cDMN for User-Friendly ASP

Simon Vandevelde[1][0000−0001−7312−3675]

KU Leuven, De Nayer Campus, Dept. of Computer Science, Belgium;
Leuven.AI; Flanders Make@KULeuven, B-3000 Leuven, Belgium
s.vandevelde@kuleuven.be

Abstract. We introduce an ASP translation for problems modeled in cDMN, a tabular decision-logic language based on the existing Decision Model and Notation standard. cDMN is designed to be readable and writable by domain experts, making knowledge representation more accessible to people without a formal logic background. We show how cDMN can be translated into ASP, enabling the use of the many tools and techniques available in the ASP ecosystem. We have also implemented this translation in the cDMN solver, allowing for automatically generating ASP encodings for any cDMN model.

Keywords: Answer Set Programming · constraint Decision Model and Notation · User-Friendly · logic programming

1 Introduction

Since the early days of Knowledge Representation (KR), accurately formalizing knowledge on a complex domain has imposed a significant hurdle for building real-life applications [11]. This problem, which is also referred to as the Knowledge Acquisition Bottleneck, is often cited as one of the main challenges for KR [14]. The issue largely stems from the need for expertise on both the application domain at hand, and on how to formalize it. Therefore, it is common practice to have a domain expert collaborate with a knowledge engineer through a process called *Knowledge Acquisition* [27]. Unfortunately, it is well-known that building an accurate knowledge base is a difficult, costly, and error-prone process, among others due to communication errors between both parties, thus causing the bottleneck.

As an approach to tackle this, the field of KR has seen some work towards KR tools and methodologies that aim to be more accessible for laypeople. The primary goal is to make knowledge formalization more approachable for non-experts, in an attempt to lift the knowledge acquisition bottleneck. In fact, the "holy grail" of user-friendly KR would be letting non-experts formally express their knowledge without any assistance from a knowledge engineer.

Typical KR formalisms, such as Prolog [10] and ASP [5], are largely viewed as too difficult for laypeople directly. Indeed, both reading and writing these languages requires extensive formal training. One popular approach to user-friendly KR is therefore to use *an alternative notation*, i.e., one that is regarded

as easier to use while still being formally translatable to the input language of a reasoning engine. Common examples of such a notation are Controlled Natural Languages (CNLs) [25], which let people describe knowledge in a restricted version of natural language. Another example is Constraint Decision Model and Notation (cDMN) [38], a table-based notation for decision logic, which is written in spreadsheet software in intuitive tabular formats.

In this paper, we take a closer look at cDMN, and show how we can translate cDMN tables into ASP. Combined with clingo as the solver, this allows us to efficiently reason over cDMN models.

2 Constraint Decision Model and Notation (cDMN)

As a preliminary, we first briefly elaborate on cDMN and its semantics. Because cDMN is an extension of a previously existing notation, DMN, we will first introduce the latter and then circle back to the former.

2.1 DMN

DMN, short for Decision Model and Notation [32], is a modeling standard published by the Object Management Group (OMG) for decision logic. Its main philosophy is to be readable *and* writeable for everyone involved in the decision modeling process, ranging from IT staff to business experts. Using off-the-shelf execution engines, DMN models can be evaluated to make decisions without requiring a separate implementation step; in this sense, DMN models are directly executable. Over the previous years, DMN has enjoyed quite some success in industry, with many companies adopting DMN(-like) approaches [33].

DMN's user-friendliness comes from its simple formalism, where decisions are represented in *decision tables* that are executed in a certain order. For instance, Fig. 1 shows a simple DMN model to decide how to hang up an object on the wall, with the inputs in green and the outputs in blue. It involves four decision variables, namely `wall`, `hole`, `weight` and `method`, used in two separate tables. The first table decides whether a hole can be made based on the wall type, and the second table decides the method based on the object's weight and whether we can make a hole. The meaning of the tables is derived from their *hit policy*, as denoted in the top-left corner. For the purpose of this paper, we will limit ourselves to the U(nique) hit policy, in which the input rows may not overlap[1]. Each row of these tables can loosely be read as an "if-then": e.g., "if the weight is less than or equal to 10 kg and a hole is possible, use nails". Due to this intuitive informal reading, decision tables are considered low in complexity and user-friendly for non-experts [21,12,17,7].

However, DMN's expressiveness is too limited for some more complex use cases. Notably, it is not possible to straightforwardly express *constraints*: instead, each decision table must always precisely *define* a symbol. Stating an expression

[1] For a full overview, we instead refer to [37, p. 22]

Decide if hole is possible		
U	wall	hole
1	Wood, Brick	Yes
2	Tile	No

Decide method to use			
U	weight	hole	method
1	≤ 10	Yes	Nails
2	(10..25]	Yes	Screws
3	≤ 15	No	Glue

Fig. 1: DMN tables for hanging up an object

like "if the weight is 15, you must use glue" is therefore possible, but "if we use glue, the weight must be below 15" is not, as the latter does not assign an explicit value to the weight.

2.2 cDMN

To overcome these expressiveness limitations, cDMN extends DMN with more expressiveness while maintaining the user-friendly tabular format. Among others, it adds constraint tables, higher-ary symbols, quantification, and more. As an example, Fig. 2 shows the Handyman tables rewritten using only constraint tables (denoted by the `E*` hit policy). When a row of a constraint table matches the values of the input variables, a constraint is applied to the output variable; if instead no row matches, nothing happens. For instance, given "method = Nails", table (A) derives that a hole must be made, based on which table (B) excludes Tile as a possible wall type. The reverse is also true: if "wall = Tile", we can derive through table (B) that a hole is not possible, and that therefore nails and screws are excluded.

(A) Method requires hole		
E*	method	hole
1	Nails, Screws	Yes

(B) No hole in tile		
E*	hole	wall
1	Yes	not(Tile)

(C) Max load		
E*	method	weight
1	Nails	≤ 10
2	Screws	≤ 25
3	Glue	≤ 15

Fig. 2: cDMN tables for hanging up an object.

Note how these two examples can be used for the same purpose, and yet do not convey the exact same meaning, due to using a different approach. The DMN tables in Fig. 1 are *prescriptive*, i.e., they express how to decide on a specific choice based on inputs. This makes the DMN model lack a certain level of explainability: one might wonder why the model suggests nails instead of screws for a weight under 10kg, whilst the latter would intuitively also work. By

contrast, the cDMN tables in Fig. 2 are *descriptive*, simply stating general logic about the problem domain without a set decision in mind. Here it is clear that both screws and nails work (at least) up to 10kg, and that therefore both are suitable. From a knowledge representation standpoint, this makes cDMN a more interesting language than its original.

Besides constraint tables, cDMN also supports typed higher-ary functions and predicates. These must be declared in *glossary tables*, which effectively represent the ontology of the model. As an example, Fig. 3 shows a modeling of the well-known map coloring problem in cDMN. In the glossary tables (in orange), we introduce types *Country* and *Color*, followed by a color function in the Function table and a *borders* predicate in the Relation table. In cDMN, each type that explicitly appears in the name of a function or predicate is considered part of its domain. For example, *color of Country* with type *Color* corresponds to a unary function $color_of : Country \to Color$, and *Country borders Country* corresponds to a binary predicate $borders : Country \times Country$. In this way, symbols in cDMN can be declared in a pseudo-Controlled Natural Language, which makes the resulting models more natural to read.

Type		
Name	**DataType**	**PossibleValues**
Country	string	BE, FR, LU, NL, DE, DK, UK
Color	string	Red, Blue, Green, Purple

Function	
Name	**DataType**
color of Country	Color

Relation
Name
Country borders Country

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	-	Yes	not(color of c2)

Fig. 3: cDMN model for map coloring

The constraint table in the example demonstrates universal quantification in cDMN, which uses a "*Type (called var)*" syntax, where *Type* is a user-declared type and *var* is an optional name for the quantification variable. All subsequent columns may then use *var* to refer to this quantification variable. In this way, the example table can be read as "For every countries `c1` and `c2` must hold that if `c1` borders `c2`, their colors cannot be the same". Note that the name is optional – if no name is given, the name of the type itself is used instead. Paired with the higher-ary symbols, quantification in cDMN it allows us to model more complex knowledge in relatively compact tables.

x	$F(x)$	x	$F(x)$
—	true	$x\{<,>,\leq,\geq\}X$	$x\{<,>,\leq,\geq\}X$
X	$x = X$	$[X,Y]$	$X \leq x \leq Y$
$not(X_1,\ldots,X_n)$	$\neg \vee_{i\in n}(x = X_i)$	(X,Y)	$X < x < Y$
$X_1,\ldots,X_n$	$\vee_{i\in n}(x = X_i)$		

Table 1: S-FEEL cell formula semantics.

3 (c)DMN semantics

We will now briefly detail the semantics of (c)DMN, beginning with decision tables in standard DMN. Formally, such a decision table is a tuple $DT = \langle I, O, R, H\rangle$ where $I = \{i_1 : D_1, \ldots, i_n : D_n\}$ is a finite set of input variables and their domains, $O = \{o : D\}$ is the output variable and its domain, $R = \{r_1, \ldots, r_m\}$ is the set of rules, and H is the hit policy.

In a decision table, each kth cell of the row r_j corresponds to a formula $F_{k,j}(x)$ with one free variable x. These cells contain expressions in S-FEEL, a language focused on simple expressions [32, p.88]. A full overview of these expressions and their corresponding formulas are shown in Table 1.

Similar to the work of Markovic et al. [28], we define the meaning of a decision table using definitional rules of the form *head* $\leftarrow$ *body* following FO(ID) [15], which extends classical logic with (inductive) definitions. In this logic, a set of definitional rules fully *defines* the variable in their head, by capturing both the *sufficient* and *necessary* conditions: if one of the bodies is satisfied, the head must be as well (sufficient), and if the head is satisfied, at least one of the bodies must be satisfied (necessary). Each row r_j of a decision table with inputs I and output O then corresponds to the following definitional rule:

$$F_{|I|+1,j}(O) \leftarrow \bigwedge_{k=1}^{|I|} F_{k,j}(I_k). \tag{1}$$

Together, these rules form a complete definition. For instance, the corresponding definition of the second handyman table in Fig. 1 is as follows:

$$\left\{ \begin{array}{c} method = Nails \leftarrow hole = Yes \wedge weight \leq 10. \\ method = Screws \leftarrow hole = Yes \wedge 10 < weight \leq 25. \\ method = Glue \leftarrow hole = No \wedge weight \leq 15. \end{array} \right\}$$

Crucially, using FO(ID) definitions ensures that values for which no row matches (e.g., "weight = 20, hole = No") lead to unsatisfiability instead of undefined behavior, in contrast to, e.g., the implication-based semantics proposed in [7].

The semantics of tables in cDMN follows that of DMN, but is modified with quantification, the higher-ary symbols, and constraint tables. We will give a short overview here, and refer to [37, p. 47] for a full overview. Firstly, table headers consisting of "*Type* (called t)" are called *variable expressions*, which are

translated into their FOL equivalent using a mapping ν. All columns following this table may use "t" (or "*Type*" if no name was given) to refer to this variable. Secondly, we define a mapping $t_\nu(\cdot)$ to translate cDMN expressions to FOL terms as follows:

- The interpretation of a constant, integer or floating point number is the constant or number itself.
- For an expression e of the form $e_1 \theta e_2$, we define $t_\nu(e) = t_\nu(e_1)\ \theta\ t_\nu(e_2)$ for $\theta \in \{+, -, *, /, <, >, \leq, \geq, =, \neq\}$.
- A variable expression v is interpreted as the corresponding variable, so $t_\nu(v) = v$.
- The interpretation of a function/predicate expression f is that function/predicate applied to the interpretation of its arguments: $t_\nu(f(a_1, \ldots, a_n)) = f(t_\nu(a_1), \ldots, t_\nu(a_n))$

Using this mapping, each row r_j of a decision table with the `U` hit policy, consisting of inputs I, output O, and a set of variable introducing headers $V \subseteq I$ corresponds to the following definitional rule:

$$\forall \nu(V) : t_\nu(F_{|I|+1,j}(t_\nu(O))) \leftarrow \bigwedge_{k=1}^{|I|} t_\nu(F_{j,k}(t_\nu(I_k))).$$

Similarly, each row r_j of a constraint table corresponds to a material implication as follows:

$$\forall \nu(V) : \Big(\big(\bigwedge_{k=1}^{|I|} t_\nu(F_{k,j}(t_\nu(I_k))) \big) \Rightarrow t_\nu(F_{|I|+1,j}(t_\nu(O))) \Big)$$

As an example, the `E*` table in Fig. 3 corresponds to:

$$\forall c_1, c_2 \in \mathit{Country} : \mathit{borders}(c_1, c_2) \Rightarrow \mathit{color}(c_1) \neq \mathit{color}(c_2).$$

cDMN also extends DMN's "multi-hit tables", which are tables that aggregate the output values of all applicable rows. One common such table is the sum table, denoted by the `C+` hit policy, which is useful for counting occurrences, calculating weights to optimize, and more. As an example, Fig. 4 shows a table calculating

Count red			
C+	Country called c1	color of c1	term
1	-	Red	2
2	-	Purple	3

Fig. 4: Sum table for extended Map Coloring

a term `term` which represents a weighted sum for each country colored red or

purple. In cDMN, the semantics of a C+ table with rows R, inputs I, outputs O is as follows:

$$\forall \nu(W) : \; t_\nu(O) = \\ \sum_{j \in R} sum\{\bar{x} : \; \bigwedge_{k=1}^{|I|} t_\nu\big(F_{k,j}(t_\nu(H_j))\big) : t_\nu(F_{|I|+1,j}((\bar{x})))\}$$

with $W \subseteq V$ the subset of variable introducing columns of which the variable appears in the output header, $\bar{x}$ the variables introduced by the remaining variable introducing columns, and $sum\{\bar{x} : \phi(\bar{x}) : F(\bar{x})\}$ the sum of all $F(x)$ for which $\phi(x)$ holds.

Finally, cDMN also adds *data tables*, which are denoted by the D hit policy. Such a table does not contain logic, but instead explicitly defines the values of a cDMN symbol, serving as its complete interpretation. An example of these last two types of tables is shown in the next section.

4 Translating cDMN to ASP

In this section, we will elaborate on how cDMN tables can be translated into ASP, allowing us to reason over the models with off-the-shelf ASP engines. As a running example, we use an extended version of the map coloring problem, modeled in Fig. 5. On top of the standard problem rules, we also wish to define "isolated countries" as those with only 1 neighbor or less, which we achieve using an additional U and C+ table.

Note that we will demonstrate rather literal translations to ASP, which might not be optimized, to ensure that the translation remains generic. This will likely make the resulting program more difficult to read as well, but as the ASP is intended only as an intermediary language after cDMN, this is not a concern to us. For performance reasons, we can also rely on tools like lpopt [4] and ngo [34] to automatically optimize the ASP encoding.

Glossary Each type of the *Type* glossary is represented by a unary predicate, for which we set the values through facts. This also includes the built-in types `bool` and `int`, with the latter being defined as a configurable range defaulting to -100 to +100. For instance, we introduce the following types for the example glossary:

```
1  bool(yes;no).
2  int(-100..100).
3  country(BE; FR; LU; NL; DE, DK, UK).
4  color(red; blue; green; purple).
```

The symbols from the *Relation* and *Function* tables are represented as functional terms. Predicates *symbol/2* and *domain/2* are used to introduce these representations and their domains, respectively. In this way, n-ary cDMN symbols correspond to n-ary functional terms. cDMN functions map on their respective types, and cDMN predicates map on `bool`.

Type		
Name	**DataType**	**PossibleValues**
Country	string	BE, FR, LU, NL, DE, DK, UK
Color	string	Red, Blue, Green, Purple

Relation
Name
Country borders Country
Country is an island

Function	
Name	**DataType**
color of Country	Color
nb nbrs of Country	Int

Define island			
U	Country	nb nbrs of Country	Country is isolated
1	-	0	Yes
2	-	1	Yes

Bordering countries may not have the same color				
E*	Country called c1	Country called c2	c1 borders c2	color of c1
1	-	not(c1)	Yes	not(color of c2)

Count neighbors				
C+	Country called c1	Country called c2	c1 borders c2	nb nbrs of c1
1	-	-	Yes	1

Bordering countries			
D	Country called c1	Country called c2	c1 borders c2
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Netherlands	Germany, Belgium	Yes
3	Germany	Belgium, France, Denmark, Luxembourg	Yes
4	France	Luxembourg, Germany, Belgium	Yes
5	Luxembourg	Belgium, Germany, France	Yes
6	Denmark	Germany	Yes

Fig. 5: Modified Map Coloring in cDMN

```
symbol(color_of_country(X0)) :- country(X0).
domain(color_of_country(X0), Y) :- country(X0), color(Y).
symbol(nb_nbrs_of_country(X0)) :- country(X0).
domain(nb_nbrs_of_country(X0), Y) :- country(X0), int(Y).
symbol(country_borders_country(X0, X1)) :- country(X0), country(X1).
domain(country_borders_country(X0, X1), Y) :- country(X0), country(X1),
                                              bool(Y).
symbol(country_is_an_island(X0)) :- country(X0).
domain(country_is_an_island(X0), Y) :- country(X0), bool(Y).
```

Predicate *value/2* is used to represent the value for each functional term. To ensure the functional nature is kept, i.e. each functional term may only be mapped on exactly one value, we add a cardinality choice rule. While this is not supported by all ASP reasoners, it can always be rewritten in standard ASP-Core-2.

```
1 { value(Y, X): domain(Y, X) } 1 :- symbol(Y).
```

Constraint Tables Constraint tables are relatively easy to represent as ASP constraints. We reuse the mapping $t_\nu(\cdot)$ presented earlier, with the exception that expressions of the form $e_1 \theta e_2$ where e_1 is a function/predicate symbol f now correspond to $value(f, V) \wedge V \theta t_\nu(e2)$, with V a universally quantified symbol over the codomain of f.

Using this modified approach, we can translate each row of the constraint table as an integrity constraint by negating the output and moving it into the body. For instance, the constraint table of the example corresponds to the following:

```
:- not V1 != V2, value(color_of_country(C1), V1),
                 value(color_of_country(C2), V2),
                 C2 != C1, V0 = yes,
                 value(country_borders_country(C1, C2), V0).
```

Decision tables Decision tables require a little more work to translate due to how predicate symbols are handled in cDMN. When defining a predicate in a cDMN decision table, the rows typically enumerate the positive cases, i.e., those where the predicate is satisfied. When no row matches a set of inputs, the predicate is implicitly set to `No`, to save users from having to enumerate all negative cases as well. For instance, the U-table in Fig. 5 defines that a country is isolated if it has 0 or 1 neighbors. In all other cases (e.g., when `nb nbrs` > 1), the predicate is implicitly unsatisfied.

We achieve this behavior as follows. For a decision table defining an n-ary cDMN predicate, we introduce an $n+3$-ary predicate `row` to keep track of which rows r of table t are satisfied for quantification variables v and output value o: `row(r, t, o1, ..., on, v)`. For example, if the second row of the example U-table matches for the UK, we would denote this as `row(0, 1, uk, yes)`, where `0` is a table-id to prevent duplicates, and `1` represents the second row counting from zero.

Using the `row` predicate for the head of the rule, each row of the decision table can now be translated like those of the constraint table. To set the value for the symbol, each row also introduces an integrity constraint:

```
row(1, 0, C, yes) :- V4 = 0, value(nb_nbrs_of_country(C), V4).
:- not value(country_is_isolated(C), V), row(1, 0, C, V).
row(1, 1, C, yes) :- V4 = 1, value(nb_nbrs_of_country(C), V4).
:- not value(country_is_isolated(C), V), row(1, 1, C, V).
```

Lastly, as we need to ensure a default `No` value for predicates, we introduce an integrity constraint in case no row of the table was applicable:

```
:- not value(country_is_isolated(C), no),
       not row(1, 0, C, yes), not row(1, 1, C, yes),
       country(C).
```

Sum tables cDMN sum tables are relatively straightforward to represent as a sum aggregate in ASP. Intuitively, the variables occurring in the output header determine the grouping of the aggregation: a separate sum is computed for each combination of their values. For instance, for the C+ table in the running example the sum is calculated for each possible value of `c1`. All other variables are quantified over in the aggregate's body, together with the rest of the input columns. The output cell corresponds to the aggregate's term. To ensure the term is unique (and thus can be counted multiple times), we also add the variables appearing in the output header as terms. Lastly, we link the aggregate to the `value` predicate as follows:

```
V3 = S :-
    S = #sum{ 1, C2:
                V0 = yes, value(country_borders_country(C1, C2), V0) },
        value(nb_nbrs_of_country(C1), V3).
```

Data tables The last type of tables, the data tables, are used to represent a problem instance. In the running example, the D(ata)-table in Fig. 5 contains information on which countries share a border. Importantly, such a data table must totally define a symbol; partial interpretations are not allowed. In ASP, we introduce an *input/2* predicate to represent the inputs as listed in the data table. Like for the decision tables, we also add an explicit rule for cDMN predicates to derive `no` when no explicit `yes` was present. To actually "apply" these inputs, we use another integrity constraint.

```
input(country_borders_country(be, fr), yes).
input(country_borders_country(be, lu), yes).
input(country_borders_country(be, nl), yes).
...
input(country_borders_country(dk, de), yes).
input(country_borders_country(X0, X1), no) :-
    not input(country_borders_country(X0, X1), yes).

:- not value(X, Y), input(X, Y).
```

5 Software

To reason over cDMN models, [38] introduces the *cDMN solver*, a Python program to translate cDMN models into FO(·) [16]. FO(·) is an extended version of classical FOL, and serves as the input language for several reasoners, including

IDP3 [13], IDP-Z3 [8] and, SLI [36]. However, as demonstrated in this paper, cDMN can also be translated into ASP, allowing us to use solvers like clingo [20] and dlv [26] instead (albeit possibly requiring minor changes for any unsupported syntactic sugars).

For some problem classes, we expect ASP solvers to be more efficient at solving them, thereby motivating our work. Moreover, a translation into ASP allows us to use some interesting tools from the existing ASP ecosystem, such as clinguin [3] and ASPChef [1], and opens the door for well-supported solving techniques in the ASP community like multi-shot solving [20], which is currently not supported by the FO(·) reasoners.

Using the original cDMN solver as a starting point, we have implemented a fully automatic translation into ASP based on the translation principles outlined in the previous section[2]. While it can already translate many of the existing cDMN examples into ASP, cDMN models involving C+ tables tend to be quite slow; presumably because our translation is not sufficiently optimal yet.

As a preliminary benchmark, Fig. 6 shows the solving speed of IDP-Z3, clingo and SLI on the "Doctor Planning" problem. In this problem, we want to find a permanence schedule for a set of doctors under a set of constraints, ensuring that each time slot has a doctor assigned. This is a simple example of a rostering problem, for which the cDMN implementation is included in the appendix. As the graph shows, clingo's solving speed outperforms IDP-Z3, and is competitive with SLI's.

6 Related Work

Developing notations to let end-users develop their own knowledge bases is not a new idea. As mentioned in the introduction, one prevalent technique class for this purpose are controlled natural languages (CNLs). By restricting the syntax and grammar of a natural language, CNLs aim to be naturally interpretable (as they can be read without prior training) while still being formal and non-ambiguous logical notations. Some examples of such CNLs are ACE [19], PENG [40], LogicalEnglish [24] and the CNL used in CNL2ASP [9]. However, the natural representation of a CNL can also be its undoing: user studies have indicated that, contra-intuitively, *expressing* knowledge is deceptively difficult [25].

Other notations for lowering the modeling threshold focus on repurposing existing standards, such as UML [18] and Feature Models [30,39]. For specific domains, a common approach is to introduce simplified, domain-specific notations, as exemplified by for instance COOM [2] for configuration problems, and PROLEG [35] for legal documents. Another popular notation format is the use of blocks-based notations to represent logic programs [31,29].

Lastly, the past advances on Large Language Models (LLMs) also offer new opportunities, as autoformalization techniques can automatically convert natural language descriptions of problem domains into logic programs [22,23,6].

[2] Available online: https://gitlab.com/EAVISE/cdmn/cdmn-solver/-/tree/asp

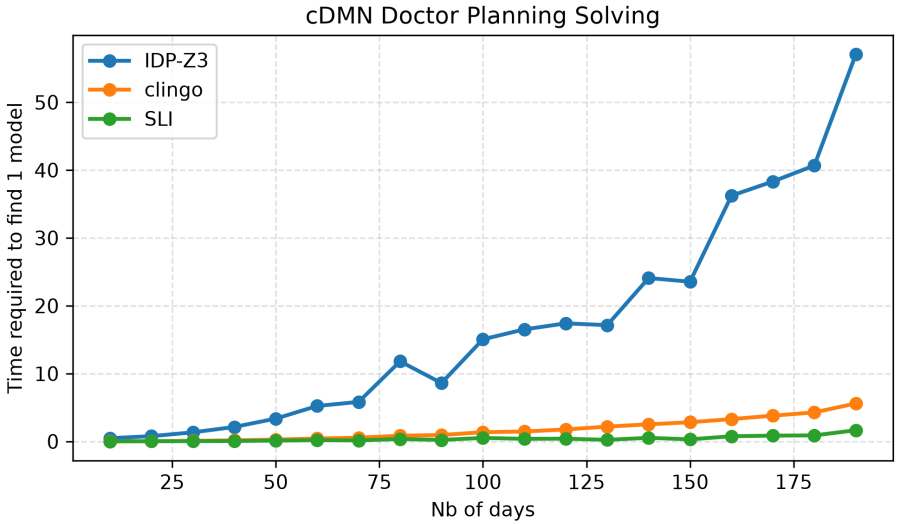


Fig. 6: Benchmark of the Doctor Planning cDMN model for finding a schedule of x days. All three systems use an encoding generated directly by the cDMN solver.

7 Conclusion

Knowledge formalization remains one of the biggest hurdles in the adoption of KR-based approaches. In this paper, we have introduced an ASP translation of the cDMN notation, which aims to make ASP more accessible to non-experts. From experience, we feel that cDMN is significantly easier to learn compared to standard KR languages, as (a) it builds off of an existing user-friendly notation, (b) it has an intuitive table-based format, and (c) it already feels familiar to non-experts that work in spreadsheet software. However, further research is required to actually measure the modeling complexity experienced by non-experts. As part of future research, we want to evaluate the ASP notation to generate better optimized encodings, and we want to test more cDMN examples to find any missing features.

8 Acknowledgements

The author thanks Javier Romero for his indispensable assistance in coming up with the initial ASP translation.

References

1. Alviano, M., Cirimele, D., Reiners, L.A.R.: Introducing asp recipes and asp chef. In: Proceedings of the International Conference on Logic Programming 2023

Workshops. vol. 3437. CEUR Workshop Proceedings (2023), https://ceur-ws.org/Vol-3437/paper4ASPOCP.pdf, copyright © 2023 by the authors. Use permitted under the Creative Commons License Attribution 4.0 International (CC BY 4.0).
2. Baumeister, J., Herud, K., Ostrowski, M., Reutelshöfer, J., Rühling, N., Schaub, T., Wanko, P.: Towards industrial-scale product configuration. In: Dodaro, C., Gupta, G., Martinez, M.V. (eds.) Logic Programming and Nonmonotonic Reasoning. pp. 71–84. Springer Nature Switzerland, Cham (2025)
3. Beiser, A., Hahn, S., Schaub, T.: ASP-driven user-interaction with clinguin. Electronic Proceedings in Theoretical Computer Science **416**, 215–228 (Feb 2025). https://doi.org/10.4204/eptcs.416.19, http://dx.doi.org/10.4204/EPTCS.416.19
4. Bichler, M., Morak, M., Woltran, S.: Lpopt: A Rule Optimization Tool for Answer Set Programming. Fundamenta Informaticae **177**(3-4), 275–296 (Dec 2020). https://doi.org/10.3233/FI-2020-1990, https://journals.sagepub.com/action/showAbstract
5. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011), https://doi.org/10.1145/2043174.2043195
6. Callewaert, B., Vandevelde, S., Vennekens, J.: Verus-lm: a versatile framework for combining llms with symbolic reasoning (2025), https://arxiv.org/abs/2501.14540
7. Calvanese, D., Montali, M., Dumas, M., Maggi, F.M.: Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge. Theory and practice of logic programming **19**(4), 536–573 (2019)
8. Carbonnelle, P., Vandevelde, S., Vennekens, J., Denecker, M.: IDP-Z3: A reasoning engine for FO(.) (2022). https://doi.org/10.48550/arXiv.2202.00343
9. Caruso, S., Dodaro, C., Maratea, M., Mochi, M., Riccio, F.: CNL2ASP: Converting Controlled Natural Language Sentences into ASP. Theory and Practice of Logic Programming pp. 1–31 (2023). https://doi.org/10.1017/S1471068423000388, https://www.cambridge.org/core/product/AF5901FADC579E49C583CFD5A10C0192
10. Colmerauer, A., Roussel, P.: The birth of prolog. Communications of the ACM **35**(9), 30–32 (1992)
11. Cullen, J., Bryman, A.: The Knowledge Acquisition Bottleneck: Time for Reassessment? Expert Systems **5**(3), 216–225 (Aug 1988). https://doi.org/10.1111/j.1468-0394.1988.tb00065.x, https://doi.org/10.1111/j.1468-0394.1988.tb00065.x
12. Dasseville, I., Janssens, L., Janssens, G., Vanthienen, J., Denecker, M.: Combining DMN and the knowledge base paradigm for flexible decision enactment. Supplementary Proceedings of the RuleML 2016 Challenge, vol. 1620. Athan, Tara, CEUR-WS.org (2016), $$Uhttp://ceur-ws.org/Vol-1620
13. De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., Denecker, M.: Predicate logic as a modeling language: The IDP system. In: Kifer, M., Liu, Y.A. (eds.) Declarative Logic Programming: Theory, Systems, and Applications, pp. 279–323. ACM (Sep 2018). https://doi.org/10.1145/3191315.3191321, https://dl.acm.org/citation.cfm?id=3191321
14. Delgrande, J.P., Glimm, B., Meyer, T., Truszczynski, M., Wolter, F.: Current and future challenges in knowledge representation and reasoning (dagstuhl perspectives workshop 22282). Dagstuhl Manifestos **10**(1), 1–61 (2024). https://doi.org/10.4230/DagMan.10.1.1, https://drops.dagstuhl.de/entities/document/10.4230/DagMan.10.1.1
15. Denecker, M., Ternovska, E.: A logic of nonmonotone inductive definitions. ACM Trans. Comput. Logic **9**(2) (Apr 2008). https://doi.org/10.1145/1342991.1342998, https://doi.org/10.1145/1342991.1342998

16. Denecker, M., Vennekens, J.: Building a Knowledge Base System for an Integration of Logic Programming and Classical Logic. In: Garcia de la Banda, M., Pontelli, E. (eds.) Logic Programming, vol. 5366, pp. 71–76. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89982-2_12, http://link.springer.com/10.1007/978-3-540-89982-2_12
17. Deryck, M., Hasić, F., Vanthienen, J., Vennekens, J.: A case-based inquiry into the decision model and notation (DMN) and the knowledge base (KB) paradigm. In: Rules and Reasoning. Lecture Notes in Computer Science, vol. 11092, pp. 248–263. Springer International Publishing, Cham (2018)
18. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: Configuration Knowledge Representation Using UML/OCL. In: Jézéquel, J.M., Hussmann, H., Cook, S. (eds.) ≪UML≫ 2002 — The Unified Modeling Language. pp. 49–62. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
19. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web: 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures, pp. 104–124. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85658-0_3, https://doi.org/10.1007/978-3-540-85658-0_3
20. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory and Practice of Logic Programming **19**(1), 27–82 (Jul 2018). https://doi.org/10.1017/s1471068418000054, http://dx.doi.org/10.1017/s1471068418000054
21. Hasic, F., De Smedt, J., Vanthienen, J.: Towards assessing the theoretical complexity of the decision model and notation (DMN) research-in-progress. In: CEUR Workshop Proceedings. vol. 1859, pp. 64–71. CEUR Workshop Proceedings (2017)
22. Ishay, A., Yang, Z., Lee, J.: Leveraging large language models to generate answer set programs. In: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning. pp. 374–383 (Aug 2023). https://doi.org/10.24963/kr.2023/37, https://doi.org/10.24963/kr.2023/37
23. Jha, S.K., Jha, S., Lincoln, P., Bastian, N.D., Velasquez, A., Ewetz, R., Neema, S.: Neuro symbolic reasoning for planning: Counterexample guided inductive synthesis using large language models and satisfiability solving (2023)
24. Kowalski, R.: Logical english. In: Logic and Practice of Programming (LPOP) 2020 (2020), https://www.doc.ic.ac.uk/~rak/papers/LPOP.pdf
25. Kuhn, T.: A Survey and Classification of Controlled Natural Languages. Computational Linguistics **40**(1), 121–170 (Mar 2014). https://doi.org/10.1162/COLI_a_00168, https://doi.org/10.1162/COLI_a_00168
26. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Trans. Comput. Logic **7**(3), 499–562 (Jul 2006). https://doi.org/10.1145/1149114.1149117, https://doi.org/10.1145/1149114.1149117
27. Leu, G., Abbass, H.: A multi-disciplinary review of knowledge acquisition methods: From human to autonomous eliciting agents. Knowledge-Based Systems **105**, 1–22 (Aug 2016). https://doi.org/10.1016/j.knosys.2016.02.012, https://www.sciencedirect.com/science/article/pii/S0950705116000988
28. Marković, D., Vandevelde, S., Vennekens, J., Denecker, M.: On the Semantics of "Null" in DMN: Undefined is not Unknown. In: Proceedings of the 16th International Rule Challenge and 6th Doctoral Consortium @ RuleML + RR. vol. 3229, pp. 1–15. CEUR Workshop Proceedings, Berlin, Germany (Mar 2022), https://ceur-ws.org/Vol-3229/

29. Morris, J.: Blawx: Web-based user-friendly rules as code. In: Proceedings of the Workshop on Goal-Directed Execution of Answer Set Programs (GDEASP 2022). CEUR-WS.org, Haifa, Israel (2022), https://ceur-ws.org/Vol-3193/paper4GDE.pdf
30. Myllärniemi, V., Tiihonen, J., Raatikainen, M., Felfernig, A.: Using Answer Set Programming for Feature Model Representation and Configuration. In: Configuration Workshop. pp. 1–8 (2014)
31. Nicolay, J., Oeyen, B., Ndung'u, S.N., Renaux, T., Démarest, M., Verbeiren, B., De Meuter, W.: Combining Flows and Rules in a Low-Code Platform for Smart Water Management. CLOUD COMPUTING 2025 pp. 88–93 (Apr 2025)
32. Object Management Group: Decision model and notation v1.5 (2024), https://www.omg.org/spec/DMN/1.5
33. Post, R., Smit, K., Zoet, M.: Adoption and implementation of the decision model and notation standard. Journal of Advanced Management Science **8**(2), 49–54 (Jun 2020). https://doi.org/10.18178/joams.8.2.49-54, https://doi.org/10.18178/joams.8.2.49-54
34. Potassco: ngo. Online, https://potassco.org/ngo/, accessed: 2026-09-28
35. Satoh, K., Asai, K., Kogawa, T., Kubota, M., Nakamura, M., Nishigai, Y., Shirakawa, K., Takano, C.: PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology. In: Onada, T., Bekki, D., McCready, E. (eds.) New Frontiers in Artificial Intelligence. pp. 153–164. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
36. Van Laer, L., Vandevelde, S., Vennekens, J.: Efficiently Grounding FOL Using Bit Vectors. In: Dodaro, C., Gupta, G., Martinez, M.V. (eds.) Logic Programming and Nonmonotonic Reasoning. pp. 167–173. Springer Nature Switzerland, Cham (2025)
37. Vandevelde, S.: Simplifying Knowledge Base Creation for Domain Experts. Ph.D. thesis, KU Leuven (Jan 2024)
38. Vandevelde, S., Aerts, B., Vennekens, J.: Tackling the DM challenges with cDMN: A tight integration of DMN and constraint reasoning. Theory and Practice of Logic Programming pp. 1–24 (2021). https://doi.org/10.1017/S1471068421000491
39. Vandevelde, S., Callewaert, B., Vennekens, J.: Interactive feature modeling with background knowledge for validation and configuration. In: Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume B. pp. 209–216 (2022)
40. White, C., Schwitter, R.: An update on PENG light. In: Proceedings of the Australasian Language Technology Association Workshop 2009. pp. 80–88 (2009)

Appendix: Full Doctor Planning cDMN Model

Type		
Name	**DataType**	**PossibleValues**
Doctor	string	Fleming, Freud, Heimlich, Eustachi, Golgi
Time	string	Early, Late, Night
Day	string	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Shiftnumber	int	[0..21]

Function	
Name	**DataType**
doctor assigned to Day and Time	Doctor
nb shifts of Doctor and Day	Shiftnumber
nb nights of Doctor	Shiftnumber
day after Day	Day

Relation
Name
Doctor is available on Day and Time

1. Only shift per day.			
E*	Doctor	Day	nb shifts of Doctor and Day
1	-	-	≤ 1

Max Nb of Nights		
E*	Doctor	nb nights of Doctor
1	Golgi	≤ 2

2. A doctor should always be available for his shift.					
E*	Doctor	Day	Time	doctor assigned to Day and Time	Doctor is available on Day and Time
1	-	-	-	Doctor	Yes

3. Subsequent night shifts or the day off.					
E*	Doctor	Day called d1	doctor assigned to d1 and Night	Time	doctor assigned to day after d1 and Time
1	-	-	Doctor	not(Night)	not(Doctor)

4. A doctor either works both days of the weekend, or none of the days.					
E*	Doctor	Day called d1	nb shifts of Doctor and d1	Day called d2	nb shifts of Doctor and d2
1	-	Saturday	1	Sunday	1
2	-	Sunday	1	Saturday	1

Count shifts per day					
C+	Doctor	Day	Time	doctor assigned to Day and Time	nb shifts of Doctor and Day
1	-	-	-	Doctor	1

Count the number of night shifts for each doctor				
C+	Doctor	Day	doctor assigned to Day and Night	nb nights of Doctor
1	-	-	Doctor	1

Doctor info				
D	Doctor	Day	Time	Doctor is available on Day and Time
1	Fleming	Friday	Early	Yes
2	...	...	...	...
3	Golgi	Sunday	Night	Yes

Define Next Day		
D	Day	day after Day
1	Monday	Tuesday
2	Tuesday	Wednesday
3	Wednesday	Thursday
4	Thursday	Friday
5	Friday	Saturday
6	Saturday	Sunday
7	Sunday	Monday