us illustrate this principle using Table 15, where the value acc has been removed from outcomes. The original outcomes other than acc have been retained. However, all acc outcomes have been converted to '*'.

Table 15: Example of deleting value acc *from outcomes.*

	A	B	C	DeleteOutValue
1	low	low	low	bad
2	low	low	med	bad
3	low	low	*high*	bad
4	low	med	low	bad
5	low	med	med	*
6	low	med	*high*	*
7	low	*high*	low	bad
8	low	*high*	med	*
9	low	*high*	*high*	*
	=================			
21	*high*	low	*high*	*
22	*high*	med	low	*
23	*high*	med	med	*
24	*high*	med	*high*	*good*
25	*high*	*high*	low	*
26	*high*	*high*	med	*good*
27	*high*	*high*	*high*	*good*

Since the above example lacks variety, let us mention that:

- Removing a value from an *interval* restricts its bounds or internal contents. Considering the interval **bad**:***good***, removing **bad** yields acc:***good***, removing acc yields **bad**:***good*** (without acc in the middle, because it is no longer there) and removing ***good*** yields **bad**:acc.
- Removing a value from a *set* is trivial, but may render an empty set.
- Removing a value from a *distribution* is equivalent to setting the corresponding value p to 0.

4.3.3 Duplicate output value

Duplicating an output value means taking some value $v \in S_y$ and inserting its copy v' to S_y, typically at the position right after v. Then, whenever the underlying decision rules map to v, they should map to v' as well. In other words, if an outcome contains v, it should contain v', too. The example in Table 16 shows that duplicating the value acc to acc2 turns all acc outcomes to the interval acc:acc2 (or equivalent set {acc, acc2}).

Table 16: Example of duplicating the output value acc *and calling the duplicate* acc2*, using interval notation.*

	A	B	C	DuplicateOutValue
1	low	low	low	bad
2	low	low	med	bad
3	low	low	*high*	bad
4	low	med	low	bad
5	low	med	med	acc:acc2
6	low	med	*high*	acc:acc2
7	low	*high*	low	bad
8	low	*high*	med	acc:acc2
9	low	*high*	*high*	acc:acc2
	=====================			
22	*high*	med	low	acc:acc2
23	*high*	med	med	acc:acc2
24	*high*	med	*high*	*good*
25	*high*	*high*	low	acc:acc2
26	*high*	*high*	med	*good*
27	*high*	*high*	*high*	*good*

4.3.4 Copy output value

Copying an output value $v_s \in S_y$ to some other output value $v_d \in S_y$ essentially just renames v_d to v_s, while still considering the two values distinct. Table 17 illustrates that the old output ***good*** has been renamed to acc, still leaving two distinct values: acc (previous) and ***acc*** (newly renamed from ***good***). While not really useful by itself, this transformation is needed for consistency with copying an input value (section 4.2.5), which typically occurs simultaneously at one level higher in the model, where the current output attribute y plays the role of some input attribute in another decision space.

Table 17: Example of copying the output value acc *to the third position, replacing **good** with **acc**.*

	A	B	C	CopyOutValue
1	low	low	low	bad
2	low	low	med	bad
3	low	low	*high*	bad
4	low	med	low	bad
5	low	med	med	acc
6	low	med	*high*	acc
7	low	*high*	low	bad
8	low	*high*	med	acc
9	low	*high*	*high*	acc
	=================			
22	*high*	med	low	acc
23	*high*	med	med	acc
24	*high*	med	*high*	*acc*
25	*high*	*high*	low	acc
26	*high*	*high*	med	*acc*
27	*high*	*high*	*high*	*acc*

4.3.5 Move output value

Moving an output value $v \in S_y$ means placing it in some other position in S_y, shifting other values if necessary. In the underlying decision table this means that all outcome values are modified so as to reflect this change of position.

The example in Table 18 shows what happens when moving output value acc from position 2 to position 3, effectively exchanging the values acc and ***good***. While retaining their names, the new ***acc*** became preferentially better than ***good***. This is exemplified in Table 18 by showing output value ordinal numbers in parentheses; the ordinal numbers of the new ***acc*** and ***good*** are 3 and 2, respectively. Unfortunately, this transformation generally results in rules violating the principle of dominance.

Table 18: Example of moving output value acc *to position 3. Output value ordinal numbers are shown in parentheses.*

	A	B	C	MoveOutValue
1	low	low	low	bad (1)
2	low	low	med	bad (1)
3	low	low	*high*	bad (1)
4	low	med	low	bad (1)
5	low	med	med	*acc* (3)
6	low	med	*high*	*acc* (3)
7	low	*high*	low	bad (1)
8	low	*high*	med	*acc* (3)
9	low	*high*	*high*	*acc* (3)
	====	====	====	====
22	*high*	med	low	*acc* (3)
23	*high*	med	med	*acc* (3)
24	*high*	med	*high*	*good* (2)
25	*high*	*high*	low	*acc* (3)
26	*high*	*high*	med	*good* (2)
27	*high*	*high*	*high*	*good* (2)

4.3.6 Exchange output values

Similarly as in section 4.2.7, exchanging two output values means exchanging positions of two values in S_y. This transformation is carried out similarly as moving output values (section 4.3.5): in all outcomes, the former value is replaced with the latter and vice versa.

Here, two values of the same scale S_s exchange positions and "drag" the original outcome assignments with them. The transformation is evident from the example in Table 14. Again, this transformation often violates the principle of dominance.

Table 19: Example of exchanging the output values **bad** *and* ***good****. Output value ordinal numbers are shown in parentheses.*

	A	B	C	ExchangeOutValue
1	low	low	low	*bad* (3)
2	low	low	med	*bad* (3)
3	low	low	*high*	*bad* (3)
4	low	med	low	*bad* (3)
5	low	med	med	acc (2)
6	low	med	*high*	acc (2)
7	low	*high*	low	*bad* (3)
8	low	*high*	med	acc (2)
9	low	*high*	*high*	acc (2)
	====	====	====	====
22	*high*	med	low	acc (2)
23	*high*	med	med	acc (2)
24	*high*	med	*high*	good (1)
25	*high*	*high*	low	acc (2)
26	*high*	*high*	med	good (1)
27	*high*	*high*	*high*	good (1)

4.3.7 Merge output values

Similarly as merging input values (section 4.2.4), this transformation turns two values $v_a, v_b \in S_y$ into a new single value v_{ab} of the transformed scale S'_y. In this case, all references to either v_a or v_b in the outcomes of entered decision rules are transformed to v_{ab}.

Table 20 illustrates merging the output values acc and ***good*** to a new value named ***good+acc***. Notice that all rules that originally contained either one of the former values now yield ***good+acc***.

Table 20: Example of merging output values acc *and* ***good*** *to a single value called* ***good+acc****.*

	A	B	C	MergeOutValue
1	low	low	low	bad

```
 2  low   low   med   bad
 3  low   low   high  bad
 4  low   med   low   bad
 5  low   med   med   good+acc
 6  low   med   high  good+acc
 7  low   high  low   bad
 8  low   high  med   good+acc
 9  low   high  high  good+acc
    =====================
22  high  med   low   good+acc
23  high  med   med   good+acc
24  high  med   high  good+acc
25  high  high  low   good+acc
26  high  high  med   good+acc
27  high  high  high  good+acc
```

4.4 Other changes

There are some other DEX-model editing operations that may affect the underlying decision tables, but have not been considered above. Most notably:

- *Changing scale orders* from ordered to unordered and vice versa. These operations do not change decision table dimensions, but often substantially affect their "correctness" in terms of logical consistency and adhering to the principle of dominance. It is hard to perform any meaningful automatic transformation in such situations. DEX typically allows such operations (silently or issuing warnings) and leaves the task of clearing out the table to the user.
- *Assigning a different scale to an attribute*: Generally, this operation is admissible only if the new scale has the same number of values as the original one. In this case, the table structure does not change; checking a possible change of the table semantics is left to the user. Any assignment of a scale with a different number of values is disallowed and should be carried out by a sequence of other operations (e.g. first delete the old attribute and then add the attribute associated with the new scale).
- *Operations involving attributes with undefined scales*, for adding such an attribute as an input to an existing table or deleting the scale of some input or output attribute. Generally, such operations are destructive and thus disallowed in DEX, but if explicitly requested, they result in a deletion of the underlying table.

5 Conclusion

The transformations discussed in this report are driven by two key requirements outlined in the DEX method:

- Decision tables are *bound to their respective decision spaces* at all times and must adapt automatically to any changes.
- While adapting to changes, the *information content* of a table must be *preserved* as much as possible.

These requirements evolved gradually with software supporting the DEX method. For instance, the old *DEXi Classic* software (https://dex.ijs.si/dexiclassic/dexiclassic.html) supported only changes of input and output scales to some extent. Whenever adding or deleting an input attribute, the affected decision table was simply deleted; this turned out to be too restrictive. The new *DEXi Suite* software (https://dex.ijs.si/dexisuite/dexisuite.html) implements all the discussed transformations, however it is still restricted to using only value intervals as decision table outcomes. A full implementation for all types of output value spaces (section 3) is being developed in software called *DEX_Library*, which has been used to generate examples presented in section 4.

Why are there so many transformations? Why do we not consider just inserting and deleting elements? For instance, why do we need duplicating and copying, as they just add a new element (new attribute, new scale value) in the given dimension? Why merging as we already have deleting? The answer is that these transformations have different semantics and affect decision rules differently. For example, when adding a new input attribute to a decision table, we do not know anything about that attribute and can hardly do anything more than just adapting table dimensions. In contrast, when duplicating an attribute, we can consider everything that we already know about that attribute and create more informed outcomes in the transformed decision table.

A few transformations are trivial, but most of them are not. Many require additional methods, such as *join* to combine multiple outcomes represented in a variety of output spaces: intervals, sets and value distributions. Actually, operations upon such representations were addressed only informally in this report. However, they are an interesting topic in their own right and warrant a more formal treatment. Also, we deliberately avoided some issues that substantially complicate practical implementation, for instance considering undefined/empty scales, using the value *undefined*, and using both scale orderings, ascending and descending.

We also deliberately excluded related work from this report. Although we did consult related literature, we were not satisfied with the results and wish to look further. We were really surprised to find out that the literature about transformation of decision tables was so scarce. Most authors take decision tables as "granted", using them for whatever purpose, but without modification. There are only a few studies addressing decision table transformations, most of which date back to before 2000. This remains a task for the future. Also, despite using a pretty standard notation for decision tables (c.f. Greco, et al., 2001; 2016), we feel there is a need for a more "fit-for-purpose" notation to express concepts related with decision tables and their transformations. In this report, we occasionally found it easier to explain the concepts vaguely in text; we wish to develop a more appropriate formal notation also for these cases. Last but not least, we wish to provide an open-source implementation of *DEX_Library*, contributing an algorithmic approach and supporting further applications in the field.

6 References

Bohanec, M.: *DEXi: Program for Multi-Attribute Decision Making, User's Manual, Version 5.04*. IJS Report DP-13100, Jožef Stefan Institute, Ljubljana, 2020.

Bohanec, M.: DEX (Decision EXpert) (2022): A qualitative hierarchical multi-criteria method. *Multiple Criteria Decision Making* (ed. Kulkarni, A.J.), Studies in Systems, Decision and Control 407, Singapore: Springer, doi: 10.1007/978-981-16-7414-3_3, 39-78.

Bohanec, M.: *Introduction to Decision Modeling Method DEX (Decision EXpert)*. Institut Jožef Stefan, Delovno poročilo IJS DP-14746, 2024.

Grabisch, M., Marichal, J.-L., Mesiar, R., Pap, E. (2009) *Aggregation Functions*. Cambridge University Press.

Greco, S., Ehrgott, M., Figueira, J. (Eds.) (2016): *Multi Criteria Decision Analysis: State of the art Surveys*. Springer, New York.

Greco, S., Matarazzo, B., Słowiński, R. (2001): Rough sets theory for multi-criteria decision analysis. *European Journal of Operational Research*, 129, 1, 1–47.

Appendix 1: Notation

Input attributes	$\boldsymbol{X} = \{x_i, i = 1, \dots, n\}$
Output attribute	y
Input attributes' scales	$S_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,k_i}\}, k_i > 0$
Output attribute scale	$S_y = \left\{v_{y,1}, v_{y,2}, \dots, v_{y,k_y}\right\}, k_y > 0$
Some (general) value scale	$S = \{v_1, v_2, \dots, v_k\}, k > 0$
Scale ordering: some scale S can be preferentially ordered or not	$S = \{v_1, v_2, \dots\}$ and $\text{ordered}(S) \Longrightarrow$ $\forall v_i, v_j \in S, j > i: v_j \succcurlyeq v_i$
Scale sizes	$\lvert S_i \rvert = k_i, i = 1, \dots, n;\ \lvert S_y \rvert = k_y; \lvert S \rvert = k$
Input space	$\boldsymbol{S} = S_1 \times S_2 \times \cdots \times S_n$
Input space size	$\lvert \boldsymbol{S} \rvert = k_1 k_2 \dots k_n$
Decision space	$DS = \langle \boldsymbol{X}, y, \boldsymbol{S}, S_y \rangle$
Value space over S_y. There are several possible definitions:	$V(S_y)$
Single value	$V(S_y) \equiv S_y$
Value interval (meaningful only for ordered scales)	$V(S_y) \equiv I(S_y) = [v_{\text{low}}, v_{\text{high}}], v_{\text{low}}, v_{\text{high}} \epsilon S_y, v_{\text{low}} \preccurlyeq v_{\text{high}}$
Value set	$V(S_y) \equiv 2^{S_y} = \{v \mid v \in S_y\}$
Value distribution	$V(S_y) \equiv D(S_y) = \begin{pmatrix} v_{y,1} & v_{y,2} & \cdots & v_{y,k_y} \\ p_1 & p_2 & & p_{k_y} \end{pmatrix}, p_i \in \mathbb{R}^+, i = 1,2, \dots, k_y$
Probability distribution	$V(S_y) \equiv P(S_y) = D(S_y)$ where $\sum_{i=1}^{k_y} p_i = 1$
Fuzzy distribution	$V(S_y) \equiv F(S_y) = D(S_y)$ where $p_i \in [0, 1], i = 1,2, \dots, k_y$
Normalized fuzzy distribution	$V(S_y) \equiv F(S_y)$ where $\max_{i=1}^{k_y} p_i = 1$
Extended value space	$XW(S_y) = W(S_y) \cup \{\text{undefined}\}$ where W is one of: $V, I, 2, D, P, F$
Decision table as function	$F: \boldsymbol{S} \to V(S_y)$
Conditions	$T = \{\boldsymbol{v} \mid \forall \boldsymbol{v} \in \boldsymbol{S}\}$
Subset of decision conditions	$T_{\text{cond}} = \{\boldsymbol{v} \in T \mid \text{cond}(\boldsymbol{v})$
q-th element of vector $\boldsymbol{v}$	$\boldsymbol{v}_q = v \in S_q{:}\text{ord}(v) = q$
Concatenation of $\boldsymbol{v}$ and value w	$\boldsymbol{v} \circ w = \{v_1, v_2, \dots, v_n, w\}, v_i \in S_i$
Vector $\boldsymbol{v}$ with removed q-th value	$\boldsymbol{v} \setminus q = \{v_1, v_2, \dots, v_{q-1}, v_{q+1}, \dots, v_n\}, v_i \in S_i$
Decision rule	$r_{\boldsymbol{v}} = \boldsymbol{v} \to c, \boldsymbol{v} \in \boldsymbol{S}, c \in V(S_y)$
Components of rule $r = \boldsymbol{v} \to c$	$\boldsymbol{v}(r) = \boldsymbol{v} \in \boldsymbol{S}; y(r) = c \in V(S_y)$
Decision table	$R = \{r_{\boldsymbol{v}} \mid \forall \boldsymbol{v} \in \boldsymbol{S}\}$
Decision table subset	$R_{\text{cond}} = \{r_{\boldsymbol{v}} \in R \mid \text{cond}(\boldsymbol{v})\}$
Entered rules	$E(R) = R_{\text{entered}(r)}$
Simplification: ordinal notation of scale S	$S = \{v_1, v_2, \dots, v_k\}$ and considering that $\text{ord}(v_i) = i, i = 1,2, \dots, k$: $Z = \{\text{ord}(v_1), \text{ord}(v_2), \dots, \text{ord}(v_k)\} = \{1, 2, \dots, k\}$
Weak preference	$v_1 \preccurlyeq v_2$: v_2 is at least as preferred as v_1