TPLP **23** (3): 535–558, 2023. © The Author(s), 2021. Published by Cambridge University Press. 535
doi:10.1017/S1471068421000491 First published online 12 November 2021

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

SIMON VANDEVELDE, BRAM AERTS and JOOST VENNEKENS
KU Leuven, De Nayer Campus, Department of Computer Science
J.-P. De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium
Leuven.AI - KU Leuven Institute for AI, B-3000 Leuven, Belgium
(*e-mails:* s.vandevelde@kuleuven.be, b.aerts@kuleuven.be, joost.vennekens@kuleuven.be)

submitted 18 December 2020; revised 6 October 2021; accepted 11 October 2021

Abstract

Knowledge-based AI typically depends on a knowledge engineer to construct a formal model of domain knowledge – but what if domain experts could do this themselves? This paper describes an extension to the Decision Model and Notation (DMN) standard, called Constraint Decision Model and Notation (cDMN). DMN is a user-friendly, table-based notation for decision logic, which allows domain experts to model simple decision procedures without the help of IT staff. cDMN aims to enlarge the expressiveness of DMN in order to model more complex domain knowledge, while retaining DMNs goal of being understandable 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. Moreover, cDMN is able to solve more challenges than any other approach.

KEYWORDS: decision model and notation, constraint reasoning, expressiveness, readability, IDP system

1 Introduction

The Decision Model and Notation (DMN) (Object Management Group 2020) standard, designed by the Object Management Group (OMG), is a way of representing data and decision logic in a table-based way. It is intended to be used directly by business experts without the help of computer scientists , and as such, aims to be low in complexity and user-friendly.

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 modelling challenge. Community members can then submit a solution, using their preferred

* This research received funding from the Flemish Government under the Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen programme.
[1] https://dmcommunity.org/

decision modelling 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, too large, or requires a specific computer science background.

In this paper, we propose an extension to the DMN standard, called cDMN. It aims to allow more complex knowledge to be represented, while remaining readable by business users. The main features of cDMN are constraint modeling, quantification, and the use of concepts such as types and functions. We test the expressiveness of cDMN on the decision modeling challenges.

In Deryck *et al.* (2019), we presented a preliminary framework for constraint modeling in DMN. In the current paper, we extend this by adding quantification, types, functions, relations, data tables, optimization, and by evaluating the resulting cDMN formalism on the DMN challenges.

This paper is an extended version of a paper we presented at the RuleML+RR 2020 conference (Aerts *et al.* 2020). It includes an updated list of challenges, changes to the semantics to make it more complete, a more in-depth description of the solver and a section on the integration of DMN into business models.

It 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 (Object Management Group 2020) describes the structure of a DMN model. The aim of the standard is to provide a user-friendly modeling notation for decision logic. It is suitable for use by business experts (Silver 2018), low in complexity (Hasic *et al.* 2017), and has already been successfully used in many case studies (Sooter *et al.* 2019; Car 2018; Hasic and Vanthienen 2020).

A DMN model consists of two components: a Decision Requirements Diagram (DRD) and a number of decision tables. The DRD is a graph that expresses the structure of a DMN model by representing the connections between inputs, decisions, knowledge sources, and more. At its core, it is a visual representation of the general structure of the model, depicting which concepts are defined in terms of which other concepts. As such, it improves the interpretability of models by end-users. Furthermore, it also enhances the traceability of decisions, as it becomes easier to see which variables influence a specific decision, and in which order different decisions depend on each other.

An example of a DRD is shown in Figure 1, which represents the decision process for the cost of an entry ticket for a museum. The two ellipses represent input data, that is, a person's age and the name of the exhibit they want to visit. Each rectangle in the graph represents a decision that needs to be made, for example, deciding whether a person is an adult.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

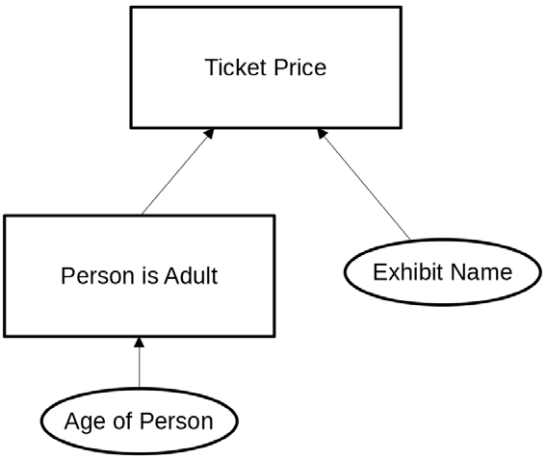

Fig. 1. Decision Requirement Diagram to decide a museum ticket price in DMN.

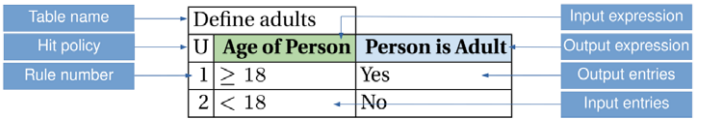

Fig. 2. Decision table to define whether a person is an adult.

The decision tables contain the in-depth business logic of a model. An example of such a decision table can be found in Figure 2. It consists of a number of input columns (darker green) and at least one 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 variable 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 behavior of a decision table is determined by its hit policy. There are a number of *single hit* policies, which cause the output variable(s) to take on a single value, even when multiple rows are applicable. In particular, these hit policies are as follows: "Unique" (for each possible set of input values, at most one row is applicable), "Any" (if multiple rows are applicable for the same input values, their outputs must be the same), and "First" (if more than one row is applicable, the first applicable row determines the value of the outputs). There exist also *multiple hit* policies such as "Collect" (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).

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 simple values, lists of values, numerical comparisons, ranges of values and arithmetic expressions. Decision tables with S-FEEL are generally considered quite readable by domain experts.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Table 1. *List of DM Community challenges and their properties. 1: Universal Quantification, 2: Constraints, 3: Optimization, 4: Need for Aggregates*

Challenge	Property	Challenge	Property
Who Killed A.?	1	Change Making	3, 4
A Good Burger	2, 3, 4	Define Dupl.	None
Coll. of Cars	None	Monkey Business	None
Vacation Days	1, 2, 4	Family Riddle	2, 4
Cust. Greeting	None	Online Dating	None
Loan Approval	4	Class. Employees	4
Soldier Payment	4	Reinder Order	None
Zoo, Buses, Kids	3, 4	Balanced Assign.	3
Vac. Days Adv.	1, 2, 4	Map Coloring	1, 2
Map Color Viol.	1, 2, 3, 4	Crack The Code	4
Numerical Haiku	1, 2, 4	Nim Rules	2
Doctor Planning	1, 2, 4	Calculator	1, 3

Table 2. *Percentage of occurrence of properties in challenges*

Property	(%)
1. Aggregates needed	54.17
2. Constraints	37.50
3. Universal quantification	33.33
4. Optimization	25.00

In addition, DMN also allows more complex FEEL statements in combination with boxed expressions, as will be illustrated in Figure 11. However, this also greatly increases complexity of the representation, which makes it unsuitable for use by domain experts without the aid of knowledge engineers.

3 Challenges overview

Of all the challenges on the DM Community website, we selected those that did not have a straightforward DMN-like solution. The list of the 24 challenges that meet this criterion can be found in Table 1. We categorize these challenges according to four different properties. Table 2 shows the list of properties, and the percentage of challenges that have this property.

The most frequent property is the need for aggregates (54.17%), 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 (37.50%). For instance, the constraint in *Map Coloring* states that two bordering countries cannot share the same color. The next property, universal quantification (33.33%), 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 25.00% of the challenges. For example, in *Zoo, Buses, and Kids* the cheapest set of buses must be found.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

The description of each challenge can be found on the DMCommunity website[2], together with their submissions. We also maintain a mirror repository[3] containing the specific challenges and submissions used in this work.

4 Related work

It has been recognized that even though DMN has many advantages, it is somewhat limited in expressiveness (Calvanese *et al.* 2019; Deryck *et al.* 2019). 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 engineers. 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 (OpenRules, Inc. 2017). It enables modellers to 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 to 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 (Object Management Group 2020, 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 (Progress 2019). It implements a basic form of constraint solving by allowing the modeler 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 that if `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. A more detailed comparison between OpenRules, Corticon and cDMN is given in Section 7.

Calvanese *et al.* (2019) propose an extension to DMN which allows for expressing additional domain knowledge in Description Logic, which would not be possible to model in DMN. In this way, they share our goal of extending DMN to express more complex real-life problems. However, they introduce a completely sepa-

[2] https://dmcommunity.org/challenge/
[3] https://gitlab.com/EAVISE/cdmn/DMChallenges

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Type

Name	Type	Values
Doctor	String	Fleming, Freud, Heimlich, Eustachi, Golgi
Day	String	Mon, Tue, Wed, Thur, Fri, Sat, Sun
Time	String	Early, Late, Night
Number	int	[0..21]

Function

Name	Type
present doctor on Day at Time	Doctor
nb shifts of Doctor on Day	Number
nb nights of Doctor	Number
day after Day	Day

Relation

Name
Doctor is on leave
Doctor is available on Day at Time

Boolean

Name
Complete

Constant

Name	Type
Head	Doctor

Fig. 3. An example cDMN glossary for the *Doctor Planning* problem.

rate Description Logic formalism, which may be too complex for a domain expert to use. While this approach makes sense if, for example, a Description Logic ontology for the domain is already available, it seems less suited for cases in which a domain expert would need to construct this. 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 modellers 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 manner. It also allows for representations that are independent of domain size by supporting types, functions, relations, and quantification. To select one or more solutions from the solution space, multiple inferences tasks are supported.

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" of 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 modeller to define their vocabulary by means of a glossary. It consists of at most five glossary tables, each enumerating a different kind of symbol. An example glossary for the *Doctor Planning* challenge is given in Figure 3.

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 `Doctor`, which contains the names of doctors. By convention, type symbols start with a capital letter.

In the `Function` table, a symbol can be declared as a *function* of one or more types to another. There is no fixed syntax for functions; all types that appear in the description are interpreted as arguments to the function (of this type) and the remaining text is the name of the function. For example, `nb nights of Doctor` has one argument of type `Doctor`, and "*nb nights of*" is its name. Intuitively, this function denotes how many nights a doctor works per week. It maps each element of type `Doctor` to an element of type `Number`. Functions with $n > 1$ can be defined by using n arguments in the name,

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Doctor works max. 1 shift per day			
E*	**Doctor**	**Day**	**nb shifts of Doctor on Day**
1	-	-	≤ 1

Fig. 4. Constraint table to express that a doctor works a maximum of one shift per day.

such as `present doctor on Day at Time`, which assigns a doctor to every pair of `Day` and `Time`. The detection of arguments is case sensitive, so `doctor` is not considered an argument, but `Doctor` is.

For each domain element, a constant with the same name is automatically introduced, which allows the modeller to refer to this domain element in constraint or decision tables. For instance, the modeller can use the constant `Fleming` to refer to the domain element `Fleming`. In addition, the *Constant* table allows also other constants to be introduced. Recall that such logical constants correspond to standard DMN variables. In our example case, we use a constant `Head` of type `Doctor`, which means it can refer to any of the domain elements `Fleming`, `Freud`, `Heimlich`, `Eustachi` or `Golgi`.

In the *Relation* table, a verb phrase can be declared as a *relation* on one or more given types. For instance, the relation `Doctor is on leave` denotes for each `Doctor` whether they are on leave. Similarly to functions, there is no strict syntax: n-ary predicates can be defined by using n arguments in the name, for example, `Doctor is available on Day at Time` is a relation with three arguments (respectively of the type `Doctor`, `Day` and `Time`), that denotes whether a doctor is available on a specific day, at a specific time.

The *Boolean* table contains *boolean* symbols (i.e. propositions), which are either true or false. An example is the boolean `Complete`, which denotes whether the planning is complete.

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. 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 4, which states that every doctor can work a maximum of one shift per day.

cDMN does not only introduce constraint tables, it also extends the expressions that are allowed in column headers, both in decision and constraint tables. There are two types of headers in cDMN: the *term-denoting* headers, and the *atom-denoting* headers. A term-denoting header can consist of the following five expressions.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

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. 5. Example of a constraint table with quantification in cDMN, defining that bordering countries can not share colors.

1. A type *Type*. Such expression introduces a new variable x of type *Type*, which is only defined in the scope of the table.
2. An expression of the form "*Type* called *name*". This expression introduces a new variable *name* of the type *Type* in the scope of the table.
3. A constant.
4. An arithmetic combination of term-denoting header expressions (such as a sum of constants).
5. A function expression such as "*Function* of arg_1 and ... and arg_n", where each of the arg_i is a term-denoting header expression, or a previously introduced variable. This expression applies the function to its arguments.

An atom-denoting header consists of a relation expression such as "*Relation* for arg_1 and ... and arg_n,", where each of the arg_i is a term-denoting header expression, or a previously introduced variable. This expression applies the relation to its arguments.

The first two kinds of term-denoting 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. Once a variable x has been introduced by an expression *Type* (item 1), subsequent uses of the expression *Type* refer back to this variable x. Similarly, once a named variable *name* has been introduced by an expression *Type* called *name* (item 2), subsequent uses of the expression *name* refer back to this variable *name*.

The table in Figure 4 shows an example of quantification in cDMN. It introduces universally quantified variables of the type *Doctor* and *Day*, places no restrictions on these variables (i.e. "-"), and hence states that every doctor can only work a maximum of one shift on every day. To illustrate the use of named variables, Figure 5 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 from a definition to a set of implications. Second, constraint tables allow S-FEEL expressions in the output columns. Third, cDMN allows quantification, functions, predicates to be used in both decision tables and constraint tables.

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

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Data Table: Declaring which countries border			
	Country called c1	**Country called c2**	**c1 borders c2**
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Germany	France, Denmark, Luxembourg, Belgium, Netherlands	Yes

Fig. 6. Data table describing countries and their neighbours.

Goal
Get 3 models

Goal
Get all models

Goal
Maximize Score

Fig. 7. Goal table examples.

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 elements of a type) are allowed in data tables. It is also possible for columns to have more than one value in a certain cell, in which case the row is instantiated for each of these values. Since functions in cDMN models are always assumed to be total, a data table for a function should be complete, that is, there should be a value defined for every possible combination of input arguments. As an example, a snippet of the data table for the *Map Coloring* challenge is shown in Figure 6.

Data tables offer several advantages.

1. There is a methodological advantage: by separating data tables from decision tables, it becomes easier to reuse the specification.
2. If the modeller 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 modeller 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.

5.4 Goal 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.

When using the cDMN solver, this is no longer the case. We can fill in as many or as few variables as we want, and use the cDMN specification to derive useful information about the not-yet-known variables. By employing a *goal* table, modellers can state what the specification is to be used for: model expansion or optimization. Model expansion is the task of finding an interpretation for each of the symbols (a "model", in the terminology of classical logic) that satisfies all of the tables, and optimization is the task of finding the model with either the lowest or highest value for a given term. Examples of such tables are given in Figure 7.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

In summary, a cDMN model consists of

- A glossary;
- A set of data tables;
- A set of constraint tables;
- A set of decision tables;
- At most one goal table.

Apart from the glossary, all other kinds of tables are optional.

5.5 Semantics of cDMN

The meaning of a cDMN specification is given by a possible world semantics. As in classical logic, a possible world is represented by a structure S for a vocabulary V. Such a structure consists of a domain D and an assignment of each symbol $\sigma \in V$ to an appropriate relation/function σ^S on D. We will define the semantics of cDMN by means of a translation to FO($\cdot$), which is a typed variant of classical FO that also extends it with a number of additional constructs such as aggregates (Bruynooghe *et al.* 2015; De Cat *et al.* 2018; Wittocx *et al.* 2008). In this typed logic, a number of unary predicates are designated as types and each structure S must be such that the interpretations t_i^S of the types t_i form a partition of the domain of S. In addition, each relation/function σ has a typing, which must be respected by the interpretation σ^S, that is, if a predicate P has typing $(T_1, \ldots, T_n)$ then $P^S \subseteq T_1 \times \ldots \times T_n$.

We will define the set of possible worlds for a cDMN model as follows. The DMN glossary defines a typed FO($\cdot$) vocabulary V in a straightforward way. The data tables, together with the glossary, define a structure S for a part $V' \subseteq V$ of this vocabulary: that is, the domain of S is defined, as well as the interpretation σ^S of the symbols $\sigma \in V'$; however, for the remaining symbols $\sigma \in V' \setminus V$, the data tables do not yet define an interpretation. We will translate the decision and constraint tables into a theory T of FO($\cdot$) sentences, such that the possible worlds of a cDMN model are precisely the structures S' that extend S with an interpretation for the remaining symbols $V' \setminus V$ in such a way that $S' \models T$, that is, that all the decision/constraint tables are satisfied.

What remains is to transform each of the decision and constraint tables into an FO($\cdot$) sentence. Decision tables retain their usual semantics as described by Calvanese *et al.* (2018). We briefly recall this semantics. Each cell (i, j) of a decision table corresponds to a formula $F_{ij}(x)$ in one free variable x. For instance, a cell "≤ 50" corresponds to the formula "$x \leq 50$". A decision table with the *Unique* or *Any* hit policy with rows R, input columns I and output columns O is a conjunction of material implications:

$$\bigwedge_{i \in R} \left(\bigwedge_{j \in I} F_{ij}(H_j) \Rightarrow \bigwedge_{k \in O} F_{ik}(H_k) \right), \tag{1}$$

where H_j is the header of column j. When a decision table is incomplete, it is possible no rows match. In this case, the output given a special *null* value, that is,

$$\left(\bigwedge_{i \in R} \neg \bigwedge_{j \in I} F_{ij}(H_j) \right) \Rightarrow \bigwedge_{k \in O} H_k = null. \tag{2}$$

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

In short, a decision table is satisfied when, for each row of which all the input conditions are met, all output conditions are also met. When no rows are applicable, the output is forced to *null* (or the table's default value, if it has one).

For example, in standard DMN (which does not contain function or relation expressions), the table in Figure 2 corresponds to the logical formula $(AgeOfPerson \geq 18 \Rightarrow PersonIsAdult = Yes) \wedge (AgeOfPerson < 18 \Rightarrow PersonIsAdult = No)$.

Data tables are simply a specific case of decision tables.

The semantics of simple constraint tables (without quantification and functions) is also a conjunction of implications, as we described in Deryck *et al.* (2019). The semantics of constraint tables and decision tables differ in the interpretation of incomplete tables: when no rows are applicable in constraint tables, its outputs can take any arbitrary value instead of being forced to null (or some default value).

We now 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 *et al.* (2018).

Similarly to Calvanese *et al.*, we translate most of the entries c in a cell (i, j) of a table into a formula $F_{ij}(x)$ in one free variable x. For an expression e, we denote by $t(e)$ the logical term that corresponds to e. In standard DMN, the only expressions we need to consider are constants and arithmetic expressions built from constants. In this case, we can simply consider $t(e) = e$. We will show below how to extend t to the other kinds of expressions in cDMN. We now define:

- If c is of the form "θe" with θ 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 $F_{ij}(x)$ is $x \neq t(e)$;
- If c is a list $e_1, \ldots, e_n$, then $F_{ij}(x)$ 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 $F_{ij}(x)$ is $x = t(e)$.
- If c is a range, for example $[e_1, e_2)$, then $F_{ij}(x)$ is $x \geq t(e_1) \wedge x < t(e_2)$.
- A special case is when c contains "Yes" or "No". In this case, the header of the column must be an atom A and we translate it into $F_{ij} = A$ or $F_{ij} = \neg A$, respectively.

We now extend this transformation to take into account the fact that certain expressions – which we call *variable expressions* – must be translated to FO variables. There are two kinds of variable expressions, as we described in Section 5.2. We define a mapping ν that maps each of these two kinds of cDMN variable expressions to a typed FO$(\cdot)$ variable x of type T, which we denote as $x[T]$. We first define a mapping ν_H for variable expressions that appear in a header H of a variable introducing column:

- The name T of a type is a variable expression. We define $\nu_H(T) = x_T[T]$, with x_T a new variable of type T.
- An expression e of the form "*Type* called v" is a variable expression. We define $\nu_H(e) = v[Type]$.

We now define a general mapping ν as follows:

- If a variable expression e appears in a header H of a variable introducing column, then $\nu(e) = \nu_H(e)$.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

- If a variable expression appears elsewhere, then its value is $\nu_H(e)$, where H is the unique variable introducing header that introduced the variable expression e (see Section 5.2).

Such a variable expression introduces a new variable in the scope of the table at hand. Given this function ν, we now define the following mapping $t_\nu(\cdot)$ of cDMN expressions to terms.

- The interpretation of a constant, integer, or floating point number expression is the constant or number itself. That is, for a constant c, $t_\nu(c) = c$; similarly, for an integer or floating point number n, $t_\nu(n) = n$;
- For an arithmetic or other expression e of the form $e_1 \theta e_2$ with $\theta \in \{+, -, *, /, <, >, \leq, \geq, =, \neq, \vee, \wedge\}$, we define $t_\nu(e) = t_\nu(e_1)\ \theta\ t_\nu(e_2)$; in other words, the interpretation of such an expression is the operator applied to the interpretation of its subexpressions.
- The interpretation of a variable expression is the corresponding variable, that is, for a variable expression v, we define $t_\nu(v) = \nu(v)$.
- If c is of the form $\#Type$, then $t_\nu(c) = \#\{x[Type] : true\}$, an FO$(\cdot)$ aggregate that denotes the number of elements in the type itself.
- The interpretation of a function expression is that function applied to the interpretation of each of its arguments. For a function expression F of the form "*Function* of arg_1 and ... and arg_n", we define $t_\nu(F) = Function(t_\nu(arg_1), \ldots, t_\nu(arg_n))$.
- The interpretation of a relation expression is that relation applied to the interpretation of each of its arguments. For a relation expression R of the form "*Relation* for arg_1 and ... and arg_n", we define $t_\nu(X) = Relation(t_\nu(arg_1), \ldots, t_\nu(arg_n))$.

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 T as the following formula ϕ_T:

$$\underset{l \in V}{\forall} \nu(H_l) : \bigwedge_{i \in R} \left(\bigwedge_{j \in I} t_\nu\big(F_{ij}(t_\nu(H_j))\big) \Rightarrow \bigwedge_{k \in O} F_{ik}\big(t_\nu(H_k)\big) \right), \tag{3}$$

where we quantify over each variable x of type U for which $x[U]$ is the variable $\nu(H_l)$ that corresponds to the variable introducing column $l \in V$. In other words, for each tuple of elements of the variables' types, all table rows should be satisfied. Such a row is satisfied when, if all input conditions are met, all its output conditions are also met.

For example, in Figure 4, $\nu(H_1) = x[Doctor]$ and $t_\nu(H_1) = x$, $\nu(H_2) = y[Day]$ and $t_\nu(H_2) = y$, $t_\nu(H_3) = nb\_shifts\_of(t_\nu(H_1), t_\nu(H_2)) = nb\_shifts\_of(x, y)$, which leads to the formula:

$$\forall x[Doctor], y[Day] : nb\_shifts\_of(x, y) \leq 1.$$

which states that every person x works a maximum of 1 shift for every day y.

This semantics generalizes that of regular DMN tables. Indeed, in regular DMN, there are no variables, thus $V = \emptyset$, and only *constant* symbols are allowed, so $F_{ij}(t_\nu(H_j)) = F_{ij}(H_j)$. As a result, equation (3) simplifies to that in equation (1).

Decision tables with multiple hit policies have a different semantics. We first describe the semantics of "C+", "C<," and "C>" tables, which are almost identical. We define

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Number of invited guests		
C+	**Item in basket of Person**	**Charge for Person**
1	Yes	Price of Item

Fig. 8. Decision table that determines the charge of a person, based on the contents of their shopping basket.

the semantics of a $C+$ table with one output header H_k:

$$\underset{w\in W}{\forall}\, \nu(H_w) : t_\nu(H_k) = \sum_{i\in R} sum\{\bar{x} : \bigwedge_{j\in I} F_{ik}\big(t_\nu(H_j)\big) : F_{ik}((\bar{x}))\}. \tag{4}$$

Here, $W \subseteq V$ is the subset of variable introducing columns V of which the variable appears in the output header $t_\nu(H_k)$, $\bar{x}$ are the variables introduced by the remaining variable introducing columns $U = V \setminus W$ (so $\bar{x} = (t_v(H))_{H\in U}$), and $sum\{\bar{x} : \varphi(\bar{x}) : F(\bar{x})\}$ denotes the sum of all $F(x)$ for which $\varphi(x)$ holds.

This formula can be explained as follows. First, when no variables are introduced (i.e. $U = V = W = \emptyset$), this formula sums the output values F_{ik} for each of the rows i that meet the input criteria $\bigwedge_{j\in I} F_{ij}$. This is precisely the definition of a standard DMN $C+$ table.

Second, when variables are introduced in a table, but the output header contains no variables ($W = \emptyset$), it is again assigned a sum of terms. For each row i and tuple $\bar{x}$ that satisfy $\bigwedge_{j\in I} F_{ij}(\bar{x})$ is satisfied, the value $F_{ik}(\bar{x})$ is included in the sum.

Third, when the output header does contain variables, the table defines the value not of a single constant H_k, but of a function $H_k(\nu(\bar{w}))$. For each appropriate tuple $\bar{a}$, the value of $H_k(\bar{a})$ is defined by the same sum as before.

We illustrate this semantics with an example. In the decision table shown in Figure 8: $W = \{Person\}$, $\nu(H_w) = p[Person]$, $t_\nu(H_k) = Charge(p)$, $\bar{x} = y[Item]$, $t_\nu(F_{1k}(\bar{x})) = Price(y)$ and $\bigwedge_{j\,\in\, I} F_{ij}(t_\nu(H_j)) = InBasket(y, p)$. This results in the logical sentence:

$$\forall\, p[Person] : Charge(p) = sum\{(Item) : InBasket(Item, p) : Price(Item)\}.$$

The semantics of $C<$ and $C>$ tables are defined analogously, where, instead of summing all values, the minimum and maximum value is selected, respectively.

Decision tables with a $C\#$ hit policy have a slightly different semantics, that is,

$$\underset{w\in W}{\forall}\, \nu(H_w) : t_\nu(H_k) = \big|\{x \mid \underset{u\in U}{\exists}\, \nu(H_u) : \bigvee_{i\in R} \big(t_\nu(F_{ik}(x)) \wedge \bigwedge_{j\in I} t_\nu(F_{ij}(H_j))\big)\}\big|. \tag{5}$$

Here, U and W are defined analogously as in equation (4).

This formula can be explained as follows: first, when the output header contains no variables ($W = \emptyset$ and $U = V$), the aggregate expression counts for how many x's there exists an assignment of values to the variables P that causes at least one row i of the table to be applicable, in the sense than both its input and output columns are satisfied. The output header is assigned the size of the set x given that there exists an expansion of variables for which one of the rules that has x as output fires.

As before, when the output header does contain variables, for each tuple $\nu(\bar{w})$, the value of $H_k(\nu(\bar{w}))$ is defined in this way.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Number of guests			
C#	**Person is Friend**	**Person is Family**	**NbInvitations**
1	Yes	—	Person
2	—	Yes	Person
3	—	Yes	Spouse of Person

Fig. 9. Decision table that counts the number of invited guests.

For instance, in the decision table in Figure 9: $W = \emptyset$, $t_\nu(H_k) = NbInvitations$, $U = \{Person\}$, $\nu(H_u) = p[Person]$. For the first row, $\big(F_{1k}(t_\nu(x)) \wedge \bigwedge_{j\in I} F_{1j}(t_\nu(H_j))$ is equivalent to $x = p \wedge isFriend(p)$. The second row is defined analogously to the first row. In the third row, $F_{3k}(t_\nu(x)) \wedge \bigwedge_{j\in I} F_{3j}(t_\nu(H_j))$ translates to $x = Spouse(p) \wedge isFamily(p)$. Consequently, the table in Figure 9 is logically equivalent to:

$$\begin{aligned} NbInvitations = \big|\{x \mid \exists p[Person] : \\ x = p \wedge isFriend(p) \vee \\ x = p \wedge isFamily(p) \vee \\ x = Spouse(p) \wedge isFamily(p) \\ \}\big|. \end{aligned} \tag{6}$$

In the table of Figure 9, the output header is a constant (*NbInvitations*), therefore no quantification is required. The value of this constant is calculated as the number of persons that are either friends, family or the spouse of family, while ensuring that duplicate persons (such as friends that are also family) are not counted multiple times.

With this, we have defined the semantics of cDMN. The goal table that can also be included in a cDMN specification does not contribute to the semantics, but simply tells the cDMN solver what to compute; this can either be a set of possible worlds (one, all, or a specific number of them) or the possible world that minimizes/maximizes a given term.

6 Implementation

Because cDMN is more expressive than DMN, it cannot be handled by existing solvers. We have therefore implemented a new solver[4], which we describe in this section. It consists of two parts: an off-the-shelf constraint solver (the IDP system (De Cat *et al.* 2018)), and a converter from cDMN to IDP input. In principle, any constraint solver could be used, but we use the IDP system because it directly supports FO(·). The input of the system is a cDMN model created in a spreadsheet in the `.xlsx` format. Such a spreadsheet allows for straightforward creation of cDMN tables, and can show a good overview over the entire model. The cDMN to IDP converter is written in Python, and works in a two-step process.

First, the converter interprets all tables in a spreadsheet, 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/Function`

[4] https://gitlab.com/EAVISE/cdmn/cdmn-solver

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

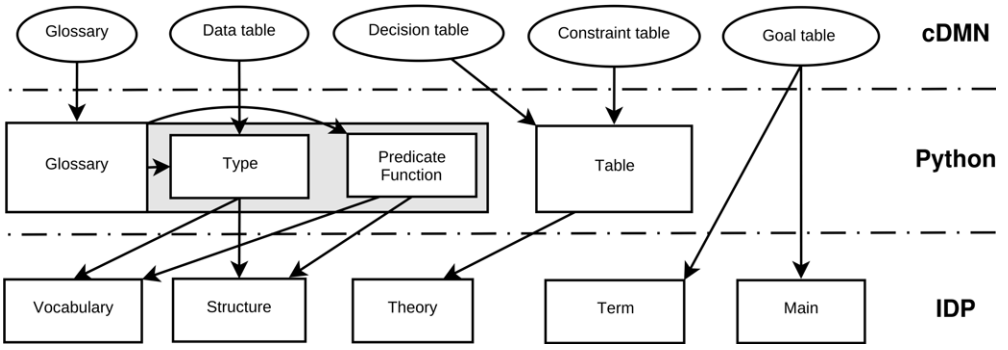

Fig. 10. An overview of the inner workings of the cDMN solver.

objects. The constraint and decision tables are then evaluated individually. A lex/yacc parser inspects each cell and parses it as a cDMN expression, such as "*Function* of arg_1 and arg_2". Such cDMN expressions are then interpreted using the constructed *Glossary* object, and transformed into an FO(·) expression, for example, "$Function(arg_1, arg_2)$". For example, the expression "nb shift of Golgi on d2" is transformed into "$nb\_shift\_of\_Doctor\_on\_Day(Golgi, d2)$". Each cDMN table is then converted to an FO(·) formula, as described in Section 5.5.

The created Python objects are then converted into IDP blocks. The knowledge in the IDP system is structured in three such blocks: the *vocabulary*, the *structure* and the *theory*. On top of these, there are also the *main* and *term* blocks. The *main* block is used to specify the logical inference method that should be applied. cDMN makes use of two of these inference methods: *model expansion* (find an expansion $S' \supseteq S$ of a structure S for part of the vocabulary of a theory T such that $S' \models T$) and *optimization* (find the model expansion of S w.r.t. T that minimizes/maximizes a given a numerical term). The *term* block is used to specify the optimization term when optimizing.

An overview summarizing all the relations between cDMN tables, Python objects and IDP blocks can be found in Figure 10. More detailed information about this conversion can be found in the cDMN documentation[5], along with an explanation of the usage of the solver and concrete examples of cDMN implementations.

Besides cDMN tables, the solver also supports most standard DMN tables and constructs. More specifically, it supports tables with the "U", "A", "F", "C+", "C>," and "C<" hit policy, and the full S-FEEL language. While there is currently no support for the "C"-tables, which collect the outputs of all matching rows in a list, it is possible to use a cDMN relation to emulate such a table's behavior.

Standard DMN specifications can be supplied in the form of a spreadsheet (as for cDMN), but also in the XML format specified in the DMN standard. As such, the solver can also be used as a drop-in replacement for standard DMN tools, allowing for a more flexible usage of the knowledge in a DMN model (Vandevelde and Vennekens 2020).

Another feature of the solver is the ability to link to the IDP-based interface *Interactive Consultant* (Carbonnelle *et al.* 2019), which is a user-friendly interface for interactively solving configuration problems. It shows users the consequences of their choices and provides explanations for these consequences. Thus, by combining cDMN and the Interactive

[5] www.cdmn.be

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

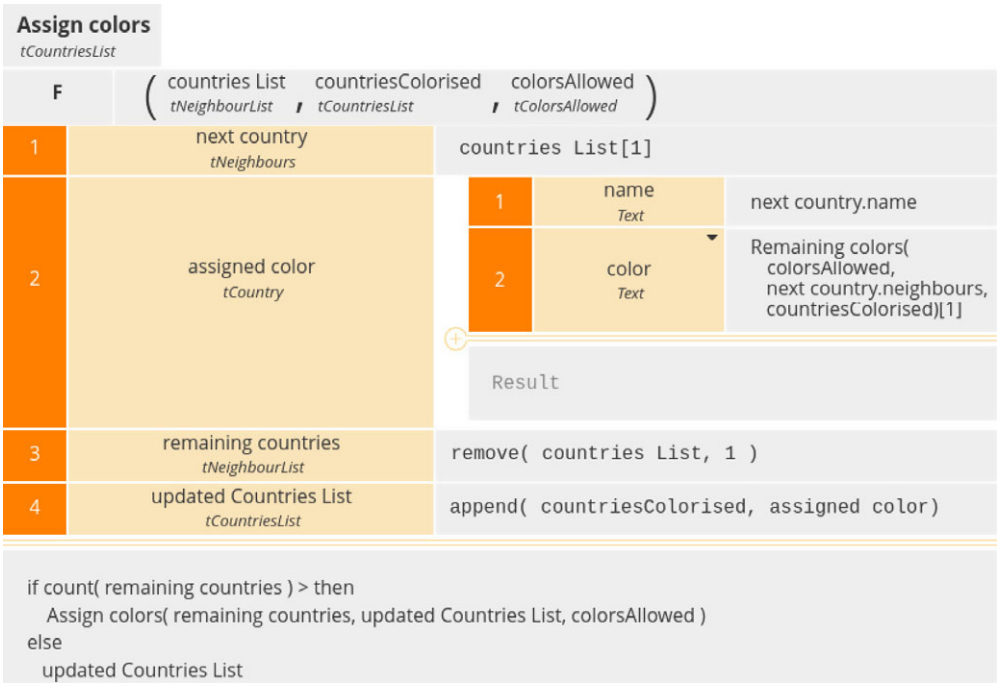

Fig. 11. An extract of the map coloring solution in standard DMN with FEEL.

Consultant interface, a KB can be both constructed and interacted with in a user-friendly manner.

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 qualitatively compare the DMN implementations from the DM Community website with our own implementation in cDMN. Afterward, we compare all challenges on size and quality. We end our discussion with a section on the integration of cDMN in business processes.

7.1 Constraint tables

Constraint tables allow cDMN to model constraint satisfaction problems in a straightforward way. For example, in *Map Coloring*, a map of six European countries must be colored in such a way that no neighboring countries share the same color. For this challenge, a pure DMN implementation was submitted, of which Figure 11 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 straightforward constraint table to solve this problem, as shown earlier in Figure 5. Together with the glossary and a data table (Figure 6), this forms a complete yet simple cDMN implementation.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

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

7.2 Quantification

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

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 12a). 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 12b). 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 13a. 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 in Figure 13b requires only a single row to express the same.

We conclude that of all the solutions that were submitted to the DM Community, only the cDMN solution has quantification powerful enough to represent the constraints of this puzzle in a way that is independent of the size of the problem instance.

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,

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Mike.fruit < > Sam.fruit	T	Mike.place < > Sam.place	T
Mike.fruit < > Harriet.fruit	T	Mike.place < > Harriet.place	T
Mike.fruit < > Anna.fruit	T	Mike.place < > Anna.place	T
Sam.fruit < > Harriet.fruit	T	Sam.place< > Harriet.place	T
Sam.fruit < > Anna.fruit	T	Sam.place < > Anna.place	T
Harriet.fruit < > Anna.fruit	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. 13. 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.

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

Goal
Maximize Score

Fig. 14. The decision tables and constraint table for *Balanced Assignment*.

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. The optimization support in cDMN allows us to represent the problem with two decision tables and one constraint table. The table *Diversity score*, shown in Figure 14, 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.

While it is possible to model this problem using the cDMN notation, the internal engine in the cDMN solver cannot find a solution in reasonable time due to the large problem size. However, our solver is only a reference implementation; it might be possible to create other solvers for cDMN that would be capable of solving this problem.

7.4 Overview of all challenges

Of the 24 challenges we considered, cDMN is capable of successfully modeling 22. In comparison, there were 12 OpenRules implementations and 12 Corticon implementations submitted. Note that we cannot rule out that 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. First, we quantitatively measure the size of the solution. This was measured by counting the number of cells used in all the decision and constraint tables. We exclude meta information (such as the cDMN glossary) and the specification of a concrete problem instance (such as the

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

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

	cDMN	Corticon	OpenRules	Others
Who Killed A.?	53	54	176	/
Change Making	26	14	/	/
A Good Burger	35	20	95	76[1]
Define Dupl.	20	19	21	/
Coll. of Cars	26	45	/	48[1]
Monkey Business	47	64	150	/
Vacation Days	38	32	31	14[2]
Family Riddle	76	22	/	/
Cust. Greeting	88	/	205	/
Online Dating	45	78	/	/
Class. Employees	36	21	70	34[3]
Reinder Order	14	64	111	370[4]
Zoo, Buses, Kids	24	/	43	/
Balanced Assign.	55	/	30	/
Vac. Days Adv.	124	/	97	/
Map Coloring	21	/	/	34[4]
Map Color Viol.	21	/	/	/
Crack The Code	48	/	/	/
Numerical Haiku	41	/	/	
Nim Rules	22	/	61	/
Doctor Planning	102	/	/	/
Calculator	33	/	/	/

cDMN data tables), because the ways in which different solvers handle this are too diverse to allow meaningful comparison. Table 3 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.

Second, we also qualitatively assess the readability and scalability of the solutions. The motivation for this is that model size, as we have defined it above, does not tell the whole story. Indeed, using very complex expressions might lead to small tables, that are nevertheless hard to figure out.

In general, we find that OpenRules implementations are usually easier to read than their Corticon counterparts. An example comparison between cDMN and Corticon can be seen in Figure 15a and 15c. 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, we introduce a type `Ingredient`, a number of functions such as `Amount of Ingredient` and `Fat in Ingredient`, and calculate the constant `Total Fat` as the product of the fat in a specific ingredient and the amount of that ingredient used. This enables the user to simply add new ingredients or change the amount of nutrition values in the data table, without having to change the model.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

	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 + cheesecount * 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

DecisionTable DefineBurgerTotals		
ActionScalProd		
Scalar Product Name	**Coefficients**	**Variables**
Total Sodium	50, 330, 310, 1, 260, 3, 160, 3	Beef Patty Items, Bun Items, Cheese Items, Onions Items, Pickles Items, Lettuce Items, Ketchup Items, Tomato Items
Total Fat	17, 9, 6, 2, 0, 0, 0, 0,	Beef Patty Items, Bun Items, Cheese Items, Onions Items, Pickles Items, Lettuce Items, Ketchup Items, Tomato Items
Total Calories	220, 260, 70, 10, 5, 4, 20, 9	Beef Patty Items, Bun Items, Cheese Items, Onions Items, Pickles Items, Lettuce Items, Ketchup Items, Tomato Items
Total Cost	25, 15, 10, 9, 3, 4, 2, 4	Beef Patty Items, Bun Items, Cheese Items, Onions Items, Pickles Items, Lettuce Items, Ketchup Items, Tomato Items

(b) OpenRules

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

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

(c) cDMN

Fig. 15. Calculating the food properties of a burger in Corticon, OpenRules, and cDMN.

The OpenRules implementation (Figure 15b) is fairly readable and modular too, but, it requires a custom scalar product decision table.

Another comparison between cDMN and OpenRules can be found in Figure 16a and 16b. Here we show a snippet of the *Who Killed Agatha?* challenge. Both show a translation of the following rule: "A killer always hates, and is not richer than, his victim." By using constraints and a constant (`Killer`), cDMN allows us to form a more scalable table. Indeed, 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 relevant problem properties. We now suggest that each property is tackled more easily by one or more of the additions cDMN proposes.

Aggregates needed Figure 15c 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.

Constraints Constraints can be conveniently modeled by constraint tables, such as the constraints in Figure 16b, which state 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.

Universal quantification Problems which contain universal quantification can be compactly represented, as can be seen in Figure 4. This table states that no doctor works more than one shift per day.

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

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

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

found in *Balanced Assignment* in Figure 14. A summary of each problem property and its cDMN answer can be found in Table 4.

7.5 Process integration

DMN models are often integrated into a larger business process model (Hasi *et al.* 2018; Bazhenova *et al.* 2019). Such a business process model consists of a sequence of steps that describe how to execute a specific process, such as for example the steps required for verifying a customer's eligibility for a bank loan. The Business Process Model and Notation is a standard published by the OMG group for this purpose.

The integration of DMN into BPMN is motivated by the *separation of concerns* paradigm (Biard *et al.* 2015), in which the decision logic is separated from the process, to increase readability and maintainability of the overall process model. If a DMN model is present in a BPMN model, it can be used to dictate the flow of the process using a so-called *gateway*, depicted by a diamond. For example, the BPMN model in Figure 17 describes the flow for buying a ticket to a museum. After a visitor has selected the exhibits they want to visit, the price of the ticket is determined by the decision model shown in Figure 1. The output of the decision model then dictates whether a payment is required, or if the ticket can be printed directly (in the event that only free exhibits were selected).

In principle, cDMN models could also be used in a BPMN model to direct the flow of a process. When a DMN model is used in BPMN, the process is always directed based on the value of the top level variable (such as $Price > 0$ or $Eligible = Yes$) of the DMN model. By contrast, the integration of cDMN also allows for other criteria. For example, the model in Figure 18 describes the process of coloring a map of countries, based on a list of countries and a list of possible colors. Here, we have added a gateway that verifies if a suitable solution was found. If none was found (because too few colors were supplied), more colors are added until a solution becomes possible. In other words, the direction of the process is based on whether or not a satisfying solution for the cDMN constraints

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

Table 4. *Comparison between the problem properties and their cDMN answers*

Property	cDMN answer
Aggregates needed	Quantification, expressive data
Constraints	Constraint tables, quantification, expressive data
Universal quantification	Quantification
Optimization	Optimization, expressive data

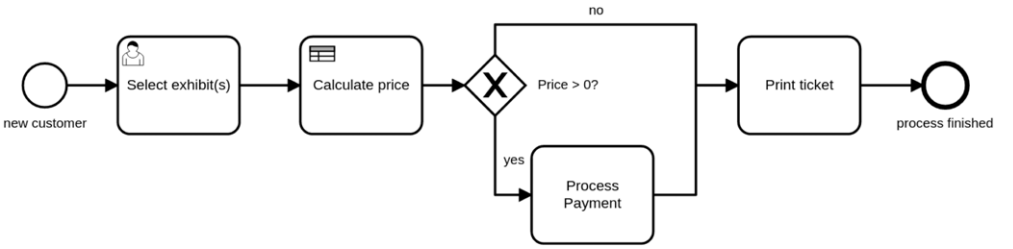

Fig. 17. Example of a BPMN model with DMN.

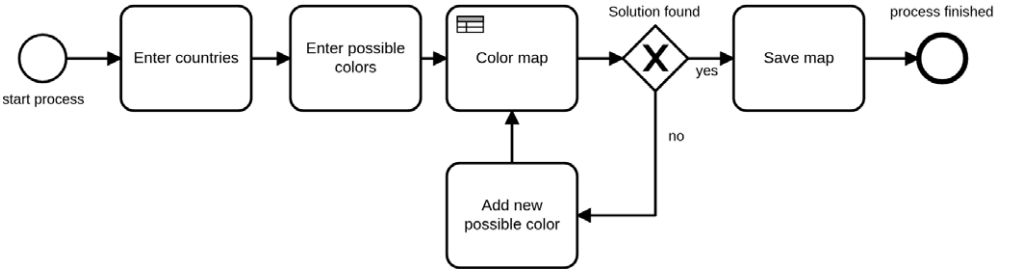

Fig. 18. Example of a BPMN model with cDMN.

could be found. Other examples of possible gateway criteria are verifying if at least n solutions exist, if a solution with a value for variable x greater than 5 exists, what the maximum value of variable y is, and more.

8 Conclusions

This paper presents an extension to DMN, which aims at solving complex problems while maintaining DMN's level of readability. This extension, which we call cDMN, adds constraint modeling, more expressive data representations (such as types and functions), and quantification.

Constraint modeling allows a user to define a solution space instead of a single solution. The cDMN solver 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 maintainability and scalability of tables.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

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 the expressiveness of DMN. Moreover, our qualitative analysis of these examples suggest that the cDMN representations are indeed typically quite readable and maintainable. In future work, we plan to investigate this in a more detailed and quantifiable way and to compare the user-friendliness and complexity of cDMN to that of DMN itself.

Other future work consists of possibly extending the cDMN notation to be able to represent disjunctions in the output of a constraint table, existential quantification, quantification in output columns, and increase compactness of the created models. Additionally, we are planning on testing this notation in a number of real-life use-cases to verify its applicability in a multitude of domains. The insights gained during the implementation of these use cases will allow us to define a graph-based representation of cDMN models, akin to the DRD for DMN.

Competing of interest

The authors declare none.

References

Aerts, B., Vandevelde, S. and Vennekens, J. 2020. Tackling the DMN challenges with cDMN: A tight integration of dmn and constraint reasoning. In *Rules and Reasoning: Fourth International Joint Conference, RuleML+RR 2020, Oslo, Norway, June 29 - July 1, 2020, Proceedings. Proceedings of RuleML+RR 2020*, 23–38.

Bazhenova, E., Zerbato, F., Oliboni, B. and Weske, M. 2019. From BPMN process models to DMN decision models. *Information Systems 83*, 69–88.

Biard, T., Le Mauff, A., Bigand, M. and Bourey, J.-P. 2015. Separation of decision modeling from business process modeling using new "Decision Model and Notation" (DMN) for automating operational decision-making. In *Risks and Resilience of Collaborative Networks*, L. M. Camarinha-Matos, F. Bénaben, and W. Picard, Eds. Springer International Publishing, Cham, 489–496.

Bruynooghe, M., BLOCKEEL, H., BOGAERTS, B., DE CAT, B., DE POOTER, S., JANSEN, J., LABARRE, A., RAMON, J., DENECKER, M. and VERWER, S. 2015. Predicate logic as a modeling language: Modeling and solving some machine learning and data mining problems with IDP3. *Theory and Practice of Logic Programming 15*, 783–817.

Calvanese, D., Dumas, M., Laurson, U., Maggi, F. M., Montali, M. and Teinemaa, I. 2018. Semantics, analysis and simplification of DMN decision tables. *Information Systems (Oxford) 78*, 112–125.

Calvanese, D., Montali, M., Dumas, M. and Maggi, F. 2019. Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge. *Theory and Practice of Logic Programming 19,* 4, 536–573.

Car, N. J. 2018. Using decision models to enable better irrigation decision support systems. *Computers and Electronics in Agriculture 152*, 290–301.

Carbonnelle, P., Aerts, B., Deryck, M., Vennekens, J. and Denecker, M. 2019. An interactive consultant. In *Proceedings of the 31st Benelux Conference on Artificial Intelligence*, K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, and P. V. Eecke, Eds. CEUR Workshop Proceedings, vol. 2491. CEUR-WS.org.

https://doi.org/10.1017/S1471068421000491 Published online by Cambridge University Press

De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G. and Denecker, M. 2018. Predicate logic as a modeling language: The IDP system. In *Declarative Logic Programming: Theory, Systems, and Applications*. ACM Books, 279–329.

Deryck, M., Aerts, B. and Vennekens, J. 2019. 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. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11784*, 171–179.

Hasic, F., De Smedt, J. and Vanthienen, J. 2017. Towards assessing the theoretical complexity of the decision model and notation (DMN) research-in-progress. In *CEUR Workshop Proceedings*. Vol. 1859. CEUR Workshop Proceedings, 64–71. ISSN: 1613-0073.

Hasic, F. and Vanthienen, J. 2020. From decision knowledge to e-government expert systems: the case of income taxation for foreign artists in belgium. *Knowledge and Information Systems 62,* 5, 2011–2028.

Hasi, F., De Smedt, J. and Vanthienen, J. 2018. Augmenting processes with decision intelligence: Principles for integrated modelling. *Decision Support Systems 107*, 1–12.

Object Management Group. 2020. *Decision Model and Notation*.

OpenRules, Inc. 2017. *Openrules*.

Progress. 2019. *Corticon*.

Silver, B. 2018. *DMN Method and Style: Business Practitioner's Guide to Decision Modeling*, 2nd ed. ed. Cody-Cassidy Press, Altadena.

Sooter, L. J., Hasley, S., Lario, R., Rubin, K. S. and Hasić, F. 2019. Modeling a clinical pathway for contraception. *Applied Clinical Informatics 10,* 5 (October), 935–943.

Vandevelde, S. and Vennekens, J. 2020. *A Multifunctional, Interactive DMN Decision Modelling Tool*.

Wittocx, J., Mariën, M. and Denecker, M. 2008. The IDP system: A model expansion system for an extension of classical logic. *Proceedings of the 2nd Workshop on Logic and Search*, 153–165.