On the Semantics of “null” in DMN: Undefined is not Unknown

Đorđe Marković[1,3,*], Simon Vandevelde[2,3,4], Joost Vennekens[2,3,4] and Marc Denecker[1,3]

[1] *Department of Computer Science, KU Leuven, Arenberg Campus, Celestijnenlaan 200A, 3001 Leuven, Belgium*
[2] *KU Leuven, De Nayer Compus, Dept. Of Computer Science*
[3] *Leuven.AI – KU Leuven institute for AI, B-3000 Leuven, Belgium*
[4] *Flanders Make – DTAI-FET*

Abstract

Decision Model and Notation (DMN) is a formalism for the representation of knowledge about decision processes. It represents a set of decision rules in an easy-to-understand tabular format, called a decision table. In this paper, we argue that current formal semantics for DMN have certain limitations and we propose a novel formal semantics. Our semantics considers a decision table as a definition. The semantics consists of two components: the first component captures the meaning of one row (rule), and the second component aggregates the meaning of the rules into meaning for the whole table. By choosing the second component in different ways, the different “hit policies” of DMN (i.e., mechanisms for deciding what happens when multiple rows of the same table are applicable) can be represented.

Our semantics can cope better with undefined and unknown values and provides a foundation for forms of reasoning different from deriving the output for a given set of inputs.

Keywords

Decision Model and Notation, Model Theoretic Semantics, Partial Functions, Knowledge Representation

1. Introduction

Decision Model and Notation (DMN) [1] is a minimalistic yet powerful knowledge representation formalism. It is increasingly popular in the industry, where a myriad of business software vendors builds tools around it. The success of DMN stems from its straightforward syntax and operational semantics. Thanks to its *tabular* format, expressing decisions requires neither knowledge of complex grammatical rules nor of connectives and quantifiers. For example, the first row of the table from Fig. 1 expresses *constraints* “< 10” and “$= 4$” on *input attributes* X_1 and X_2, and, when satisfied, the assignment of $true$ to *decision attribute* D. The entire table derives a decision iff for given values of X_1 and X_2 there is a unique rule that applies. This is specified by the *hit policy* (“U”nique) which is a mechanism for deciding what happens when multiple rows of the same table are applicable. If for some combination of input values there is no matching rule we say that the rule is *missing*, and if there is more than one matching rule, we say that they are *overlapping*.

RuleML+RR’22: 16th International Rule Challenge and 6th Doctoral Consortium, September 26–28, 2022, Virtual
[*] Corresponding author.
dorde.markovic@kuleuven.be (Đ. Marković); s.vandevelde@kuleuven.be (S. Vandevelde); joost.vennekens@kuleuven.be (J. Vennekens); marc.denecker@kuleuven.be (M. Denecker)

Example			
U	X_1	X_2	D
1	< 10	$= 4$	$true$
2	≥ 10	$= 4$	$false$

Figure 1: A DMN table with X_1 and X_2 input attributes, D decision attribute and (U)nique hit policy.

In the past, there have been multiple attempts to formalize the semantics of DMN [2, 3] as a theory in first-order logic (FO). We argue that these semantics only approximately formalize the informal definition of DMN. Furthermore, we argue that there are three significant problems: **(1)** translating DMN tables to FO is not very well suited for capturing the nature of DMN decision tables as combinations of rules under various hit policies, **(2)** FO is not well suited for dealing with *partial* attributes (i.e., they may be *undefined*), and **(3)** there is no way of distinguishing between epistemic and objective relations while decisions are of epistemic nature. This reflects the problem of the distinction between *unknown* value and *non-existing* value. Some of these issues are appearing in examples encountered by the DMN community, such as in [4, 5].

In the current DMN formalism, there are two possible ways of going around this issue. Firstly, DMN has a special "null" value, which is used when no rule of a table fires [1, p.166]. However, it is also used in various other cases, such as for *notANumber*, *negativeInfinity* and *positiveInfinity* [1, p.104], which makes its meaning not perfectly clear. Moreover, many commercial DMN rule engines allow it in different places (e.g., as a value of an input attribute, as a constraint of a row) and implement its behavior differently. An alternative solution is to simulate non-existing values by introducing a new value, such as "*does_not_exist*". This approach brings room for all kinds of mistakes. This *special* value requires special treatment in each statement where the variable ranges over that value, in other words, it pollutes the domain. Furthermore, this special value should be introduced for each type, breaking the uniform way of treating undefinedness. In both cases, there is a semantic mismatch with FO. This stems from FO's assumption that every logical constant is *total*, i.e., has a value. Hence, the introduction of partial attributes requires subtle changes in the current semantics and syntax.

In this paper, we present a general solution for these problems by proposing an alternative semantics for DMN that is declarative, yet, much closer to the intuitive operational understanding of DMN. We define the semantics in two steps: semantics of individual rules, and semantics of a decision table (i.e., a sequence of rules). Individual rules are defined as decision value derivation functions, and decision tables are defined as an aggregate function (corresponding to the hit policy) over values derived by the rows of the table. With this semantics we investigate the proper distinction between undefined (non-existing) values and unknown values in three different situations: (i) some attribute is known to be undefined, (ii) it is not known whether an attribute is defined or not, and (iii) it is known that an attribute is defined, but its value is unknown.

The remainder of the paper is structured as follows. First, an overview of the most important related work and issues with current approaches are provided. Next, we present the preliminary version of the novel semantics for DMN. Further, we discuss the benefits and practical applications of our semantics. We close the paper with notes on future work and a conclusion.

2. Related Work

Although DMN is a relatively young formalism, quite some theoretical work has already been done. This section provides an overview of the most related ideas to this paper and points out the important differences.

In [3], Calvanese et al. define a model semantics for DMN decision tables by translating tables to first-order logic statements. In the paper, the rules in a table with *unique* hit policy are translated into a set of material implications. Each rule is represented as a material implication from the rule's conditions to the assignment of values to decision attributes. Rule conditions on an input attribute without value, are assumed to be false. This approach was developed under the assumption that tables do not have *missing* or *overlapping* rules and works correctly in these cases. However we claim that even when this assumption does not hold, such tables still have a meaning (Section 3), and hence the formal semantics should capture them. Furthermore, this semantics does not generalize well, as each hit policy requires a separate translation to FO (problem (1) from the introduction).

Another approach is based on defining an input-output relation based on a DMN decision table and its hit policy. The *rule semantics* described in [2] by Calvanese et al. follows this idea. This relation contains per rule all tuples of input and decision values such that the rule is satisfied for the input and derives decision values. In this way, the input-output relation correctly approximates undefinedness and over-definedness in case no or multiple rules are applicable for some input values. This approach resolves the first problem from our list, while the other two are still present.

Finally, the official DMN specification [1] suggests using *null* values in certain situations (e.g., missing row in a table with a unique hit policy). However, the specification does not specify how to treat *null* values as input to the table and distinguish them from existing but unknown values. Furthermore, the specification does not provide model semantics but instead focuses on deriving decisions from tables based on input values. This is a problem for the generalization of different inference methods which are based on possible world analysis as will be briefly demonstrated in Section 5. The aim of the formal semantics defined below is to formalize the intuitions of the official DMN specification in full generality while filling in gaps left open in them.

3. Analysis of Existing Semantics

In this section, we briefly illustrate the three problems discussed in Section 1.

Let us consider the non-realistic simple[1] example from Fig. 2 of two tables with unique hit policy, one of which (2a) has a *missing* rule and the other one (2b) has *overlapping* rules for input value 10. One can argue that these tables should not be considered since they are not "correct". But Table 2a will nevertheless provide a sound decision for any combination of input values except for the missing one, which makes it usable. On the other hand, Table 2b has a conflict in case $Input = 10$, resulting in a clear *error*. In this paper, we take the simple approach

[1]Note that our examples do not contain default values, which is allowed by the standard [1, p. 134].

Missing		
U	Input	Dec
1	< 10	Yes
2	> 10	No

(a) Missing rule

Overlapping		
U	Input	Dec
1	≤ 10	Yes
2	≥ 10	No

(b) Overlapping rules

Salutation			
U	Gender	MStatus	Salut
1	Male	-	Mr
2	Female	Married	Mrs
3	Female	Single	Ms

(c) Salutation decision table

Figure 2: Examples of decision tables with missing and overlapping rules, and DMN decision table for salutation based on gender and marital status.

to derive *undefined*) for the decision attribute in this case[2]. We proceed to show that FO is not well suited for capturing these cases. Following [3], the two tables (a+b) are translated in the following material implications.

$$(Input < 10 \Rightarrow Decision = Yes) \land (Input > 10 \Rightarrow Decision = No) \tag{1}$$

$$(Input \leq 10 \Rightarrow Decision = Yes) \land (Input \geq 10 \Rightarrow Decision = No) \tag{2}$$

It is not hard to see that these translations do not match the basic intuition in the case when $Input = 10$. Formula (1) is trivially satisfied as both antecedents are *false* and hence $Decision$ can take any value. But according to the DMN specification, in this case, $Decision$ should take the distinguished value *null*. On the other hand, the formula (2) will be unsatisfiable since $Decision$ should be both Yes and No, which is not possible. This case is not well covered by the DMN specification, but intuitively there is something wrong with this table due to its inconsistency. This has to be distinguished from unsatisfiability which is representing impossible situations (more in Section 5.3). The *input-output relation* [2] for these two examples would respectively assign an empty set $R_{io}(10) = \emptyset$ and a set $R_{io}(10) = \{Yes, No\}$ for the value 10. This captures our intuition of being undefined and over-defined, but due to the inherited limitations of the first-order logic, the problem with non-existing values for input attributes still remains.

Another major issue with the existing semantics of DMN is support for unknown values and distinction between objective and epistemic relations. The greeting example [5] is an example set forth by industry practitioners demonstrating these issues. Concretely, it discusses the automatic generation of a greeting and lists some pitfalls to consider. Among other things, it brings to attention that the gender and marital status of a person might not be known to the system and that a table such as in Fig. 2c might not suffice. The problem with DMN is that there is no good formalization that would allow reasoning in cases when something is unknown. For example, in case we do not know the gender of a person but we know that they are single we would probably say something like "Dear Mr., Ms." and certainly not "Mrs". To circumvent these issues one can use the *null* value, or introduce another value simulating that something is unknown. But these attempts have a major downside: they are not standardized by the formalization and hence can lead to many issues. This causes a lot of difficulties for users since they have to make very peculiar choices in order to get the desired effects. Furthermore,

[2]It is straightforward to extend the semantics with over-definedness and capture these *errors*.

there are strong limitations to current approaches; e.g., in some applications or contexts, there may be information that "all female persons are married" and we cannot express or exploit that information.

4. DMN Formal Syntax and Semantics

DMN is recognizable for its simple and clean syntax. It supports certain data types described with object symbols, characteristic relations (e.g., the order relation), and characteristic functions (e.g., the addition of numbers). Since our intention is to define a general semantics, we will not focus on a particular data type in this paper, but rather focus on an abstract data type.

Definition 4.1. *A **data type** $\mathcal{D}$ is a triple (Dom, P, F) where Dom is a set of objects (object domain), P is a set of **predicate symbols** and F is a set of **function symbols**. Predicate and function symbols have arity denoted as p/n and f/n. Function symbols of arity 0 are called **object symbols**. A data type interprets all predicate and function symbols as: Per predicate symbol p/n it assigns a relation $p^{\mathcal{D}} \subseteq Dom^n$ and per function symbol f/n a function $Dom^n \rightarrow Dom$ denoted with $f^{\mathcal{D}}$. $\mathcal{D}_{Dom}$, $\mathcal{D}_P$, and $\mathcal{D}_F$ respectively denote projections of Dom, P, and F of a data type $\mathcal{D}$.*

The set of all terms for a particular data type is defined as:

Definition 4.2. *Given a set of function symbols $\mathcal{D}_F$ of a data type $\mathcal{D}$, the set of all terms over $\mathcal{D}$, denoted as $Term(\mathcal{D})$, is defined as:*

- *An object symbol $f/0 \in \mathcal{D}_F$ is a term,*
- *If $t_1, \ldots, t_n$ are terms and $f \in \mathcal{D}_F$ is a function of arity n then $f(t_1, \ldots, t_n)$ is a term.*

Based on terms, a DMN expressions are defined as follows.

Definition 4.3. *For a given data type $\mathcal{D}$ we define a DMN **value expression** v as $v := t$ where t is a term from $Term(\mathcal{D})$ and **constraint expression** C as:*

$$\begin{aligned} A :=& p(v_1, \ldots, v_{i-1}, \_, v_{i+1}, \ldots, v_n) \mid \texttt{not}\ \ A && \text{(Atom and negation)} \\ E :=& \text{“} - \text{”} \mid \text{“}defined\text{”} \mid \text{“}undefined\text{”} && \text{(Expressions)} \\ C :=& A \mid E \mid C, C && \text{(Compound)} \end{aligned}$$

Where p is a n-ary predicate symbol from $\mathcal{D}_P$ applied to $n-1$ values leaving i-th argument unspecified, denoted with "_" (when obvious from the context, "_" is omitted). We call this language $\mathcal{DMN}_c$ (for DMN constraints). Additionally, $C[X]$ denotes an FO logic expression obtained by replacing all occurrences of "_" in C with the attribute symbol X, replacing "not" with $\neg$ and "," with $\vee$.

Note that only atomic constraints can be negated, negating other DMN constraints is not needed as not $defined = undefined$ and vice versa, while negating " $-$ " would mean that nothing matches this constraint, which would be equivalent to omitting the rule. Following is the example of DMN constraints over the data type of natural numbers.

Example 4.1. *Consider the data type of natural numbers equipped with comparison predicates and standard binary functions:*

$$\mathcal{N} = (\mathbb{N}, \{< /2, > /2, \leq /2, \geq /2\}, \{+/2, -/2, \times/2, \div/2, \mathbf{0}/0, S/1\})$$

We use $\mathbf{1}, \mathbf{2}, \ldots$ *to denote* $S(\mathbf{0})$, $S(S(\mathbf{0}))$, … *The examples of terms are:* $\mathbf{0}$, $S(\mathbf{0})$, $\mathbf{10} \times \mathbf{5}$, … *The atomic constraint expressions are defined as follows:*

$$A := (v..v) \mid [v..v] \mid (v..v] \mid [v..v) \mid < v \mid > v \mid \leq v \mid \geq v \mid = v$$

An example of an atomic constraint expression is $(1..34)$. *When this occurs in a cell below attribute* X, *it expresses the constraint* $1 < X < 34$, *while* $(1..34]$ *expresses* $1 < X \leq 34$, *and similar for the others.*

Section 3 demonstrates that current formal semantics are not aligned with the human intuition behind the decision tables. Hence, based on the informal operational understanding of DMN decision tables, we propose a two-step declarative semantics[3]. First defining the formal semantics of rules individually and then decision tables as the hit policy function applied on the sequence of values produced by all the rules. We start with definitions of rules and tables where a rule corresponds to a row in a DMN table, and a sequence of rules corresponds to the full table.

Definition 4.4. *For a given sequence* $\vec{X} = \langle X_1, \ldots, X_n \rangle$ *of input attributes and a decision attribute* D*: A DMN* ***rule*** r *is a pair* $(\vec{C}, v)$ *where* $\vec{C}$ *is a (finite) sequence of constraint expressions* $\langle C_1, \ldots, C_n \rangle$ *and* v *an value expression. The logical format of* r *is the expression* $D := v \leftarrow C_1[X_1] \wedge \cdots \wedge C_n[X_n]$ *where the left/right hand side of the* $\leftarrow$ *is head/body denoted as* $H(r)/B(r)$. *A DMN* ***decision table*** *is a pair* $(\vec{R}, h)$ *where* $\vec{R}$ *is a (finite) sequence of rules (for* $\vec{X}$ *and* D*) and* h *a hit policy expression.*

We intend to define semantics that can cope with undefined input and decision attributes. In order to define a formal model semantics, we need a notion of an *interpretation* assigning values to the attributes and hit policy functions. But introducing partiality brings the issue of constraining the existence of a value. According to Kleene's principle of *regularity* for three-valued logic [6, p.334] comparing an undefined object with anything results in undefinedness. For that reason we extend DMN syntax with two new constraints $defined$ and $undefined$ (Definition 4.3) with semantics defined in 4.7.

Definition 4.5. *Given a data type* $\mathcal{D}$ *and set* A *of attribute symbols, a DMN interpretation* $\mathfrak{A}$ *over* A *in* $\mathcal{D}$ *consists of:*

- *A domain* $\mathbb{D}$ *which is a non empty set* $[\![\mathbb{D}]\!]^{\mathfrak{A}} = \mathcal{D}_{Dom}$.
- *Per attribute symbol* $x \in A$ *an element* $d \in [\![\mathbb{D}]\!]^{\mathfrak{A}}$ *or distinguished value* $\perp$, *denoted as* $[\![x]\!]^{\mathfrak{A}}$.
- *Per hit policy function symbol* h *an aggregate function* $[\![h]\!]^{\mathfrak{A}}$ *that maps the sequence of contributions of the rules into a unique value for the output attribute, as in definition 4.6.*

[3]To preserve simplicity, we restrict DMN decision tables to the simpler form without losing generality in which all attributes are of the same type, decision tables have only one decision attribute and there are only two hit policies.

Each DMN constraint expression C denotes the (obvious) set $[\![C]\!]^{\mathfrak{A}} \subseteq [\![\mathbb{D}]\!]^{\mathfrak{A}} \cup \{\bot\}$, and each DMN value expression v denotes an element from the domain $[\![v]\!]^{\mathfrak{A}} \in [\![\mathbb{D}]\!]^{\mathfrak{A}}$ according to the predicate and function symbol interpretations in $\mathcal{D}$. Definition 4.7 demonstrates this with the data type of natural numbers $\mathcal{N}$.

Definition 4.6. *Let $\vec{V} = \langle V_1, \ldots, V_n\rangle$ be a sequence of values, then: The* ***unique*** *(i.e., exactly one rule matches) hit policy is $u(\vec{V}) =$ if $\exists_1 i \in \{1..n\} : V_i \neq \bot$ then $V \in \vec{V} \mid V \neq \bot$ else $\bot$ (where $\exists_1$ means "exists exactly one"), and* ***sum*** *(i.e., summation of values of all matching rules) hit policy is $c_+(\vec{V}) =$ if $(\vec{V}_{\not\bot}) \neq \emptyset$ then $sum(\vec{V}_{\not\bot})$ else $\bot$. Where $\vec{V}_{\not\bot}$ represents a sequence obtained by removing all values equal to $\bot$ from $\vec{V}$.*

Definition 4.7. *For a given structure $\mathfrak{A}$ over data type of natural numbers $\mathcal{N}$, a DMN* ***value expression*** *v denotes an element from domain $[\![\mathbb{D}]\!]^{\mathfrak{A}}$ (in this case $\mathbb{N}$), which is denoted with $[\![v]\!]^{\mathfrak{A}}$ and defined as: if v is a object symbol then $[\![v]\!]^{\mathfrak{A}} = v^{\mathcal{N}}$, if it is a compound term $v = f(t_1, \ldots, t_n)$ then $[\![v]\!]^{\mathfrak{A}} = f^{\mathcal{N}}(t_1^{\mathcal{N}}, \ldots, t_n^{\mathcal{N}})$. A DMN* ***constraint expression*** *C (over $\mathcal{N}$) denotes a subset of $[\![\mathbb{D}]\!]^{\mathfrak{A}}_{\bot}$ (the domain of $\mathfrak{A}$ augmented with $\bot$, in this case $\mathbb{N}_\bot$), denoted as $[\![C]\!]^{\mathfrak{A}}$. We first define the value of atomic expression A:*

$$[\![[i..j]]\!]^{\mathfrak{A}} = \{x \mid x \in \mathbb{N} \wedge [\![i]\!]^{\mathfrak{A}} \leq^{\mathcal{N}} x \leq^{\mathcal{N}} [\![j]\!]^{\mathfrak{A}}\} \qquad [\![(i..j)]\!]^{\mathfrak{A}} = \{x \mid x \in \mathbb{N} \wedge [\![i]\!]^{\mathfrak{A}} <^{\mathcal{N}} x <^{\mathcal{N}} [\![j]\!]^{\mathfrak{A}}\}$$

$$[\![(i..j]]\!]^{\mathfrak{A}} = \{x \mid x \in \mathbb{N} \wedge [\![i]\!]^{\mathfrak{A}} <^{\mathcal{N}} x \leq^{\mathcal{N}} [\![j]\!]^{\mathfrak{A}}\} \qquad [\![\oplus i]\!]^{\mathfrak{A}} = \{x \mid x \in \mathbb{N} \wedge x \oplus^{\mathcal{N}} [\![i]\!]^{\mathfrak{A}}\}$$

$$[\![[i..j)]\!]^{\mathfrak{A}} = \{x \mid x \in \mathbb{N} \wedge [\![i]\!]^{\mathfrak{A}} \leq^{\mathcal{N}} x <^{\mathcal{N}} [\![j]\!]^{\mathfrak{A}}\} \qquad [\![not \;\; A]\!]^{\mathfrak{A}} = \mathbb{N} \setminus [\![A]\!]^{\mathfrak{A}}$$

where $\oplus \in \{=, <, >, \leq, \geq\}$, and i and j are value expressions. The value of other DMN expressions is defined as:

$$[\![defined]\!]^{\mathfrak{A}} = \mathbb{N} \qquad [\![undefined]\!]^{\mathfrak{A}} = \{\bot\} \qquad [\![-]\!]^{\mathfrak{A}} = \mathbb{N}_\bot$$

Compound constraint, constructed with "or" operator (in DMN ","), is defined as:

$$[\![C_1, \ldots, C_n]\!]^{\mathfrak{A}} = [\![C_1]\!]^{\mathfrak{A}} \cup \cdots \cup [\![C_n]\!]^{\mathfrak{A}}$$

Note that atomic expressions are always specifying a subset of the domain, meaning that each of these constraints carries an assumption that the value exists.

If not stated differently we assume that each interpretation is over data type of natural numbers $\mathcal{N}$. Henceforth data type is not mentioned explicitly in the following definitions.

The semantics of a rule is defined as a function that maps an interpretation of the input attributes to the *contribution* of the rule, the value for the decision attribute.

Definition 4.8. *For a given sequence of n input attributes $\vec{X} = \langle X_1, \ldots, X_n\rangle$ and a decision attribute D, let $r = (\vec{C}, v)$ be a DMN rule with n constraint expressions $\vec{C} = \langle C_1, \ldots, C_n\rangle$ and value expression v, and let $\mathfrak{A}$ be a DMN interpretation (over $\vec{X}$ and D), then $\mathbb{C}_r(\mathfrak{A})$, the value contributed by r in $\mathfrak{A}$, is $[\![v]\!]^{\mathfrak{A}}$ when each constraint is satisfied (in $\mathfrak{A}$) and $\bot$ otherwise; formally:*

$$\mathbb{C}_r(\mathfrak{A}) = \text{ if } \forall i \in \{1..n\} : [\![X_i]\!]^{\mathfrak{A}} \in [\![C_i]\!]^{\mathfrak{A}} \text{ then } [\![v]\!]^{\mathfrak{A}} \text{ else } \bot$$

Finally, the function associated with the hit policy aggregates all contributions of rules into one value.

Definition 4.9. *For a given input attributes $\vec{X}$ and a decision attribute D, let $T = (\vec{R}, h)$ be a DMN table with k rows and $\mathfrak{A}$ be a DMN interpretation (over $\vec{X}$ and D) then $\mathfrak{A}$ **satisfies** table T, or $\mathfrak{A}$ is a **model** of T, (denoted as $\mathfrak{A} \models T$) iff the value $[\![D]\!]^{\mathfrak{A}}$ equals to the value obtained by applying the hit policy function ($[\![h]\!]^{\mathfrak{A}}$) on the sequence of values $\langle \mathbb{C}_{r_1}(\mathfrak{A}), \dots, \mathbb{C}_{r_k}(\mathfrak{A}) \rangle$ derived by rules ($r_i \in \vec{R}$) application; formally:*

$$[\![D]\!]^{\mathfrak{A}} = [\![h]\!]^{\mathfrak{A}}(\langle \mathbb{C}_{r_1}(\mathfrak{A}), \dots, \mathbb{C}_{r_k}(\mathfrak{A}) \rangle)$$

5. Discussion

In this section, we discuss the results of the proposed model semantics, preliminary ideas, and future work. The model semantics defined in the Section 4 distinguishes itself from other approaches in the following aspects:

1. It is straightforward to extend the semantics with new hit policies. Also, it is possible to define hit policies that are domain-specific (e.g., if rule 3 derives value ≥ 5 take the value produced by the rule 4 and otherwise the value produced by the rule 5).
2. The distinction between undefined and unknown is made clear (more in Section 5.1).
3. Combining decision tables is straightforward, and tables with *missing* or *overlapping* rules does not require any special treatment (more in Section 5.3).
4. Employing formal reasoning methods is possible and hence solving many different problems without implementing new custom algorithms but rather using existing solvers. This is the principle of knowledge representation and reasoning as explained in [7, 8, 9], and in [10] such an approach based on the semantics of [3] is demonstrated with the use of the IDP system [8].
5. View of DMN tables as set of definitional rules directly supports *relevance* inference (as implemented in [11, 12]). Relevance inference, for a given DMN table and partial interpretation, determines the set of attributes whose value does not affect the model anymore. However, translating tables to FO destroys dependency relation between attributes and would require the development of custom techniques for relevance in DMN tables. Such an algorithm for relevance inference in DMN tables is implemented in [10], but our view on rows as rules allows us to exploit the structure of rules and detect relevance easier.

5.1. Unknown vs Undefined

One of the main contributions of this paper is the disambiguation of unknown and undefined values of DMN attributes. This subsection provides a detailed discussion on this matter.

We start from the usual DMN task, which is computing the value of the decision attribute for given values of input attributes. Let $X_1, \dots, X_n$ be input attributes and D the decision attribute of the DMN table T. The problem is then to compute the value of D when values of $X_1, \dots, X_n$ are given by value expressions $v_1, \dots, v_n$. To solve this problem using model semantics proposed in this paper it is necessary to find the interpretation $\mathfrak{A}$ such that $[\![X_i]\!]^{\mathfrak{A}} = [\![v_i]\!]^{\mathfrak{A}}$ for $i \in \{1, \dots, n\}$ and $\mathfrak{A} \models T$. Finding such an interpretation would require a solver capable of determining whether some constraint is satisfied in an interpretation or not; practically implementing definition 4.7 and building up to answer the question of whether some interpretation is a model of a DMN table. For the moment we assume the existence of such a solver and discuss it further in Section 5.2. Taking into account that values of input attributes are exact and that value of the decision is obtained with an aggregate function, it follows that there exists exactly one interpretation satisfying the above requirements, and hence the decision value is $[\![D]\!]^{\mathfrak{A}}$.

Although solving this kind of problem is already very useful, in real-life scenarios it is often the case that the user does not know exact values for all input attributes but rather a set of possible values. For example, a user might know that input attribute X_1 is between numerical values 3 and 10. Hence we propose to allow users to specify constraints (using $\mathcal{DMN}_c$) on attributes rather than exact values. By combining different constraints users can express their knowledge about an attribute value. This idea is not new, cDMN [13] is DMN extension that among other functionalities supports constraints on input attributes in the form of material implications. The following definition formalizes the idea of imposing a set of constraints on each attribute using $\mathcal{DMN}_c$.

Definition 5.1. ***DMN model generation with attribute constraints*** *is an inference method defined as: For a given input: (i) A set of input and decision attributes: $A = \{X_1, \ldots, X_n, D\}$, (ii) A DMN table: T, (iii) Per attribute $X \in A$ a set of $\mathcal{DMN}_c$ constraints : C_X, the output is the set of interpretations such that each interpretation satisfies all attribute constraints and is a model of the table, formally: $\{\mathfrak{A} \mid \forall X \in A : \forall C \in C_X : [\![X]\!]^{\mathfrak{A}} \in [\![C]\!]^{\mathfrak{A}}$ and $\mathfrak{A} \models T\}$.*

Constraining attributes in this way allow users to distinguish between unknown and undefined values. Following are constraints corresponding to the three situations from the introduction:

1. Constraint "$undefined$" expresses that some attribute is known to be undefined.
2. Constraint "$defined$" expresses that some attribute is known to be defined, but its value is unknown.
3. The absence of constraints means that it is not known whether an attribute is defined or not (nor anything about its value).

Furthermore, by combining different constraints one can express much more. For example, "$undefined$, [3..10]" means: "if the attribute is defined then its value is between 3 and 10". However this is not the most convenient way of expressing such constraints, hence extending the language of attribute constraints would be useful. We propose to use the language $\mathcal{L}_{ac}$ defined as the set of formulas consisting of $\mathcal{DMN}_c$ constraints closed under the logical connectives $\wedge$,$\vee$,$\neg$, $\Rightarrow$, and $\Leftrightarrow$. The constraint from above then becomes "$defined \Rightarrow$ [3..10]". The semantics of this language is straightforward as it can be defined by a slight extension of standard DMN constraints. The main difference is that DMN constraints do not support $\wedge$ connective, or negation of constraints connected with "or" (in DMN ",") like "`not` $(defined, [3..10])$". Following is the definition of the value of $\mathcal{L}_{ac}$ expressions for $\vee$ and $\neg$, as other connectives can be expressed in terms of these.

Definition 5.2. *For a given DMN interpretation $\mathfrak{A}$, a $\mathcal{L}_{ac}$* ***attribute constraint expression*** *C denotes a set $[\![C]\!]^{\mathfrak{A}}_{ac}$ which is subset of the $[\![\mathbb{D}]\!]^{\mathfrak{A}} \cup \{\bot\}$ (denoted as $[\![\mathbb{D}]\!]^{\mathfrak{A}}_{\bot}$), defined as:*

$$[\![C_1 \vee C_2]\!]^{\mathfrak{A}}_{ac} = [\![C_1]\!]^{\mathfrak{A}}_{ac} \cup [\![C_2]\!]^{\mathfrak{A}}_{ac} \qquad [\![\neg C]\!]^{\mathfrak{A}}_{ac} = [\![\mathbb{D}]\!]^{\mathfrak{A}}_{\bot} \setminus [\![C]\!]^{\mathfrak{A}}_{ac} \qquad [\![C]\!]^{\mathfrak{A}}_{ac} = [\![C]\!]^{\mathfrak{A}} \text{ if } C \in \mathcal{DMN}_c$$

5.2. Solver

To make practical use of the semantics proposed in this paper, a *solver* implementing it is needed. The solver should be able to answer whether an interpretation is a model of one or more DMN tables. On top of that, the solver should be intuitive to extend with different functionalities, such as generating all models of a table, extending partially specified interpretations, and more. However, building such a solver is a difficult task and a bit redundant since there are existing solvers for many languages, especially those based on FO. While this suggests that a translation of DMN tables to some FO language is required, as we pointed out earlier, standard FO will not suffice. Indeed, support for partial functions, definitions, and (optionally) aggregates is required. At the moment of writing this paper, we are not aware of a solver that provides all of these options. But, one solver, the IDP system [8], provides most of these with limited support for partial functions. Limitations of the IDP system for partial functions are mostly related to recursive function definitions which do not occur in DMN tables. Hence, we propose a translation to FO(·), first-order logic language extended with *inductive definitions* [14], *aggregates*, *partial functions*, and *types* (summarized in [8]), which is the language used by the IDP system.

To implement the idea from Definition 5.1 we create three FO(·) theories. The first theory is specifying attribute constraints (i.e., $\mathcal{L}_{ac}$ constraint expressions). The second theory represents DMN table translation to a definition; a set of rules of the form $D := v \leftarrow C_1[X_1] \wedge \cdots \wedge C_n[X_n]$ (Definition 4.4). However, this is not sufficient as the sequence of rules is lost, hence translation has to include the numerical identifier (id) assigned to the row in the DMN table. The most convenient way with IDP system is to replace the head of rule with relation: $R_D(id, v) \leftarrow C_1[X_1] \wedge \cdots \wedge C_n[X_n]$. The third theory defines the hit policy as aggregate functions over the R_D relation; for example unique hit policy:

$$\forall v : D = v \leftarrow (\exists r : R_D(r, v)) \wedge \#\{r_1, v_1 : R_D(r_1, t_1)\} = 1.$$

The IDP solver can generate all models satisfying these three theories. These models correspond to models defined in definition 5.1.

We demonstrate this approach in the greeting example from Fig. 2c. Presented are snippets of theories while the whole formalization is provided online[4]. First, the theory that expresses the user's knowledge about the attributes of the table is declared (e.g., $Gender = Male$).

```
theory T_attribute_constraints : V_salutation{
    Gender = Male.
}
```

Following is the theory expressing the contributions of the rules (with identities).

```
theory T_rules_derivation : V_salutation_decision {
    {
        R_Salutations(1,Mr) ← Gender = Male.
        R_Salutations(2,Mrs) ← Gender = Female ∧ MaritalStatus = Married.
        R_Salutations(3,Ms) ← Gender = Female ∧ MaritalStatus = Single.
    }
}
```

[4]https://bitbucket.org/krr/dmn-to-idp-poc/src/main/

Finally, the theory defining unique hit policy is instantiated for this particular table.

```
theory T_unique_hit_policy : V_salutation_decision {
    {
        ∀ t[Titles] : Salutation = t ← (∃r[Rule]:R_Salutations(r,t))
            ∧ #{r1[Rule], t1[Titles]:R_Salutations(r1,t1)}=1.
    }
}
```

The model of theories above obtained by the use of the IDP solver would look like this:

```
structure  : V_salutation {
    Gender = Female
    MaritalStatus = Single
    Salutation = Ms
}
```

The IDP system is sufficient for building this proof of concept example. Furthermore, it provides a variety of inference methods and hence is a good candidate to serve as a solver for DMN model semantics proposed in this paper.

5.3. Combining Tables

So far in this paper, we focus on the semantics of DMN tables individually, while they are almost always used as a part of Decision Requirements Graphs (DRG) where tables can share inputs or be connected (the decision of one table is an input of another). The reason for this is that model semantics defined in this paper naturally scale up to multiple tables.

Definition 5.3. *For a given sequence of attributes $\vec{X}$, let S be a set of DMN tables $\{T_1, \dots, T_n\}$ where each table is over some subset of attributes, and let $\mathfrak{A}$ be a DMN interpretation (over $\vec{X}$) then $\mathfrak{A}$ **satisfies** set of DMN tables S, or $\mathfrak{A}$ is a **model** of S, (denoted as $\mathfrak{A} \models S$) iff $\mathfrak{A}$ is a model of each table $T \in S$; formally:*

$$\mathfrak{A} \models S \Leftrightarrow \forall T \in S : \mathfrak{A} \models T$$

In the introduction, it is mentioned that the current semantics based on the translation to FO would result in undefinedness in the case of tables with *overlapping* rules. This would prevent decision derivation for other tables that perhaps are deriving a decision. For examples, combining tables from Fig. 2b and 2c in case that user knows that $Input = 10$ for table 2b and $Gender = Male$ for table 2c. Translation of rules to material implications would result in a theory that is unsatisfiable because of the overlapping rules in table 2b. However, table 2c derives the decision $Salutation = Mr$ based on gender. Our semantics produces one model in such scenario, decisoin of table 2b is non-existing (i.e., undefined) and decision of table 2c is Mr. As pointed out earlier, it is more natural to consider overlapping rules (with a unique hit policy) as an error. The model semantics defined in this paper can be refined with over-defined values which would correctly capture the value in these cases and allow different interpretations from undefinedness.

5.4. Epistemic DMN

In Sections 5.1 and 5.2 it is demonstrated how to get models of a DMN table where user can express the knowledge about input/output attributes. These ideas are briefly demonstrated in the greeting challenge example [5]. However, getting models for salutation, in case of partially known data about someone (i.e., gender and marital status) is not enough to solve the problem. For example, if the only information known about someone is that $MaritalStatus = Single$ there will be two models, in one of them $Salutation =$ "Mr" and in another $Salutation =$ "Ms". This is because the table from Fig. 2c does not specify the decision of salutation to be used in the greeting message, but rather the objective relation between gender, marital status, and salutation. To get the salutation that should be used in a greeting message it is necessary to observe all the worlds possible according to this objective relation. In our example, there are two possible salutations "Mr" and "Ms", hence reasonable decision would be to just use "*Dear customer*".

Expressing such statements, about all possible models, is not possible in DMN. This forces users to use encoding tricks which ultimately results in a bad knowledge representation. While as a matter of fact, the nature of this knowledge is epistemic (e.g., If we know that salutation is "Mr", taking into account what is known about gender and marital status, then use greeting "*Dear Mr.*"). It is important to observe that there are two levels of knowledge in this statement. The first is about values of gender and marital status. The second is about *salutation message* with respect to all the possible salutations (according to what is known about the person). This idea is known as ordered epistemic logic and it is worked out in detail in [15]. The basic idea is to first get all models of the salutation according to the user's information on gender and marital status, as explained in section 5.2. The next step is to evaluate epistemic statements like "If it is known that salutation is "Mr"", or formally (using modal epistemic K operator) $K(Salutation = \text{"}Mr\text{"})$. This can be done by verifying that $Salutation$ is indeed equal to "Mr" in all models of the first theory.

We propose to extend DMN with modal epistemic K operator and in particular with ordered epistemic logic. This extension comes naturally to DMN since tables are not allowed to have cycles in dependency. Moreover, this extension would not require a new specialized solver as it is possible to simulate it using the IDP system (as demonstrated in [15]). The table from Fig. 3 shows how the DMN epistemic decision table would look like for the salutation example.

Salutation message		
U	Salutation	Salutation message
1	$K(Mr)$	"Dear Mr."
2	$K(Mrs)$	"Dear Mrs."
3	$K(Ms)$	"Dear Ms."
4	$K(Mrs \vee Ms) \wedge \neg K(Mr) \wedge \neg K(Mrs) \wedge \neg K(Ms)$	"Dear Madam"
5	$\neg K(Mrs \vee Ms) \wedge \neg K(Mr) \wedge \neg K(Mrs) \wedge \neg K(Ms)$	"Dear Customer"

Figure 3: Salutation message decision based on salutation value, epistemic DMN

The first three rows are simply stating that in cases when an exact salutation is known then it should be used in the message. The fourth row expresses that in cases when $Mrs \vee Ms$ is known and no exact salutation is known message is "*Dear Madam*". This rule is quite cumbersome because it is necessary to eliminate all the previous cases. Likewise, the last rule states that if non of the above is known "*Dear Customer*" message is appropriate.

The size of the epistemic formulas in the table from Fig. 3 is simply unmanageable. However, it is possible to optimize it by the use of different hit policies. For example, *first* hit policy derives the value of the first rule that produces a value where the order of the rules corresponds to the numbers in the table. The simplified table is presented in Fig. 4.

Salutation message		
F	Salutation	Salutation message
1	$K(Mr)$	"Dear Mr."
2	$K(Mrs)$	"Dear Mrs."
3	$K(Ms)$	"Dear Ms."
4	$K(Mrs \vee Ms)$	"Dear Madam"
5	-	"Dear Customer"

Figure 4: Salutation message decision based on salutation value with first hit policy

The informal interpretation of this table is slightly different from the previous one, but they have the same meaning. The first three rules are expressing that it is *safe* to use a message with the exact salutation only if that salutation is known. According to the fourth rule it is *safe* to use message "*Dear Madam*" if $Mrs \vee Ms$ is known. The last rule specifies that message "*Dear Customer*" is always *safe*. The first hit policy then selects the most precise message where precision is following the order of rules. We provide a proof of concept for this example (using table from the Fig. 3) using the IDP system[5].

5.5. Future Work

The topics that are opened with this paper are the formalization of epistemic DMN with ordered epistemic logic [15] and the implementation of the solver for the proposed semantics. The first version of the solver can be implemented as an automated translation of DMN tables to FO(·) as demonstrated in Section 5.2.

6. Conclusion

In this paper, we proposed and partially formalize the extension of DMN with undefinedness, uncertainty, and epistemic K-operator. This is an important problem in real-life, which is not yet addressed in current semantics. Our model semantics provides clear disambiguation between undefined and unknown values. Furthermore, extending it with ordered epistemic logic is straightforward. We discuss the importance and benefits of our approach and provide a proof of concept for implementing it.

[5]https://bitbucket.org/krr/dmn-to-idp-poc/src/main/

Acknowledgements

This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programma.

This paper benefited from reviews and constructive comments from Maurice Bruynooghe.

References

[1] Object Modelling Group, Decision model and notation, https://www.omg.org/spec/DMN/, 2021.

[2] D. Calvanese, M. Dumas, Ü. Laurson, F. M. Maggi, M. Montali, I. Teinemaa, Semantics and analysis of dmn decision tables, in: International Conference on Business Process Management, Springer, 2016, pp. 217–233.

[3] D. Calvanese, M. Montali, M. Dumas, F. M. Maggi, Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge, Theory and practice of logic programming 19 (2019) 536–573.

[4] Bruce Silver, Dmn: Dealing with nothing, https://www.trisotech.com/dmn-dealing-with-nothing/, 2015 - 2022.

[5] D. Community, Greeting a customer with unknown data, https://dmcommunity.org/challenge/challenge-aug-2016/, 2016.

[6] S. C. Kleene, Introduction to metamathematics (1952).

[7] M. Denecker, J. Vennekens, Building a knowledge base system for an integration of logic programming and classical logic, in: International Conference on Logic Programming, Springer, 2008, pp. 71–76.

[8] B. De Cat, B. Bogaerts, M. Bruynooghe, G. Janssens, M. Denecker, Predicate logic as a modeling language: the idp system, in: Declarative Logic Programming: Theory, Systems, and Applications, 2018, pp. 279–323.

[9] P. Van Hertum, I. Dasseville, G. Janssens, M. Denecker, The kb paradigm and its application to interactive configuration, Theory and Practice of Logic Programming 17 (2017) 91–117.

[10] S. Vandevelde, V. Etikala, J. Vanthienen, J. Vennekens, Leveraging the power of IDP with the flexibility of DMN: a multifunctional API, Proceedings of RuleML+RR 2021, 2021.

[11] J. Jansen, B. Bogaerts, J. Devriendt, G. Janssens, M. Denecker, Relevance for sat (id), in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, volume 2016, AAAI Press, 2016, pp. 596–603.

[12] J. Jansen, J. Devriendt, B. Bogaerts, G. Janssens, M. Denecker, Implementing a relevance tracker module, arXiv preprint arXiv:1608.05609 (2016).

[13] S. Vandevelde, B. Aerts, J. Vennekens, Tackling the DM challenges with cDMN: A tight integration of DMN and constraint reasoning, Theory and Practice of Logic Programming (2021) 1–24. doi:10.1017/S1471068421000491.

[14] M. Denecker, Extending classical logic with inductive definitions, in: International Conference on Computational Logic, Springer, 2000, pp. 703–717.

[15] H. Vlaeminck, J. Vennekens, M. Bruynooghe, M. Denecker, Ordered epistemic logic:

Semantics, complexity and applications, in: Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning, 2012.