# Tackling the DMN Challenges with cDMN: A Tight Integration of DMN and Constraint Reasoning

Bram Aerts, Simon Vandevelde, Joost Vennekens

KU Leuven, De Nayer Campus, Dept. of Computer Science
{b.aerts, s.vandevelde, joost.vennekens}@kuleuven.be

**Abstract.** This paper describes an extension to the Decision Model and Notation (DMN) standard, called cDMN. DMN is a user-friendly, table-based notation for decision logic. cDMN aims to enlarge the expressivity of DMN in order to solve more complex problems, while retaining DMN's goal of being readable by domain experts. We test cDMN by solving the most complex challenges posted on the DM Community website. We compare our own cDMN solutions to the solutions that have been submitted to the website and find that our approach is competitive, both in readability and compactness. Moreover, cDMN is able to solve more challenges than any other approach.

## 1 Introduction

The Decision Model and Notation (DMN) [4] standard, designed by the Object Management Group (OMG), is a way of representing data and decision logic in a readable, table-based way. It is intended to be used directly by business experts without the help of computer scientists.

While DMN is very effective in modeling deterministic decision processes, it lacks the ability to represent more complex kinds of knowledge. In order to explore the boundaries of DMN, the Decision Management Community website[1] issues a monthly decision modeling challenge. Community members can then submit a solution, using their preferred decision modeling tools or programming languages. This allows solutions for complex problems to be found and compared across multiple DMN-like representations. So far, none of the available solvers have been able to solve all challenges. Moreover, the available solutions sometimes fail to meet the readability goals of DMN, because the representation is either too complex or too large.

In this paper, we propose an extension to the DMN standard, called cDMN. It allows more complex problems to be solved, while remaining readable by business users. The main features of cDMN are constraint modeling, quantification, and

[1] https://dmcommunity.org/

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

the use of concepts such as types and functions. We test the usability of cDMN on the decision modeling challenges.

In [3], we presented a preliminary version of constraint modeling in DMN. In the current paper, we extend this by adding quantification, types, functions, relations, data tables, optimization and by evaluating the formalism on the DMN challenges.

The paper is structured as follows. In Section 2 we briefly describe the DMN standard. Section 3 gives an overview of the challenges used in this paper. After this, we touch on the related work in Section 4. We discuss both syntax and semantics of our new notation in Section 5. Section 6 briefly discusses the implementation of our cDMN solver. We compare our notation with other notations and evaluate its added value in Section 7, and conclude in Section 8.

## 2 Preliminaries: DMN

The DMN standard [4] describes the structure of a DMN model. Such a model consists of two components: a Decision Requirements Diagram (DRD), which is a graph that expresses the structure of the model, and Decision Tables, which contain the in-depth business logic. An example of such a decision table can be found in Figure 1. It consists of a number of input columns (darker green) and a single output column (lighter blue). Each row is read as: if the input conditions are met (e.g., if "Age of Person" satisfies the comparison "$\geq$ 18" ), then the output expression is assigned the value of the output entry (e.g. "Person is Adult" is assigned value "Yes"). Only single values, such as strings and numbers, can be used as output entries. In the case where no row matches the input, then each output is either set to the special value *null* (which is typically taken to indicate an error in the specification) or to the output's default value, if one was provided.

The behaviour of a decision table is determined by its hit policy. There are a number of *single hit* policies, which define that a table can have at most one output for each possible input, such as "Unique" (no overlap may occur), "Any" (if there is an overlap, the outputs must be the same) and "First" (if there is an overlap, the first applicable row should be selected). There exist also *multiple hit* policies such as C (collect the output of all applicable rows in a list) and C+ (sum the output of all applicable rows). Regardless of which hit policy is used, each decision table uniquely determines the value of its output(s).

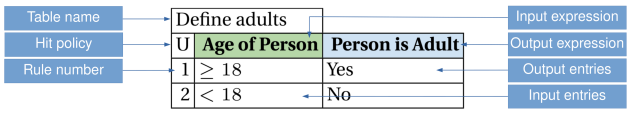


Fig. 1: Decision table to define whether a person is an adult.

The entries in a decision table are typically written in the (Simple) Friendly Enough Expression Language, or (S-)FEEL, which is also part of the DMN standard. S-FEEL allows to express simple values, lists of values, numerical

comparisons, ranges of values and calculations. Decision tables with S-FEEL are generally considered quite readable by domain experts.

In addition, DMN also allows more complex FEEL statements in combination with boxed expressions, as will be illustrated in Figure 6. However, this also greatly increases complexity of the representation, which makes it unsuitable to be used by domain experts without the aid of knowledge experts.

## 3 Challenges Overview

Of all the challenges on the DM Community website, we selected those that did not have a straightforward DMN-like solution submitted. The list of the 21 challenges that meet this criterion can be found in the cDMN documentation[2].

We categorize these challenges according to four different properties. Table 1 shows the list of properties, and the percentage of challenges that have this property.

The most frequent property is the need for aggregates (57.14%), such as counting the number of violated constraints in *Map Coloring with Violations* or summing the number of calories of ingredients in *Make a Good Burger*. The second most frequent property is having constraints in the problem description (33.33%). E.g., in *Who Killed Agatha*, the killer hates the victim and is no richer than her; or the constraint in *Map Coloring* states that two bordering countries can not share the same color. The next property, universal quantification (28.75%), is that a statement applies to every element of a type, for example in *Who Killed Agatha?*: nobody hates everyone. The final property, optimization, occurs in 23.81% of the challenges. For example, in *Zoo, Buses and Kids* the cheapest set of buses must be found.

| Property | (%) |
|---|---|
| 1. Aggregates needed | 57.14 |
| 2. Constraints | 33.33 |
| 3. Universal quantification | 28.75 |
| 4. Optimization | 23.81 |

Table 1: Percentage of occurrence of properties in challenges.

## 4 Related Work

It has been recognized that even though DMN has many advantages, it is somewhat limited in expressivity [1,3]. This holds especially for decision tables with S-FEEL, the fragment of FEEL that is considered most readable. While full FEEL is more expressive, it is not suitable to be used by domain experts without the aid of knowledge experts. Moreover, it does not provide a solution to other shortcomings, such as the lack of constraint reasoning and optimization.

One of the systems that does effectively support constraint solving in a readable DMN-like representation is the OpenRules system [5]. It enables users to

[2] `https://cdmn.readthedocs.io/en/latest/community.html`

define constraints over the solution space by allowing "Solver Tables" to be added alongside decision tables. In contrast to standard decisions, which assign a specific value to an output, Solver Tables allow for setting constraints on the output space. OpenRules offers a number of *DecisionTableSolve-Templates*, which can be used to specify these constraints. It is possible to either use these predefined templates, or define such a template manually if the predefined ones are not expressive enough. Even though this system extends the range of applications that can be handled, there are three reasons why it does not offer the ease of use for business users that we are after. First, because of the wide range of available templates for solver tables, which differ from that of standard decision tables, using the OpenRules constraint solver entails a steep learning curve. Second, the solver's functionality can only be accessed through the Java API, which goes against the DMN philosophy [4, p. 13]. Third, because of the lack of quantification in OpenRules, solutions are generally not independent of domain size, which reduces readability.

Another system that aims to increase expressiveness of DMN is Corticon [6]. It implements a basic form of constraint solving by allowing the user to filter the solution space. While this approach indeed improves expressiveness, it decreases readability. Moreover, some constraints can only be expressed by combining a number of rules and a number of filters. For example, when expressing "all female monkeys are older than 10 years", this is split up in two parts; (1) a rule that states `Monkey.gender = female & Monkey.Age < 10 THEN Monkey.illegal = True` and (2) a filter that states that a monkey cannot be illegal: `Monkey.illegal = False`. There are no clear guidelines about which part of the constraints should be in the filter and what should be a rule.

In [1], Calvanese et al propose an extension to DMN which allows for expressing additional domain knowledge in Description Logic. They share our goal of extending DMN to express more complex real-life problems. However, they introduce a completely separate Description Logic formalism, which seems too complex for a domain expert to use. Unfortunately, they did not submit any solutions to the DMN Challenges, which leaves us unable to compare its expressiveness in practice.

## 5 cDMN: Syntax & Semantics

While DMN allows users to elegantly represent a deterministic decision process, it lacks the ability to specify constraints on the solution space. The cDMN framework extends DMN, by allowing constraints to be represented in a straightforward and readable manner. It also allows for representations that are independent of domain size by supporting types, functions, relations and quantification.

We now explain both the usage and the syntax of every kind of table present in cDMN.

### 5.1 Glossary

In logical terms, the "variables" used in standard DMN correspond to constants (i.e., 0-ary functions). cDMN extends these by adding n-ary functions and n-ary relations. Similarly to OpenRules and Corticon, we allow the user to define their vocabulary by means of a glossary. This glossary contains every symbol used in a cDMN model. It consists of at most five glossary tables, each enumerating a different kind of symbol. An example glossary for the *Who Killed Agatha?* challenge is given in Figure 2.

| Type | | |
|---|---|---|
| **Name** | **Type** | **Values** |
| Person | string | Agatha, Butler, Charles |
| Number | int | [0..100] |

| Relation |
|---|
| **Name** |
| Person hates Person |
| Person is richer than Person |

| Boolean |
|---|
| **Name** |
| Suicide |

| Constant | |
|---|---|
| **Name** | **Type** |
| Killer | Person |

| Function | |
|---|---|
| **Name** | **Type** |
| Hatees of Person | Number |

Fig. 2: An example cDMN glossary for the *Who Killed Agatha?* problem.

In the *Type* table, *type* symbols are declared. The value of each type is a set of domain elements, specified either in the glossary or in a data table (see section 5.3). An example is the type `Person`, which contains the names of people.

In the *Function* table, a symbol can be declared as a *function* of one or more types to another. The infix operator `of` is used to apply the function to its argument(s). For example, the `Hatees of Person` function denotes how many people a person hates. It maps each element of type `Person` to an element of type `Number`. Functions with $n > 1$ arguments can be declared by separating the $n$ arguments by the keyword `and`.

For each domain element, a constant with the same name is automatically introduced, which allows the user to refer to this domain element in constraint or decision tables. For instance, the user can use the constant `Agatha` to refer to the domain element `Agatha`. In addition, the *Constant* table allows other constants to be introduced. Recall that such logical constants correspond to standard DMN variables. In our example case, we use a constant `Killer` of the type `Person`, which means it can refer to any of the domain elements `Agatha`, `Butler` or `Charles`.

In the *Relation* table, a verb phrase can be declared as a *relation* on one or more given types. For instance, the relation `Person is Adult` denotes for each `Person` whether they are an adult. This relation translates to the unary predicate *isAdult*. n-ary predicates can be defined by using $n$ arguments in the name, e.g. `Person is richer than Person` is a relation with two arguments (both of the type `Person`), that denotes whether one person is richer than another.

The *Boolean* table contains *boolean* symbols (i.e. propositions), which are either true or false. An example is the boolean `Suicide`, which denotes whether the murder is a suicide.

### 5.2 Decision Tables and Constraint Tables

As stated earlier in Section 2, a standard decision table uniquely defines the value of its outputs. We extend DMN by allowing a new kind of table, called a *constraint table*, which does not have this property.

Whereas decision tables only allow single values to appear in output columns, our constraint tables allow arbitrary S-FEEL expressions in output columns, instead of only single values. Each row of a constraint table represents a logical *implication*, in the sense that, if the conditions on the inputs are satisfied, then the conditions on the outputs must also be satisfied. This means that if, for instance, none of the rows are applicable, the outputs can take on an arbitrary value, as opposed to being forced to *null*. In constraint tables, no default values can be assigned. Because of these changes, a set of cDMN tables does not define a single solution, but rather a solution space containing a set of possible solutions.

We introduce a new *hit policy* to identify constraint tables. We call this the *Every* hit policy, denoted as *E**, because it expresses that every implication in the table must be satisfied. An example of this can be found in Figure 3, which states that each person hates less than 3 people.

cDMN does not only introduce constraint tables, it also extends the expressions that are allowed in column headers, both in decision and constraint tables. Such a header can consist of the following expressions: (1) a type *Type*; (2) an expression of the form "*Type* called *name*"; (3) a constant; (4) an expression of the form "*Function* of $arg_1$ and ... and $arg_n$", where each of the $arg_i$ is another header expression; (5) an arithmetic combination of header expressions (such as a sum).

The first two kinds of expressions are called *variable* header expressions. They allow *universal quantification* in cDMN. Each input column whose header consists of such a *variable* expression either introduces a new universally quantified variable (we call this a *variable-introducing* column), or refers back to a variable introduced in a preceding *variable-introducing* column. Subsequent uses of the same type name (in case of the first kind of variable-introducing expression) or of the variable name (in case of the second kind) then refer back to this universally quantified variable. Whenever a type or variable name appears in a header of a column that is itself not variable-introducing, a unique preceding variable-introducing column that has introduced this variable must exists.

The table in Figure 3 shows an example of quantification in cDMN. It introduces a universally quantified variable of the type *Person*, stating that every person hates less than three others. To illustrate the use of named variables, Figure 4 defines variables `c1` and `c2`, both of the type `Country`, and states that when those countries are bordering, they cannot have the same color.

In summary, this section has discussed three ways in which cDMN extends DMN. First, the hit policy *E** changes the semantics of the table. Second, constraint tables allow S-FEEL expressions in the output columns. Third, cDMN allows quantification, functions, predicates and calculations to be used in both decision tables and constraint tables.

| Noone hates all | | |
|---|---|---|
| E* | **Person** | **Hatees of Person** |
| 1 | - | <3 |

Fig. 3: Part of the implementation of "Nobody hates everyone" in *Who Killed Agatha?*.

| Bordering countries can not share colors | | | | |
|---|---|---|---|---|
| E* | **Country called c1** | **Country called c2** | **c1 and c2 are Bordering** | **Color of c1** |
| 1 | - | - | Yes | Not(Color of c2) |

Fig. 4: Example of a constraint table with quantification in cDMN, defining that bordering countries can't share colors.

### 5.3 Data Tables

Typically, problems can be split up into two parts: (1) the general logic of the problem, and (2) the specific problem instance that needs to be solved. Take for example the map coloring problem: the general logic consists of the rule that two bordering countries cannot share a color, whereas the instance of the problem is the specific map (e.g. Western Europe) to color. cDMN extends the DMN standard to include *data tables*, which are used to represent the problem instances, separating them from the general logic. The format of a data table closely resembles that of a decision table, with a couple of exceptions. Instead of a hit policy, a data table has "data table" in its name. Furthermore, only basic values (integers, floats and domain elements) are allowed in data tables. It is also possible for columns to have more than one value in a certain row, in which case the row is instantiated for the combination of each of the values of the columns. As an example, a snippet of the data table for the *Map Coloring* challenge is shown in Figure 5.

This use of data tables offers several advantages.

1. There is a methodological advantage: by separating the data tables from the decision tables, reusing the specification becomes easier.
2. If the user chooses to enumerate the domain of a type in the glossary, then the system checks that each value in a data table indeed belongs to the domain of the appropriate type. This helps to prevent errors or typos in the input data or glossary. If the user chooses not to enumerate a type in the glossary, then the type's domain defaults to the set of all values in the data table.
3. The cDMN solver is able to compute solutions faster, due to a different internal representation between data tables and decision tables.

| Data Table: Declaring which countries border | | | |
|---|---|---|---|
| | **Country called c1** | **Country called c2** | **c1 and c2 are Bordering** |
| 1 | Belgium | France, Luxembourg, Netherlands, Germany | Yes |
| 2 | Germany | France, Denmark, Luxembourg, Belgium, Netherlands | Yes |

Fig. 5: Data table describing countries and their neighbours

### 5.4 Execute Table

A standard DMN model defines a deterministic decision procedure. It is typically always used in the same way: the external inputs are supplied by the user, after which the values of the output variables are computed by forward propagation.

In cDMN, this is no longer the case. We can fill in as many or as few variables as we want, and use the model to derive useful information about the not-yet-known variables. By employing an *execute table*, users can specify what the model is to be used for: model expansion or optimization. Model expansion creates a given number of solutions, and optimization looks for the solution with either the lowest or highest value for a given term.

### 5.5 Semantics of cDMN

We describe the semantics of cDMN by translating it to the FO($\cdot$) language used by the IDP system [2,7]. FO($\cdot$) is a rich extension of First Order Logic, adding concepts such as types, aggregates and inductive definitions. The semantics of cDMN is defined by the semantics of each of its sub-components.

It is straightforward to translate the glossary into an FO($\cdot$) vocabulary: types, functions, constants, relations and booleans are each translated to their FO($\cdot$) counterpart.

Decision tables retain their usual semantics as described by Calvanese [1]. We briefly recall this semantics. Each cell of a decision table $(i, j)$ corresponds to a formula $F_{ij}(x)$ in one free variable $x$. For instance, a cell "$\leq 50$" corresponds to the formula "$x \leq 50$". A decision table with rows $R$, input columns $I$ and output columns $O$ is a conjunction of material implications:

$$\bigwedge_{i \in R} \left( \bigwedge_{j \in I} F_{ij}(H_j) \Rightarrow \bigwedge_{k \in O} F_{ik}(H_k) \right)$$

where $H_j$ is the header of column $j$. For example, the table in Figure 1 corresponds to the logical formula $(\mathit{AgeOfPerson} \geq 18 \Rightarrow \mathit{PersonIsAdult} = \mathit{Yes}) \wedge (\mathit{AgeOfPerson} < 18 \Rightarrow \mathit{PersonIsAdult} = \mathit{No})$.

Data tables are simply a specific case of decision tables.

In [3], the semantics of simple constraint tables (without quantification and functions) is introduced, which is also a conjunction of implications. The semantics of constraint tables and decision tables differ in the interpretation of incomplete tables: when no rows are applicable in decision tables, the output is forced to *null* (i.e., the implicit default value is *null*), while the output in constraint tables can take any value.

Now we extend this semantics to take variables and quantification into account. Our first step is to define a function that maps cDMN expressions to terms. For the most part, this definition corresponds to that of Calvanese [1]. However, we extend it to take into account the fact that certain expressions – which we call *variable expressions* – must be translated to FO variables. There are three kinds of variable expressions. We now define a mapping $\nu$ that maps each of these three kinds of cDMN variable expressions to a typed FO variable $x$ of type $T$, which we denote as $x[T]$:

- The name $T$ of a type is a variable expression. We define $\nu(T) = x_T[T]$, with $x_T$ a new variable of type $T$.
- An expression $e$ of the form "*Type* called $v$" is a variable expression. We define $\nu(e) = v[\mathit{Type}]$.
- If the header of a column contains an expression "*Type* called $v$", then $v$ is a variable expression in all subsequent columns of the table and in its body. We define $\nu(v)$ as $v[\mathit{Type}]$.

Given this function $\nu$, we now define the following mapping $t_\nu(\cdot)$ of cDMN expressions to terms.

- For a constant $c$, $t_\nu(c) = c$; similarly, for an integer or floating point number $n$, $t_\nu(n) = n$;
- For an arithmetic expression $e$ of the form $e_1 \theta e_2$ with $\theta \in \{+, -, *, /\}$, we define $t_\nu(e) = t_\nu(e_1)\ \theta\ t_\nu(e_2)$;
- For a variable expression $v$, we define $t_\nu(v) = \nu(v)$.
- For a function expression, i.e. "*Function* of $arg_1$ and ... and $arg_n$": $t_\nu(X) = \mathit{Function}(t_\nu(arg_1), ...., t_\nu(arg_n))$ .

Similarly to Calvanese, we translate each entry $c$ in a cell $(i, j)$ of a table into a formula $F_{ij}(x)$ in one free variable x:

- If $c$ is of the form "$\theta e$" with $\theta$ one of the relational operators $\{\leq, \geq, =, \neq\}$, then $F_{ij}(x)$ is the formula $x\ \theta\ t(e)$;
- If $c$ is of the form *Not* $e$, then $F_{ij}$ is $x \neq t(e)$;
- If $c$ is a list $e_1, \ldots, e_n$, then $F_{ij}$ is $x = t(e_1) \vee \ldots \vee x = t(e_n)$. As a special case, if $c$ consists of a single expression e, then $F_{ij}$ is $x = t(e)$.
- If $c$ is a range, e.g. $[e_1, e_2)$, then $F_{ij}$ is $x \geq t(e_1) \wedge x < t(e_2)$.

We are now ready to define the semantics of a constraint table. If I is the set of input columns of the table, $O$ the set of output columns and $V \subseteq I$ the set of variable introducing columns, we define the semantics of the table as:

$$\underset{l \in V}{\forall} \nu(H_l) : \bigwedge_{i \in R} \Big( \bigwedge_{j \in I} F_{ij}\big(t_\nu(H_j)\big) \Rightarrow \bigwedge_{k \in O} F_{ik}\big(t_\nu(H_k)\big) \Big)$$

For example, in Figure 3, $\nu(H_1) = x[\mathit{Person}]$ and $t_\nu(H_1) = x$, $t_\nu(H_2) = \mathit{Hatee}(t_\nu(H_1)) = \mathit{Hatee}(x)$, which leads to the formula:

$$\forall x[\mathit{Person}] : \mathit{Hatee}(x) < 3.$$

The above transformation turns each decision or constraint table $T$ into an FO$(\cdot)$ formula $\phi_T$. The glossary and data tables together define a structure $S$ for part of the vocabulary. The domain of $S$ consists of the union of the interpretations $I_t$ of all the types $t$. If $t$ is enumerated in the glossary, then $I_t$ is this enumeration. Otherwise, $I_t$ consists of all the values that appear in a data table in a column of type $t$. The structure $S$ interprets all the relations / functions for which a data table is provided, and it interprets them by the set of tuples / the mapping that is given in this table.

The set of "solutions" of a cDMN model is the set $MX(\Phi, S)$ of all model expansions of the structure $S$ w.r.t. the theory $\Phi = \{\phi_T \mid T$ is a constraint or decision table$\}$, i.e., the set of all structures $S' \models \Phi$ that extend $S$ to the entire vocabulary.

## 6 Implementation

This section gives an overview of the inner workings of the cDMN solver[3]. It is a brief overview, as the solver is not the main focus of this paper. The solver consists of two parts: a constraint solver (the IDP system), and a converter from cDMN to IDP input. In principle, any constraint solver could be used, but we chose the IDP system because of its flexibility.

The cDMN to IDP converter is built using Python3, and works in a two-step process. It first interprets all tables in a `.xslx` sheet and converts them into Python objects. For example, the converter parses all the glossary tables and converts them into a single `Glossary` object, which then creates `Type` and `Predicate` objects. The created Python objects are then converted into IDP blocks. More detailed information about this conversion can be found in the cDMN documentation[4], along with an explanation of the usage of the solver and concrete examples of cDMN implementations.

## 7 Results and discussion

In this section we first look at three of the DM Community challenges, each showcasing a feature of cDMN. For each challenge, we compare the DMN implementations from the DM Community website with our own implementation in cDMN. Afterwards, we compare all challenges on size and quality.

### 7.1 Constraint tables


Fig. 6: An extract of the map coloring solution in standard DMN with FEEL

Constraint tables allow cDMN to model constraint satisfaction problems in a straightforward way. For example, in *Map Coloring*, a map of six European

[3] `https://gitlab.com/EAVISE/cdmn/cdmn-solver`
[4] `www.cdmn.be`

countries must be colored in such a way that no neighbouring countries share the same color. For this challenge, a pure DMN implementation was submitted, of which Figure 6 shows an extract. The implementation uses complicated FEEL statements to solve the challenge. While these statements are DMN-compliant, they are nearly impossible for a business user to write without help. In cDMN, we can use a single straight-forward constraint table to solve this problem, as shown in Figure 4. Together with the glossary and a data table (Figure 5), this forms a complete yet simple cDMN implementation.

### 7.2 Quantification

Quantification is useful in the *Monkey Business* challenge. In this challenge, we want to know for four monkeys what their favorite fruit and their favorite resting place is, based on some information. There are two DMN-like submissions for this challenge: one using Corticon, and one using OpenRules.

| DecisionTable SituationRules | | | | | |
|---|---|---|---|---|---|
| ConditionVarOperValue | | | ConclusionVarOperValue | | |
| String attribute | Oper op | String value | String attribute | Oper op | String value |
| ... | ... | ... | ... | ... | ... |
| Mike's Resting Place | = | Rock | Mike's Fruit | = | Apple |
| Sam's Resting Place | = | Rock | Sam's Fruit | = | Apple |
| Anna's Resting Place | = | Rock | Anna's Fruit | = | Apple |
| Harriet's Resting Place | = | Rock | Harriet's Fruit | = | Apple |
| ... | ... | ... | ... | ... | ... |

(a) Open Rules

| Monkey Constraints | | | |
|---|---|---|---|
| E* | **Monkey** | **Place of Monkey** | **Fruit of Monkey** |
| ... | ... | ... | ... |
| 2 | — | Rock | Apple |
| ... | ... | ... | ... |

(b) cDMN

Fig. 7: An extract of *Monkey Business* implementation in (a) OpenRules and (b) cDMN, specifying "The monkey who sits on the rock is eating the apple".

One of the pieces of information is: `The monkey who sat on the rock ate the apple.` The OpenRules implementation has a table with a row for each monkey, which states that if this monkey's resting place was a rock, their fruit was an apple (Figure 7a). In other words, for $n$ monkeys, the OpenRules implementation of this rule requires $n$ lines. Because of quantification, cDMN requires only one row, regardless of how many monkeys there are (Figure 7b). The Corticon implementation also uses a similar quantification for this rule.

Another rule states that no two monkeys can have the same resting place or fruit. In both the Corticon and OpenRules implementations, this is handled by two tables with a row for each pair of monkeys. The Corticon tables are shown in Figure 8a. Each row either states that two monkeys have different fruit, or that they have different place. Therefore, $n$ monkeys require $\frac{n\times(n-1)}{2}$ rows. By contrast, the cDMN implementation seen in Figure 8b requires only a single row to express the same.

| | |
|---|---|
| Mike.fruit<>Sam.fruit | T |
| Mike.fruit<>Harriet.fruit | T |
| Mike.fruit<>Anna.fruit | T |
| Sam.fruit<>Harriet.fruit | T |
| Sam.fruit<>Anna.fruit | T |
| Harriet.fruit<>Anna.fruit | T |

| | |
|---|---|
| Mike.place<>Sam.place | T |
| Mike.place<>Harriet.place | T |
| Mike.place<>Anna.place | T |
| Sam.place<>Harriet.place | T |
| Sam.place<>Anna.place | T |
| Harriet.place<>Anna.place | T |

(a) Corticon

| Different Preferences | | | | |
|---|---|---|---|---|
| E* | **Monkey called m1** | **Monkey called m2** | **Place of m1** | **Fruit of m1** |
| 1 | — | not(m1) | not(Place of m2) | not(Fruit of m2) |

(b) cDMN

Fig. 8: An extract of the *Monkey Business* implementation in (a) Corticon and (b) cDMN, defining that no monkeys share fruit and no monkeys share the same place.

### 7.3 Optimization

In the *Balanced Assignment* challenge, 210 employees need to be divided into 12 groups, so that every group is as diverse as possible. The department, location, gender and title of each employee is known. This is quite a complex problem to handle in DMN. As such, of the four submitted solutions, only one was DMN-like: an OpenRules implementation, using external CP/LP solvers. The logic for these external solvers is written in Java. Although the code is fairly compact, it cannot be written without prior programming knowledge. Because optimization is built-in in cDMN, we can solve the problem with two decision tables and one constraint table. The table *Diversity score*, shown in Figure 9, adds 1 to the total diversity score if two similar people are in a different group. Maximizing this score then results in the most diverse groups.

| Diversity score | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| C+ | **Person called p1** | **Person called p2** | **Department of p1** | **Location of p1** | **Gender of p1** | **Title of p1** | **Group of p1** | **Score** |
| 1 | - | - | = Department of p2 | - | - | - | not(Group of p2) | 1 |
| 2 | - | - | - | = Location of p2 | - | - | not(Group of p2) | 1 |
| 3 | - | - | - | - | = Gender of p2 | - | not(Group of p2) | 1 |
| 4 | - | - | - | - | - | = Title of p2 | not(Group of p2) | 1 |

| **Execute** |
|---|
| Maximize Score |

Fig. 9: The decision tables and constraint table for Balanced Assignment.

### 7.4 Overview of all challenges

Of the 21 challenges we considered, cDMN is capable of successfully modeling 19. In comparison, there were 12 OpenRules implementations and 12 Corticon implementations submitted. Note that we have not examined whether OpenRules and Corticon might be capable of modeling more challenges than those for which a solution was submitted.

To compare cDMN to other approaches, we focus on two aspects: quantity (how big are they?) and quality (how readable and how scalable are they?). The size of implementations was measured by counting the number of cells used in all the decision tables. Glossaries, data tables and equivalents thereof were not included in the count. Table 2 shows that cDMN and Corticon alternate between having the fewest cells, and that OpenRules usually has the most. In general, OpenRules implementations require many cells because each cell is very simple. For instance, even an "=" operator is its own cell. The Corticon implementations, on the other hand, contain more complex cells, rendering them more compact.

| | Who Killed A.? | Change Making | A Good Burger | Define Dupl. Coll. of Cars | Monkey Business | Vacation Days | Family Riddle | Cust. Greeting | Online Dating | Class. Employees | Reindeer Order | Zoo, Buses, Kids | Balanced Assign. | Vac. Days Adv. | Map Coloring | Map Color Viol. | Crack the Code | Numerical Haiku |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cDMN | 53 | 26 | 35 | 20 | 26 | 47 | 38 | 76 | 88 | 45 | 36 | 14 | 24 | 55 | 124 | 21 | 48 | 77 | 41 |
| Corticon | 54 | 14 | 20 | 19 | 45 | 64 | 32 | 22 | | 78 | 21 | 64 | | | | | | | |
| OpenRules | 176 | | 95 | 21 | | 150 | 31 | | 205 | | 70 | 111 | 43 | 30 | 97 | | | | |
| Others | | | 76[1] | | 48[1] | | 14[2] | | | | 34[3] | 370[4] | | | | 31[4] | | | |

Table 2: Comparison of the number of cells used per implementation. Other implementations: 1. FEEL, 2. Blueriq, 3. Trisotech, 4. DMN

Because of this, OpenRules implementations are usually easier to read than their Corticon counterparts. An example comparison between cDMN and Corticon can be seen in Figure 10a and 10b. Each figure shows a snippet of their *Make a Good Burger* implementation, in which the food properties of a burger are calculated. While the Corticon implementation is more compact, it is less interpretable, less maintainable and dependent on domain size. If the user wants to add an ingredient to the burger, complex cells need to be changed. In cDMN, simply adding the ingredient to the data table suffices.

A comparison between cDMN and OpenRules can be found in Figure 11a and 11b. Here we show a part of their implementations of the *Who Killed Agatha?* challenge. They both show a translation of the following rule: "A killer always hates, and is no richer than his victim." By using constraints and a constant (`Killer`), cDMN allows us to form a more readable and more scalable table. If the police ever find a fourth suspect, they can easily add the person to the data table without needing to change anything else.

In Section 3, we identified four different problem properties. We now suggest that each property is tackled more easily by one or more of the additions cDMN proposes.

**Aggregates needed** Figure 10b shows how aggregates are both more readable and scalable when using quantification. Moreover, cDMN allows the use of aggregates for more complex operations such as optimization or defining constraints.

| | Conditions | 1 |
|---|---|---|
| a | beefPatties.count * 50 + bun.count * 330 + cheese.count * 310 + onion.count + ketchup_lettuce.count * (3+160) + pickle_tomato.count * (260+3) | < 3000 |
| b | beefPatties.count * 17 + bun.count * 9 + cheese.count * 6 + onion.count*2 | < 150 |
| c | beefPatties.count * 220 + bun.count * 260 + cheese.count * 70 + onion.count*10 + ketchup_lettuce.count * (4+20) + pickle_tomato.count * (5+9) | < 3000 |

(a) Corticon

| Determine Nutrition | | | | | |
|---|---|---|---|---|---|
| C+ | **Item** | **Total Sodium** | **Total Fat** | **Total Calories** | **Total Cost** |
| 1 | - | Number of Item * Sodium of Item | Number of Item * Fat of Item | Number of Item * Calories of Item | Number of Item * Cost of Item |

| Nutrition Constraints | | | |
|---|---|---|---|
| E* | **Total Sodium** | **Total Fat** | **Total Calories** |
| 1 | <3000 | <150 | <3000 |

(b) cDMN

Fig. 10: Calculating the food properties of burger in Corticon and cDMN.

**Constraints** Constraints can be conveniently modeled by constraint tables, such as the constraints in Figure 11b, which states that the killer hates Agatha, but is no richer than her. The addition of constraint tables allows for an obvious translation from the rule in natural language to the table.

| DecisionTable KillerHatesAndNoRicherHisVictim | | | | | |
|---|---|---|---|---|---|
| ConditionXoperY | | | ActionXoperY | | |
| IF | | | THEN | | |
| BUTLER KILLED AGATHA | = | 1 | Butler Hates Agatha | = | 1 |
| BUTLER KILLED AGATHA | = | 1 | Butler Richer Than Agatha | = | 0 |
| CHARLES KILLED AGATHA | = | 1 | Charles Hates Agatha | = | 1 |
| CHARLES KILLED AGATHA | = | 1 | Charles Richer Than Agatha | = | 0 |
| AGATHA KILLED AGATHA | = | 1 | Agatha Hates Agatha | = | 1 |
| AGATHA KILLED AGATHA | = | 1 | Agatha Richer Than Agatha | = | 0 |

(a) OpenRules

| Killer constraints | | |
|---|---|---|
| E* | **Killer hates Agatha** | **Killer richer than Agatha** |
| 1 | Yes | No |

(b) cDMN

Fig. 11: Implementation of "A killer always hates and is no richer than their victim" in OpenRules and cDMN

**Universal quantification** Problems which contain universal quantification can be compactly represented, as can, among others, be seen in Figure 3. This table states that each person hates less than 3 people.

**Optimization** Because cDMN directly supports optimization, problems containing this property are easily modeled. Furthermore, by the addition of more complex data types, optimization can be used in a more flexible manner. An example can be found in Figure 9.

## 8 Conclusions

This paper presents an extension to DMN, which is able to solve complex problems while still maintaining DMN's level of readability. This extension, which we call cDMN, adds constraint modeling, more expressive data and quantification.

Constraint modeling allows a user to define a solution space instead of a single solution. A user can generate a desired number of models, or generate the model which optimizes the value of a specific term. Unlike DMN, which only knows constants, cDMN also supports the use of functions and predicates, which allow for more flexible representations. Together with quantification, this allows tables to be constructed in a compact and straightforward manner, while being independent of the size of the problem. This improves readability, maintainability and scalability of tables.

By comparing our cDMN implementations to the implementations of other state-of-the-art DMN-like solvers, we can conclude that cDMN succeeds in increasing expressivity while retaining the simplicity of standard DMN.

## References


1. Calvanese, D., Montali, M., Dumas, M., Maggi, F.: Semantic dmn: Formalizing and reasoning about decisions in the presence of background knowledge. Theory and Practice of Logic Programming **19**(4), 536–573 (2019)
2. De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., Denecker, M.: Predicate logic as a modeling language: The idp system. In: Declarative Logic Programming: Theory, Systems, and Applications, pp. 279–329. ACM Books (2018)
3. Deryck, M., Aerts, B., Vennekens, J.: Adding constraint tables to the dmn standard: Preliminary results. In: Rules and Reasoning: Third International Joint Conference, RuleML+RR 2019, Bolzano, Italy, September 16–19, 2019, Proceedings. vol. 11784, pp. 171–179. Springer (2019)
4. Object Modelling Group: Decision model and notation (2019), `http://www.omg.org/spec/DMN/`
5. OpenRules, Inc.: Openrules (2017), `http://openrules.com`
6. Progress: Corticon (2019), `progress.com/corticon`
7. Wittocx, J., Mariën, M., Denecker, M.: The idp system: a model expansion system for an extension of classical logic. In: Proceedings of the 2nd Workshop on Logic and Search. pp. 153–165. ACCO; Leuven (2008)