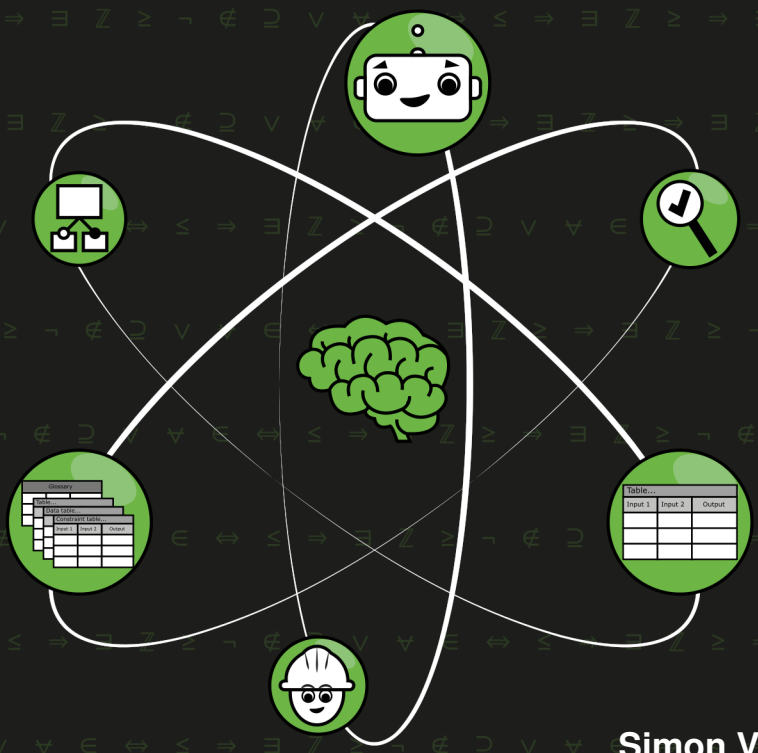
KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Technology

Simplifying Knowledge Base Creation for Domain Experts



Simon Vandevelde

Supervisor:
Prof. dr. J.Vennekens

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Technology (PhD)

January 2024

Simplifying Knowledge Base Creation for Domain Experts

Simon VANDEVELDE

Examination committee:
Prof. dr. ir. Jan Genoe, chair
Prof. dr. J.Vennekens, supervisor
Prof. dr. ir. G. Janssens
B. Coppens (Intelli-Select)
Prof. dr. ir. P. Karsmakers
Dr. R. Markovich
(University of Luxembourg)
Prof. dr. S. Vansummeren
(University of Hasselt)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Technology (PhD)

January 2024

Uitgegeven in eigen beheer, Simon Vandevelde, Jan Pieter Denayerlaan 5, 2860 Sint-Katelijne-Waver (Belgium)

Preface

In 2019, I finished my studies at KU Leuven De Nayer campus and became a software engineer. Suddenly, I had to make a decision – what do I do now? Torn between pursuing other studies or going into industry, I was suddenly offered a third option: starting a PhD at the faculty of Engineering Technology. A mix of industry and academia, and a chance to postpone "real life" for around four more years? Sounds perfect!

The thesis you are now reading is the result of those four years, which were filled with exploration, learning, and enjoyment. For this, I am incredibly grateful to Joost. Thanks for offering me the chance to start a PhD, and for guiding me throughout. A big thank you also goes out to the members of my examination committee for their valuable feedback during my PhD trajectory. This research would also not have been possible without the Flemish Government, who funded it under the "Onderzoeksprogramma Artificiële Intelligentie (AI)" programme.

I would also like to thank the many colleagues at EAVISE, DTAI and LIRIS that I interacted with, with a specific mention for the colleagues in our C204 office. Of course, I also cannot forget those colleagues that wrote a song about me (merci Laurent![1]).

Throughout the thesis, I also use many diagrams and images that have been created for me by others. Astrid, thank you for the beautiful cover page, and for prettifying my (very) ugly diagrams. Kaat, thank you for contributing the lovely robot drawings for FOLL-E. Lastly, Sari, thank you for your amazing digital drawing of my face just so that I could embellish my chapter introductions. :-)

Lastly, I express my gratitude collectively to all those who have been a part of my personal life, providing support, love, and understanding. Each person has played a unique role in shaping the person I am today, and for that, I am forever thankful.

[1] `https://suicidopolis.bandcamp.com/track/song-for-simon` – if you read a page of this thesis every 56 seconds, you can listen to it 45 times!

While I am sad that my PhD track is over, I am incredibly grateful for the experiences I got. :-)

Abstract

Knowledge Representation and Reasoning (KRR) is a sub-field of AI in which we model (a part of) the world in a declarative Knowledge Base (KB), so that computers can "reason" on it. The process of constructing such a KB typically starts with a knowledge extraction phase, in which an AI expert obtains the required knowledge from a domain expert. This "knowledge transfer" is often regarded as a costly, time-consuming, and error-prone process. One approach to facilitate the process is by helping the domain expert understand the KB: in this way, they can interpret and validate the knowledge, and in some cases, formalize it themselves.

In this thesis, we investigate how to make knowledge representation (KR) easier for domain experts by looking into user-friendly formalisms. First, we draw inspiration from two industry standards, DMN and Feature Modelling, and show that they are well-suited for KR. Not only are they designed to be user-friendly, our knowledge-based approach also allows us to repurpose the knowledge in a DMN model or feature diagram, which enables us to apply them to more problems than originally intended. In this way, these intuitive formalisms can be used to model and solve various real-life problems.

While DMN's simplicity and deterministic nature are some of its main benefits, there are also cases in which it is not sufficiently expressive. To this end, we introduce two extensions, cDMN and pDMN, which respectively extend DMN with constraint reasoning and with probabilistic reasoning. The idea is that by using DMN as a foundation, we can effectively extend the expressiveness while still maintaining the intuitive format of the original. We show for both notations that they are meaningful extensions which allow modelling new types of problems in a user-friendly way.

To further support the creation of KB's using DMN, we looked at improving DMN verification mechanisms. When modelling any KB, it is often a good idea to regularly check its correctness. For DMN however, we found the

state-of-the-art verification capabilities lacking. To overcome this, we present a knowledge-based approach for verification, and we show that it is more thorough than any other tool.

Throughout the course of this PhD, we have also worked on a use case concerning a decision support tool for adhesive selection. The main goal of this use case is to apply and evaluate our approaches. Through qualitative interviews with the domain experts, we gauge their opinions on our knowledge-based approach. In short, they appreciate the explainable and interactive nature of our tools, and are happy with the progress.

To better guide future research on formalisms, we additionally hold multiple qualitative interviews with modelling experts. The main goal of these interviews is to elucidate their modelling intuitions and instincts, and to discuss concepts such as "user-friendliness". Here, we learn many intricate details of their modelling process, such as the crucial role of tooling and their KISS (Keep It Simple, Stupid) approach.

Lastly, we present an "extreme" example of a user-friendly formalism as part of a tool to teach children First Order Logic (FOL). While FOL plays an important role in many fields, there is little research around teaching it at elementary school level. To address this gap, we created FOLL-E: a fun and engaging environment in which children can freely experiment with FOL. Through multiple workshops with children, we have found that children enjoy playing around with it, stating that it is intuitive and inviting.

Summarized, our work looks into simplifying KB creation for non-AI experts. By using tried and tested notations such as DMN and feature modelling, we can ensure that they are user-friendly. Moreover, even if such a notation is not sufficiently expressive, they still form an excellent foundation to build an extension on top of. When combined with proper tooling support, such formalisms significantly lower the threshold for non-AI experts to build their own KBs.

Beknopte samenvatting

Kennisrepresentatie en redeneren (KRR) is een onderdeel van AI waarin we (een deel van) de wereld modelleren als een declaratieve kennisbank (KB), zodat computersystemen hierover kunnen "redeneren". Om zo'n KB te maken starten we typisch met "kennisextractie", waarbij een AI expert kennis over een probleemdomein verkrijgt van een domeinexpert. Dit proces wordt aanzien als kostelijk, tijdrovend, en zeer kwetsbaar voor fouten. Een mogelijke manier om dit proces makkelijker te maken is het opleiden van de domeinexperten om de KB zelf te begrijpen: op deze manier kunnen ze de kennis zelf interpreteren en valideren. Sterker nog, in sommige gevallen zullen ze zelfs in staat zijn om hun eigen kennis te formaliseren.

In deze thesis gaan we na hoe we kennisrepresentatien (KR) toegankelijker kunnen maken voor domeinexperten met behulp van gebruiksvriendelijke formalismen. Om te beginnen halen we de mosterd bij twee industriestandaarden, namelijk DMN en Feature Modelling, en tonen we aan dat beide formalismen zich goed verlenen tot KR. Deze formalismen zijn niet enkel gemakkelijk in gebruik, maar ze zijn ook bruikbaar voor meer problemen dan oorspronkelijk bedoeld, dankzij onze kennisgebaseerde aanpak.

Hoewel eenduidigheid een van DMN's voornaamste troeven is, zijn er toch gevallen waarin we tegen expressiviteitslimieten lopen. Om dit te overwinnen introduceren we twee uitbreidingen: cDMN, welke "constraint reasoning" toevoegt, en pDMN, welke "probabilistic reasoning" toevoegt. In beide gevallen is het basisidee om DMN te gebruiken als toegankelijk fundament dat we uitbreiden met meer expressiviteit. We tonen voor beide notaties aan dat ze nuttige uitbreidingen vormen dat elk op hun beurt nieuwe types van problemen kunnen aanpakken op een gebruiksvriendelijke manier.

Om verder het gebruik van DMN als KR-formalisme te ondersteunen, hebben we ook diens verificatiemechanismen onder de loep genomen. Wanneer we een KB modelleren is het namelijk altijd een goed idee om deze frequent te verifiëren.

Echter, de huidige verificatiemechanismen laten in onze ogen nog steken vallen. Om dit op te lossen stellen we onze eigen kennisgebaseerde verificatie-aanpak voor, en tonen we dat het grondiger werkt dan eenieder andere aanpak.

Doorheen het doctoraat hebben we verder aan een kennisgebaseerd beslissingssysteem voor lijmselectie gewerkt. Het voornaamste doel hiervan was om onze aanpak te testen en te evalueren op een probleem van realistische schaal. Aan de hand van qualitatieve interviews met de domeinexperten proberen we hun ervaringen te achterhalen. De voornaamste bevindingen hierbij zijn dat ze de verklaarbaarheid en interactiviteit van onze aanpak ten sterkste waarderen, en dat ze dan ook tevreden zijn met de vooruitgang tot dusver.

Om toekomstig onderzoek over formalismen beter te sturen hebben we ook modelleerexperten bevraagd via qualitatieve interviews. Het voornaamste doel hiervan is om hun modelleerinstincten en -intuities te achterhalen, en om concepten zoals "gebruiksvriendelijkheid" te bespreken. Hierdoor leren we veel details over hun modelleerprocessen, zoals de belangrijke rol van een model-ontwikkelingomgeving en het KISS principe (Keep It Simple, Stupid).

Tot slot presenteren we een "extreme" vorm van een gebruiksvriendelijk formalisme in de vorm van een applicatie om eerste-orde logica aan te leren aan kinderen. Hoewel eerste-orde logica een belangrijke rol speelt in verscheidene velden is er weinig onderzoek omtrent het aanleren ervan aan kinderen. Om hier een oplossing voor te bieden hebben wij FOLL-E gemaakt: een interactieve applicatie waarin kinderen vrij kunnen experimenteren met eerste-orde logica. Aan de hand van meerdere workshops hebben we reeds bevonden dat kinderen zich effectief amuseren met FOLL-E, en dat de applicatie uitnodigend en intuïtief werkt.

Samengevat gaat deze thesis over het vergemakkelijken van KB-creatie door niet-AI experten. Door reeds bestaande notaties als DMN en Feature Modelling te gebruiken zijn we verzekerd over hun gebruiksvriendelijkheid. Bovendien, in het geval dat zulke notatie niet expressief genoeg zou zijn, vormen ze alsnog een uitstekende basis om op verder te bouwen. Wanneer we dit combineren met een goed uitgeruste ontwikkelingsomgeving wordt het maken van kennisbanken toegankelijker te maken voor niet-AI experten.

List of Abbreviations

ACE Attempto Controlled English. 129

AD Annotated Disjunction. 67

AI Artificial Intelligence. iii, v, vi, 1, 3, 4, 12, 18, 114, 115, 125, 146, 160, 171, 173, 174, 196

AMG Abstract Model Generation. 104

API Application Programming Interface. 14, 18, 28–31, 33–35, 40, 75, 155, 172

ASP Answer Set Programming. 83, 89–91, 129, 166

BMI Body Mass Index. 19, 21, 25, 26, 31–33

BPMN Business Process Model and Notation. xviii, 61, 62

cDMN constraint Decision Model and Notation. iii, v, xii, xvii–xix, 25, 37, 38, 40–49, 52–63, 68, 105, 110, 111, 124, 125, 128, 129, 139, 142, 143, 162–164, 166–168, 172–174, 190

CNL Controlled Natural Language. xiii, xix, 129–131, 139, 163

CP Constraint Programming. 56

CS Computer Science. 146, 147, 159, 160, 173

DMN Decision Model and Notation. iii–vi, xi, xvii–xix, 14, 17–19, 21–35, 37, 38, 40–48, 50, 53–56, 61–63, 65, 66, 68, 69, 72, 77, 93–95, 97, 101–103, 105, 106, 125, 128, 133, 137, 139, 142, 143, 162–167, 171–174

DRD Decision Requirements Diagram. 19, 29, 35, 63, 77, 110

DTAI Declaratieve Talen en Artificiële Intelligentie (Declarative Languages and AI). xx, 132, 142, 167, 168

FEEL Friendly Enough Expression Language. 22, 40

FM Flanders Make. 108, 109, 115, 116

FO(·) FOL extended with types, inductive definitions, aggregates, and more. xi, xix, 5–8, 11, 12, 18, 19, 24–26, 47, 49, 53, 81, 84, 86, 87, 89, 90, 92, 101, 103–106, 110, 125, 128, 130–132, 134–141, 143, 144, 171–173, 186, 188–193, 195, 196

FOL First Order Logic. iv, xiv, 5–7, 11, 12, 14, 15, 22, 49, 80, 101, 129, 146–148, 150, 151, 154, 159, 160, 173, 174

FOLL-E First Order Logic Learning Environment. vi, xviii, 13, 15, 147, 150, 155–158, 160, 168, 169, 173

GUI Graphical User Interface. 9, 10, 150, 155, 160

IC Interactive Consultant. 10, 140, 144, 165, 166, 173

JML Joining Materials Lab. 108, 115–117, 120, 123

K-12 From kindergarten to 12th grade, between 5 and 18 years old. 159, 168

KB Knowledge Base. iii–vi, 2, 3, 6, 8–12, 88, 89, 101–104, 110–112, 115, 124, 125, 130, 138–140, 143, 144, 162–164, 166, 171, 173, 174, 196

KBP Knowledge Base Paradigm. 2–5, 26, 138

KISS Keep It Simple, Stupid. iv, vi, 138, 139, 143, 196

KR Knowledge Representation. iii, v, xx, 132, 142, 163, 171, 172, 174

KRR Knowledge Representation and Reasoning. iii, v, xiii, 1, 2, 4, 12, 13, 17, 129, 131, 171

lex Lexical Analyzer Generator. 53

LGPLv3 GNU Lesser General Public License. 26, 167, 168

LP Logic Programming. 56

MIT MIT copyright license. 167, 168

NL Natural Language. 12, 130, 139

OMG Object Management Group. 18, 24, 61

pDMN probabilistic Decision Model and Notation. iii, v, xviii, 66, 68–77, 128, 164, 166, 167, 172

PENG Processable English. 129

PLP Probabilistic Logic Programming. 65, 66, 172

PyPI Python Package Index. 75

S-FEEL Simple Friendly Enough Expression Language. xix, 22–24, 40, 43, 44, 53

SAT Boolean Satisfiability Problem. 83, 146, 155

SPL Software Product Line. 79, 80

SSI Semi-Structured Interviews. 115

STEM Science Technology Engineering Maths. 155, 168, 169

XLSX Microsoft Excel Open XML Spreadsheet. 53, 74

yacc Yet Another Compiler-Compiler. 53

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In today's computer age, we see a mass trend towards computer-based automation. Entire industries abound with processes that are currently performed manually or are based on inefficient software. Here, effective software solutions promise an enormous gain in performance, freeing up employees to focus on more important aspects of their work.

This automation trend has recently been accelerated further by the sudden explosion of interest in "Artificial Intelligence". For many companies, AI almost seems inescapable, as they find themselves in an "adapt or die" situation. Yet, there are problems for which AI-based automation might seem infeasible, such as when an expert should remain in the loop, or when explainability and trust are of high importance. Here, knowledge-based AI might be the key: by explicitly representing knowledge on a problem domain, we can build computer programs to support human experts in an explainable way.

1.2 Knowledge Representation & Reasoning and the Knowledge Base Paradigm

Knowledge Representation and Reasoning (KRR) is a sub-field of Artificial Intelligence (AI), in which the goal is to model (a part of) the world in a symbolic representation so that computers can "reason" on it. In doing so, these

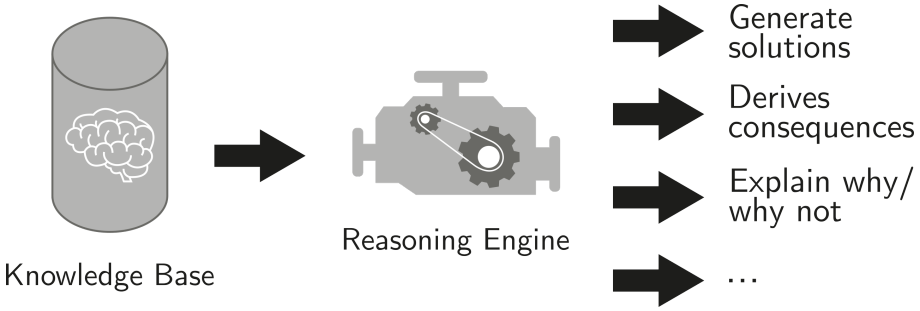


Figure 1.1: Diagram of the Knowledge Base Paradigm

computers can solve complex tasks "like a human would", by reasoning on our own knowledge.

The field of KRR contains many different approaches, each with their own unique characteristics and strengths. One of these approaches is the Knowledge Base Paradigm (KBP) [44], which distinguishes itself from the others by "returning to the classical view of logic as a formal language to express pure knowledge in a declarative way" [35]. The main idea of the KBP is as follows: knowledge is stored in a knowledge base (KB), independent of its intended use, so that different forms of inference can be applied in order to solve different problems. The key aspect here is solving different problems using the *same* KB, without requiring modifications. This KB contains a purely declarative representation of a problem domain – it does not describe *how* to solve a specific problem, but rather *what* the domain looks like. It is then up to a general reasoning engine to put this knowledge to use, e.g., by generating possible solutions or deriving consequences, as shown in Fig. 1.1. The main benefits of this approach are four-fold.

Natural representation As the knowledge base does not contain an algorithmic description of how to solve a problem, but rather a description of the problem itself, it is more intuitive to express knowledge on a problem. Expressing knowledge in the KBP leads to formulas that are more closely aligned to our natural intuitions, making them easier to understand and write.

Flexible Problems in the real world are not static – they change over time, thereby varying in size and complexity. Ideally, software should be able to adapt to these changes with a minimal number of modifications. Consider for example a snippet of law text describing a person's eligibility for social

assistance and how to calculate their total benefits. Such law is typically subject to frequent change, and any software tools dealing with social assistance need to be updated accordingly each time. In traditional computer programs, knowledge on a problem is usually "scattered" in the code base, making it rigid and tough to update. In the KBP on the other hand, the knowledge is represented *crisply*, "as-is", and is therefore typically straightforward to modify. Indeed, to incorporate any changes to the law we would only need to update the relevant snippets in the KB; the functionality of the reasoning engine remains the same.

Quick to build In general, the KBP lends itself well to building prototypes relatively quickly. This is due to the fact that we only need to focus on representing the problem domain, and not worry about the solving; this is "off-loaded" to the reasoning engine. One downside of using a general reasoning engine however is that its solving time for a specific problem is typically higher than that of an optimized algorithm. At the same time, there are cases in which development time is more costly than runtime: designing an optimized algorithm takes a lot of time and effort, whereas building a KB typically requires a fraction of the time.

Explainable Explainable AI is currently a hot topic in computer science, as many (data-based) AI approaches function as black boxes that cannot elaborate on why an output was produced. Yet, explainability is important: the people that experience the impact of an AI-based decision will, of course, want to know *why* a decision was made. Such research on explainable AI ties in nicely with other topics such as *fair* and *ethical* AI, especially when considering problem domains with sensitive information. AI systems that follow the KBP will always be explainable, as the knowledge on which the system reasons is explicitly present. In other words, it is possible to trace back the outputs of a decision and highlight the relevant knowledge; in fact, the reasoning engine can most likely perform this operation automatically. This explainability increases trust in the system. Though, it should be noted that explainable does not necessarily entail understandable – it is possible that laymen do not understand an explanation due to a lack of knowledge on the logic formalism. However, it is certainly understandable to the experts who built the system, which is a major benefit w.r.t. validation.

Of course, every rose has its thorns: as with any approach, there are also some disadvantages linked to the KBP. For example, as mentioned, the computational efficiency of tools following the KBP philosophy is typically lower than that of tools built to solve specific problems. However, an arguably bigger downside of

the KBP (and of KRR in general) is the "knowledge acquisition" bottleneck. Knowledge acquisition is the process of condensing real-life knowledge (from one or more experts) into a formal representation, and it is sometimes regarded as the most difficult part of creating knowledge-based tools. There are a few reasons for this difficulty. Firstly, the knowledge an expert has is *tacit*, and often difficult to put into words; they frequently make assumptions internally, and forget to explain them. Secondly, because the domain expert is not an AI expert (i.e., they cannot formalize their own knowledge), a "knowledge engineer" is needed. Communication between the former and the latter is error-prone and costly however, as neither are experts in both fields and miscomprehensions slip in quickly. Thirdly, once the knowledge is formalized, it can be difficult to a domain expert without a good grasp on logic to validate it, as they might not fully understand what is written down.

One knowledge acquisition methodology that aims to tackle these problems is *Joint Interactive Modelling* (Fig. 1.2). This methodology is straightforward to understand by dissecting its name:

Joint While knowledge acquisition is always a collaborative process, Joint Interactive Modelling takes it a step further by actively including the domain expert in the formalisation. Here, a requirement is that they should be able to actively partake in the modelling of the knowledge. In other words, they should be able to see what is written down, understand the notation its written in, and be able to formalize their knowledge together with the knowledge expert. In this way, the expert can immediately perform an initial validation: "Does what I explained correspond to what the knowledge engineer and I just wrote down?".

Interactive In Joint Interactive Modelling, building the knowledge base should be *interactive*, in the sense that prototypes should be made quickly and frequently. This ensures that a more thorough, incremental validation is possible, by letting the domain expert "play around" with the formalized knowledge. If any errors are discovered, they can be dealt with quickly.

So, in general, the idea of Joint Interactive Modelling is to include the domain expert in every step of the process, and to support tight feedback loops.

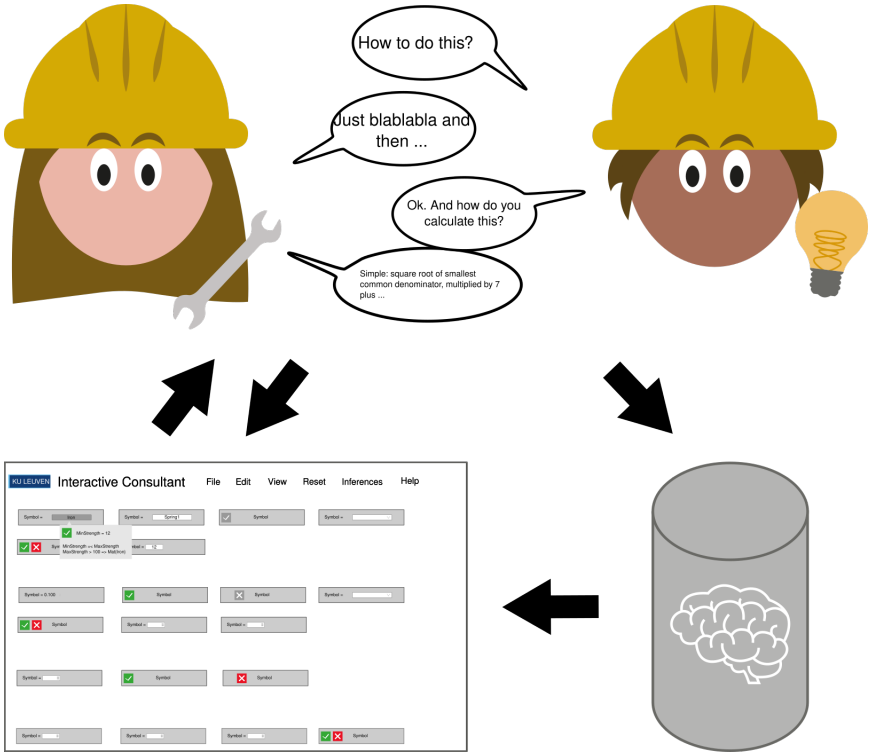


Figure 1.2: Diagram of the Joint Interactive Modelling process

1.3 The IDP System

The IDP system [42, 35] is an implementation of the KBP. Here, the knowledge in the KB is represented in a rich extension of First Order Logic (FOL), called FO(·) (pronounced *"eff-oh-dot"*), which extends FOL with types, aggregates, inductive definitions and more. The name "IDP system" is an umbrella-term for a number of KBP reasoning engines for FO(·), the latest of which is IDP-Z3[1]. As IDP-Z3 is used throughout this thesis, we will now give a brief introduction on the FO(·) formalism, on IDP-Z3's inference tasks and on an interactive, IDP-Z3-based interface.

[1] `www.IDP-Z3.be`

1.3.1 FO(·)

FO(·) is an expressive and flexible knowledge representation language, capable of modelling complex domains. Knowledge in an FO(·) KB is structured in three kinds of blocks: *vocabularies*, *structures* and *theories*. We will briefly go over each block, and illustrate their usage with an FO(·) implementation of the map colouring problem.

A *vocabulary* specifies a set of symbols. A symbol is either a type, function, or a predicate. A type represents a range of values, e.g., an enumeration list of countries called *Country* or the domain of real numbers $\mathbb{R}$. A function symbol represents a function from the Cartesian product $T_1 \times \ldots \times T_n$ of a number of types to a type T_{n+1}. In the listing below, the function $colour\_of : Country \rightarrow Colour$ maps each country on a colour. A predicate symbol expresses a function on range $\mathbb{B}$, i.e., each set of arguments is either *true* or *false*. For example, the predicate $borders : Country \times Country \rightarrow \mathbb{B}$ can be used to express when two countries share a border.

```
vocabulary {
    type Country := {BE, NL, FR, LU, DE}
    type Colour := {Red, Green, Blue, Yellow}
    colour_of: Country → Colour
    borders: Country × Country → 𝔹
}
```

A *(partial) structure* specifies an interpretation for (some of) the symbols of a given vocabulary. A structure is *total* if it specifies an interpretation for each symbol of the vocabulary. The structure below contains an interpretation for *borders*, which intuitively denotes whenever two countries border.

```
structure {
    borders := {(BE, NL), (BE, FR), (BE, LU), (BE, DE),
                (LU, FR), (LU, DE), (LU, BE),
                (DE, FR), (DE, NL)}.
}
```

A *theory* contains a set of logical formulae in FO(·). For example, the formula in the theory below expresses that "For every country c_1 and c_2 must hold that if they border, they do not have the same color".

```
theory {
    ∀c1, c2 in Country: borders(c1, c2) ⇒ color_of(c1) ≠ color_of(c2).
}
```

As an extension of FOL, FO(·) uses the standard logical connectives and quantifiers shown in Table 1.1. On top of them, it adds types, definitions,

aggregates, and more. We will briefly go over the former three; for detailed explanations of all extensions, we refer to the FO(·) standard [33].

Types have already been introduced in the previous paragraphs as domains of elements. IDP-Z3 supports four[2] built-in types: $\mathbb{B}$ (Bool), $\mathbb{Z}$ (Int), $\mathbb{R}$ (Real) and *Date*. We can also introduce custom types, e.g., containing strings or specific number enumerations, such as *Country* and *Colour*. Quantifiers in formulas must always be typed to prevent domain mismatches.

FO(·) also introduces definitions, which define a symbol by enumerating a set of necessary and sufficient conditions in the form of rules. A definition may also be *inductive*, i.e., may define a symbol in terms of the symbol itself. For example, the definition below expresses when a country $c2$ can be reached from a country $c1$. The first rule expresses that a country $c2$ is reachable if there is a border, and the second rule expresses that a country $c2$ is reachable if there is a reachable country $c3$ which in turn can reach $c2$.

```
{
    ∀c1, c2 in Country: reachable(c1, c2) ← borders(c1, c2).
    ∀c1, c2 in Country: reachable(c1, c2) ← ∃c3: reachable(c1, c3)
                                              ∧ reachable(c3, c2).
}
```

A third extension to FOL is support for aggregates; more specifically, the *sum, min, max* aggregates and the cardinality expression. These are operators which evaluate (multi)sets and respectively return their sum, minimum value, maximum value or their cardinality. For example, the formula below counts how many neighbours each country has, using a cardinality.

```
∀c in Country: nb_neighbours(c) = #{c2 in Country: borders(c, c2)}.
```

IDP-Z3 also implements some convenient notations in the form of syntactic sugar for expressions that frequently occur. Examples of such expressions are the *in*-operator and direct quantification, which are both demonstrated below.

```
// Below formulas are semantically equivalent.
colour_of(BE) in {Red, Green, Yellow}.
colour_of(BE) = Red ∨ colour_of(BE) = Green ∨ colour_of(BE) = Yellow.

// Below formulas are semantically equivalent.
∀(c1, c2) in borders: colour_of(c1) ≠ colour_of(c2).
∀c1, c2 in Country: borders(c1, c2) ⇒ colour_of(c1) ≠ colour_of(c2).
```

[2]Excluding the ones introduced for predicate and function symbols as *Concept* – see `https://fo-dot.readthedocs.io/en/latest/Concept.html`.

Table 1.1: Logical operators in FO(·)

Operator	Meaning
$\phi \wedge \psi$	ϕ and ψ must hold
$\phi \vee \psi$	ϕ or ψ must hold (inclusive)
$\neg\phi$	ϕ is false
$\phi \Rightarrow \psi$	ϕ implies ψ
$\phi \Leftrightarrow \psi$	ϕ and ψ are equivalent
$\forall x\ in\ T : \phi(x)$	$\phi(x)$ must hold for every x in type T
$\exists x\ in\ T : \phi(x)$	$\phi(x)$ must hold for at least one x in type T

The IDP-Z3 documentation [34] maintains an exhaustive list of all these "extra" expressions that are allowed when building a KB.

1.3.2 Inference tasks

By itself, a KB cannot be executed: it is merely a "bag of knowledge", without information on how it should be used. IDP-Z3 supports many different inference tasks that can be applied to this knowledge. We will briefly go over the inference tasks that are relevant to this work.

Model expansion Given a partial structure $\mathcal{I}$, model expansion extends it to a complete I that satisfies the theory T ($I \models T$). Note that this partial structure can also be empty. In plain English, model expansion generates complete models based on the information that is already known.

Propagation Given a partial interpretation $\mathcal{I}$ for the vocabulary of a theory T, propagation derives the consequences of $\mathcal{I}$ according to T, resulting in a more precise partial interpretation $\mathcal{I}'$. In other words, propagation will derive truth values which hold in all possible models, i.e., are "shared" between them.

Optimization Optimization is similar to model expansion, but looks for the model with the lowest/highest value for a given term. If multiple models exist, IDP-Z3 will instead return a partial structure containing only those interpretations that are shared between these models. This is different from previous versions of IDP.

Explanation Given a (partial) structure $\mathcal{I}$ which does not satisfy the theory T ($\mathcal{I} \not\models T$), explanation will find minimal subsets of the interpretations in $\mathcal{I}$ and the formulas in T which together explain why the structure does not satisfy the theory. Note that this explanation is not necessarily minimal in the sense that it is the smallest possible explanation, but instead minimal in the sense that it contains no irrelevant information. IDP-Z3 cannot guarantee that it generates the smallest possible explanation for an unsatisfiability.

Abstract Model Generation Given a theory T and a partial interpretation $\mathcal{I}$, abstract model generation searches for a set C of simple formulas that imply the theory, i.e., such that for all I that extend $\mathcal{I}$, $I \models C \rightarrow T$. Each interpretation that satisfies all of the formulas C is a model of T; in this sense, the formulas form an abstract representation of a class of models of T.

Relevance Given a theory T and a partial interpretation $\mathcal{I}$, the relevance inference will determine which symbols are still *relevant*. In other words, it will look for the variables which are an essential part of at least one model.

In the past, the IDP system has already proven itself as a suitable tool in a number of use cases [3, 4, 6, 45, 47, 48]. Its approach works well to tackle complex problems in an interactive way, typically by supporting experts in leveraging their own knowledge. These problems can be found in many domains, such as manufacturing, finance, logistics, law, and more. IDP-Z3 typically performs best if there are many constraints over relatively small domains.

An example of IDP-Z3 running in production is the OSCAR tool developed by Intelli-Select[3], which has been integrated in the toolkit of one of the global leading financial institutions [86].

1.3.3 Interactive Consultant

The Interactive Consultant [32] is a graphical user interface for IDP-Z3, aimed at facilitating interaction between a user and the system. It is a generic interface, in the sense that it is capable of generating a view for any syntactically correct KB. In short, each symbol of the KB is represented using a *symbol tile*, which allows users to set or inspect that symbol's value. In this way, the GUI represents a partial structure to which we can add and remove values. Each time a value is added, removed or modified, IDP's propagation is performed and the interface is updated: symbols for which the value was propagated are updated accordingly,

[3] `https://intelli-select.com/`

and values that are no longer feasible for the other symbols are removed. In this way, a user is *guided* towards a correct solution: they cannot enter a value that would make the partial structure represented by the current state of the GUI inconsistent with the theory.

An example of the interface in action is shown in Fig. 1.3. Here, a knowledge base dealing with event organisation according to the Belgian 2021 Covid-19 guidelines has been loaded in the Interactive Consultant. The user has already made two choices, namely that there are 10 participants, and that they do not have a Covid Safe Ticket. Based on this information and the rules in the KB, IDP-Z3 derives that *Indoor sport* is no longer feasible (as indicated by the grey cross). Moreover, it has also derived that "end time() $\leq$ 11" is now irrelevant, as it will have no influence on potential models.

At any point in time the user can ask for an explanation of a system-derived value, e.g., when the user does not understand it or agree with it. The system will then respond with the relevant formulas and user-made assignments that lead to the derived value, as shown for *Indoor sport* in the example. In this sense, the tool is explainable, leading to more trust in the system.

A similar functionality is in place for the rare cases in which a user manages to reach an *inconsistent* state, i.e., a set of assignments that can no longer be extended to a solution. While the IC removes values that have become impossible, it cannot do so for variables belonging to unbounded integer or real domains. For example, if the theory contains a rule stating that the cost of a product must always be below a maximum cost value, it would still be possible to input something which contradicts this. A user could for instance input "Max Cost = 20" followed by "Cost = 30", leading to an obvious conflict. As both of these variables can have any value between 0 to $+\infty$, the Interactive Consultant uses a free-form input field for these variables instead of a limited drop-down list. If such an inconsistency arises, the interface alerts the user and explains why no solutions are possible by showing the relevant design choices and laws.

One of the design goals of the Interactive Consultant is to empower users to interactively explore a problem space. Indeed, due to its intuitive interface and powerful solver, users can seamlessly interact with the knowledge in the KB. The Interactive Consultant interface has already proven itself to be useful in a number of applications in the past [4, 48].

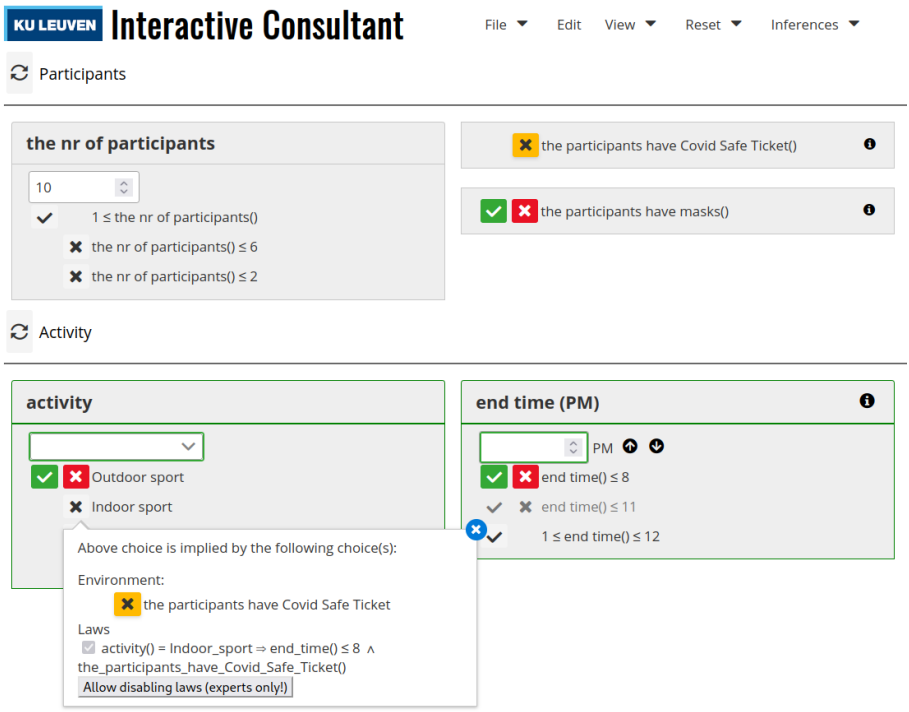


Figure 1.3: Screenshot of IC with Covid guidelines example.

1.4 Research Goal

The research goal of this thesis is to simplify the process of creating knowledge bases for domain experts. While IDP-Z3 is a powerful, multi-faceted reasoning engine, creating an FO(·) knowledge base remains a hurdle in the conception of IDP-based tools. Though the knowledge acquisition methodology of joint interactive modelling already ameliorates the process, FO(·) is not an ideal formalism. While it is possible that domain experts with a formal mathematics background (e.g., engineers) have a strong knowledge of FOL, most domain experts might find the language quite daunting and will not be able to express their knowledge in it, let alone interpret FO(·) written by someone else. Therefore, the main goal of this thesis is to facilitate the Joint Interactive Modelling process by investigating and/or designing alternative knowledge formalisms. For such a notation to be suitable as a KB language, there are a few requirements imposed on it.

User-friendly The user-friendliness of a notation is an important factor, as non-AI experts should be able to understand it sufficiently well in order to be able to express their knowledge. Ideally, we want a notation for which the basics can be explained quickly, and for which the more intricate details can be learned during the first modelling session(s). Here, intuitiveness is key: if a domain expert can "naturally" interpret what is written down, they will be able to quickly build instincts on how to use and interpret the formalism.

As mentioned, FO(·) might not meet these requirements for laypeople. At the same time, we do not need to re-invent the wheel by making up our own formalisms: there are many industry standards for representing knowledge in one way or another which could also lend themselves well to KRR. For this reason, some chapters in this thesis will focus on evaluating or extending existing standards on their KRR capabilities.

Expressive Besides user-friendly, a good notation should also be sufficiently expressive to correctly model a problem. Note that "correctly" here consists of two aspects: (1) the knowledge in the KB should accurately model the problem, and (2) it should do so in a way intended by the notation, without "abusing" it. The latter aspect is important, as having to resort to notation abuse will make the resulting KB less intuitive to interpret.

Expressiveness and user-friendliness typically counter-act each other, as there exists a trade-off between the two. Indeed, a user-friendly notation might have limited expressiveness to ensure it remains "simple to use", while an expressive notation could support many complex constructs which lead to a steeper learning curve.

Precise A good notation is as precise and non-ambiguous as possible. As an example, consider natural language (NL). NL is user-friendly (we are all taught it from birth), and sufficiently expressive to model many problems. Yet, at the same time, it is very difficult to convert NL texts into FOL. Take for instance the sentence "One morning I shot an elephant in my pyjamas."[4]. While the meaning might seem straightforward, the sentence can actually be interpreted in multiple ways – was I wearing the pyjamas, or the elephant? As our end goal is to use formalisms in conjunction with IDP-Z3, they should have a clear and unambiguous semantics, that is formally defined and straightforward to translate into FOL. Note that this does not mean that the work of this thesis is IDP-specific; on the contrary, being straightforward to translate into FOL also allows a notation to be translated to other common languages for reasoning

[4] "One morning I shot an elephant in my pyjamas. How he got into my pyjamas I'll never know." – Groucho Marx

engines, such as SMT-LIB [9], ASP-Core [25] and MiniZinc [100]. In this sense, the output of this work is also useful for a myriad of logic-related tools and software.

1.5 Contributions

While user-friendly KRR formalisms are the main focus of the thesis, we have also worked on other aspects of the Joint Interactive Modelling process. The contributions of the thesis can be divided into five categories.

1. We have examined the KRR-potential of two existing industry modelling standards, namely "Decision Model and Notation" (Chapter 2) and "Feature Modelling" (Chapter 5), and built tooling to support their application.

2. We have designed two extensions of the Decision Model and Notation to further enhance its expressiveness. Firstly, "constraint Decision Model and Notation" (Chapter 3) builds upon the notation by extending it with constraint reasoning and related concepts. Secondly, "probabilistic Decision Model and Notation" (Chapter 4) adds probabilistic reasoning to the notation. Here too, we built proper tooling to support their use for solving concrete problems.

3. We have extended the verification capabilities of the Decision Model and Notation standard with background knowledge (Chapter 6) and integrated them in our tool. These verification capabilities are an excellent addition to support Joint Interactive Modelling, as they form yet another layer of verification and validation of the knowledge.

4. We have performed two qualitative studies. Firstly, we gauged domain expert's opinions on our IDP-based approach in the context of a use case on adhesive selection (Chapter 7). The output of this study is a valuable source of information to guide further development of our tools and approaches. Secondly, we performed a study on knowledge modelling formalisms by interviewing multiple knowledge experts (Chapter 8). Here, the aim is to get a grasp on the instincts of the modellers w.r.t. which aspects are important when selecting a KRR formalism. The output of the chapter is therefore a condensed opinion of multiple knowledge experts, which can serve as a guide to pick the correct formalism for a problem.

5. We have developed an "extreme" example of a user-friendly notation in the form of a didactic tool for children, named FOLL-E (Chapter 9). FOLL-E

combines a physical, block-based notation for FOL with an embedded IDP-Z3 application into an educational game for children.

An overview of all software developed or contributed to throughout the course of this thesis is available in Appendix A.

1.6 Outline

In the next chapter, we will begin by introducing the Decision Model and Notation standard (DMN), and elaborate on its semantics. We also introduce two tools to leverage DMN using IDP-Z3: an integration between a DMN editor and the Interactive Consultant, and a flexible Python API for our IDP-Z3-based DMN engine.

In Chapter 3, we present our first extension, the constraint Decision Model and Notation. Here, the goal is to extend DMN's expressiveness while maintaining its user-friendly, tabular format. We describe the syntax and semantics of the notation, and compare it to other approaches based on DMCommunity challenges.

Chapter 4 describes our probabilistic DMN extension. The motivation for this extension is similar to the previous one: we wish to extend the expressiveness (with probabilistic reasoning), while maintaining the user-friendliness of the original formalism. We show the syntax of our extension, and describe how it can be translated to a state-of-the-art probabilistic reasoning engine.

We introduce Feature Modelling in Chapter 5. This is an industry standard in the field of software product lines that lends itself well to modelling configuration problems. Based on a use case, we show the benefits of extending it with background knowledge and present an integration of feature modelling in the Interactive Consultant for that purpose.

In Chapter 6, we extend the existing verification capabilities of DMN to also include background knowledge. We show that this approach leads to a more thorough verification compared to existing approaches, and present an IDP-Z3-based verification suite.

Chapter 7 introduces the use case that has been worked out through this thesis. This use case proved an excellent opportunity to perform a qualitative study on our knowledge-based approach, and as such, the chapter summarizes the thoughts and comments of the domain experts.

A second qualitative study, in which we interview modelling experts, is presented in Chapter 8. The main goal of these interviews is elucidating the modelling approaches of the experts based on four real-life use cases. As such, the output of this chapter is a valuable source of information to guide future research w.r.t. modelling formalisms.

In Chapter 9, we present FOLL-E, a learning environment for first order logic targeted at children. Here the contribution is two-fold. Firstly, we have designed a tangible, block-based notation for FOL that allows children to express formulas without worrying about mathematical symbols or syntax errors. Secondly, we have incorporated this notation in an interactive, captivating tool to teach FOL.

Finally, we finish the thesis by concluding in Chapter 11 and we discuss possible valorisation of the research output in Chapter 10.

Chapter 2

Decision Model and Notation

The Decision Model and Notation standard (DMN) is an industry standard for decision logic. Due to its intuitive table-based format, DMN is considered very user-friendly. However, its use in industry is limited to expressing straightforward decision modelling problems, where given a set of inputs, we want to derive an output. In this chapter, we investigate the applicability of DMN as a full-fledged KRR formalism, and build tooling to support it.

This chapter is based on work presented at the Benelux Conference on Artificial Intelligence (BNAIC), November 2020 [130]; the Joint Conference on Rules and Reasoning (RuleML+RR), September 2021 [128]; and DecisionCAMP 2021, September 2021 [131]. Parts of this research were performed in collaboration with Vedavyas Etikala and Jan Vanthienen of the LIRIS research group.

2.1 Introduction

The Decision Model and Notation standard [102], designed by the Object Modelling Group (OMG), is a user-friendly, table-based notation for modelling decision logic. Its main goals are to make decision knowledge readable by everyone involved in the decision process (business people, IT experts), and to be executable. Since its start in 2015, DMN has quickly gained popularity in both industry [31, 69, 116] and academia [5, 40, 48].

Typically, DMN is used to automate day-to-day business decisions. Most DMN tools therefore focus on supporting the required functionalities for this specific use.

However, we believe that more ambitious uses of DMN are also possible. In particular, the knowledge that is contained in a DMN model could be used to build knowledge-based AI systems, that can implement various sorts of intelligent behaviour. Consider, for instance, a cobot tasked with assisting an operator in product assembly. It seems likely that the domain knowledge that such a cobot would need can be expressed in DMN, and, moreover, doing so would allow the domain knowledge to be written and maintained directly by the operators themselves, instead of requiring programmers or knowledge engineers as middle men.

To actually implement such a system, the functionality of typical DMN tools does not suffice. For instance, the cobot would need to figure out which sensor input is necessary for specific operations, and such functionality is not supported by the state-of-the-art DMN execution engines.

Here, a reasoning engine such as IDP-Z3 could form the missing link, as applying its powerful logical inference algorithms could effectively "unlock" more functionalities for a DMN model. Indeed, Dasseville et al. [40] already demonstrated this approach by manually translating DMN into FO($\cdot$). However, to build truly useful intelligent systems, this alone does not suffice: it is also necessary to combine these different inference tasks in a suitable way. Moreover, this should be done using the concepts and terminology from the original DMN model (instead of those from the FO($\cdot$) theory that the DMN model is translated to behind the scenes).

To further support the combination of DMN and IDP, we present two approaches.

- An integration between a DMN editor and the Interactive Consultant.
- A versatile Python API that combines DMN as a notation with the IDP system as a reasoning engine.

Both approaches share the same idea: tightly integrating the accessible DMN formalism with the powerful reasoning capabilities of the IDP system, thereby combining their advantages.

The second approach is similar in spirit to previous work on the PyIDP API [136], which exposes the functionality of IDP to Python programmers, allowing also the knowledge base itself to be represented in a pythonic syntax, rather than the usual syntax of FO(·). The difference to our work is that we now bring DMN into the mix to allow the knowledge to be maintained by domain experts, rather than Python programmers.

This chapter is structured as follows. First, we introduce DMN in Section 2.2, and elaborate on its semantics and the state-of-the-art execution methods. We then present a first approach for combining DMN and IDP-Z3 in Section 2.3, and follow it up by a second approach in Section 2.4. In Section 2.5, we demonstrate how this second approach can be used to build a *naive* chat bot based on DMN. We finish with a comparison to other DMN tools in Section 2.6 and a conclusion in Section 2.7.

2.2 Preliminary: Decision Model and Notation

DMN consists of two main components: a Decision Requirements Diagram (DRD), and decision tables. The DRD is a graph representing the *decision flow* throughout the DMN model. It shows a graphical overview of which decision tables are present, how they connect, which input variables are used, which data sources are needed, and more. Fig. 2.1 shows an example of a DRD with three decision tables, as represented by the rectangles, and four input variables, as represented by the ovals. The arrows between them represent the flow of information: the value of *BMI* is defined by the value of the inputs *Weight* and *Length*, the value of *BMI Level* is decided based on *BMI* and *Sex*, and the value of *Risk Level* is decided based on *BMI Level*, *Sex*, and *Waist*.

The second component of DMN consists of *decision tables*, which are tabular representations of straightforward decisions. Due to their format, these tables are seen as intuitive and user-friendly, with a low conceptual complexity [68]. Fig. 2.2 shows the decision tables for our BMI running example, taken from [53]. In DMN, decision tables contain one or more input variables (in green, below the table header) and one output variable (in blue), each corresponding to a column. A decision table defines the value of the output variable in terms of the value of the input variables. Each row of the table corresponds to a decision rule. We say that a rule *fires* whenever the actual value of the input variables match the values listed in the rule's cells. For example, if *BMI* = *27* and *Sex*

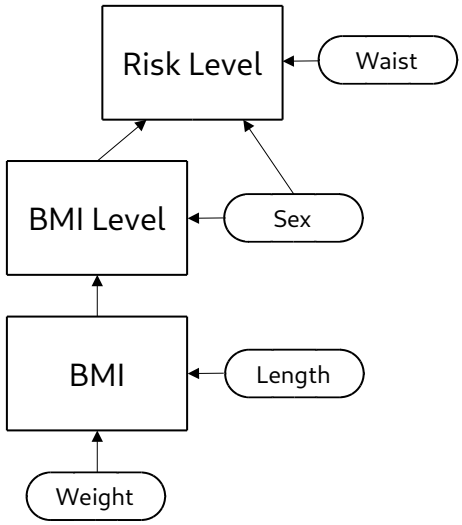


Figure 2.1: DRD for the BMI example

= *Female*, the fifth rule of the second table will fire, leading to a *BMILevel* of *Overweight*.

The way in which the inputs define the output depends on the *hit policy* of the table, as denoted in its top-left cell. There are two types of hit policies: single-hit policies, which denote that the output is determined by a single row, and multi-hit policies, which denote that the output is determined by a set of rows. More specifically, the single-hit policies are as follows:

U (-nique): the rules must be mutually exclusive, and may not overlap.

A (-ny): multiple rules may fire, as long as they specify the same output value.

F (-irst): the first applicable rule of the table is chosen to fire.

P (-riority): rules are assigned a priority, and the applicable rule with the highest priority fires.

The multi-hit policies are as follows:

C (-ollect): collect all output values of the fired rows in a list

C+ : sum all output values of the fired rules together.

BMI			
U	Weight	Length	BMI
1	—	—	Weight/(Length*Length)

BMI Level			
U	BMI	Sex	BMILevel
1	< 18.5	Female	Underweight
2	< 25	Male	Underweight
3	[18.5..25]	Female	Normal
4	(25..30]	Male	Normal
5	(25..30]	Female	Overweight
6	> 30	—	Obese

Risk Level				
U	BMILevel	Sex	Waist	Risk Level
1	Normal	—	—	Low
2	Underweight	—	—	High
3	Overweight	Male	≤ 102	Increased
4	Overweight	Male	> 102	High
5	Overweight	Female	≤ 88	Increased
6	Overweight	Female	> 88	High
7	Obese	Male	≤ 102	High
8	Obese	Male	> 102	Very High
9	Obese	Female	≤ 88	High
10	Obese	Female	> 88	Very High

Figure 2.2: Decision tables for the BMI example

C> : select the largest output value of the applicable rules.

C< : select the smallest output value of the applicable rules.

The example DMN model shown in Fig. 2.2 consists of three tables in total, defining BMI, BMI Level and Risk Level. In this example, all tables have the U hit policy, meaning that only a single row can fire per table. A cell containing "—" signifies that the value of this variable does not matter. For instance, if the BMI Level is underweight, the Risk Level is always high, regardless of sex and waist size. Note also that the table is *incomplete*, as there is no rule for $BMI = 25$ and $Sex = Male$. This means that the table is incorrect according to the DMN standard, as will be explained later and in Chapter 6.

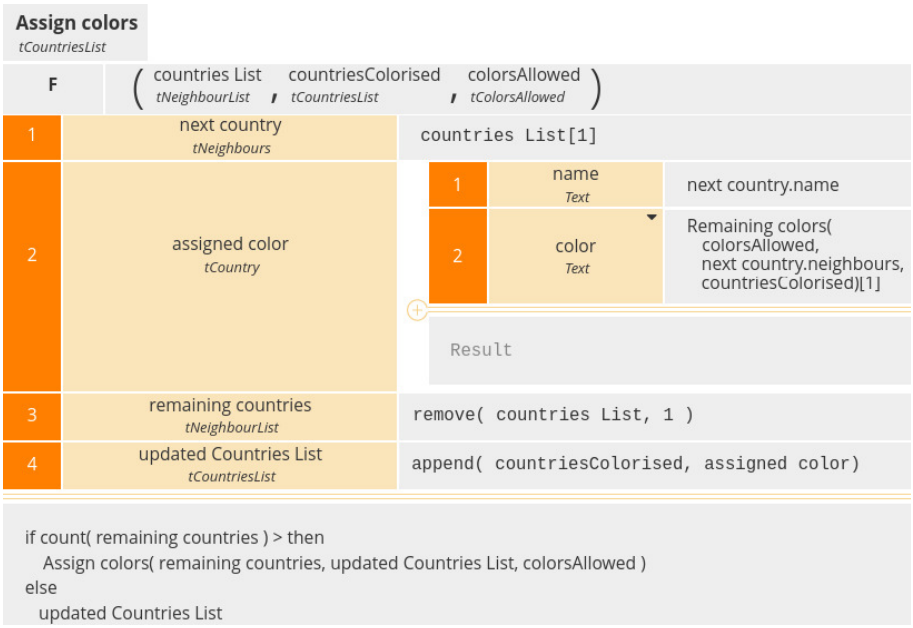


Figure 2.3: Example of boxed expression using full FEEL.

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.

In addition, DMN also allows more complex FEEL statements, in combination with other forms of tables called "boxed expressions". Fig. 2.3 shows an example of such a boxed expression. Note that the "F" in the top-left corner of the table does not denote the first hit policy, but rather that the boxed expression contains FEEL. Use of full FEEL and boxed expressions other than decision tables greatly increases complexity of the representation, which makes it unsuitable for use by domain experts without the aid of knowledge engineers. Therefore, we will limit ourselves to decision tables with S-FEEL expressions.

2.2.1 DMN Semantics

Most work on DMN and logics rely on the FOL semantics proposed by Calvanese et al. [27], in which rows are represented by implications that together in a

conjunction represent a table. The main shortcoming of this semantics is that it assumes all tables to be complete (there is an applicable row for each possible set of input values) and sound (there are no conflicting rules). To illustrate, consider the *BMI Level* table in our running example (Fig. 2.2) according to the semantics of [27]:

$$\begin{aligned} & BMI < 18.5 \wedge Sex = Female \Rightarrow BMILevel = Underweight \\ \wedge \quad & BMI < 25 \wedge Sex = Male \Rightarrow BMILevel = Underweight \\ \wedge \quad & 18.5 \leq BMI \leq 25 \wedge Sex = Female \Rightarrow BMILevel = Normal \\ \wedge \quad & 25 < BMI \leq 30 \wedge Sex = Male \Rightarrow BMILevel = Normal \\ & \ldots \end{aligned} \tag{2.1}$$

As mentioned, this table is not complete: there is no row specified for the combination of $BMI = 25$ and $Sex = Male$. According to the DMN standard, the table is therefore incorrect and it would thus be natural for the corresponding semantics to be inconsistent. However, for the semantics outlined above, this is actually not the case. Instead, the value of the *BMILevel* variable would be unrestricted, and could take on anything from *Underweight* to *Obese* for these input values. This side-effect of the semantics could lead to many problems if used to reason on DMN tables.

Analogously to the semantics proposed by Marković et al. [90], the semantics in this work relies on definitions instead of implications to represent rows. This has two benefits: firstly, we overcome the issue with incorrect tables outlined earlier, and secondly, definitions have better support for the relevance inference as there is a clear distinction between "input variable" and "output variable". We will now go over this semantics.

Each cell (i, j) of a decision table corresponds to a formula $F_{ij}(x)$ in one free variable x. For example, a cell containing "< 18.5" translates to the formula "x < 18.5". Table 2.1 shows an overview of all legal S-FEEL cell expressions, and their corresponding formula. The semantics of a table row depends on the hit policy of the table. In a table T with hit policy U or A, n input columns I and an output column o, each row i is represented by the following definitional rule:

$$F_{io}(x_o) \leftarrow \bigwedge_{j \in I} F_{ij}(x_j). \tag{2.2}$$

The rows of a table T with hit policy F are represented similarly, but are also appended by a negation of the disjunction of the previous rows. This ensures that a row cannot fire if a row higher in the table is already applicable.

$$F_{io}(x_o) \leftarrow \bigwedge_{j \in I} F_{ij}(x_j) \wedge \neg \Big(\bigvee_{k \in 1..i-1} \big(\bigwedge_{j \in I} F_{kj}(x_j) \big) \Big). \tag{2.3}$$

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

Table 2.1: S-FEEL cell formula semantics

A DMN decision table is then represented by a finite sequence of such definitional rules, one for each row of the table. Note that such a definition captures both the *necessary* and the *sufficient* conditions for the concept it defines. By the former we mean that if the head is satisfied, at least one of the bodies must be satisfied as well, while the latter signifies that if one of the bodies is satisfied, the head will be satisfied as well.

For example, the first two tables of our running examples are translated to the following definitions in FO(·):

$$
\begin{aligned}
&\{\\
&BMI() = Weight()/(Lenght() \times Lenght())\\
&\}\\
&\{\\
&BMILevel() = Underweight \leftarrow BMI() < 18.5 \wedge Sex() = Female.\\
&BMILevel() = Underweight \leftarrow BMI() < 25 \wedge Sex() = Male.\\
&BMILevel() = Normal \leftarrow 18.5 \leq BMI() \leq 25 \wedge Sex() = Female.\\
&BMILevel() = Normal \leftarrow 25 < BMI() \leq 30 \wedge Sex() = Male.\\
&BMILevel() = Overweight \leftarrow 25 < BMI() \leq 30 \wedge Sex() = Female.\\
&BMILevel() = Obese \leftarrow 30 < BMI().\\
&\}
\end{aligned}
\tag{2.4}
$$

2.2.2 Execution Methods

Since the introduction of DMN by OMG, software companies such as Camunda [30], OpenRules [103] and Signavio [114] offer decision modelling software based on this standard. Besides assisting the user in modelling and verifying decisions, some of them also provide execution mechanisms for the models.

Execution of decision tables goes back to Decision Table Solvers [135]. Practically, all of these tools all support the same execution method: the *bottom-to-top* approach. This execution method, which is seen as the "standard" method, requires the user to input a value for every input variable present in the model, after which the execution engine decides the value of all other variables. In our running example, this correspondents with supplying a value for *Weight*, *Length*, *Waist* and *Sex* to derive the value of *Risk Level*. While this is considered the standard usage of a DMN model, some tools also support additional execution methods.

One such method is *reasoning on sub-decisions*: instead of evaluating every decision table in a model, it is sometimes preferable to evaluate only a specific subset of decisions. If we are only interested in the *BMI Level*, for example, we do not need to evaluate the *Risk Level* table. The advantage of reasoning on sub-decisions is that not all input variables are required, i.e., *Waist* is irrelevant as long as we do not need to know the *RiskLevel*. Examples of tools capable of this execution method are Camunda and OpenRules, both of which can evaluate a decision table in isolation. By reasoning on a single table at a time, they allow only evaluating the tables necessary for a sub-decision.

Another alternative execution method is the *"wildcard" mode*, such as the one provided by Camunda and Signavio, in which users can evaluate a decision model with partial input values. This is most interesting in cases where not all input variables are known, but a user still wants to see what top-level values remain possible. For example, if the value of *Sex* is unknown, a wildcard value can be used instead, in which case the engine returns a set of all possible output values.

2.3 DMN in Interactive Consultant

As a first way of leveraging the user-friendliness of DMN together with the benefits of the IDP-Z3 system, we integrated a DMN editor (dmn-js [24]) into the Interactive Consultant. In this way, this new tool (called DMN-IDP) functions as a full-fledged DMN tool, capable of supporting users in reasoning on the information in the model.

Fig. 2.4a shows a screenshot of the BMI running example (Fig. 2.2) modelled in DMN-IDP. Models can be imported and exported from DMN-IDP in the DMN Exchange format, which is part of the DMN specification [102, p217]. The translation of DMN to FO($\cdot$) is performed by the cDMN solver [125] (see also Chapter 3), based on the semantics outlined in Section 2.2.1. This FO($\cdot$) is

then passed on to the Interactive Consultant, which automatically generates an interface using the variables from the DMN model, as shown in Fig. 2.4b.

Like any other DMN engine, DMN-IDP can derive the value of the top-level decision based on the values of the input variables. But, unlike the others, DMN-IDP also supports reasoning on the knowledge for other purposes, thanks to IDP-Z3 and the KBP. Indeed, an FO(·) representation of a DMN model is merely a "bag of knowledge", without a predefined use in mind: we do not need to limit ourselves to bottom-to-top decisions.

As an example, consider a man that is 1.79m tall, weighs 100kg and has a waist size of 98. Based on the length and the weight, the system first derives that the man has a BMI of 31.2 and is therefore obese. This behaviour is already different from standard DMN engines, as these latter tools typically require all inputs to be known before making any decisions. DMN-IDP on the other hand is "always live", and will interactively derive consequences where possible.

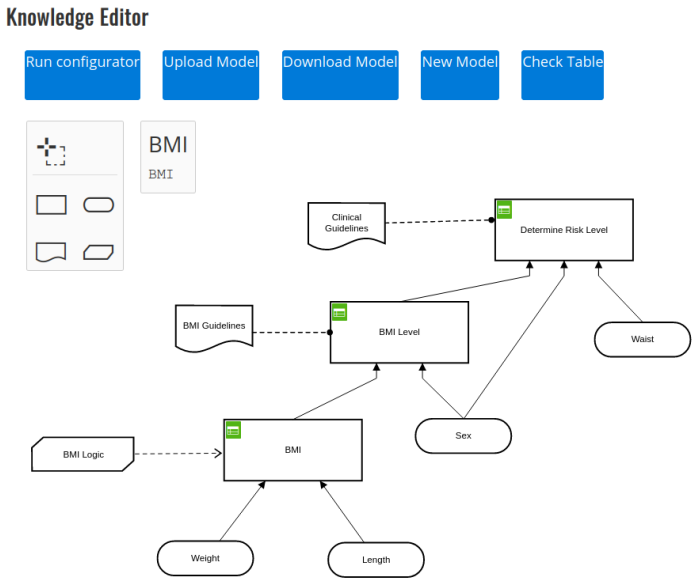
Once the man also enters his waist size, the tool derives that he has a high risk level. Naturally, the man wants a low risk level to be healthy, so he would now like the same tool to assist him in finding a target weight. Even though the knowledge necessary to find this information is present in the DMN model, state-of-the-art DMN engines cannot infer this information.

In contrast, DMN-IDP effectively allows the man to "reason backwards", by removing his weight and instead selecting a low risk level. He can now calculate a healthy target by maximizing the value for weight, resulting in a weight of 96.12kg at a BMI of 29.99, which according to our DMN model is considered healthy.

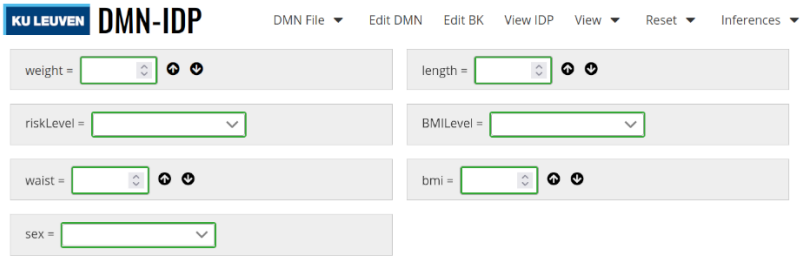
As highlighted earlier, DMN-IDP is also capable of reasoning on incomplete information. Together with IDP-Z3's *relevance* inference task, this ensures that users do not need to input more information than necessary. As an example, Fig. 2.4c shows the interface for a person that is 1.79m tall, and weighs 79kgs. Notice that, regardless of their waist or sex, the system has derived that their risk level is always Low, which saves them from inputting this information. In the context of some applications (e.g., chat bots), this can greatly reduce the number of queries needed, as highlighted later in this chapter.

A hosted version of DMN-IDP is freely available online[1]. Thanks to the DMN Exchange format, any DMN model made in a different modeller can also be imported in DMN-IDP, thereby "unlocking" more ways in which the model can be used. DMN-IDP is open source[2] under the LGPLv3 license.

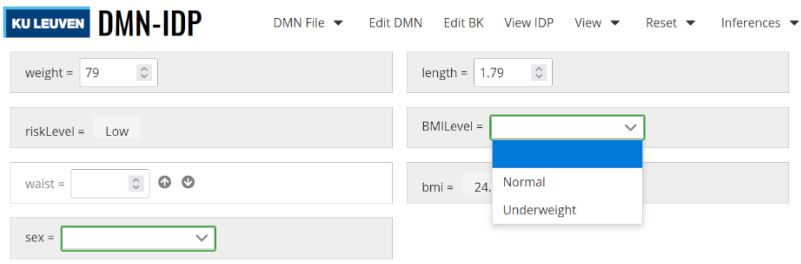
[1] `https://dmn-idp.onrender.com/`
[2] `https://gitlab.com/EAVISE/cdmn/DMN-IDP`



(a) The BMI example implemented in the DMN-IDP editor.



(b) The automatically generated interface for the BMI example.



(c) Example of reasoning on incomplete information in DMN-IDP.

Figure 2.4: Screenshots of the DMN-IDP tool.

2.4 IDP-Z3-based DMN reasoning API

To build actual applications on top of DMN models, DMN-IDP does not suffice as it cannot easily be integrated into larger software. To overcome this, our second approach to integrating DMN and IDP-Z3 is an easy-to-use Python API (named DMN-IDPy) that supports the same fundamental reasoning functionalities as DMN-IDP, and more. To illustrate its usefulness, we will first enumerate its capabilities in this section. For each capability, we briefly mention what it is, why it is important and we show a short code snippet to show it in action. Afterwards, we will show an example application based on the API, followed by a comparison between our API and that of other state-of-the-art decision engines.

2.4.1 Bottom-Up Decision Calculation

Like any other DMN tool, our API supports the "bottom-to-top" functionality. In the example shown in Fig. 2.2, this corresponds to setting the values for *Weight*, *Length*, *Sex* and *Waist* to then calculate the decisions in the following order: *BMI* → *BMILevel* → *RiskLevel*.

```
spec = DMN('bmi.dmn')
spec.set_value('weight', 74)
spec.set_value('length', 1.79)
spec.set_value('sex', 'Male')
spec.set_value('waist', 90)
```

→

```
>>> spec.model_expand(1)
Model 1
==========
bmi := 23.09540900720951.
riskLevel := Low.
BMILevel := Underweight.
sex := Male.
waist := 90.
weight := 74.
length := 1.79.
```

2.4.2 Reasoning with Incomplete Information

In contrast to most state-of-the-art execution engines, DMN-IDPy can reason with incomplete information. For example, to calculate the value of one or more sub-decisions without requiring the values of all input variables: if we are exclusively interested in the value of *BMILevel*, we should be able to perform this decision using only *Weight* and *Length* as inputs.

```
spec.set_value('Weight', 74)
spec.set_value('Length', 1.79)
spec.propagate()
```

→

```
>>> spec.value_of('BMI')
23.09540900720951
```

By supporting reasoning with incomplete information, every DMN model that consists of more than one table can directly and efficiently be used for multiple purposes by reasoning on sub-decision trees.

2.4.3 Relevance

One of the goals of our API is to allow *generic* tools to be built. As such, we want to avoid the need for hard-coding an assignment order to variables. DMN-IDPy instead allows querying which variables are relevant for making a certain decision, at runtime. For example, because *BMI* is defined by *Length* and *Weight*, these latter two variables should both be known in order to decide the value of *BMI*. By implementing this functionality in the API, tools can be built with a more generic nature.

Note that by "inputs" we do not only mean the inputs of a decision table, but rather all *upstream* variables needed for a decision to be made. For example, while the *BMI Level* table has two input variables, ons of those in turn has two input variables of its own. So, in reality, there are three dependencies for *BMI Level*, but at two different levels of the DRD. In the API, we show the number of extra *node hops* necessary to reach the variable to clearly denote this difference. This information is generated from the DMN file, without making use of the IDP system.

```
>>> spec.dependencies_of('BMI')
{'Weight': 0, 'Length': 0}

>>> spec.dependencies_of('BMILevel')
 {'BMI': 0, 'Weight': 1, 'Length': 1, 'Sex': 0}
```

As mentioned in Section 2.4.2, this can help optimize the required operations needed to decide a variable's value.

2.4.4 Multidirectional Reasoning

With the aim of getting as much use out of a DMN model as possible, being able to reason on decisions "in any direction" unlocks a lot of additional functionality. The knowledge to do so is already present in the decision tables, and our DMN-IDPy API is the first to offer it.

However, this means we need a way to directly assign values to decision variables (instead of only to inputs). For example, if the value for BMI is already known beforehand, we can directly assign that value to the decision variable. In this way, we derive the value of BMI Level without entering a weight and height.

```
spec.set_value('BMI', 31)
spec.set_value('Sex', 'Female')
spec.propagate()
```

→

```
>>> spec.value_of('BMILevel')
Obese
```

In the previous section, we discussed an obese man that is 1.79m tall, who wants to know what his target weight should be. This example of "backwards" reasoning can be implemented in DMN-IDPy as follows:

```
spec.set_value('BMI', '25')
spec.set_value('Length', 1.79)
spec.propagate()
```

→

```
>>> spec.value_of('Weight')
80.1025
```

Here, only a single value for *Weight* remains, because we set both *BMI* and *Length*. However, if we only set *BMI*, multiple values for *Weight* (and *Length*) are still possible, and no equality $Weight = x$ would be propagated. Indeed, instead of a single solution, we now have a *solution space*.

There are multiple ways to traverse this solution space in order to find a single solution. Assigning values to more variables will decrease the size of the space, possibly up until the point where there is only one solution left. If there are no variables left and there are still multiple solutions possible, we can generate solutions via IDP's model expansion inference (as demonstrated in the example in Section 2.4.1). Alternatively, we can search for the solution with the maximal/minimal value for a specific variable, as further explained in the Section 2.4.7.

2.4.5 Known variables

Because of the API's interactive approach, in which any variable can be assigned a value at any time, it is important to be able to keep track of which variables are *known*. A variable is considered known when it has been assigned a value, either by the user or by the reasoning engine via propagation. Consider for instance a case where a user has calculated their BMI level by entering their length and weight, as demonstrated in Section 2.4.2. If they want to calculate their risk level afterwards, they should only have to enter their sex and waist, as that is the only information that is still missing for this decision.

```
spec.set_value('Length', 1.79)
spec.set_value('Weight', 79)
spec.propagate()
```

→

```
>>> spec.is_certain('BMI')
True
>>> spec.is_certain('Sex')
False
```

2.4.6 Variable type and values

Every variable in a DMN model has a data type, such as *Int*, *Float*, *String* or other. Intuitively, these denote the type of data that a variable represents. To avoid errors such as assigning a numerical value to a variable of data type *String*, the API allows querying a variable's type via *type_of*.

String is a special case of data type: where *Int*, *Float*, etc are considered to have infinite ranges, *String* is limited to a predefined list of possible values. Indeed, it makes sense that only those values that appear in a table can be assigned to a variable. For instance, the value of BMI level is limited to {*Underweight, Normal, Overweight, Obese*}. To prevent assigning impossible values to a string variable, the API can give a list of all possible values by either returning the variable's predefined list, or, if no list was predefined, by returning a list of all string values which appear at least once for that variable.

```
>>> spec.type_of('BMI')
Real
>>> spec.type_of('Sex')
String
>>> spec.possible_values_of('BMILevel')
['Obese', 'Underweight', 'Overweight', 'Normal']
```

2.4.7 Optimization

Optimization allows us to find the solution with the highest or the lowest value for any given numerical variable. Consider a patient that has just entered their weight and length to find out that they have an *Overweight* BMI Level. A logical next question would be: "What should my target weight be if I want to have a normal BMI Level?". To answer this, they can enter their length and set the value of *BMI Level* to *Normal*. If they then maximize the value of *Weight*, the system will calculate the maximum weight that still results in a normal BMI Level.

```
>>> spec.set_value('Length', 1.79)
>>> spec.set_value('BMILevel', 'Normal')
>>> spec.maximize('Weight')
Length := 1.79.
Weight := 95.80259.
BMILevel := Normal.
```

2.5 DMN-IDPy Application Example

We will now sketch a naive implementation for a chat bot based on DMN-IDPy, implemented in less than 25 lines of Python. Its main goal is to allow users to calculate any of the intermediary or top-level variables of a DMN model. In order to achieve this, the bot goes through a few steps. First, it fetches the list of variables and asks the user which variable should be calculated.

```
spec = DMN(sys.argv[1], auto_propagate=True)
vars = spec.get_outputs() + spec.get_intermediary()
req_var = input('Which variable to calculate? {}\n>'.format(variables))
```

Next, the program finds out which input variables should be known in order to make this calculation. Input variables without any effect on the value of the requested variable are not included.

```
deps = spec.dependencies_of(req_var)
missing_vars = [x for x in deps if x in spec.get_inputs()]
print("\nThe following variables are still unknown:")
print(missing_vars)
```

Finally, it loops over every unknown variable and queries the user for its value. A different question is asked based on the data type of the variable, to prevent type clashes. If the program requests the value of a *String*-based variable, it

should also supply the user the list of possible values. Similarly for numerical variables, the user should be notified if the variable is an integer or a float.

```
for var in missing_vars:
    # Ask for the variable's value. Based on var type, ask different question.
    var_type = spec.type_of(var)
    if var_type in ['Real', 'Int']:
        msg = "Value for {} ({}) unknown.\n>".format(var, var_type)
    else:
        pos_vals = spec.possible_values_of(var)
        msg = "Value for {} unknown.\n"\
              "Possible values: [{}]\n>".format(var, pos_vals)
    value = input(msg)
    spec.set_value(var, value)

    if spec.is_known(req_var):
        break

req_var_val = spec.value_of(req_var)
print('Calculated value for {}:\n{}'.format(req_var, req_var_val))
```

Note that at the end of every loop cycle, the program checks whether the variable is known yet. While this might not make much sense at first, because the program specifically fetched the list of necessary inputs for the decision, there are cases where not all inputs might be necessary. Consider for example the decision table for *RiskLevel*. Here, if the values for *Weight* and *Length* are queried first and they lead to a BMI Level that is neither overweight nor obese, then the values of *Sex* and *Waist* will have no impact on this decision.

```
>>> python bot.py bmi.dmn
Which variable to calculate? ['RiskLevel', 'BMILevel', 'BMI']
> Risk Level
The following variables are still unknown:
['Weight', 'Length', 'Sex', 'Waist']
Value for Weight (Real) unknown.
> 79
Value for Length (Real) unknown.
> 1.79
Calculated value for Risk Level:
Low
```

The program example shown above works for any DMN model that is supported by the API. For example, if we want a chat bot for personal taxes, we only need to switch out the underlying DMN model, with no code modifications required. Research on such DMN-based chat bots has recently been emerging

more and more in the literature [52, 53]. In this context, DMN-IDPy has already successfully been integrated in a full-fledged chat bot by Goossens et al. [63].

2.6 Comparison to other tools

To the best of our knowledge, there is no other approach that offers such a flexible yet powerful use of DMN models. While there exist tools that support more than exclusively the bottom-to-top calculation, none are capable of performing all features discussed in this work. Table 2.2 shows a comparison of the functionalities of DMN-IDPy, the OpenRules API and the Camunda API.

As expected, all compared APIs support the bottom-to-top execution. Additionally, they all also support reasoning on incomplete information, but only up to a varying degree. Both OpenRules and Camunda are capable of using incomplete information by reasoning on sub-decisions, as they allow the evaluation of a single decision table isolated from the rest of the DMN model. Thus, it is possible to e.g. use the Risk Level model to only calculate a patient's BMI, as discussed in the example in Section 2.4.2. However, as the API's only allow reasoning on either the entire model or a single specific table, attempting to reason on a sub-decision consisting of multiple tables (e.g. *BMI* followed by *BMILevel*) requires quite a bit of extra overhead: for each table, we would need to (a) manually supply the inputs, (b) evaluate, and (c) extract the outputs to use as inputs for the next table. In our API, no such workarounds are needed, as it suffices to enter all input values followed by calling the propagation inference. As such, the process of using sub-decisions with DMN-IDPy is much more streamlined.

The wildcard mode, as featured in Camunda, is possible in DMN-IDPy by leveraging its ability to reason on incomplete information. After entering a partial set of input values, we can generate all remaining solutions using the model expansion inference.

Neither OpenRules nor Camunda support multidirectional reasoning or optimization.

The main downside of our approach is the efficiency of the reasoning engine itself. Where other engines have specific optimized algorithms to perform the bottom-to-top calculation, we use a general purpose reasoning engine. As such, our calculation times will often be a magnitude higher compared to the other state-of-the-art engines. However, we feel that we make up for it with the increased flexibility that the API offers.

	DMN-IDPy	OpenRules	Camunda
Bottom-to-top	X	X	X
Incomplete Information	X	o	o
Wildcard mode	X		X
Multidirectional Reasoning	X		
Optimization	X		

Table 2.2: Comparison between functionalities of DMN-IDPy, and state-of-the-art DMN execution engines. (X = full support, o = partial support)

2.7 Conclusion

This chapter introduced the link between DMN as a user-friendly formalism for modelling knowledge, and IDP-Z3, as a powerful inference engine to put the knowledge to use. We have shown that DMN models can be used in many more scenarios other than bottom-to-top calculation, allowing applications with complex behaviour to be built on top of DMN. For this to be possible however, DMN needs to be supported by a flexible reasoning tool. To this end, we have presented two tools: an integration of DMN into the Interactive Consultant, and a multi-functional Python API for reasoning on DMN.

DMN-IDP allows users to *plug-and-play* their DMN models in the Interactive Consultant, thereby unlocking new functionalities "for free". The tool is capable of assisting users to traverse the knowledge in their DMN model, in more ways than other state-of-the-art tools.

DMN-IDPy enables the IDP reasoning system as an embeddable execution engine for DMN. In this way, it provides the required building blocks to construct intelligent tools. The main additions of the API are:

- Support for reasoning in any direction (e.g. going in the other direction of the DRD);
- Support for reasoning on incomplete data (allowing for sub-decision calculations);
- Optimization of variable values.

In order to showcase DMN-IDPy in action, we created a naive implementation of a chat bot in under 25 lines of Python code. The implementation is generic in the sense that it can be used with any DMN model, without having to change a line of code.

Chapter 3

Constraint Decision Model and Notation

In this chapter, we introduce our own extension of DMN, called cDMN (constraint Decision Model and Notation). Its aim is to extend DMN's expressiveness, while maintaining its intuitive tabular format. In this way, modelling complex problems could be more accessible to non-AI experts. We elaborate on cDMN's syntax and semantics, and compare it to standard DMN implementations of decision management challenges.

This chapter is based on work presented at the Joint Conference on Rules and Reasoning (RuleML+RR) (RuleML+RR), September 2020 [5]; DecisionCAMP, September 2020 [124]; and on work published in Theory and Practice of Logic Programming, 2021 [125]. The research was performed in collaboration with Bram Aerts.

3.1 Introduction

While DMN is very effective in modelling 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 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 chapter, 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 modelling, quantification, and the use of concepts such as types and functions. We test the expressiveness and capabilities of cDMN on decision modelling challenges set forth by the Decision Management Community.

In [46], the authors presented a preliminary framework for constraint modelling in DMN. In this chapter, we extend this by adding quantification, types, functions, relations, data tables, optimization and by evaluating the resulting cDMN formalism on the DMN challenges.

3.2 Challenges Overview

Of all the challenges on the DMCommunity website, we selected those that did not have a straightforward DMN-like solution. In total, the website has set forth 64 challenges[2], of which 24 meet our criterion. A list of these challenges is shown in Table 3.1. We categorize these challenges according to four different properties. Table 3.2 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

[1] https://dmcommunity.org/
[2] last checked 2023/07/12

Table 3.1: List of DMCommunity 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 3.2: 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

instance, the constraint in *Map Coloring* states that two bordering countries can not 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.

The description of each challenge can be found on the DMCommunity website[3], together with their submissions. We also maintain a mirror repository[4] containing the specific challenges and submissions used in this work.

[3] `https://dmcommunity.org/challenge/`

[4] `https://gitlab.com/EAVISE/cdmn/DMChallenges`

3.3 Related Work

It has been recognized that even though DMN has many advantages, it is somewhat limited in expressiveness [28, 46]. 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 [103]. 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 [102, p. 13]. Third, because of the lack of quantification in OpenRules, solutions are generally not independent of domain size, which reduces readability.

Another DMN engine which supports a way to increase the expressiveness of DMN is Corticon [105]. It implements a basic form of constraint solving by allowing the modeller 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 3.6.

Calvanese et al. [28] 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 separate Description Logic formalism, which may be too complex for a domain expert to use. While this approach makes sense if, e.g., 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 have not submitted any solutions of DMN Challenges, which leaves us unable to compare its expressiveness in practice.

3.4 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.

3.4.1 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 3.1.

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 3.4.3). An example is the type `Doctor`, which contains the names of doctors. By convention, type symbols start with a capital letter. Besides user-declared types, there are two built-in types: *Int* and *Real*, which respectively represent the (infinite) domains of integer and real numbers.

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)

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

Function	
Name	**Type**
present doctor on Day at Time	Doctor
nb shifts of Doctor on Day	Int
nb nights of Doctor	Int
day after Day	Day

Constant	
Name	**Type**
Head	Doctor

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

Boolean
Name
Complete

Figure 3.1: An example cDMN glossary for the *Doctor Planning* problem.

and the remaining text is the name of the function. For example, `nb nights of Doctor` has one argument of type `Doctor`. Intuitively, this function denotes how many nights a doctor works per week: it maps each element of type `Doctor` to an integer number. Functions with arity $n > 1$ can be defined by using n arguments in the name, 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.

3.4.2 Decision Tables and Constraint Tables

As stated earlier in Section 2.2, 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 3.2, 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.

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).

Doctor works max 1 shift per day			
E*	Doctor	Day	nb shift of Doctor on Day
1	—	—	≤ 1

Figure 3.2: Constraint table to express that a doctor works a maximum of one shift per day.

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 3.2 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 3.3 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 subsection 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.

Bordering countries cannot share colors				
E*	Country called c1	Country called c2	c1 borders c2	Color of c1
1	—	—	Yes	Not(Color of c2)

Figure 3.3: Example of a constraint table with quantification in cDMN, defining that bordering countries can not share colors.

3.4.3 Data Tables

Typically, problems can be split up into two parts: (1) the general logic of the problem, and (2) the specific problem instance that needs to be solved. Take for example the map coloring problem: the general logic consists of the rule that two bordering countries cannot share a color, whereas the instance of the problem is the specific map (e.g., Western Europe) to color. cDMN extends the DMN standard to include *data tables*, which are used to represent the problem instances, separating them from the general logic. The format of a data table closely resembles that of a decision table, with a couple of exceptions. Instead of a hit policy, a data table has "data table" in its name[5]. 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, i.e., 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 3.4.

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]Starting with the soon-to-be-released cDMN v3.0.0 however, data tables use the `D` hit policy instead. We have found that this makes more sense from a user perspective.

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, Luxembourg, Belgium, Netherlands	Yes

Figure 3.4: Data table describing countries and their neighbours

Goal	Goal	Goal
Get 3 models	Get all models	Maximize Score

Figure 3.5: Goal table examples

3.4.4 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 3.5.

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.

3.4.5 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($\cdot$).

The 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: i.e., 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 $S' \models T$, i.e., 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 in Section 2.2.1. 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 "$x \leq 50$". The semantics of a table row depends on the hit policy of the table. In a table T with hit policy U or A, n input columns I and an output column o, each row i is represented by the following definitional rule:

$$F_{io}(y_o) \leftarrow \bigwedge_{j \in I} F_{ij}(x_j). \tag{3.1}$$

The rows of a table T with hit policy F are represented similarly, but are also appended by a negation of the disjunction of the previous rows. This ensures that a row cannot fire if a row higher in the table is already applicable.

$$F_{io}(x_o) \leftarrow \bigwedge_{j \in I} F_{ij}(x_j) \wedge \neg\Big(\bigvee_{k \in 1..i-1} \big(\bigwedge_{j \in I} F_{kj}(x_j) \big) \Big). \tag{3.2}$$

A DMN decision table is then represented by a finite sequence of such definitional rules, one for each row of the table. As an exception, tables with an atom-denoting header as an output are treated slightly differently: rows in which the output value is "False" are not included in the sequence. Indeed, such rows are already implicitly represented, as definitions represent both the sufficient and necessary conditions. For example, in standard DMN (which does not contain function or relation expressions), the table in Figure 3.6 corresponds to:

Define Adults		
U	Age of Person	Person is Adult
1	≥ 18	Yes
2	< 18	No

Figure 3.6: Decision table to define whether a person is an adult.

$$\{Person\_is\_Adult \leftarrow Age \geq 18.\} \tag{3.3}$$

Data tables are simply a specific case of decision tables.

The semantics of simple constraint tables (without quantification and functions) is a conjunction of implications, as was described in [46]. 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 [27].

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, e.g. $[e_1, e_2)$, 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 FOL variables. There are two kinds of variable expressions, as we described in Section 3.4.2. 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 $\nu_H(e) = v[\mathit{Type}]$.

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 $\nu(e) = \nu_H(e)$.
- 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 3.4.2).

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 sub-expressions.
- The interpretation of a variable expression is the corresponding variable, i.e., for a variable expression v, we define $t_\nu(v) = \nu(v)$.
- If c is of the form $\#\mathit{Type}$, then $t_\nu(c) = \#\{x[\mathit{Type}] : \mathit{true}\}$, 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 $t_\nu(F) = \mathit{Function}(t_\nu(arg_1), \ldots, t_\nu(arg_n))$.
- 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 $t_\nu(X) = \mathit{Relation}(t_\nu(arg_1), \ldots, t_\nu(arg_n))$.

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:

$$\bigwedge_{l \in V} \nu(H_l) : \bigwedge_{i \in R} \Big(\bigwedge_{j \in I} t_\nu\big(F_{ij}(t_\nu(H_j))\big) \Rightarrow \bigwedge_{k \in O} F_{ik}\big(t_\nu(H_k)\big) \Big) \tag{3.4}$$

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.

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

$$\bigwedge_{w \in W} \nu(H_w) : t_\nu(H_k) = \sum_{i \in R} \mathit{sum}\big\{\bar{x} : \bigwedge_{j \in I} F_{ik}\big(t_\nu(H_j)\big) : F_{ik}((\bar{x}))\big\} \tag{3.5}$$

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 $\mathit{sum}\{\bar{x} : \varphi(\bar{x}) : F(\bar{x})\}$ 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

Number of invited guests				
C+	Person	Item	Item in basket of Person	charge for Person
1	—	—	Yes	price of Item

Figure 3.7: Decision table that determines the charge of a person, based on the contents of their shopping basket.

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 3.7: $W = \{Person\}$, $\nu(H_w) = p[Person]$, $t_\nu(H_k) = charge(p)$, $\bar{x} = y[Item]$, $t_\nu(F_{1k}(\bar{x})) = price(y)$ and $\bigwedge_{j \in I} F_{ij}(t_\nu(H_j)) = in\_basket(y, p)$. This results in the logical sentence:

$$\forall\, p[Person] : charge(p) = sum\{Item : in\_basket(Item, p) : price(Item)\}$$

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, i.e.,

$$\underset{w \in W}{\forall} \nu(H_w) : t_\nu(H_k) = \left|\left\{x \mid \underset{u \in U}{\exists} \nu(H_u) : \bigvee_{i \in R} \left(t_\nu(F_{ik}(x)) \wedge \bigwedge_{j \in I} t_\nu(F_{ij}(H_j))\right)\right\}\right| \tag{3.6}$$

Here, U and W are defined analogously as in Equation 3.5.

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.

For instance, in the decision table in Figure 3.8: $W = \emptyset$, $t_\nu(H_k) = nb\_invitations$, $U = \{Person\}$, $\nu(H_u) = p[Person]$. For the first row,

Number of guests				
C#	Person	Person is friend	Person is family	nb invitations
1	—	Yes	—	Person
2	—	—	Yes	Person
3	—	—	Yes	spouse of Person

Figure 3.8: Decision table that counts the number of invited guests.

$\big(F_{1k}(t_\nu(x)) \wedge \bigwedge_{j\in I} F_{1j}(t_\nu(H_j))$ is equivalent to $x = p \wedge is\_friend(p)$. 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 $x = spouse(p) \wedge is\_family(p)$. Consequently, the table in Figure 3.8 is logically equivalent to:

$$\begin{aligned} nb\_invitations = \big|\{x \mid \exists p[Person] : \\ x = p \wedge is\_friend(p) \vee \\ x = p \wedge is\_family(p) \vee \\ x = spouse(p) \wedge is\_family(p) \\ \}\big| \end{aligned} \tag{3.7}$$

In the table of Figure 3.8, the output header is a constant (*nb_invitations*), 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 number of possible worlds (one, all, or a specific number of them), the possible world that minimizes/maximizes a given term, or the consequences of the model (propagation).

3.5 Implementation

Because cDMN is more expressive than DMN, it cannot be handled by existing solvers. We have therefore implemented a new solver[6], which we describe in this section. It consists of two parts: IDP-Z3 as the reasoning engine, and a converter from cDMN to 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` 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, e.g. "$Function(arg_1, arg_2)$". 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 3.4.5.

An overview summarizing all the relations between cDMN tables, Python objects and IDP blocks can be found in Figure 3.9. More detailed information about this conversion can be found in the cDMN documentation[7], 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 behaviour.

Another feature of the solver is the ability to link to the IDP-based interface *Interactive Consultant* [32], 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

[6] https://gitlab.com/EAVISE/cdmn/cdmn-solver
[7] www.cdmn.be

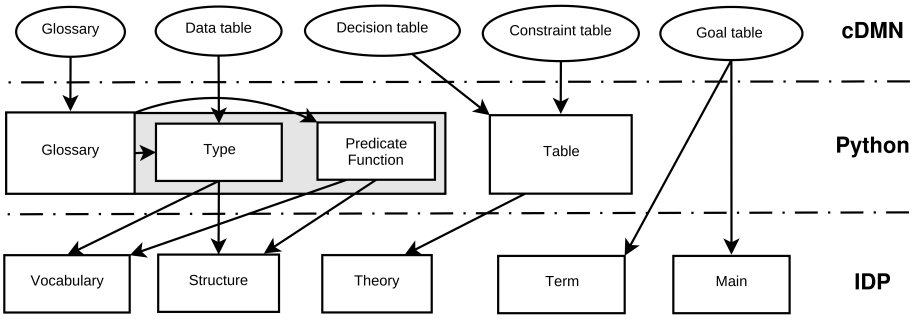


Figure 3.9: An overview of the inner workings of the cDMN solver

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

3.6 Results and discussion

In this section we first look at three of the DMCommunity challenges, each showcasing a feature of cDMN. For each challenge, we qualitatively compare the DMN implementations from the DMCommunity website with our own implementation in cDMN. Afterwards, we compare all challenges on size and quality. We end our discussion with a section on the integration of cDMN in business processes.

3.6.1 Constraint tables

Constraint tables allow cDMN to model constraint satisfaction problems in a straightforward way. For example, in *Map Colouring*, a map of six European countries must be coloured in such a way that no neighbouring countries share the same color. For this challenge, a pure DMN implementation was submitted, of which Figure 3.10 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 3.3. Together with the glossary and a data table (Figure 3.4), this forms a complete yet simple cDMN implementation.

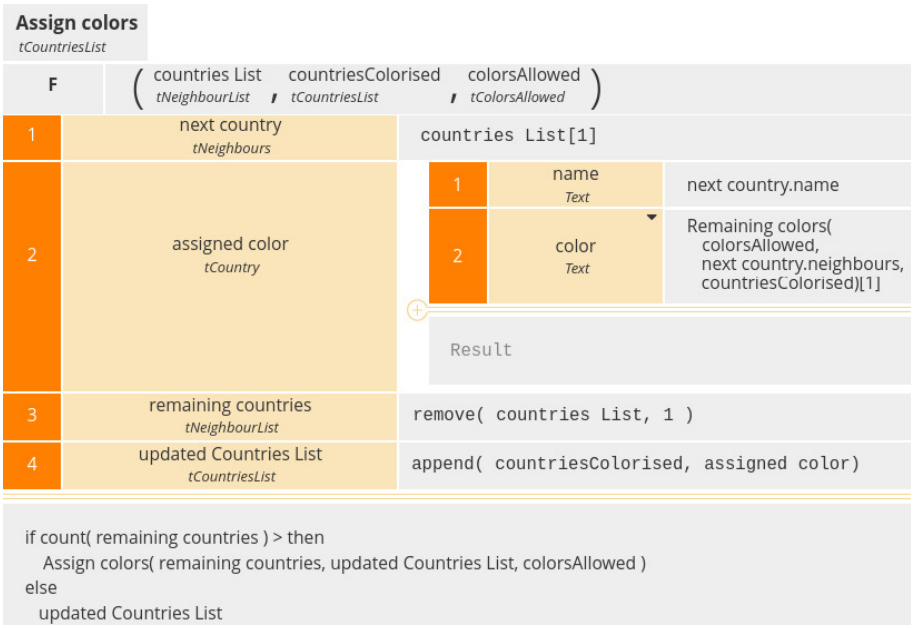


Figure 3.10: An extract of the map coloring solution in standard DMN with FEEL.

3.6.2 Quantification

Among others, 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 3.11a). 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 3.11b). 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

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
1	...	...	...
2	—	Rock	Apple
3	...	...	...

(b) cDMN

Figure 3.11: An extract of *Monkey Business* implementation in (a) OpenRules and (b) cDMN, specifying "The monkey who sits on the rock is eating the apple".

in Figure 3.12a. 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 3.12b requires only a single row to express the same.

We conclude that of all the solutions that were submitted to the DMCommunity, 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.

3.6.3 Optimization

In the *Balanced Assignment* challenge, 210 employees need to be divided into 12 groups, so that every group is as diverse as possible. The department, location, gender and title of each employee is known. This is quite a complex problem to handle in DMN. As such, of the four submitted solutions, only one was DMN-like: an OpenRules implementation, using external CP/LP solvers. The logic for these external solvers is written in Java. Although the code is fairly compact, it cannot be written without prior programming knowledge. The

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

Figure 3.12: 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	Dept of p1	Loc of p1	Gender of p1	Title of p1	Group of p1	Score
1	—	—	Dept of p2	—	—	—	not(Group of p2)	1
2	—	—	—	Loc of p2	—	—	not(Group of p2)	1
3	—	—	—	—	Gender of p1	—	not(Group of p2)	1
4	—	—	—	—	—	Title of p1	not(Group of p2)	1

Figure 3.13: The decision tables and constraint table for *Balanced Assignment*.

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 3.13, 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.

3.6.4 Overview of all challenges

Of the 24 challenges we considered, cDMN is capable of successfully modelling 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 modelling 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 cDMN data tables), because the ways in which different solvers handle this are too diverse to allow meaningful comparison. Table 3.3 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 3.14a and 3.14c. 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. The OpenRules implementation (Figure 3.14b) 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 3.15a and 3.15b. 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 3.2, 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.

Table 3.3: 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			

Aggregates needed Figure 3.14c 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 modelled by constraint tables, such as the constraints in Figure 3.15b, 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 3.2. This table states that no doctor works more than one shift per day.

	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

Nutrition					
C+	Item	Total Sodium	Total Fat	Total Calories	Total Cost
1	—	Nb of Item * Sodium in Item	Nb of Item * Fat in Item	Nb of Item * Calories of Item	Nb of Item * Cost of Item

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

(c) cDMN

Figure 3.14: Calculating the food properties of a burger in Corticon, OpenRules and cDMN.

Table 3.4: 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

Optimization Because cDMN directly supports optimization, problems containing this property are easily modelled. Furthermore, by the addition of more complex data types, optimization terms can be defined in a more flexible manner. An example can be found in *Balanced Assignment* in Figure 3.13. A summary of each problem property and its cDMN answer can be found in Table 3.4.

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		
E*	Killer hates Agatha	Killer richer than Agateha
1	Yes	No

(b) cDMN

Figure 3.15: Implementation of "A killer always hates and is no richer than their victim" in OpenRules and cDMN.

3.6.5 Process integration

DMN models are often integrated into a larger business process model [70, 14]. 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 (BPMN) 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 [19], 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 3.16 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 a decision model, as indicated by the table icon. 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

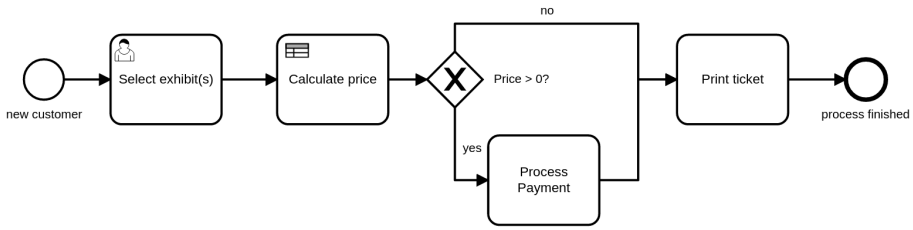


Figure 3.16: Example of a BPMN model with DMN

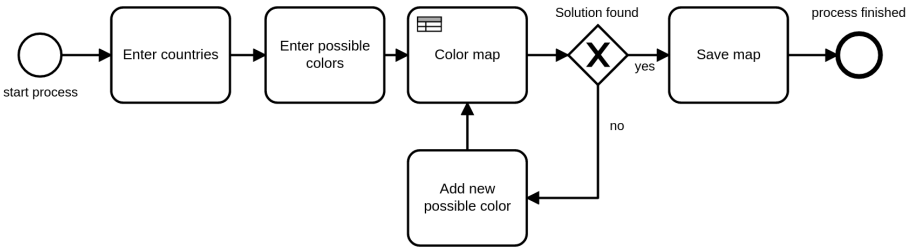


Figure 3.17: Example of a BPMN model with cDMN

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 3.17 describes the process of colouring a map of countries, based on a list of countries and a list of possible colours. Here, we have added a gateway that verifies if a suitable solution was found. If none was found (because too few colours were supplied), more colours 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 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.

3.7 Conclusion

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

Constraint modelling 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.

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.

Chapter 4

Probabilistic Decision Model and Notation

Probabilistic modelling systems such as ProbLog combine logical reasoning with probabilities. However, ProbLog rules are often difficult to interpret for domain experts with no familiarity with Probabilistic Logic Programming (PLP). The goal of this chapter is to investigate a DMN extension for PLP, to make probabilistic modelling more accessible.

This chapter is based on work presented at the Tenth International Workshop on Statistical Relational AI, October 2021 [133]; and DecisionCAMP 2023, September 2023 [123]. The research was performed in collaboration with Victor Verreet and Luc De Raedt.

4.1 Introduction

While DMN's deterministic nature is one of the main advantages of the notation, there exist applications for which it is necessary to be able to model uncertainty. Indeed, in real life, uncertainties can be found in many problem domains. While it is technically possible to "mimic" probabilities in DMN in an ad-hoc way, the user-friendliness of the resulting models would be significantly reduced. As such, a more structural approach based on DMN is preferred; after all, DMN already forms an excellent foundation to build on when designing a user-friendly notation for probabilistic logic.

In the state-of-the-art, there exist many concrete tools for probabilistic reasoning. One such tool is ProbLog [43], which combines logical reasoning with probabilities. It supports many inference tasks, such as marginal and conditional probability calculations, allowing it to be used for problems such as Bayesian reasoning and inference in social networks [66]. However, ProbLog rules can be difficult to interpret for domain experts with no familiarity with Probabilistic Logic Programming (PLP).

In this chapter, we present a preliminary version of Probabilistic Decision Model and Notation (pDMN): a DMN-like notation for probabilistic logic that aims to combine DMN's intuitive notation with ProbLog's powerful probabilistic reasoning capabilities. Our goal is to close the gap between ProbLog experts and domain experts, by lowering the threshold to understand and interpret probabilistic models, and to possibly allow domain experts to create the models themselves.

The contributions of this chapter are as follows:

1. the pDMN notation for probabilistic programming, which aims to be user-friendly;
2. the translation principles of pDMN into ProbLog;
3. an implementation of a ProbLog-based solver for pDMN.

This chapter is structured as follows. First, we go over the ProbLog concepts necessary for this work in Section 4.2. We then introduce our pDMN syntax in Section 4.3, and elaborate on how it differs from standard DMN. We also present the translation principles of pDMN into ProbLog in Section 4.4, and very briefly go over our implementation of a pDMN solver in Section 4.5. Afterwards, Section 4.6 shows a full pDMN implementation of a well-known example, to show the notation in action. Finally, we conclude our preliminary work in Section 4.7, and lay out the future work ahead.

4.2 Probabilistic Logic Programming

ProbLog [43] is a probabilistic extension of Prolog. A ProbLog program consists of a set of probabilistic facts and a set of Prolog rules. Probabilistic facts are of the form $P_f :: f$, with $P_f \in [0, 1]$ a probability and f an atom. The atom f is true or false with probability P_f and $1 - P_f$ respectively. Rules are written as $h \ :- \ b_1, b_2, \ldots, b_n$ where the atom h is called the head and b_i are the body atoms. The head of a rule may never occur in a probabilistic fact. Whenever all the atoms in the body of a rule are true, the head atom is true as well. A rule can also be annotated with a probability, but this is syntactic sugar for adding a unique atom to the body which is true with the annotated probability. Symbolically, the rule

$$P_r :: h \ :- \ b_1, b_2, \ldots, b_n \tag{4.1}$$

is translated into

$$h \ :- \ b_1, b_2, \ldots, b_n, f_r \quad \text{and} \quad P_r :: f_r \tag{4.2}$$

with f_r a newly created atom. ProbLog also allows annotated disjunctions (ADs), written as

$$P_1 :: f_1; P_2 :: f_2; \ldots; P_n :: f_n \tag{4.3}$$

with $\sum_i P_i \leq 1$. An AD denotes a probabilistic choice where every atom f_i is selected to be true with probability P_i, but at most one atom in the AD can be selected. If $\sum_i P_i < 1$ it is possible that none are true.

An interpretation is a truth value assignment to every atom occurring in the program. A model of the program is an interpretation that satisfies every rule and follows the closed world assumption. The closed world assumption states that an atom can only be true in a model whenever it can be derived through at least one rule. The probability of any model M of the program is the product of the probabilities of the facts in the model. The probability of an atom q is the sum of the probabilities of the models in which that atom is true. Hence,

$$P(q) = \sum_{M \models q} \prod_{f \in M} P(f) \tag{4.4}$$

where the sum runs over all models M in which q is true and the product runs over all the probabilistic facts f in the model M. The probability $P(f)$ is the user given value P_f if f is true in the model M, and $1 - P_f$ otherwise. An example of a ProbLog program is given in Example 1.

Example 1. Consider the program

$$\begin{array}{ll} 0.8 :: a. & c \ :- \ a. \\ 0.3 :: b(1); 0.5 :: b(2); 0.2 :: b(3). & c \ :- \ b(1). \end{array} \tag{4.5}$$

where we are interested in the probability of c. This program has 6 models, $\{a, b(1), c\}$, $\{a, b(2), c\}$, $\{a, b(3), c\}$, $\{\text{not}(a), b(1), c\}$, $\{\text{not}(a), b(2), \text{not}(c)\}$ and $\{\text{not}(a), b(3), \text{not}(c)\}$, where c is only true in the first 4 models. Therefore, the probability of c is

$$P(c) = 0.8 \cdot 0.3 + 0.8 \cdot 0.5 + 0.8 \cdot 0.2 + 0.2 \cdot 0.3 = 0.86 \tag{4.6}$$

In the past ProbLog has been applied to a variety of problems, such as object tracking in robotics [104], modelling protein interactions [64], predicting missing edges in biological networks [43] and spreadsheet completion [41]. Furthermore, ProbLog has been extended with neural networks [88], decision-theoretic reasoning [120] and continuous distributions [50], allowing for a broad range of use cases.

4.3 pDMN: Syntax

We now elaborate on the syntax of pDMN, our DMN extension for probabilistic logic programming. In pDMN, there are three types of tables: glossary tables, decision tables, and the query table.

4.3.1 Glossary

Variables in pDMN, in contrast to standard DMN, are typed n-ary functions and predicates. In order to correctly identify these variables and their arguments, pDMN introduces three glossary tables in which these should be declared: the *Type* table, the *Predicate* table and the *Function* table. These glossary tables contain the required meta-information to correctly interpret the pDMN model. This is analogous to the approach used in cDMN.

The *Type* table declares the *types* used in a pDMN model, together with their *domain of elements*. For example, the *Type* table in Fig. 4.1 declares a type *Person*, which consists of two elements, *ann* and *bob*, and a type *Vaccine*, which consists of the elements *a*, *b* and *n(one)*.

The *Predicate* table declares n-ary predicates. There is no fixed naming syntax for predicates; the arguments of a predicate are those types that appear in its description, and the remaining string is considered the predicate's name. For example, in the glossary of Fig. 4.1, *Person is infected* represents a unary predicate *is_infected*, which denotes for every *Person* (i.e., *ann* and *bob*) whether they are infected. Similarly, *Person contacted Person* is a binary predicate *contacted* that denotes contact between people.

Type	
Name	**Elements**
Person	ann, bob
Vaccine	a, b, n

Function	
Name	**Type**
vaccine of Person	Vaccine

Predicate
Name
Person is infected
Person contacted Person

Figure 4.1: Example of a pDMN glossary

The *Function* table declares n-ary functions. Analogously to predicates, the function's name contains its arguments. In contrast to predicates, however, functions map their arguments to the type listed in the *Type* column of the glossary table, instead of to a boolean. For example, *vaccine of Person* denotes the *Vaccine* for each *Person*, i.e., it maps every person (*ann* and *bob*) to a vaccine (*a, b, n*).

4.3.2 Decision Tables

pDMN extends standard DMN decision tables with three new concepts: probabilities, the new *Ch(oice)* hit policy, and quantification. We will briefly touch on each concept, and show an example. Firstly, pDMN allows probabilities in the cells of an output column. For example, the *h1* and *h2* tables shown in Fig. 4.2a respectively define a probability of 0.5 and 0.6 to flip a coin on its head. Note that we use *Yes* and *No* to represent *true* and *false* for predicates. In a table containing probabilities, the output values (such as *Yes*) are not listed in the rules directly, but rather in a separate row above the rules, which contains only output values. If the conditions of a rule are met, the probability of the output variable taking on a specific value is equal to the value that is listed below this output value in that particular row.

The second new concept is the *Ch(oice)* hit policy, which denotes that the output values for the output variable are mutually exclusive (i.e., only one can be assigned to the variable). This is demonstrated in the table in Fig. 4.2b, which states in its first row that an ordinary die has an equal 1/6 chance for any die value, and in its second row that a biased die has a higher chance of resulting in six. However, because of the *Choice* hit policy, the die can never e.g. be assigned both "one" and "two" at the same time. If, for instance, the

h1	
U	heads1
	Yes
1	0.5

h2	
U	heads2
	Yes
1	0.6

heads				
U	heads1	heads2	twoHeads	someHeads
1	Yes	Yes	Yes	Yes
2	Yes	No	No	Yes
3	No	Yes	No	Yes
4	No	No	No	No

(a) Example pDMN implementation describing two coinflips.

Throwing Dice							
Ch	biased	die value					
		one	two	three	four	five	six
1	No	1/6	1/6	1/6	1/6	1/6	1/6
2	Yes	0.1	0.1	0.1	0.1	0.1	0.5

(b) Example of a pDMN table with the "Choice" hit policy.

Vaccine			
Ch	vaccine of X		
	a	b	n
1	0.36	0.63	0.01

Infection				
U	X contacted Y	Y is infected	vaccine of X	X is infected
				Yes
1	Yes	Yes	n	0.8
2	Yes	Yes	a	0.1
3	Yes	Yes	b	0.2

(c) Snippet of a pDMN model implementing infections with vaccination.

Figure 4.2: Snippets of various pDMN examples.

Query
twoHeads
someHeads

(a)

Query
vaccine of bob
X is infected

(b)

Query
die value = six

(c)

Figure 4.3: Example *Query* tables.

table had the *Unique* hit policy, it would be possible to have an outcome in which the die has multiple face values at once.

The third and final addition in pDMN is quantification. For example, the *Vaccine* table shown in Fig. 4.2c expresses that "For every Person X, there is a chance of 36% that they have received vaccine a, a 63% chance on vaccine b, and a 1% chance of being unvaccinated." The X here represents a quantification variable of type *Person*. Similarly, the *Infection* table expresses that every person X who had contact with an infected person Y could now also be infected, depending on their vaccine's performance, or lack thereof.

4.3.3 Query

The *Query* table is the third type of table present in a pDMN model, and is used to denote which symbols' probability should be calculated. Querying the probability of a predicate is done by adding it to the query table, either with specific elements of a type or with a quantification variable. To query a function, the table should contain a cell of the form $func\_name(arg) = val$. Here too, it is allowed to write down a specific element of a type or a quantification variable. Examples of query tables are shown in Fig. 4.3. The table in Fig. 4.3a verifies the probability of flipping two heads and some heads with coins. Fig. 4.3b demonstrates querying predicates with a specific variable value (*bob*), or a quantification variable (X). In the latter case, the probability of the predicate is calculated for every element of the type *Person*. Lastly, Fig. 4.3c, in which we want to know the probability that a die lands on a six, shows the querying syntax for functions.

4.4 Translating pDMN to ProbLog

To apply pDMN models to solve problems, we translate them into ProbLog. We will now go over the general translation principles. Intuitively, every row of

a U-table represents a rule in ProbLog, with the input variables forming the body, and the output variable forming the head. If there are multiple output variables present, a rule is created for each of them. For example, the *heads* table in Fig. 4.2a translates to the ProbLog rules shown in (4.7). Note that the rows in which the output was *No* are not translated, as these do not need to be explicitly formulated in ProbLog due to the closed world assumption.

$$\begin{aligned} &\mathit{twoHeads} \ :\!- \ \mathit{heads1}, \mathit{heads2}. \\ &\mathit{someHeads} \ :\!- \ \mathit{heads1}, \mathit{heads2}. \\ &\mathit{someHeads} \ :\!- \ \mathit{heads1}, \mathrm{not}(\mathit{heads2}). \\ &\mathit{someHeads} \ :\!- \ \mathrm{not}(\mathit{heads1}), \mathit{heads2}. \end{aligned} \tag{4.7}$$

If the output rows of a table contain probabilities, these are added to their respective ProbLog rules or facts. E.g., the *h1* table in Fig. 4.2a translates to the fact $0.5 :: \mathit{heads1}$.

As explained before, DMN also provides the F(irst) hit policy. Consider again the *heads* table in Fig. 4.2a, except we now consider it as an F-table. To translate the first hit behaviour to ProbLog, for any row in the table we need to add the negation of all the previous rows to the body of the translation. To do this, dummy variables are introduced, representing whether a row has fired or not. The resulting ProbLog translation for this example is shown in (4.8), where *r1*, *r2* and *r3* represent the dummy variables.

$$\begin{aligned} &\mathit{r1} \ :\!- \ \mathit{heads1}, \mathit{heads2}. && \mathit{twoHeads} \ :\!- \ \mathit{r1}. \\ &\mathit{r2} \ :\!- \ \mathit{heads1}, \mathrm{not}(\mathit{heads2}). && \mathit{someHeads} \ :\!- \ \mathit{r1}. \\ &\mathit{r3} \ :\!- \ \mathrm{not}(\mathit{heads1}), \mathit{heads2}. && \mathit{someHeads} \ :\!- \ \mathit{r2}, \mathrm{not}(\mathit{r1}). \\ & && \mathit{someHeads} \ :\!- \ \mathit{r3}, \mathrm{not}(\mathit{r1}), \mathrm{not}(\mathit{r2}). \end{aligned} \tag{4.8}$$

Tables with the newly introduced *Ch(oice)* hit policy are translated into ProbLog's annotated disjunctions. For example, the table shown in Fig. 4.2b assigns a value to the 0-ary *die value* function. In ProbLog, n-ary functions are represented by an $(n+1)$-ary predicate, resulting in the unary *die_value* predicate:

$$\begin{aligned} &1/6 :: \mathit{die\_value}(\mathit{one}); \ldots; 1/6 :: \mathit{die\_value}(\mathit{six}) \ :\!- \ \mathrm{not}(\mathit{biased}). \\ &1/5 :: \mathit{die\_value}(\mathit{one}); \ldots; 1/2 :: \mathit{die\_value}(\mathit{six}) \ :\!- \ \mathit{biased}. \end{aligned} \tag{4.9}$$

Types declared in the *Type* table in pDMN are represented by unary predicates in ProbLog, as the latter is not a typed language. Additionally, the contents

of the *Elements* column are translated into facts. E.g., *Person*, as shown in the *Type* table in Fig. 4.1, translates to the facts $person(ann)$ and $person(bob)$. When translating a decision table containing quantification, the type of the quantification variable(s) is derived from the glossary, and an atom is added to the ProbLog rule for each variable to denote its type.

For example, the pDMN model in Fig. 4.2c translates to the ProbLog program shown in (4.10). Note that we shortened some of the variable names to better fit the page size. Now consider e.g. the *Infection* table: it contains two quantification variables, X and Y, both of type *Person*. As such, this is denoted in the ProbLog rules by adding two atoms to their bodies, $person(X)$ and $person(Y)$, to represent the types of the variables.

$$
\begin{aligned}
&person(ann). \quad person(bob).\\
&vacc(a). \quad vacc(b). \quad vacc(n)\\
&0.36 :: vacc(X,a); 0.63 :: vacc(X,b); 0.01 :: vacc(X,n) \ :- \ person(X).\\
&0.8 :: inf(X) \ :- \ vacc(X,n), inf(Y), contacted(X,Y), person(X), person(Y).\\
&0.1 :: inf(X) \ :- \ vacc(X,a), inf(Y), contacted(X,Y), person(X), person(Y).\\
&0.2 :: inf(X) \ :- \ vacc(X,b), inf(Y), contacted(X,Y), person(X), person(Y).
\end{aligned}
\tag{4.10}
$$

The *Query* table is represented in ProbLog by *query* statements. For every cell of the table, a new *query* statement is added. For example, the three query tables shown in Fig. 4.3 translate to the following ProbLog statements:

$$
\begin{aligned}
&query(twoHeads).\\
&query(someHeads).\\
&\\
&query(vaccine\_of\_Person(bob)).\\
&query(person\_is\_infected(X)).\\
&\\
&query(die\_value(six)).
\end{aligned}
\tag{4.11}
$$

If no *Query* table is present in a pDMN model, it is assumed that the probabilities of all symbols of the model should be queried. In such a case, a ProbLog *query* rule is generated for every entry in the *Predicate* and *Function* glossary tables.

Type	
Name	**Elements**
Person	john, mary
Intensity	heavy, mild, none

Predicate
Name
burglary
alarm
Person calls
anycalls

Function	
Name	**Type**
earthquake	Intensity

Burglary	
U	burglary
	Yes
1	0.7

Earthquake			
Ch	earthquake		
	heavy	mild	none
1	0.01	0.19	0.8

Calls		
U	alarm	X calls
		Yes
1	Yes	0.8
2	No	0.1

Alarm			
U	burglary	earthquake	alarm
			Yes
1	Yes	heavy	0.9
2	Yes	mild	0.85
3	Yes	none	0.8
4	No	mild	0.1
5	No	heavy	0.3

anycalls		
U	X calls	anycalls
1	Yes	Yes

Query
X calls
anycalls

Figure 4.4: Full pDMN model for the *Earthquake* example

4.5 Implementation

To automatically translate pDMN models to ProbLog and execute them, the translation principles described earlier have been implemented in a solver[1]. This solver is largely based on the solver which we created for cDMN, due to the similar nature of the notations. The input for the solver is a pDMN model in the form of a `.XLSX` spreadsheet. Concretely, the solver works in three steps.

First, it interprets all glossary tables in the spreadsheet, beginning with the *Type* table. For every entry, the solver creates internal *Type* objects, necessary to interpret the arguments used in the *Predicate* and *Function*

[1] `https://gitlab.com/EAVISE/cdmn/pdmn`

tables. The solver then evaluates every decision table one-by-one, using a lex/yacc parser to parse every cell and transform them into a pDMN expression. For example, an expression of the form "*vaccine of bob*" is translated into "*vaccine_of_person(bob)*".

Next, all decision tables, are converted into ProbLog rules in the manner described earlier. At the same time, the *Query* table is parsed and converted into ProbLog *query* statements.

Lastly, the generated specification is executed using ProbLog's Python API, after which the queried probabilities are shown. In this way, the pDMN execution process consists of a closed pipeline between pDMN modelling and ProbLog execution.

The pDMN solver is available as a Python package, and can be downloaded from its PyPI repository[2].

4.6 Full example

In the previous sections, every example only consisted of limited snippets of pDMN models. To give a view of what a complete pDMN model looks like, this section shows a concrete implementation of the well-known *Earthquake* example. In this example, a house alarm can be triggered by a burglary, by an earthquake of a certain intensity (heavy, mild or none), or by a combination of the two. Both the burglary and the intensities of the earthquake have a probability associated with them. If the alarm rings, the neighbours *John* and *Mary* both could either call the home owner, or they could dismiss the alarm as incorrect and ignore it. We now want to find out the probabilities of either neighbour calling.

The pDMN model for this example is shown in Fig. 4.4, and consists of the glossary tables, five decision tables and a query table. In the glossary tables, we first introduce two types, *Person* and *Intensity*, which respectively represent the neighbours and the earthquake intensities. In the *Predicate* table, we declare four predicates: the 0-ary predicates *burglary*, *alarm* and *anycalls*, and the unary predicate *Person calls*. To denote the intensity of the earthquake, we make use of the 0-ary function *earthquake*, which will thus either be *heavy*, *mild*, or *none*.

Of the five decision tables, two are straightforwardly used to set the probabilities of a burglary and the earthquake intensities. As these concepts do not depend

[2] `https://pypi.org/project/pdmn`

on anything, their decision tables contain no input columns. The *Alarm* table contains a rule for every possible combination of burglary and earthquake to represent the probability of the alarm triggering. Note that it does not contain a rule in which neither a burglary or an earthquake take place, as the alarm will never trigger in such a situation, thus allowing us to leave out that rule. The fourth decision table, named *Calls*, expresses that every person X has a certain probability to call the home owner, depending on whether the alarm rings. Finally, the last decision table defines $anycalls = Yes$ whenever any person X calls.

To find the probability of each neighbour calling separately, and the probability of either of them calling, the *Query* table is added to the model in order to finish it. Translating this model to ProbLog using the pDMN solver results in the following code:

```
% facts
intensity(heavy). intensity(mild). intensity(none).
person(john). person(mary).
% Burglary
0.7 :: burglary.
%Earthquake
0.01 :: earthquake(heavy); 0.19 :: earthquake(mild); 0.8 :: earthquake(none).
% Alarm
0.9 :: alarm :- burglary, earthquake(heavy).
0.85 :: alarm :- burglary, earthquake(mild).
0.8 :: alarm :- burglary, earthquake(none).
0.1 :: alarm :- not(burglary), earthquake(mild).
0.3 :: alarm :- not(burglary), earthquake(heavy).
% Calls
0.8 :: person_calls(X) :- alarm, person(X).
0.1 :: person_calls(X) :- not(alarm), person(X).
% anycalls
anycalls :- person_calls(X).
query(person_calls(X)).
query(anycalls).
```

We can also use the pDMN solver to run ProbLog directly, thereby "executing" the example. This results in the following output:

```
>>> pdmn Examples.xslx -x -n "Earthquake"
{person_calls(mary): 0.501765, person_calls(john): 0.501765,
 anycalls: 0.6319415}
```

4.7 Conclusion

This chapter presents a preliminary version of pDMN, a notation for Probabilistic Logic Programming based on the DMN standard. While DMN's deterministic nature is one of the main advantages of the notation, there are many application domains in which probabilities appear. To this end, the goal of pDMN is to extend DMN with probabilistic reasoning, while maintaining the readable and user-friendly format of the original. It extends DMN with probabilities, predicates, quantification, and a new hit policy to represent annotated disjunctions. We have laid out the general translation principles of converting pDMN into ProbLog code, allowing for the execution of the pDMN models. These principles have also been implemented in an automatic conversion tool, which is available for general use.

In future work, we plan on further extending the notation (e.g., with support for more hit policies), formalizing the complete pDMN semantics, extending the DRD to support probabilities and making a user-friendly interface for the system.

Chapter 5

Feature Modelling

Feature Models are straightforward diagrams representing a product's configuration. While they are typically used as part of a Software Product Line (SPL), feature models also have much potential to be used in a manufacturing context. In particular, they could help to create design tools which support the engineers in creating correct designs. The goal of this chapter is to investigate this link.

This chapter is based on work presented at the 24th International Workshop on Configuration, September 2022 [127]. The research was performed in collaboration with Benjamin Callewaert.

5.1 Introduction

Feature modelling [74] is frequently used in industry to model variability in a product. The goal of this approach is to create a diagram that contains all of the product's features and their underlying relations. In this way, we create an overview of all possible configurations of the product, assisting in the development of new designs, which is important in times of *mass customizability*. Usage of feature modelling decreases time-to-delivery, costs, product risk and labour [101].

Feature models are almost exclusively used in Software Product Lines (SPL), where the goal is to write software that is re-usable across different variations of products. In this context, feature models are used to keep an overview on the product, its features, and the software libraries needed to support them. From a configuration point of view, the following question then arises: "If a feature model represents all possible configurations of a product, why is it not also used to build software that helps in configuring designs?"

However, for realistic applications of this kind, feature modelling and related tools fall short: to create real designs, we also require knowledge from "outside" the design, such as knowledge on the working environment, which a feature model cannot elegantly represent. Thus, to optimally apply feature models to designs in a manufacturing context, they need to be extended to allow us to represent background knowledge. However, this addition of background knowledge leads to a rise in the conceptual complexity of the model, making it harder to ensure its correctness.

In general, the correctness of a feature model is of paramount importance. To ensure this correctness, two distinct operations can be performed: *verification*, which is "building the system right", and *validation*, which is "building the right system" [20]. In other words, verification is checking whether the system conforms to specifications, free of syntax errors, whereas validation compares the system against the real world [62].

For large and complex feature models, verification and validation are difficult to perform without the support of proper tooling. Hence, much research has been performed on automated reasoning on feature models. Such systems include tools based on FOL [89], constraint programming [16], logic programming [76] and description logics [55]. They are capable of performing various automated verification and validation tasks, such as generating all possible configurations, verifying if at least one solution is feasible, detecting so-called *dead features* and *false optional features*, etc.

While verification can be performed fully automatically, validation benefits

from interaction with the domain expert. During such interactive validation, the domain experts "play around" with the system, to check if their actions lead to the results that they expect. However, for this process to be effective, supporting tools are required. Such a tool should (a) be interactive, (b) allow to explore the configuration space in a flexible way, and (c) should be explainable. The latter is specifically important in the case of incorrect behaviour, to quickly identify the error.

In this chapter, we present our interactive feature modelling tool, FM-IDP. The tool distinguishes itself from the state of the art by (1) extending feature models with background knowledge and (2) combining this with a user-friendly, interactive and explainable configuration interface. On the one hand, it allows modellers to enhance their feature models with background knowledge in FO($\cdot$). With this additional knowledge, feature modelling becomes better suited to tackle design problems in a manufacturing context, as we will motivate with a real-life use case. On the other hand, the modeller can interact with the entirety of the knowledge via an interactive interface, empowering them to quickly and interactively explore the problem domain. This also makes FM-IDP well-suited to assist product engineers in creating new designs in a real-life context.

In short, this chapter presents a full-fledged tool to support the design of new products. The contributions are as follows:

- We extend feature modelling with background knowledge in FO($\cdot$)
- We support interaction with the resulting model using the IDP-Z3 reasoning engine paired with a user-friendly and explainable interface
- We motivate our approach by a real-life industrial use-case

The chapter is structured as follows: we begin by introducing feature modelling and related works in Section 5.2. Next, we elaborate on a running example for feature modelling with our approach in Section 5.3. We present our FM-IDP tool in Section 5.4, and perform a brief comparison and evaluation in Section 5.5. Finally, we conclude in Section 5.6.

5.2 Background & Related Work

Feature modelling aims to provide an overview of a product and its variability. Introduced in 1990 [74], it has since gained wide popularity in the design of Software Product Lines. We will briefly go over the concepts and components of a feature model diagram, using Fig. 5.1 as an example. In this example, we

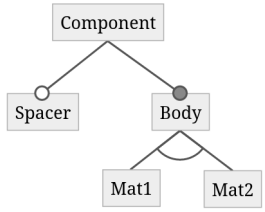


Figure 5.1: Example of a feature model

have modelled the design of a *Component*, which consists of multiple *features* as denoted by its child nodes. These nodes can be connected through multiple types of relationships.

- A *mandatory* feature is included in every configuration in which its parent of the feature is included. This is denoted by an edge ending in a filled-in circle, such as for *Body*.
- An *optional* feature may or may not be included if its parent is included. This is denoted by an edge ending in an empty circle, such as for *Spacer*.
- Children can be *alternative* features to each other, meaning that exactly one of them must be selected, if their parent is included. This is denoted by a curve between the edges of the child nodes, as shown for *Mat1* and *Mat2*.
- Children can be in an *or* relationship, meaning that at least one child (instead of precisely one, as is the case for alternatives) must be selected. This relationship is not shown in the example, but is denoted by a filled-in curve between the edges.

Besides these parent-child relationships, there are also two types of *cross-tree constraints* possible. Firstly, a feature x can *require* a feature y, meaning that each configuration containing the former must also contain the latter. Secondly, a feature x and y can *exclude* each other, such that no legal configuration may contain both features at the same time. These cross-tree constraints are typically either drawn on the model itself using an arrow between the two nodes, or are specified in a separate text.

There have been many extensions to feature modelling proposed throughout the years. The two most prominent ones are cardinality-based feature modelling [38] and extended feature modelling [18]. In cardinality-based feature modelling, features can be annotated with a cardinality $[p..q]$ to denote that there should be between p and q copies of the child. For example, the *mandatory* and

optional relations correspond to [1..1] and [0..1] respectively. In extended feature modelling, features can be extended with attributes. An attribute is typically a measurable characteristic of a feature, such as *cost*, with its own feature domain.

Since the introduction of feature models, much research has gone into their verification and validation. For our purposes, verification is "building the system right" (i.e., there are no structural errors) while validation is "building the right system" (i.e., there are no knowledge errors). In this work, we focus on validation and will therefore not discuss verification any further, but refer to [17, 51, 59, 93] for more information.

In [117], the authors describe a method to automatically validate feature models based on textual specifications, by checking that the features and relations in the feature model are described by the text, and vice versa. Another tool supporting automated validation is DIRECTDEBUG [83], where the model can be compared against sets of positive and negative tests cases. This approach is especially useful when correct products have been designed before, serving as excellent tests.

Besides these automated validations, there exist some tools that support interactive validation. Botterweck et al. outline the importance of a visual, interactive tool for configuring and understanding software product lines [23], and present their own tool for this purpose. In [22], Botterweck et al. elaborate on an interactive feature configurator. By performing a translation from feature models to SAT, the configurator can use a SAT solver to automatically derive the consequences of user input. The tool is also capable of generating explanations for those consequences which the user does not understand.

Another example of an interactive tool is FeatureIDE [119], a full-fledged framework for development of Software Product Lines, which contains a feature configurator with similar functionalities to the one by Botterweck et al. Additionally, it also allows more expressive propositional cross-tree constraints, by allowing combinations of all logical connectives through an intuitive constraint editor.

The need for adding background knowledge to feature modelling has also been recognized in literature. To address this need, Myllärniemi et al. [99] present transformations from feature modelling to intermediary languages, to "enable the product line engineer to operate on domain-specific modelling constructs.". This intermediary language can then be translated into an Answer Set Programming (ASP) specification to perform configuration. In this way, the engineer can add the background knowledge required for a correct design in a user-friendly manner.

The modelling tool Clafer [8] also combines feature models with more complex knowledge, in a language built on top of first-order logic. As explained in [7]: "Clafer offers an expressive constraint language that allows [the modeller] to specify any first-order logic formulas, which reduces the amount of tacit knowledge". It is capable of converting the models into input for multiple solvers, such as Z3 [98], which can then be used to generate valid configurations. To support interactive configuration, they have introduced the Clafer Configurator [7], where a given number of possible configurations are generated at the start, after which the user can explore them through toggling on or off features.

In this work, we propose an interactive feature model configurator based on IDP-Z3. In our tool, modellers can complement the feature model with background knowledge in FO($\cdot$). As we will motivate by a real-life use case, an expressive language to represent background knowledge, e.g., on the environment of a product, is essential to select feasible configurations. Additionally, our tool comes with an interactive interface, allowing the users to explore the problem domain, validate the feature model and select an appropriate configuration. To facilitate such interactions between the user and the entirety of the knowledge, our tool supports several functionalities: the consequences of design choices are clearly visible to the user, the tool can generate feasible configurations and is able to provide explanations. Although a number of existing systems already provide interactive capabilities or allow background knowledge, our work is unique in providing the combination of the two. As we will demonstrate in our use case, it is precisely this combination that is required for a system to be suitable as a support tool for product design in a manufacturing context.

5.3 Use Case

In this section we describe the industry use case which we use to support our claims. It was previously presented in [4], and comes from a company that wants to support their engineering staff in designing machine components.

The company has specified a few important goals. To begin with, the design support should be interactive: the engineers want to incrementally explore the consequences of different design choices. Additionally, explainability is of high importance, to ensure that the engineers can track *why* their choices lead to these consequences. This helps the engineers to understand the model and to trust the tool. To properly support this interactivity, the tool should be responsive enough to avoid that the engineers have to wait for a result. Another important goal for this tool is that the company's engineers should be able to

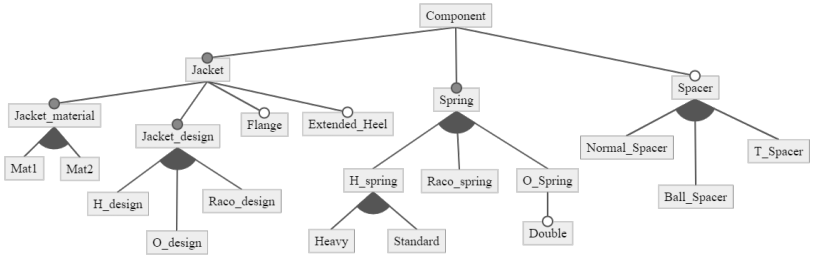


Figure 5.2: Feature model of the Component use case

maintain the model themselves: if new information about the problem domain is discovered, or they want to modify existing information, they should be able to update it without help from IT staff. This goal puts the engineers in the driver's seat of the tool: they model their own knowledge, which the system uses to support them in the design process.

We will briefly discuss a simplified version of the problem domain, which nevertheless retains most of its important characteristics. The design of the component may consist of three sub-components: a jacket, a spring and an optional spacer. A jacket can be constructed out of two possible materials, named "Mat1" and "Mat2" for simplicity. There are multiple options for its design: either an "H-design", an "O-design" or a "RACO-design". Optionally, the jacket may also have an extended heel and/or a flange. The spring, also a mandatory sub-component, is either an "H-spring", a "RACO-spring" or an "O-spring". Here, there are 2 types of "H-springs": they are either standard or heavy. Additionally, the "O-spring" can also be a double spring. Finally, the component may also have a spacer. A spacer is either a normal spacer, a ball spacer or a "T-spacer". Together with one of the company's design engineers, we constructed a feature model representing this knowledge, as shown in Fig. 5.2.

There are a few design constraints applicable on these components, which are modelled using cross-tree constraints. Firstly, the design of the jacket requires a corresponding spring design, and vice versa. Next, if an "H-design" is chosen for the jacket, the component also needs a spacer. Inversely, the "RACO-design" prohibits the use of a spacer. Lastly, ball spacers and "T-spacers" always require either an "H-design" or a double "O-spring".

To design a correct component, it is important to incorporate the influences of the working environment, such as the working temperatures. We were not able to directly model these in the feature model. Some examples of such knowledge are:

- If back pressure is present in the environment, the component requires a spacer.
- "Material1" is not heat resistant, and cannot be used above temperatures of $200°C$.
- If the temperature is below $-50°C$, either a heavy "H-spring" or double "O-spring" is needed to compensate this.

To summarize, the implementation of this use case has the following requirements:

R1. an expressive notation, capable of modelling the problem

R2. a notation which the engineers can update themselves

R3. support for expressing background knowledge

R4. interactivity between the user and the system

R5. explainability

R6. support for unbounded integer and real numbers

5.4 FM-IDP Tool

The FM-IDP tool combines the simplicity and intuitiveness of feature modelling with the interactivity of the Interactive Consultant. It aims to support interactive exploration of configuration space with background knowledge, which is useful both for an expert to validate feature models and for making concrete, error-free configurations. The tool's interface is split into two tabs: one for knowledge editing, and one for knowledge interaction.

Knowledge editing As there are two sources of knowledge in the FM-IDP tool (a feature model and its accompanying background knowledge), there are also two editors. The feature model editor, as visible on the left side of the screenshot in Fig. 5.3, is a graphical editor for basic feature modelling. Creating feature models is performed by clicking on the canvas to create a new node, naming it, clicking its parent node to form a connection and selecting the connection type. Once all features have been added, the model is converted into an FO($\cdot$) vocabulary and theory. These are visible to the user through a read-only text editor, present in the bottom-right of the screen. The conversion

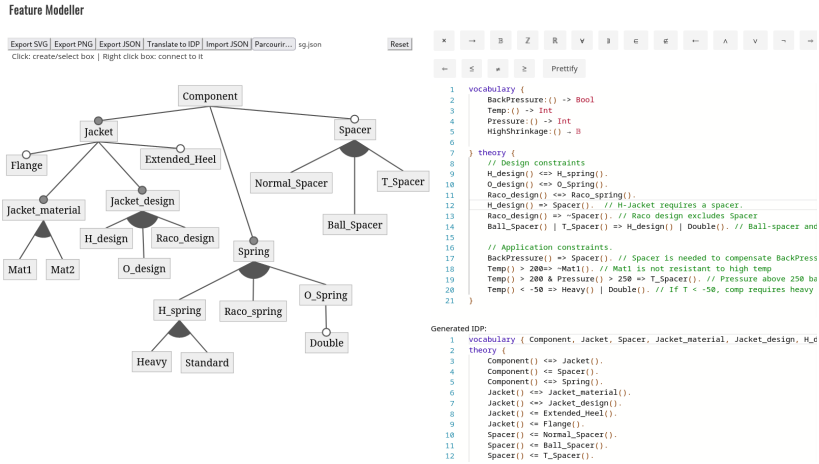


Figure 5.3: Screenshot of the Component use case modelled in the knowledge editor

to FO(·) relies on the translation as used in [12], in which each node of the diagram is represented by a proposition. A brief overview of how to transform different relations is shown in Table 5.1. As an example, the feature model shown in Fig. 5.1 is translated to the following FO(·) formulae:

$$Spacer() \Rightarrow Component().$$

$$Body() \Leftrightarrow Component().$$

$$Body() \Leftrightarrow Mat1() \vee Mat2().$$

$$Body() \Leftrightarrow ((Mat1() \vee Mat2()) \wedge \neg(Mat2() \wedge Mat1())).$$

After creating the feature model, background knowledge is modelled in the background knowledge editor in the top-right corner. This editor allows expressing two things:

1. In the vocabulary, we can declare additional concepts, such as *BackPressure* and *Temperature*.
2. In the theory, we can write additional FO(·) formulae, such as "$BackPressure \Rightarrow Spacer$" and "$Mat1 \Rightarrow Temperature < 200$".

Behind the scenes, the vocabulary and theory blocks are merged with their counterparts generated from the feature model. In this way, they form one

Table 5.1: Translation from feature relations to propositional logic

Relation	Propositional Representation
c is mandatory for p	$p \Leftrightarrow c$
c is optional for p	$p \Leftarrow c$
p has alternative $c_0 \ldots c_n$	$p \Leftrightarrow \bigvee_{i=0}^{n} c_i \wedge \bigwedge_{0 \leq j \leq k \leq n} \neg(c_j \wedge c_k)$
p has OR $c_0 \ldots c_n$	$p \Leftrightarrow (c_0 \vee \ldots \vee c_n)$
x requires y	$x \Rightarrow y$
x excludes y	$x \Rightarrow \neg y$

complete KB containing both the knowledge of the feature model, and its required background knowledge.

The tool also supports exporting/importing knowledge in the form of a custom-formatted json file, which describes all the nodes in the feature model, their connection, and the present background knowledge.

Knowledge interaction Interaction with the knowledge is facilitated through the Interactive Consultant, as shown in the screenshot in Fig. 5.4. This screenshot demonstrates how for each symbol in the KB a *tile* is generated that can be used to assign it a value. E.g., a proposition is displayed with a green check mark and a red cross, which either assert or negate the proposition when clicked. In the example, *BackPressure* has been asserted by a user, as indicated by the yellow check mark. Based on this, the Interactive Consultant automatically derived that, among others, *Spacer* should be true as well, as indicated by the grey check mark, and that a *Raco design* is no longer possible for the jacket, as indicated by the grey cross. If the user does not fully understand a consequence, they can ask for an explanation by clicking the grey button: i.e., the consequence for *Raco design* is explained by the fact that the user selected *BackPressure*, which requires a spacer to be present, which in turn excludes *Raco* jacket design.

This process of selecting features in the interface and immediately viewing the consequences results in a tight feedback loop between the user and the knowledge. Moreover, the ability to generate explanations ensures explainability and interactivity. It is for these reasons that the configurator in FM-IDP lends itself excellently to the interactive exploration of the problem space. As such, it is a useful tool in the context of expert validation of the feature diagram. For example, just by opening the tool a user can make several observations:

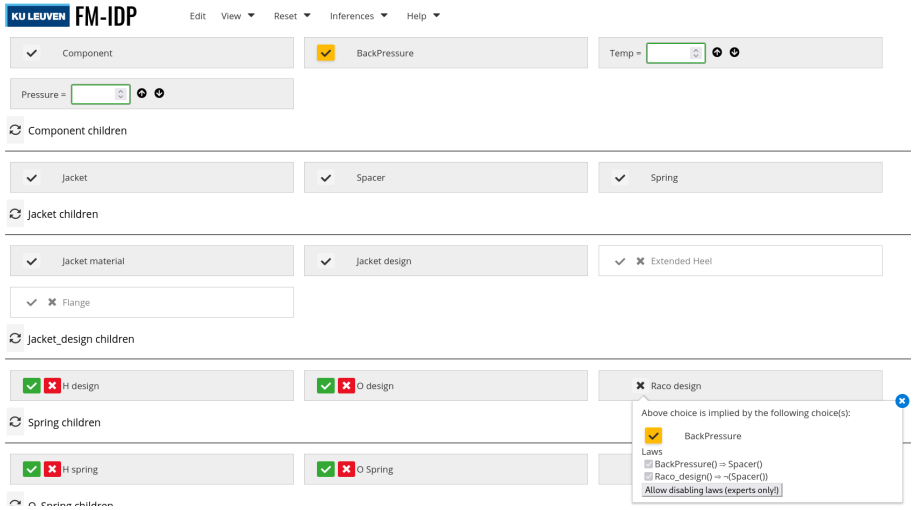


Figure 5.4: Screenshot of the Component use case in the FM-IDP configurator

a. whether a legal configuration is possible, if the root's proposition symbol does not show a grey cross mark

b. which features are *core features* that are always present, as the interface always shows them with a grey tick mark

c. which features are *dead features* that are never present, as the interface always shows them with a grey cross

These third features are most likely due to a mistake in the knowledge, which should be fixed. While tracing back such an error could be difficult in a large and complex feature model, this is made easy due to the interface's explanation feature. Indeed, these explanations can be interpreted as *corrective explanations*, which tell the user what things can be changed to achieve a different outcome.

Our motivation for using the IDP-Z3 system in our FM-IDP tool instead of other off-the-shelf reasoning engines, such as those for ASP or Prolog, is threefold. Firstly, while some of IDP's inference tasks are also supported by other systems, such as model expansion and propagation, the benefit of the IDP-Z3 system is that it implements all of them together in the same solver. This multi-functional approach is quite unique, and allows tools such as the Interactive Consultant to be built. Secondly, the FO($\cdot$) language used to express knowledge is easy-to-read, yet expressive. The background knowledge is not limited to propositional logic; when extending the knowledge in the KB, modellers can benefit from n-ary

symbols and FO(·)'s large expressiveness. Lastly, IDP-Z3 and the Interactive Consultant have both already proven themselves to be effective in the context of configuration problems [4, 48].

An additional benefit of the FM-IDP tool is that modellers and users get to know FO(·), IDP-Z3 and the Interactive Consultant. In this sense, it can be used as an accessible introduction to propositional logic, driven by the intuitive nature of feature models, which can then be extended to encompass more complex expressions.

The source code of FM-IDP is available online[1]. A hosted instance of the tool is available for testing online[2].

5.5 Evaluation and Comparison

In this section we evaluate the FM-IDP tool and make a brief comparison with the state-of-the-art. We use the requirements listed in Section 5.3 as basis for evaluation. Table 5.2 shows an overview of this evaluation in a comparison with the other works discussed in this section.

Requirements R1–R3 are met by our combination of feature modelling and FO(·) as complementary notation languages, where the former allows to intuitively model the product and its features, and the latter allows us to straightforwardly extend it with background knowledge. By then utilizing the IDP-Z3 reasoning engine, we meet requirement R6 as IDP-Z3 supports reasoning over unbounded integer and real numbers, and delivers the required computational performance. Indeed, in the full use case, which consists of 55 variables (of which 27 are integers/reals), each value assignment takes on average 2 seconds to propagate. Finally, the Interactive Consultant's interactivity and explainability ensure that requirements R4 & R5 are satisfied as well. This demonstrates that our approach is indeed capable of capturing the variability of real-life machine components and offering all of the functionality that design engineers expect from a design support system.

The ASP approach presented in [99] and Clafer [8] both allow adding additional knowledge to the feature model: the former supports an intermediate language with concepts familiar to the product line engineer, while the latter extends feature modelling with a language built on first-order logic. However, neither system supports interactive configuration using the knowledge. The ASP solution does not support interactive configuration [96]. In the case of Clafer,

[1] `https://gitlab.com/EAVISE/featuremodel/feature-model-IDP`
[2] `https://fm-idp.onrender.com`

Table 5.2: Comparison between functionalities of FM-IDP and other tools. X = full support, o = partial support.

Work	R1	R2	R3	R4	R5	R6
ASP-approach [99]	X	X	X			
Clafer Conf. [7]	X	o	X	o		
Botterweck et al. [22]		X		X	X	
FeatureIDE [119]		X		X	X	
FM-IDP	X	X	X	X	X	X

while the Clafer Configurator [7] aims at supporting interactive exploration of solutions, it is not sufficiently versatile. Indeed, the tool only allows a user to explore a number of initially generated configurations. There is also no support for generating explanations. It does not show the consequences of choices in general (i.e., beyond those initially generated configurations), and does not provide explanations for consequences.

Note that generating a fixed number of configurations as Clafer does is not enough to simulate the kind of propagation offered by FM-IDP. To see why, consider our use case with two important integer environment variables, *Temperature* and *Pressure*, both of which have a possibly infinite domain. As a result, many of the configurations generated by Clafer will be essentially the same, modulo tiny difference in the value of these variables. This makes it hard for the engineers to see the forest for the trees. Moreover, there is no guarantee that all values which do make a difference will actually be covered.

The tool by Botterweck et al. [22] supports interactive configuration, in a similar fashion to FM-IDP. However, it does not support background knowledge, thereby limiting the use of the tool for real-life configuration. Similarly, while the configurator of FeatureIDE [119] supports a constraint editor, this editor is limited to cross-tree constraints without the possibility of adding new symbols to express constraints on. Furthermore, constraints can only be expressed in propositional logic, whereas FM-IDP supports full FOL expressions.

One recurring difference between our work and most of the compared state-of-the-art is that FM-IDP is restricted to basic feature modelling, whereas the other tools typically support cardinality-based modelling and/or extended modelling. However, this design choice is motivated by two reasons. Firstly, as Schobbens et al. [112] point out, original feature models are expressively complete. Secondly, FM-IDP is meant to showcase, as a prototype, that our approach of combining feature modelling with IDP and the Interactive Consultant opens up new possibilities. Basic feature modelling suffices to show

this synergy. In future work, the modeller will be extended to also support cardinality-based and extended feature modelling.

5.6 Conclusion

Feature models are an intuitive notation for modelling a product's features, components, and the relations between them. But for feature models to be useful in real-life configuration design problems, the feature model should allow complex background knowledge and be supported by an interactive interface to explore the problem domain.This is where the current state-of-the-art falls short. Although some of the considered systems do have interactive functionalities or allow to add background knowledge, no system fully supports the combination of these two aspects.

In this chapter, we presented our own feature modelling tool FM-IDP. It combines the simplicity and intuitiveness of feature modelling with the advantages of the IDP-Z3 system and the Interactive Consultant. It allows the user to interactively explore the design problem and validate the correctness of the feature model. Furthermore, the feature model can be extended with background knowledge in FO($\cdot$), an easy-to-understand extension of FOL. The combination of an interactive modeller and the possibility to add expressive background knowledge makes the FM-IDP tool unique w.r.t. other state-of-the-art FM tools.

We evaluated our tool against the requirements for a design support system of a real-life industrial use case and demonstrated that our approach is indeed capable of capturing the variability of real-life machine components, while offering all of the functionality that design engineers expect from such a system.

Chapter 6

Context-Aware Verification of DMN tables

To ensure correctness of DMN models, all decision tables should be both *sound* and *complete*. State-of-the-art verification strategies focus on decision tables in isolation, without regard for any context. Yet, this context, while in-model or as background knowledge, can influence the output of the table. The goal of this chapter is to look into extending DMN verification capabilities with context.

This chapter is based on work presented at the Hawaii International Conference on System Sciences (HICSS), January 2021 [126]. The research was performed in collaboration with Benjamin Callewaert.

6.1 Introduction

An important aspect of decision tables is that they should be both *complete* and *sound*. A table is complete if it contains an applicable row for every possible set of input values. The soundness of a table depends on its hit policy: U tables should not contain overlapping rows of any kind, while A tables allow them as long as they do not have conflicting outputs. Tables with the F hit policy are allowed to have overlap, and as such, are always sound. Tables lacking these correctness properties are considered erroneous. On top of soundness and completeness, tables should also be without *unfireable* rules, i.e., rules that can be omitted without changing the meaning of the table.

Efficient tools have been created to perform automated table verification. However, as we point out in this chapter, most of these tools are unable to (sufficiently) reason on the *context* of the table and the model. Indeed, most approaches verify each table in isolation, i.e., without any regard to the rest of the model. We are aware of two approaches that do take context into account, but they either do not make enough use of context, or are unable to sufficiently pin-point specific errors, as we will discuss later.

The contributions of this chapter are threefold: (a) demonstrating the importance of context via concrete use cases, (b) formally defining context-aware verification, and (c) building a tool capable of verification *within context*. It is structured as follows. We first look at the tools that are already available in Section 6.2. Afterwards in Section 6.3, we give concrete examples in which context is important. In Section 6.4 we formally explain the correctness criteria, and extend them to include context. These definitions are then used in our implementation, as explained in Section 6.5. We briefly compare and evaluate our tool in Section 6.6, and finally we conclude in Section 6.7.

6.2 Related Work

Smit et al. [115] conducted a study on DMN verification in a real-life context. In total, they identified eight different verification capabilities for decision tables. As shown in Table 6.1, five of these capabilities can be considered as specific kinds of soundness. Indeed, all these errors are caused by the same root cause: two or more (fully) overlapping rules. While they do differ in the actions required to fix them, we will nevertheless combine these five capabilities together in this work, as this already showcases the benefit of our context-aware approach. The sixth verification capability, “Missing Rules”, corresponds to table completeness. The two remaining verification capabilities, “Unnecessary fact verification” and

Table 6.1: Translation between verification capability terminology.

Smit et al. terminology	Our terminology
Identical rules Equivalent rules Subsumed rule Indeterminism Overlapping fact value range	Soundness
Missing Rules	Completeness
—	Unfireable rule
Unnecessary fact verification	—
Specific partial reduction	—

"Specific partial reduction", are not considered as verification by us and are therefore beyond our scope. There is no counterpart for unfireable rules in the framework of Smit et al.

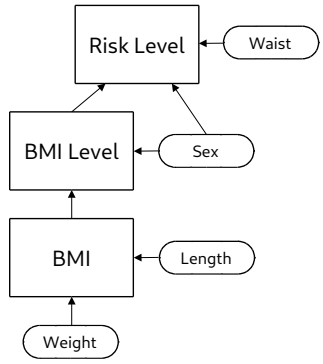
Table 6.2 shows an overview of previous works on decision table verification, and which verification capabilities they support. With the exception of Hasic et al. [67] and Calvanese et al. [29], all of the proposed algorithms verify a decision table in isolation from the rest of the model. For example, when verifying the *BMILevel* table in Fig. 6.1, the information in the (upstream) *BMI* table or in the (downstream) *Risk Level* table would simply be ignored.

The tool by Hasic et al. [67] is one of the exceptions that does incorporate some context from the rest of the model in the verification process. Indeed, when verifying a table, it also checks whether every output value of the directly upstream table(s) appears as input value in the current table, and checks for every output value of the current table whether it appears as input value in the directly downstream table(s). For example, when verifying the *Risk Level* table, they check whether every possible output for *BMILevel* (Underweight, Normal, Overweight) appears at least once as an input.

In [29], Calvanese et al. outline a conceptual framework for *semantic DMN*, as a way to reason on DMN tables together with background knowledge. Together with this framework, they also extend their verification methods from [27] to include forms of background knowledge. However, when verifying table completeness for example, their algorithm can only tell *if* rules are missing, but not *which* rules are missing. Depending on the size of the DMN model and the amount of background knowledge, this can drastically reduce the

BMILevel		
U	BMI	BMILevel
1	< 18.5	Underweight
2	[18.5..25]	Normal
3	> 25	Overweight

(a) Example of a decision table.



(b) Example of a Decision Requirements Diagram

Figure 6.1: DMN example

Table 6.2: Verification tools and their capabilities. (X = full support, o = partial support, * = does not distinguish between types of soundness, † = boolean result)

Work	Soundness	Completeness	Unfireable rules	Context
Calvanese et al. (2016) [26]	o	X		
Laurson et al. (2016) [82]	o	X		
Batoulis et al. (2017) [13]	o	X		
Calvanese et al. (2018) [27]	o	X		
Corea et al. (2019) [37]	X	X		
Calvanese et al. (2019) [29]	o†	X†	X†	X
Hasic et al. (2020) [67]	X	X		o
Our tool	X*	X	X	X

actual usefulness of the verification method. Indeed, if a large table is found to be incomplete, manually finding all the missing rules is both difficult and time-consuming.

6.3 Types of Context

We identify two types of context, each with a different meaning and scope.

1. *In-model context*: the information contained in the rest of the model, i.e., in all tables apart from the one currently being verified.
2. *Background knowledge*: additional knowledge about the domain that is not part of the DMN model itself.

This section elaborates on both types, and gives concrete examples of cases where this context matters.

6.3.1 In-model context

The first type of context is the in-model context, which consists of all decision tables in the model that are not the target table to be verified. For example, consider the model in Fig. 6.2: if we want to perform table verification on the "Risk Level" table, its context would consist of the "BMI" and the "BMILevel" tables.

When verified in isolation, the "Risk Level" table passes our three tests: it is sound, complete, and has no unfireable rules. However, when taking the in-model context into consideration, the table has two rows that are actually *unfireable*. Indeed, both the third and the fourth row of the table can never fire, as the input combination *BMILevel* = *Overweight* and *Sex* = *Male* is not possible, according to the "BMILevel" table; this table has no rule for which the input value of *Sex* = *Male* leads to output value *Overweight* for *BMILevel*.

This is an example in which the error cannot be detected by considering either table in isolation: it can only be detected by looking at both tables together. There are two possible root causes for this error. Either rule 3 and 4 are indeed redundant and should be removed, or, perhaps more likely, the "BMILevel" table is missing a rule in which a man can be overweight.

6.3.2 Background knowledge

The second type of context is the *background knowledge*: information about the domain that is not explicitly present in the model. Typically, a DMN model contains only the knowledge needed to make certain decisions in a domain. However, the domain experts may have a lot more knowledge that, even though

BMI			
U	Weight	Length	BMI
1	—	—	Weight/(Length*Length)

BMI Level			
U	BMI	Sex	BMILevel
1	< 18.5	Female	Underweight
2	< 25	Male	Underweight
3	[18.5..25]	Female	Normal
4	(25..30]	Male	Normal
5	(25..30]	Female	Overweight
6	> 30	—	Obese

Risk Level				
U	BMILevel	Sex	Waist	Risk Level
1	Normal	—	—	Low
2	Underweight	—	—	High
3	Overweight	Male	≤ 102	Increased
4	Overweight	Male	> 102	High
5	Overweight	Female	≤ 88	Increased
6	Overweight	Female	> 88	High
7	Obese	Male	≤ 102	High
8	Obese	Male	> 102	Very High
9	Obese	Female	≤ 88	High
10	Obese	Female	> 88	Very High

Figure 6.2: Decision tables defining a patient's BMILevel

it plays no direct role in the decision process, can be useful when verifying the model.

Consider for instance the decision table shown in Fig. 6.3, derived from a real-life use case at a company that develops information systems for trains. Based on information on a train and its station, the table defines a sequence ID that can then be used to decide what information should be announced in the train.

Concretely, the table has three inputs: the type of the station (major, minor, airport), the location of the station in the train's route (origin, intermediate, terminating) and the status of the train (departing, in between stations, arriving).

The following background knowledge is obvious to the domain experts:

Sequence ID				
U	Station Type	Location	Status	ID
1	—	origin	departure	s1a
2	minor	intermediate	departure	s1b
3	major	intermediate	departure	s1c
4	airport	intermediate	departure	s1d
5	—	—	in between	s2
6	—	intermediate	arrival	s3a
7	minor	terminating	arrival	s3b
8	major	terminating	arrival	s3b
9	airport	terminating	arrival	s3c

Figure 6.3: Decision table defining a sequence ID

Sequence ID: missing rules?				
U	Station Type	Location	Status	ID
1	—	terminating	departure	?
2	—	origin	arrival	?

Figure 6.4: Missing rules (?) of the table in Fig. 6.3

- If the station is the terminating station, the train's status can never be "departure".
- If the station is the station of origin, the train's status can never be "arrival".

Because of this background knowledge, some input combinations are not needed in the decision table. For instance, there is no rule for *Location = origin* and *Status = arrival*.

If the table is verified without background knowledge, it would seem to be incomplete, and a verification tool might suggest to add at least two new rules, shown in the table in Fig. 6.4. However, from a modeller's point of view, such rules of course do not make much sense.

6.4 Formal correctness criteria

In this section, we formally define correctness of decision tables, taking into account context. First, we explain the semantics of single hit decision tables

and their completeness criterion as described by Calvanese et al. [27]. Following that, we describe the criterion for unfireable rules, and extend the correctness criteria with both types of context. We will only consider tables with the U hit policy in this section for the sake of simplicity, but the semantics are trivial to extend to the A and F hit policy.

6.4.1 Decision table semantics, completeness and soundness

Calvanese et al. [27] already define a few table correctness criteria, including table completeness and soundness. We will briefly go over their formalisation, which we slightly modified to fit our semantics presented in Chapter 2.2.1.

First, recall that each cell of a decision table (i, j) corresponds to a formula $F_{ij}(x)$ in one free variable x. For example, a cell containing "< 18.5" translates to the formula "$x < 18.5$". To denote if a value x belongs to the domain of a variable, we also introduce a unary predicate $Legal_i$ for each input column i. For example, these predicates would be interpreted as follows for the *Risk Level* table in Fig. 6.2:

$$Legal_0 \in \{Normal, Underweight, Overweight, Obese\}.$$

$$Legal_1 \in \{Female, Male\}.$$

$$Legal_2 \in \{0 \ldots +\infty\}.$$

This prevents us from checking table correctness using nonsensical input values, such as BMILevel = Red, Sex = 3 and Waist = Yes.

Completeness: a decision table is complete if it contains an applicable rule for every legal configuration of input values. In other words, it should not be possible that, for a given input, no rule fires.

$$\forall \vec{x} : \left(\bigwedge_{i \in I} Legal_i(x_i) \right) \Rightarrow \bigvee_{r \in R} \left(\bigwedge_{i \in I} F_{ir}(x_i) \right)$$

with $\vec{x}$ representing a set of input values, I the number of inputs, R the number of rows, and x_i the *i-th* input value of $\vec{x}$. The above formula can be read as "For every possible set of legal input values, at least one row must match".

Soundness: a decision table with the U hit policy is sound whenever the rules of the table are mutually exclusive. As the soundness of a decision table does not change when taking context into account, we refer to the work of Calvanese et al. [27] for the formal criterion.

6.4.2 Unfireable rules

While Calvanese et al. [27] also specify a criterion for unfireable rules, they only do so for F tables. Indeed, in F tables certain rules can be *masked* by others, and thus prevented from firing. However, it is also possible (though unlikely) to have unfireable rules in both U and A tables. As such, we propose our own criterion for such rules: for every row in the decision table, there should be a legal set of input values that satisfies it.

$$\bigwedge_{r \in R} \left(\exists \vec{x} : \bigwedge_{i \in I} \left(Legal_i(x_i) \wedge F_{ir}(x_i) \right) \right)$$

with $\vec{x}$ representing a set of input values, I the number of inputs, R the number of rows, and x_i the $i-th$ input value of $\vec{x}$.

Note that this criterion will only rarely be violated, namely only if a condition rules out all possible values for an input variable, such as a condition "< 0" for a natural number variable.

6.4.3 Correctness criteria with context

Before extending the correctness criteria of completeness and unfireable rules with context, we elaborate on both types of context. In-model context consists of all tables that are not the table currently being verified.

Background knowledge, in the form of FO($\cdot$) formulae, is contained in the knowledge base *KB*. For example, the knowledge base of the *Sequence ID* example given in Section 6.3.2 consists of the following two implications:

$$Location = terminating \Rightarrow Status \neq departure.$$

$$Location = origin \Rightarrow Status \neq arrival.$$

The "variables" of DMN are represented by constants (e.g., *Location* and *Station*) in the FOL KB and in the FOL semantics Sem_T of a DMN table. In order to define our context-sensitive correctness criteria, however, we will need to be able to quantify over the DMN variables. Therefore, in the correctness criteria, the DMN variables will need to be represented by FOL variables. Let $\vec{V}$ be the set of all FOL constants that correspond to the DMN variables of the model. We introduce for each $V \in \vec{V}$ a unique FOL variable x_V and define $\vec{x} = (x_V)_{V \in \vec{V}}$ and denote by $\phi[\vec{x}]$ the result of replacing each constant V in the formula ϕ by

Sequence ID		
U	Station	ID
1	city	0
2	major	1

(a)

Stop		
U	ID	Stop
1	0	No
2	1	Yes

(b)

Figure 6.5: Example incorrect DMN tables

the variable x_V. For example, if ϕ is the formula

$$\mathit{Location} = \mathit{terminating} \Rightarrow \mathit{Status} \neq \mathit{departure}$$

then $\phi[\vec{x}]$ is the formula

$$x_1 = \mathit{terminating} \Rightarrow x_2 \neq \mathit{departure}$$

where $x_1 = x_{Location}$ and $x_2 = x_{Status}$. We now extend the completeness and unfireable rules criteria with both types of context. We use $Sem_T(V_1, \ldots, V_{m+n})$ to represent the table semantics as outlined in Chapter 2.2.1, with V_i representing the DMN variable in the heading of the ith table column and m/n respectively representing the number of input and output columns.

Completeness. When verifying the completeness criterion for a table T_j, we require that the set of variable values $\vec{x}$ satisfies all other tables T_i, $i \neq j$ and the background knowledge in the KB. The completeness property for table T_j with input variables $W_1, \ldots, W_m$ and rows R is defined as:

$$\forall \vec{x} : \bigwedge_{i \neq j} Sem_{T_i}[\vec{x}] \wedge KB[\vec{x}] \wedge \bigwedge_{k \in 1..m} Legal_i(x_{W_k})$$

$$\Rightarrow \bigvee_{r \in R} \bigwedge_{k \in 1..m} F_k r(x_{W_k}).$$

For example, consider the completeness verification of the table in Fig. 6.5a, with variables for *Station*, *ID* and *Stop*.

$$\forall p,q,r : \Big((q = 0 \Rightarrow r = \mathit{No}) \wedge (q = 1 \Rightarrow r = \mathit{Yes}) \Big)$$

$$\wedge \mathit{Legal}_p(\mathit{minor}) \wedge \mathit{Legal}_{ID}(q) \wedge \mathit{Legal}_{Stop}(r)$$

$$\Rightarrow (p = \mathit{major} \vee p = \mathit{city}))$$

We can conclude that the table is incomplete, as there is no row for *Station* = *minor* or for *Station* = *airport*.

Unfireable rules. We extend the unfireable criterion in a similar manner to the completeness criterion, by adding the KB together with the other tables. In other words, for every row of table T_j, there should be a set of variable assignments $\vec{x}$ that satisfies all other tables, the KB, and the inputs of the row itself.

$$\bigwedge_{r \in R} \Bigg(\exists \vec{x} : \bigwedge_{i \neq j} Sem_{T_i}[\vec{x}] \wedge KB[\vec{x}] \\ \wedge \bigwedge_{k \in 1..m} Legal_i(x_{W_k}) \wedge \bigwedge_{i \in I} \Big(F_{ir}(x_i) \Big) \Bigg)$$

For example, consider the verification of the first rule of the table in Fig. 6.5b, using variables for *Station* and *ID*.

$$\exists p, q : (p = city \Rightarrow q = 0) \wedge (p = major \Rightarrow q = 1) \\ \wedge Legal_{Station}(p) \wedge Legal_{ID}(q) \wedge q = 0$$

Because the *city* is not a legal value for the variable *Station*, q can never be 0. As such, this first rule is unfireable.

6.5 Implementation

To show the practical applicability of this work, we have created a tool capable of decision table verification with context. Concretely, each of the three verification capabilities has been implemented using IDP-Z3. As will be shown later on, this separation between knowledge and its use facilitates the reuse of the same knowledge for different purposes. To reason on the KB, three of IDP-Z3's inference methods are used in this work: propagation, model expansion and abstract model expansion.

To verify a table j, we convert the DMN model into an FO($\cdot$) theory that consists of $KB \cup \bigcup_{i \neq j} Sem_{T_i}$ (both types of context) and the formula

$$\bigwedge_{r \in R} Row(r) \Leftrightarrow \bigwedge_{i \in I} F_i r(H_i)$$

to represent table j, with *Row* a new predicate to represent if a row has fired. We will now go over every verification capability and explain how IDP-Z3 can perform them.

Completeness To verify a table's completeness, we look for a set of assignments for which no row fires. If the IDP system is then unable to find any solution, we can conclude that the table is complete. We perform this check by adding a constraint to the KB stating that no row is allowed to fire.

$$\forall r : \neg Row(r).$$

We then run IDP-Z3's *model expansion* inference to find a structure that satisfies the adapted KB theory T'. Because this KB includes the representations of the other tables and the background knowledge in the form of FO($\cdot$) formulae, we are effectively able to verify completeness w.r.t. both types of context. If no solution can be found by model expanding, the table is complete.

One weakness of this verification is that, in the case of an incomplete table, it is not easy to pin-point the exact gaps in its rules. We cannot use model expansion to generate all value assignments for which no rule is applicable, because there can be an infinite number of them when reasoning with integers and floats. However, we overcome this issue by using IDP-Z3's Abstract Model Generation (AMG, see Section 1.3.2). Indeed, this inference allows us to find the gaps in a decision table, with every abstract model representing a different gap in the rules.

Soundness. To detect overlapping rules, we use IDP-Z3 to find a set of value assignments that results in multiple rules of a table firing. Similarly to table completeness, we can do so by adding a new constraint:

$$\#\{r : Row(r)\} > 1.$$

This constraint can be read as "The number of rows for which *Row* is satisfied should be greater than 1". If model expansion is then incapable of finding a solution (i.e., no two rows can fire at the same time), the table is sound. Else, the solution will contain which specific rows overlap, allowing us to present this information.

Unfireable rules. IDP-Z3's propagation inference task is used to detect unfireable rules. If the propagation derives that an atom of the form $Row(i)$ must be False, we know that row i cannot fire. If it does not derive any atoms of this form, the table is free of unfireable rules.

All of these described table verification algorithms have been implemented in DMN-IDP. It is available for demonstration online[1], and includes all the examples discussed in this thesis. Internally, the conversion from DMN to FO(·) is done using the cDMN solver, which has been extended to generate the FO(·) representation for table verification. The specific version of the IDP system used in this work is IDP-Z3 [32].

6.6 Comparison and Evaluation

In this section, we give a brief comparison between our tool, the verification algorithms described by Calvanese et al. [29] and the state-of-the-art verification tool by Hasic et al. [67]. When compared to the approach by Calvanese et al., our work distinguishes itself in two ways. Firstly, our completeness verification does not only return a boolean output, but is also capable of identifying the missing rules. Similarly, our overlap detection and unfireable rule detection are also able to pin-point the specific rules causing the error. Secondly, we have created a concrete implementation of our verification methods, which has also been integrated in a DMN tool. In this way, we show that our approach is also practically feasible.

On the language level, the description logic $\mathcal{ALC}$ used by Calvanese et al. can be seen as a fragment of the FO(·) language, making our approach at least as expressive in theory. However, because FO(·) is not decidable (unlike $\mathcal{ALC}$), the IDP system can actually only reason with theories that obey certain restrictions on the domains over which the variables range. Variables that range over a finite domain are not a problem, and neither are certain uses of numerical variables within infinite domains. In particular, the theories that result from translating DMN variables with an infinite numerical domain can be handled. However, in general, IDP-Z3 cannot perform all of the Open World reasoning of typical description logic reasoning engines over infinite domains. The relation between FO(·), IDP and description logics has been further described in [137].

To compare our implementation to the one by Hasic et al., we used both tools to verify the DMN models in Fig. 6.2 and Fig. 6.3. In the first model (3 tables, 17 rows, 4 inputs), as mentioned in Section 6.3.1, the *Risk Level* table contains two unfireable rules. The tool by Hasic et al. is unable to detect this. Our tool on the other hand is able to correctly identify the two unfireable rules, and considers the table error-free after their removal. By contrast, the tool by Hasic et al. generates a false positive for the fixed table, reporting missing rules for the cases that cannot occur.

[1] `https://dmn-idp.onrender.com/`

Table 6.3: Comparison between table verification time in milliseconds.

	BMI (Risk Level)	Train sequence
Hasic et al. [67]	118	97
Our tool	1245	287

The second example (1 table, 9 rows, 3 inputs), as described in Section 6.3.2, is considered to be incomplete by the tool of Hasic et al., which states that it is missing the rules listed in Fig. 6.4. In contrast, our tool is able to use the background knowledge and conclude that this table is indeed correct.

The increased functionality of our tool comes at a computational cost, however. As shown in Table 6.3, the verification time of our tool is a magnitude higher compared to the tool by Hasic et al. when verifying the examples given in this chapter. These timings were measured as the time it took for the servers to respond with the verification results. The reasons for the difference in efficiency are twofold: firstly, by keeping in mind the context of a table, we increase the verification complexity, as more information has to be verified. Secondly, we employ a general logic-based solver for our verification instead of highly optimized procedures.

6.7 Conclusion

Most state-of-the-art verification DMN verification algorithms tend to verify a table "in isolation", without regards to the rest of the model or to background knowledge.

In this chapter, we have first explained the importance of context, extended the formal correctness criteria, and then proceeded to present a *context-aware* DMN verification tool. Instead of verifying isolated tables, this tool always keeps the other tables in mind. On top of this, the tool allows the addition of background knowledge in the form of FO($\cdot$), thus ensuring a more correct verification.

As we discussed in Section 6.6, our tool offers more functionality than the context-aware methods of Hasic et al. and Calvanese et al. The verification tool has also been implemented as part of an existing DMN editor, allowing anyone to freely test it.

Chapter 7

Adhesive Selector

This chapter presents a use case in collaboration with the Flanders Make Joining Materials Lab. Selecting the correct adhesive for a gluing task is an arduous and time-consuming process. The goal of this chapter is to build a knowledge-based decision support system which can guide the adhesive experts towards correct adhesives. Through a thorough user study, we then gauge the opinions of Flanders Make's experts on our approach. These opinions are a valuable source of knowledge for our further development.

This chapter is based on work presented at the 32nd CIRP Design Conference, March 2022 [72]; the 16th International Conference on Logic Programming and Non-Monotonic Reasoning, September 2022 [129]; and an extended version has been submitted to Theory and Practice of Logic Programming. The research was performed in collaboration with Jeroen Jordens, Maarten Witters and Bart Van Doninck, our Flanders Make partners.

7.1 Introduction

The Flanders Make Joining & Materials Lab (FM JML) is specialized in adhesive bonding. They support companies in selecting the most appropriate adhesive for a specific use case, by accounting for characteristics such as strength, temperature resistances, adhesive durability, and more. Currently, this is done manually by one of the *adhesive experts* that work at the lab. Selecting a suitable adhesive is a time-consuming and labor intensive task, due to the large number of adhesives available on the market, each with extensive data sheets. At the moment, the experts do not use any supporting tools to help them perform the selection, because the current generation of tools does not meet their requirements.

This paper describes our work on a logic-based tool which supports the experts in the selection process. It is structured as follows. We start by describing the process of selecting an adhesive and the state-of-the-art tools in Section 7.2. Next, we present our Adhesive Selector Tool in Section 7.3, where we discuss the process of Knowledge Acquisition, how the system handles unknown parameter values, and how the experts interface with the knowledge. We share the results of our preliminary three-fold validation in Section 7.4 and the results of our comprehensive user study in Section 7.5. Finally, we describe our lessons learned in Section 7.6 and conclude in Section 7.7.

7.2 Adhesive Selection and Current Tools

As there is no universally applicable adhesive, the selection of an adhesive is an important process. There are many factors that influence the choice of an adhesive: structural requirements such as bonding strength and maximum elongation, environmental factors such as temperature and humidity, economic factors, and more. Due to the complexity of the problem, there is quite a potential for tools that support this selection process. Yet, Ewen [54] concludes that "there is a severe shortage of selection software, which is perplexing especially when the task of adhesive selection is so important".

Currently, when tasked with a use case, the experts work in two steps. First, they try to identify requirements, such as temperature ranges or values for parameters like minimum strength. Based on this list of requirements, they perform an initial selection by manually looking through various data sheets while keeping track of which adhesives are suitable. In the second step, the inititally chosen adhesives are put to the test by performing real-life experiments in FM's lab, to ensure suitability. However, this testing step is costly and time-

consuming, so it is important that the initial selection is as precise as possible. While there are tools available for this process, the FM experts do not use them because they are either too simplistic, or not sufficiently flexible.

The most straightforward selection tools are websites offering simple interfaces[1] [39]. Based on a series of questions, they provide advice to support selection. However, they still require the expert to look up and process the information themselves.

There are also a number of expert systems to be found in the literature [75, 80, 84, 95, 97, 108, 118]. Here, domain knowledge is captured and formalized in the form of rules, which can be used for adhesive selection by forward chaining and often also for generating explanations by backward chaining. However, these systems have a number of downsides: they are low in both interpretability and maintainability by the expert, often not all required knowledge can be expressed, and they generally only contain a low number of adhesives or substrates. Finally, forward and backward chaining are not capable of providing all the functionality the expert needs. For instance, a situation might arise in which an adhesive is already pre-defined (e.g., left-over from a previous gluing operation), and the selection of a second substrate is required. While this selection requires the same knowledge, the expert tools are not capable of performing this operation.

7.3 Adhesive Selector Tool

This section outlines the creation and usage of the tool, and the main challenges that were faced in that process.

7.3.1 Knowledge Acquisition

The creation of the knowledge base is an important element in the development process of knowledge-based tools. It requires performing knowledge acquisition, which is traditionally the most difficult step, as the knowledge about the problem domain needs to be extracted from the domain expert to be formalized by the knowledge engineer. Here, we applied the Joint Interactive Modelling method (see Chapter 1). As a refresher, the central principle of this method is to formalize knowledge in a common notation for both domain expert and knowledge engineer, so that both sides actively participate in the formalization process.

[1] such as `www.adhesivestoolkit.com` and `www.adhesives.org`

We started by organizing three knowledge articulation workshops, each lasting between three and four hours. Each of these workshops was held with a group of domain experts. While typically a single domain expert would suffice for knowledge extraction, having a group present helps as an initial form of knowledge validation, as the experts discuss their personal way of working amongst themselves, before coming to a consensus. For the common notation we used the cDMN (see Chapter 3).

The first workshop consisted of identifying all relevant adhesive selection parameters and using them to create an initial DRD, of which a fragment is shown in Fig. 7.1. It is structured in a bottom-to-top way, similar to how the experts would reason: they start by calculating the thermal expansions, and then work their way up to the calculation of the maximum stress.

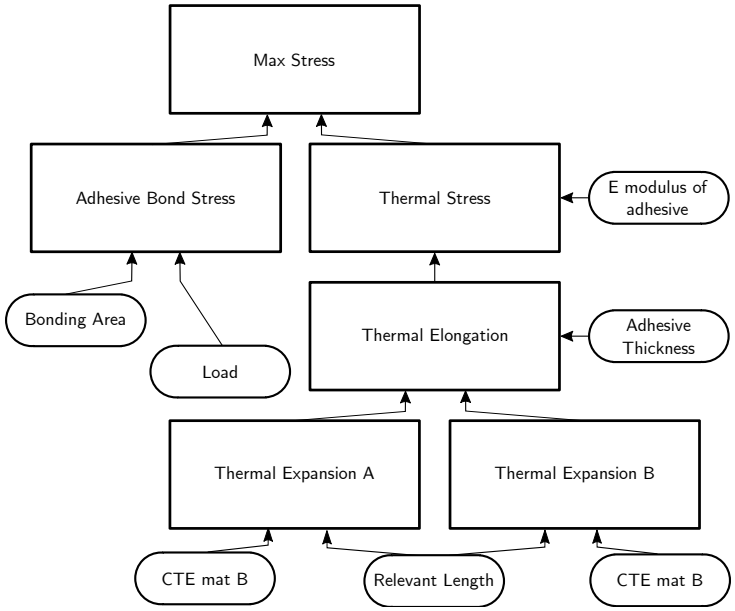


Figure 7.1: Snippet of the created DRD.

During subsequent workshops, the rest of the model was fleshed out. This consists of *decision tables* and *constraint tables*. As an example, Fig. 7.2 shows one of the decision tables (Fig. 7.2a) and one of the constraint tables (Fig. 7.2b) that were generated in the workshops.

After these three initial workshops, the cDMN model was converted into an FO(·) KB using the cDMN solver. Since then, multiple one-on-one workshops were held between the knowledge engineer and the primary domain expert to

MinElongation		
U	Support	MinElongation
1	free	0.5 × deltaLength / BondThickness
2	fixed	deltaLength / BondThickness

(a) Calculation of MinElongation

BondStrength constraint		
E*	Strength is Known	Max Stress
1	Yes	≥ MinStrength

(b) BondStrength constraint table.

Figure 7.2: Example cDMN tables

further fine-tune the KB. Among others things, this included adding a list of adhesives, substrates, and their relevant parameter values, and further validating the knowledge. In total, the current version of the KB contains information on 55 adhesives and 31 substrates. For the adhesives, the KB contains 21 adhesive parameters, such as temperature resistances, strength and maximum elongation. Similarly, it contains 11 parameters for the substrates, such as their water absorption and their solvent resistance. These parameters are a mix of discrete and continuous: in total, 15 are continuous, and 17 are discrete.

7.3.2 Unknown adhesive parameters

One of the main challenges in the formalization of the KB was handling unknown adhesive data. Indeed, often an adhesive's data sheet does not list all of its properties. This raises the question of how the tool should deal with this: should the adhesive be excluded, or should it simply ignore the constraints that mention unknown properties? Together with the experts we agreed on a third approach, in which we first look at the adhesive's family. Each adhesive belongs to one of 18 families, for which often some indicative parameter values are known. Whenever an adhesive's parameter is unknown, we use its family's value as an approximation. If the family's value is also unknown, then the constraint is ignored. This best corresponds to how the experts typically work.

This way of reasoning is formalized in the KB. For example, the constraint that an adhesive should have a minimum required bonding strength is written as

follows:

$$\forall p \in param : Known(p) \Leftrightarrow (KnownAdh(p) \vee KnownFam(p))$$

$$KnownAdh(strength) \Rightarrow BondStrength = StrengthAdh(Adhesive).$$

$$\neg KnownAdh(strength) \Rightarrow BondStrength = StrengthFam(Family(Adhesive)).$$

$$Known(strength) \Rightarrow BondStrength \geq MinBondStrength.$$

with *StrAdh* and *StrFam* representing respectively the specific adhesive's and its family's bonding strength. This approach is used for all 21 adhesive parameters.

One caveat to this approach is that IDP-Z3 currently does not support partial functions, i.e., all functions must be totally defined. To overcome this, we assign the value *-1000* to unknown parameter values, and define that the value is only *known* if it is different from this number. We chose -1000 as there is no adhesive parameter for which it is a realistic value.

$$\forall p \in param : KnownAdh(p) \Leftrightarrow StrengthAdh(Adhesive) \neq -1000.$$

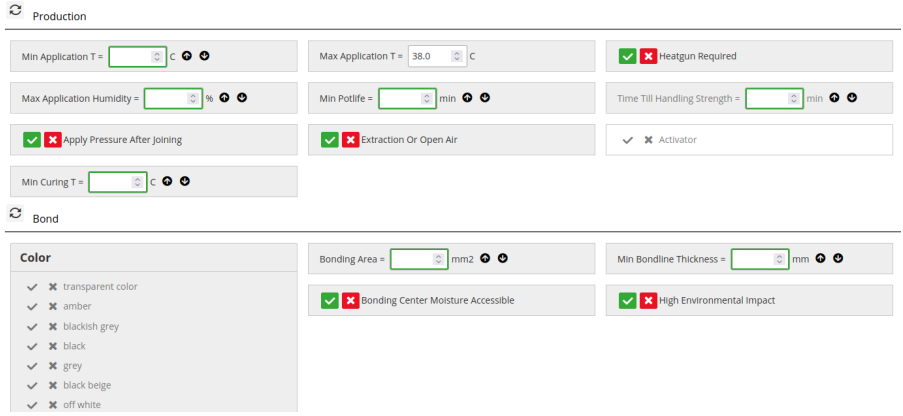
$$\forall p \in param : KnownFam(p) \Leftrightarrow StrengthFam(Adhesive) \neq -1000.$$

7.3.3 Interface

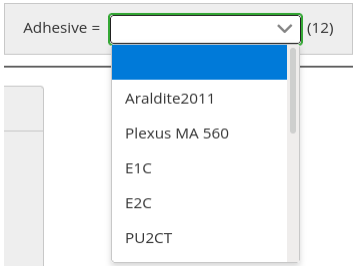
A crucial requirement of this application is the ability to interactively explore the search space. To this end, our tool integrates the Interactive Consultant to facilitate interaction with the KB. This interface makes use of several functionalities of the IDP system to make interactive exploration possible: the **propagation** inference algorithm is used to show the consequences of each choice, the **explain** inference is used to help the user understand why certain propagations were made, the **optimize** inference is used to compute the best adhesive that matches all of the choices made so far.

When using the interface, the user fills in symbol tiles, each representing a different symbol of the KB, and the system each time computes the consequences. For example, Fig. 7.3a shows a segment of the interface in which a user set a maximum application temperature of $38°C$ as a requirement. To make it easier to navigate the symbol tiles, they are all divided in five *categories*: Performance, Production, Bond, Substrate A and Substrate B. In the top-right of the interface, the number of adhesives that remain feasible is shown: e.g., after setting the temperature constraint, it drops from 55 to 12, as shown in Fig. 7.3b.

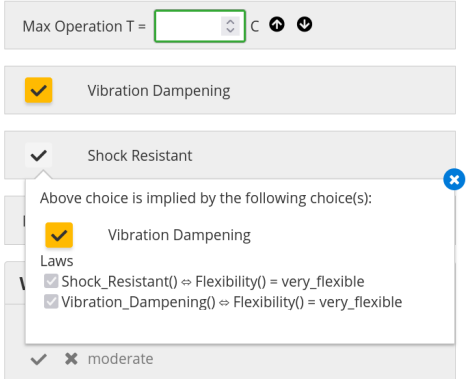
The tool is also capable of generating two types of explanations. Firstly, if the user does not understand why a certain value was propagated, they can click



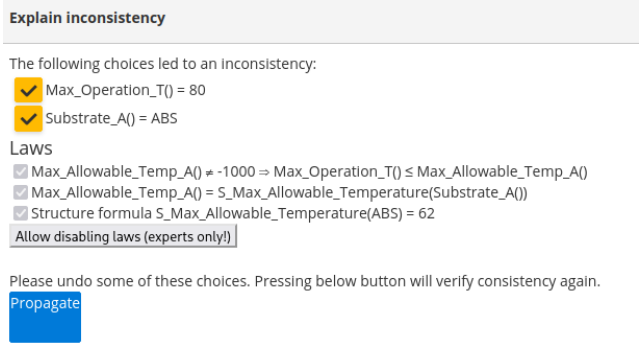
(a) Some of the symbol tiles available in the interface, divided into two categories.



(b) List of remaining suitable adhesives during selection.



(c) Example of explanation.



(d) Inconsistency window.

Figure 7.3: Screenshots of the interface.

on that value to receive a clarification, as demonstrated in Fig. 7.3c. Secondly, if the user manages to reach an inconsistent state, the tool will try to help resolving the issue by listing what is causing it. For example, Fig. 7.3d shows an inconsistency in which a substrate is selected that can not handle the required operating temperature.

Besides generating a list of all the adhesives that meet certain requirements, the tool can also find the optimal adhesive according to a specific criterion, such as lowest price or highest strength.

7.4 Preliminary Validation

Initially, we performed three types of validation for this tool: a benchmark to measure the efficiency, a survey to measure the opinion of the adhesive experts and a discussion with the Flanders Make AI project lead.

Benchmark In an initial benchmark, an adhesive expert was tasked with finding a suitable adhesive for an industrial use case which the company received. In total, it took the expert about three hours to find such an adhesive, after delving through multiple data sheets. We then used our tool for the same use case, and were able to find the exact same adhesive within three minutes. Interestingly, the reasoning of the tool closely mimicked that of the expert: for example, they both excluded specific families for the same reasons.

Survey After a demonstration of the tool to four adhesive experts, we asked them to fill out a short quantitative survey to better gauge their opinion. Their answers can be summarized as follows.

- The experts find the tool most useful for finding an initial list of adhesives to start performance testing with.
- The tool will be most useful for newer, less knowledgeable members of the lab. They can use the tool to learn more about the specifics of adhesive selection, or to verify if their result is correct.
- However, it is also useful for senior experts as they can discover adhesives which they have not yet used before.

The main criticism of the tool given by the experts is that more adhesives should be added, to make the selection more complete.

Project Lead Discussion As part of a discussion with Flanders Make's project lead, who oversees multiple AI-related projects, they outlined their perception of our tool. They see many advantages. Firstly, as there is not much data available on the process of adhesive selection (e.g., previous use cases and the selected adhesives), and data generation is quite expensive, data-based approaches are not feasible. Therefore, building a tool based on a formalization of the knowledge they already have is very interesting. Secondly, by "storing" the expert knowledge formally in a KB they can retain this information, even when experts leave the company. Thirdly, having a formal representation also makes the selection process more uniform across different experts, who typically use different heuristics or rules-of-thumb. Lastly, they indicated that there is trust in the system, because the knowledge it contains is tangible. This makes it more likely that the experts will agree with the outcome of the tool.

The project lead also expressed that there is potential to maintain and extend this tool themselves, which would be a significant advantage compared to their other AI systems. However, we currently have not yet focused on this aspect.

7.5 User Study

On top of the preliminary validation presented in Section 7.4, we performed a user study with members of the Flanders Make JML group. This study distinguishes itself from the preliminary validation in two ways: (1) all experts actually got to use the tool themselves and (2) it is a qualitative study instead of quantitative w.r.t. the input of the experts. Our main motivation for this study is to thoroughly validate the Adhesive Selector, with the following goals in mind:

1. Gauge the tool's effectiveness in a real-life setting.
2. Observe how new users interact with the tool.
3. Get feedback on the different aspects of the tool (interactivity, interface, explanations, ...).

For this validation, we asked the FM JML members to work out two real-life use cases, after which we performed one-on-one semi-structured interviews (SSI). In the following subsections we first elaborate on our methodology, then discuss the results of our interviews followed by describing the limitations of our approach.

7.5.1 Methodology

In our study, we held interviews with four members of the FM JML, who each possess knowledge on adhesives but have varying degrees of involvement in adhesive selection. Two interviewees are adhesive experts that often perform adhesive selection. The other two do not perform adhesive selection as part of their job, but are knowledgeable on glues in general. We included such "non-experts" to explore whether the tool is capable of making adhesive selection more accessible. We held an online one-on-one session with each of the interviewees. First the interviewee was asked to perform adhesive selection using the tool, and then we conducted a semi-structured interview to gather their opinions.

Adhesive selection. Two real-life bonding cases were selected as test cases for the study: both are cases that the JML lab has received from companies in the past. In the first one, a plastic door needs to be glued to the body of an industrial harvester. Originally, the process of finding the correct glue took weeks, as the requirements are fairly tight. Moreover, the specification contains an inconsistency in which a higher temperature is required than allowed by the substrate, which is the same inconsistency as shown in Fig. 7.3d. The second case details a join between a plastic component and an aluminum body, and is less challenging.

Both cases were presented to the participants as a short description of the gluing operation, together with a table containing the actual requirements. For the first example, the table lists requirements such as "Material A = Virgin ABS", "Gap filling of min 1 mm", "Application between $15°C$ to $35°C$", etc. We have taken care here to specify requirements in the same terminology as the tool. In total, the use cases consist of nine and seven requirements respectively.

Before letting the experts work on the cases, we also gave a brief introduction on how to use the tool. We explained how to enter information, where they could find the list of possible adhesives and how to see that the tool was performing calculations.

We observed each tester during the selection process, to gather information on how they interacted with the tool. To better understand their thought process, the interviewees were encouraged to talk to themselves about their process as a way to elicitate their thoughts. We did not intervene while they were working out these use cases, even when errors were made, and only made suggestions on what to do if they got stuck or an unexpected bug popped up.

Interviews. Right after the selection, we held an interview with each participant. Because our goal is to explore the opinions of the JML members, we opted for a semi-structured interview set-up, for which we prepared four main questions to serve as a general guide:

1. Do you see a role for the tool in your job?
2. Do you feel that you understand what the tool does when you use it?
3. What was your experience working with the tool?
4. How would you compare our tool to the ones that you are used to working with?

In addition to these main questions, we asked additional questions to zoom in on specific aspects of the answers given by the participants. The interviews were audio-recorded and transcribed, so that they could be analysed thoroughly.

After transcribing the recordings, we followed the guidelines of Richards and Hemphill [111] and performed *open coding* followed by *axial coding* [36]. Here, the goal is to identify various *codes* that pop up during the interviews, and then further group them into several main categories. These two steps were also performed separately by an external member of the research group, after which the results were compared and adapted to mitigate biases (*consensus coding*).

7.5.2 Results

Based on the interview transcripts, we identified 26 codes in total. Table 7.1 shows an overview of the interview statistics. The last column of this table shows the *Code distribution*, calculated as the cumulative percentage of codes discovered after each interview. This is an important parameter that indicates whether *data saturation* is reached, i.e., the point at which the same themes keep recurring and new interviews would not yield new results. According to Guest et al. [65], this point is reached when the difference in code distribution between the current and previous interview is $\leq 5\%$ (i.e., less than 5% of new information was found). As the information threshold between the third and fourth interview is 4%, we conclude that we have reached data saturation. A more detailed table showing the codes per interview is included in Appendix B.

We sub-divided the codes into five main themes: Knowledge, Interactivity, Expert, Interface and Explainability. These themes, together with their codes, are visualised in the graph in Fig. 7.4. We will now briefly go over each theme and highlight the most important findings.

Table 7.1: Interview statistics

Use Case Duration (min)	Interview Duration (min)	Words	Total codes	Code distribution
31	26	2092	15	58%
55	39	2107	16	88%
66	47	3497	19	96%
41	49	3680	13	100%

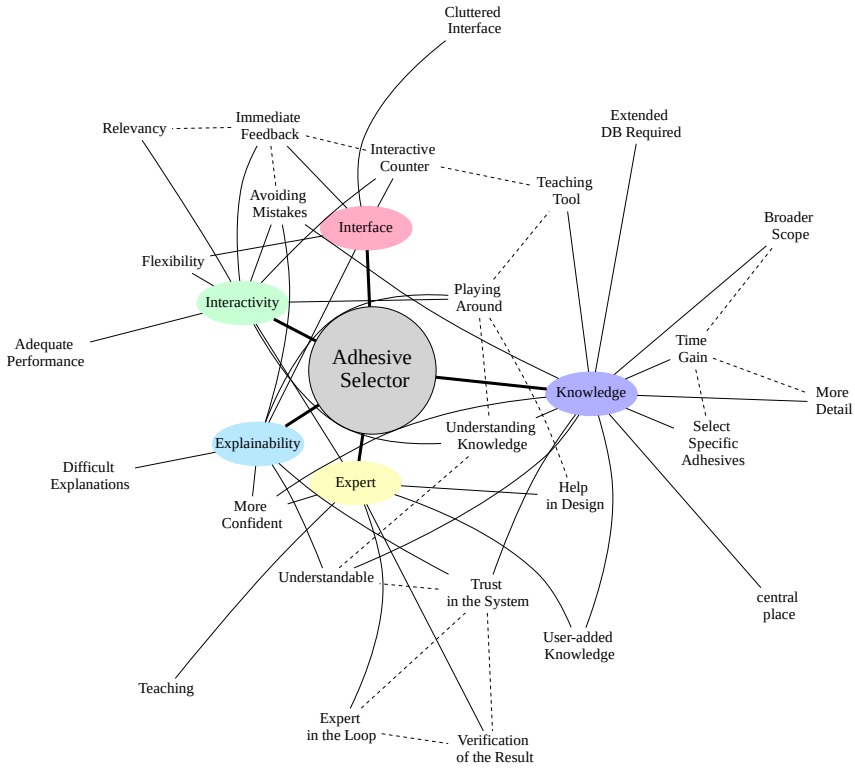


Figure 7.4: Graph showing connections between the interview themes and their codes.

Knowledge

A major advantage of our tool is that it reasons on the knowledge of the adhesive experts "like they themselves would". This is affirmed by the participants, who particularly liked its level of detail in two ways. Firstly, instead of being limited to selecting adhesive families, as is the case in most other selector systems, our tool helps them find specific adhesives. Secondly, the number of parameters available in the tool is unparalleled, allowing for a more fine-grained search.

> "[...] the good thing that I like about this tool is that it's quite detailed. It's really one or multiple steps further, multiple steps deeper than these other tools. [...] So there it's super helpful, definitely."

Similarly, the tool can also efficiently reason on a broader number of specific adhesives than an expert. As the latter typically knows some approximate parameter values by heart for a handful of glues and families, they tend to look at these adhesives first before widening their search to others when needed. Here, the Adhesive Selector can help them to find additional adhesives that they would not have considered without the tool, while also saving them from having to manually go through their data sheets.

> "You might encounter situations in which you find an adhesive that you hadn't thought about before. That quite increases your search scope, I think. It could also help in mitigating bias."

> "If you want to understand the glue that you want to use you have to read a lot of data sheets, while having this tool I think optimized the time in a way that is crazy."

The main criticism expressed by the participants is that the number of specific adhesives in the knowledge base is still rather limited. While 55 adhesives is a good start, adding more adhesives will definitely make the tool more effective. Similarly, some participants remarked that the knowledge base should contain more expert knowledge such as the environmental impact and re-usability of adhesives.

Interactivity

Throughout the interviews, interactivity was an often recurring topic. For example, one aspect that the participants all appreciated was that they could immediately see the effects of entering requirements in the interface.

> "Interactively choosing the glue now feels like online shopping – I can select more options, and see the total number of adhesives go down until I have entered all requirements."

Besides making the number of adhesives go down, each time a new requirement is entered the interface also greys out parameters that have become irrelevant, and removes parameter values that are no longer possible. This immediate feedback helps preventing mistakes in the selection.

> "I think it's easier to spot if there are some problems, like the ones that popped up."

> "It's safer, you avoid wasting materials and time."

Moreover, the immediate feedback also allows the experts to "play around" with the knowledge in the tool. In this way, they can get a feel for the effect of certain parameter values on the suitable adhesives.

> "Us engineers typically want to play with things. They want to see what happens when they change something, thereby implicitly performing a sensitivity analysis."

However, some participants felt that always having to enter the requirements one-by-one is too inefficient. While they all agreed that the tool is sufficiently fast, they stated that they would like to "bulk update" choices to be more efficient in cases where they already know that they are correct.

Expert

As our tool is designed to support JML's experts based on their own knowledge, they have played a big role in the creation of the tool and will continue to play a big role in its further development and use. It is necessary to always have an expert in the loop: the tool cannot be used by laymen, who lack the specific

knowledge required to extract requirements from a description of a use case and who cannot understand the technical jargon.

Besides supporting the experts in making suitable selections, the participants described two other ways in which the Adhesive Selector can help them. Firstly, the tool covers sufficient knowledge to also assist in the design of the entire joint. This is different from adhesive selection in that many "environmental parameters" are still left open, e.g., substrates might need to be picked, a decision on joining method needs to be made, etc.

> "This tool can already help me to list all these requirements [required for the dimensioning of the joint] so that I have a bird's eye view of the whole design."

The second additional use of the tool identified by the participants is its potential to be used as a teaching tool. For instance, newer members of the lab could use it to gain their footing when starting out.

> "You can use it to teach people 'If you select these requirements, these are the consequences, which means you can no longer use these adhesives'."

> "[The tool] can also give some confidence, if they say 'I would select this' and the tool confirms it, you would feel more certain in your selection"

Interface

Having the participants talk out loud while working the use cases proved to be a valuable source of information, allowing us to gain insight in how someone without prior experience interacts with the tool. Some of these insights are fairly minor, e.g., that we should order the values of the drop down lists alphabetically and support folding in/out entire categories to make the interface easier to navigate. Other insights are more major, such as how much difficulty the participants experienced with the structure of the interface – due to the large number of parameter tiles (78 in total), it typically took them around 10 seconds to find the right one. Moreover, the participants often lost overview of which choices they had already entered in the system, as these are spread all over the interface.

One positive note that the experts really liked was the automatically updating counter showing the remaining number of suitable adhesives:

> "I can look at the number of adhesives and see that we are converging, converging, converging. So it's fun, to say it like that."

This ties in nicely with the idea of the Adhesive Selector as a didactical tool: if a choice rules out many adhesives at once, the user can assume it is more "important" than a choice that only removes a handful. Moreover, another participant suggested using diagrams to annotate symbols with their intended meaning, e.g., a diagram showing two bonded substrates with arrows pointing to the "bond line", the "bonding surface", etc. This would make the tool more self-explanatory to people less familiar with these terms, such as the newer members of the team.

Explainability

While explainability is one of the focuses of our knowledge-based approach, the experts were not yet fully convinced of this functionality. When prompted with the "inconsistency" window (Fig. 7.3d), none of the participants knew what to do. Only two of them quickly understood what the cause of the inconsistency was, but none were able to resolve it by themselves. As one participant later remarked in an interview:

> "There were multiple sentences below each other, I didn't know if it was three remarks or a single one. [...] I was confused, and could not see the information I needed"

In other words, they had some difficulties navigating the inconsistency window: partly due to its layout, but also due to the complexity of the knowledge. However, they did appreciate the potential that this feature holds, for example to assist in experimenting with the knowledge.

The difficult explanations also did not have an impact on the expert's trust in the system. Indeed, it seems that it is more important to know that its behaviour is derived from the lab's own knowledge, than to actually 100% understand the explanations.

> "I trust it, because it contains our knowledge. So, well, I do trust it, but only because I know it's ours."

Other Results A participant with a slightly more business background pointed out an interesting result that could not be codified under the other themes. They see the tool as a "uniform way of collecting data relevant to adhesive selection", which could help drive the team's decision making. Examples of relevant information are the use cases performed by experts, the substrates that were used, which adhesives are typically picked, what constraints were present, etc. This data could be useful for:

- finding trainings specifically for the most commonly picked adhesives
- identifying target industries that could also be helped by JML
- deciding which adhesives to keep in stock, and what equipment should be purchased

This is a unique take that we had not yet considered for our tool.

7.5.3 Limitations of the Study

As with any study, ours is not without its limitations. The main limitation of this study is our low number of interviewees. However, as pointed out in Section 7.5.2, we do reach data saturation when looking at the code distribution of our interviews. Therefore, we feel that the low number of interviewees does not have a major impact.

Another limitation is that all interviewees are from the same organization and would therefore have a "common" approach to selecting glues, while external experts might have a different focus that could result in additional feedback. However, as the tool has specifically been developed for use within Flanders Make JML, we believe that this feedback would be less relevant.

During the testing of the tool, we continually observed the interviewees to study their interaction with the tool. Because of this, the users might have felt pressured to work in a more "efficient" way, as a form of the Hawthorne effect [94]. In fact, one participant said so explicitly, when talking about their experience with the inconsistency window:

> "I think that if I were alone, in the lab, I would have taken more time to read the pop-up. I wanted to be a bit quick."

Because we were observing, we might have inhibited the users to truly "play around" with the tool and test it to their heart's content.

7.6 Lessons learned

Typically, knowledge acquisition is a time-consuming and difficult process. We have found that the use of a common notation such as cDMN can help facilitate this process. The use of a formal representation that the experts can also understand helps to keep them in the loop and allows them to actively participate in the formalization process. This way of working is less error-prone, as it functions as a preliminary validation of the knowledge.

After our three initial workshops, we mainly held one-on-one meetings with one of the experts to add information on the adhesives, and to further fine-tune the knowledge. This resulted in a tight feedback loop, which turned out to be a key element in our formalization. Indeed, thanks to thorough examinations of the tool by the expert, we were able to discover additional bugs in our KB. Here, the Interactive Consultant was of paramount importance: each time the KB was modified, the expert could immediately play around with it using the generic interface. In this way, the knowledge validation of the tool could happen immediately after the modifications, allowing for a swifter detection of any errors.

Having knowledge in a declarative format, independent of how it will be used, has multiple advantages. To begin with, it allows using the knowledge for multiple purposes, even when this initially might not seem useful. Furthermore, it increases the experts' trust in the system, as it reasons on the same knowledge as they do, and is interpretable.

The main advantage of using IDP-Z3 does not lie in any one of its inference algorithms, but rather in the fact that it allows all of the functionalities that are required for interactive exploration of the search space to be performed by applying different inference algorithms to a single knowledge-base.

The validation of the tool by the actual end-users proved to be a source of valuable feedback. Through our observations, we have gained insights on how the users interact with the tool. By means of semi-structured interviews, we gathered their opinions, thoughts and suggestions. These two sources of input combined will help shape further development of our tool.

7.7 Conclusions and Future Work

This chapter presents the Adhesive Selector, a tool to support adhesive selection using a knowledge-based approach. The Knowledge Base was constructed by conducting several workshops and one-on-one meetings, using a combination of

DMN and cDMN. Our current iteration of the tool contains sufficient knowledge to assist an expert in finding an initial list of adhesives. Compared to the state-of-the-art, it is declarative, more explainable, and more extensive. The KB is also not limited to just adhesive selection, but can also be used to perform other related tasks.

In future work, we plan on converting the entire FO(·) KB into cDMN, and evaluating its readability and maintainability from the perspective of the domain experts. Besides this, we intend to test the tool using more real-life use cases, to quantify the gain in efficiency. Additionally, we are also collaborating with an external research group to develop an AI-based tool capable of extracting adhesive information from data sheets, to efficiently add more adhesives to our KB.

Chapter 8

How do experts choose a modelling language?

Selecting an appropriate knowledge modelling formalism is typically one of the first steps an AI expert undertakes when building a knowledge-based tool. However, they do so mainly based on intuition and past experiences, and cannot straightforwardly explain their process. This chapter sets out to elucidate this tacit knowledge through qualitative interviews, based on which we can distil the expert's reasoning processes.

We would like to thank (in alphabetical order) Benjamin Callewaert, Pierre Carbonnelle, Marjolein Deryck, and Jo Devriendt for their time and contributions to the interviews of this chapter.

8.1 Introduction

Throughout the thesis, we have introduced and discussed four formalisms for representing knowledge: DMN, cDMN, pDMN and Feature Modelling. In an ideal world, this thesis would now contain a chapter comparing these formalisms, and would present a complete and comprehensive guide giving advice on what formalism to use for what types of problems. However, in the real world, this is unfortunately not possible due to a few reasons.

- As (co-)author of multiple works on these formalisms, we are inherently biased.
- This chapter would be entirely based on the opinions and experiences of a single person, which would make the output insufficiently general.
- When comparing implementations in different formalisms, a lot depends on how the problem is modelled in the first place. In other words, having multiple implementations is key here.
- A perfect categorisation of formalisms and problems is unattainable due to the highly complex nature of such an undertaking. Indeed, such a categorisation would either be too specific and incomprehensible, or too high-level and incomplete. In either case, it would be unusable[1].

Instead, as an alternative, this chapter sets out to distil the experiences and opinions of multiple modelling experts through qualitative interviews. By giving them four problems to model in their formalism of choice and interviewing them afterwards, we can take a closer look at their reasoning: what formalism(s) did they use, what design choices were made, how user-friendly do they estimate their solution is, and more. The overall goals of this chapter are therefore as follows:

- Get a grasp on the instincts of expert modellers w.r.t. formalisms.
- Condense the modelling knowledge of multiple experts.
- Find out what makes a formalism *user-friendly*.

As IDP-Z3 is the common denominator of all chapters in this work, we have specifically selected experts that are well-versed in FO(·). In total, four such experts have been involved in this interview, who combined used FO(·), DMN,

[1] "Reality is captured in the categorical nets of Language only at the expense of fatal distortion" – Friedrich Nietzsche

cDMN, Feature Modelling and Controlled Natural Language (CNL). This latter formalism has not been introduced yet, and is the focus of the next section. Next, we introduce the methodology of this research in Section 8.3, and we discuss the results of the interviews in Section 8.4. We briefly touch on the limitations of this work in Section 8.5, and finish by concluding in Section 8.6.

8.2 Preliminary: Controlled Natural Language

Kuhn [79] defines CNL as follows: "A controlled natural language is a constructed language that is based on a certain natural language, being more restrictive concerning lexicon, syntax, and/or semantics, while preserving most of its natural properties." The main idea of CNLs is increase the precision of a natural language by restricting it in some way. There are two variants of CNL: "human-oriented" (to streamline communication between human parties) and "computer-oriented" (to be directly interpretable by computers). For the purpose of this chapter, we will only consider the latter.

8.2.1 CNL for KRR

In the context of KRR, research in CNL aims to bridge the gap between natural language and formal language. The idea is that a carefully designed CNL could combine the benefits of both approaches: excel in user-friendliness and expressiveness, while at the same time being precisely defined to support automated reasoning. When designing such a CNL, there are two main approaches, namely general languages and domain-specific languages.

General languages are designed without an application domain in mind – they have a very large lexicon to try and be as generic as possible. Examples of such languages are Attempto Controlled English (ACE) [57] and Processable English (PENG) [139], both of which can be translated to FOL. The latter also supports direct translation to ASP [113].

Domain-specific languages, as the name implies, are designed to work in a specific problem domain. By limiting themselves to a specific domain, they are typically easier to work with – at the cost of generalizability. An example of such a language is described in [47], where a CNL was designed to describe financial assets which could then be reasoned on using IDP-Z3.

While research shows that CNLs certainly improve readability and user-friendliness, they are not a panacea solving all problems. Indeed, they themselves have a limitation known as "the writability problem" [78], which is the difficulty

of writing sentences that conform to the restrictions of the language. In this way, CNL might sometimes be more difficult to write than formal languages, as the line between correct and incorrect syntax is more vague – i.e., it is tempting to write NL sentences that do not conform to the CNL's grammar.

8.2.2 CNL in IDP-Z3

Since version 0.10.3, IDP-Z3 natively supports CNL-type terms in the theory. The idea here is that by adding NL-like alternatives for logical connectives and other constructors, KBs in IDP-Z3 can be more user-friendly. Table 8.1 shows an overview of all these CNL terms and their FO(·) equivalent. Note that they are merely synonyms for existing terms (i.e., "syntactic sugar"), and that they therefore do not modify the structure of formulas themselves.

The listing below shows an example of a simple FO(·) formula and its CNL counterpart. The variable names have been slightly modified in the second formula to increase the NL-like "flow" of the sentence.

```
nr_of_participants() < 3 ∧ Covid_Safe_Ticket() ⇐ activity() = Cinema.

the_nr_of_participants() is less than 3
and the_participants_have_Covid_Safe_Ticket()
    are necessary conditions for activity() is Cinema.
```

For laypeople without prior experience in logic, the second formula is indeed most likely easier to read and interpret. However, people with a basic understanding of logical connectives might prefer the top formula, as it is more concise and less cluttered. Moreover, the second formula reads quite a bit stilted due to the fact that the structure of the two formulas remains the same, making the sentence less natural from an NL point of view. Compare this to an NL description of the rule: "There must be less than 3 people and all of them must have covid safe tickets if the activity is cinema."

It is important to remember however that the CNL developments in IDP-Z3 are still relatively young, and are subject to change. An up-to-date overview of the syntax is available in the documentation[2].

[2] https://docs.idp-z3.be/en/stable/summary.html

Table 8.1: CNL terms in IDP-Z3

IDP-Z3 CNL	FO(·)
for all T x:	$\forall x \in T :$
there is a T x:	$\exists x \in T :$
p() or q()	$p() \vee q()$
p() and q()	$p() \wedge q()$
if p(), then q()	$p() \Rightarrow q()$
p() are sufficient conditions for q()	$p() \Rightarrow q()$
p() are necessary conditions for q()	$p() \Leftarrow q()$
p() if q().	$p() \leftarrow q()$
p() are necessary and sufficient conditions for q()	$p() \Leftrightarrow q()$
p() is the same as q()	$p() \Leftrightarrow q()$
x is y	$x = y$
x is not y	$x \neq y$
x is strictly less than y	$x < y$
x is less than y	$x \leq y$
x is greater than y	$x \geq y$
x is strictly greater than y	$x > y$
the sum of f(x) for each T x such that p(x)	$sum\{\{f(x) \mid x \in T : p(x)\}\}$

8.3 Interview Methodology

In this section, we will discuss our approach to the interviews. Our design choices consist of three key questions:

- **Selection:** Based on what parameters do we select the interviewees?
- **Preparation:** What do we give the experts as preparation for the interviews?
- **Questions:** What questions will we ask during the interviews?

8.3.1 Experts

The expert modellers for this chapter have been selected according to three criteria: they should have expert knowledge of FO(·), have at least two years of experience with KRR, and should have collaborated extensively with domain experts on a use case in the past. Following these criteria, we were able to identify four expert modellers, each of which are either current members

Table 8.2: Modeller confidence, with years of experience and DTAI-KR member status.
(X = full confidence, o = familiar but not expert)

Modeller	FO(·)	DMN	cDMN	FM	CNL	YoE	Member
M1	X	o	X		o	10	Ex
M2	X				X	5	Current
M3	X	X	X	o		5	Ex
M4	X	o		X		3	Current

Table 8.3: Problem descriptions. †Private correspondance.

Problem	Source	Problem Type
Concrete Selector	[73, 129]	Decision Support System
Phone Configurator	[17]	Configuration Problem
Vacation Days	[134]	Legislation
Planning Problem	Kylian Van Dessel†	Planning Problem

or ex-members of KULeuven's DTAI-KR research group. While this means that all four of them have expert knowledge of IDP-Z3 and FO(·), they are not necessarily experienced with the other formalisms discussed in this thesis. Table 8.2 shows an overview of the modellers and how familiar they are with the various formalisms.

8.3.2 Problem Descriptions

To prepare the experts for the interviews, we have selected four small use cases for them to model. As these use cases greatly influence the quality of the output of this chapter, we have paid extra attention to certain requirements when selecting them. In general, they should meet three requirements:

R1 "Rooted" in real-life: the problems should describe real-life applications.

R2 Diverse: there should be minimal overlap between the types of problems.

R3 New to the experts: they should not have modelled it in the past already.

To meet these requirements, we have decided to derive smaller versions of cases that have already been published in past literature. In this way, we know that the problem cases are sufficiently meaningful (R1). Moreover, we have

selected use cases from four distinct application domains, to ensure diversity in our problems (R2). While the majority of the problems will be new to the modellers (R3), one specifically might be well known to them already. To mitigate any risk we have created our own problem that closely "mirrors" the original but with different concepts. Table 8.3 shows an overview of the problems, their source and their application domains. We will now briefly go over each one and highlight the important aspects.

P1. Concrete Selector is a synthetic problem created to be similar to the Adhesive Selector use case (Chapter 7). It describes an application in which engineers must select proper mixes of concrete to meet certain criteria. To do this, they have a standard library of mixes ($\approx$ adhesive families) to which they can make small variations ($\approx$ specific adhesives). However, some parameter values are unknown for some mixes, and this should be handled with care.

P2. Phone Configurator is derived from a feature model presented by Benavides et al. [17], for which we have manually created a problem description that matches it. Here, a phone producer wishes to develop a tool capable of supporting its design staff. In essence, this is a standard configuration problem.

P3. Vacation Days is a classic DMN example, published by Jan Vanthienen via the DMCommunity website [134]. It describes how to calculate the number of vacation days for a person depending on their age and years of service. This problem is very straightforward, but can still be modelled in many different ways.

P4. Planning Problem is a simplified version of a real-life rostering problem by Kylian Van Dessel. It is a fairly standard scheduling problem, in which a first-aid officer and a back-up must be present at all times in a school. Our simplified version of the problem does not contain any soft constraints.

The modelling experts were given these problems ahead of the interview, and told to consider them "as if given by a domain expert with the goal of building an interactive knowledge-based tool". They were also given carte blanche w.r.t. the formalism and design focus when representing the problems. The full versions of the problem descriptions can be found in Appendix C.

8.3.3 Interviews

After an expert had finished modelling all four problems separately, we held an interview with them. Similar to the interview described in Section 7.5, we opted for a semi-structured interview set-up. This approach is ideal for capturing the expert's opinions, as we have preselected questions to direct the interview while maintaining sufficient freedom to ask about specific aspects. To this end, we prepared four main questions to ask for each model:

Q1. Why have you chosen this formalism, and was this your first choice?

Q2. Try to describe the elements of the use case that led to choosing this formalism. I.e., what specifically made you pick this formalism?

Q3. Here is a list of a number of different aspects. On which did you focus the most, and where do you think your model is strongest?

- Readability for you
- Readability for non-expert
- Extensibility
- Efficient execution of certain inference tasks
- Compactness
- Something else?

Q4. How long do you think it would take to teach a domain expert enough of the notation/formalism, to allow them to maintain the model without your help?

Similar to the qualitative interview in Section 7.5, we will perform *open coding* and *axial coding* [36] as per the guidelines of Richards and Hemphill [111]. In other words, we will go through the transcriptions to identify *codes*, and then aggregate these codes into overarching categories (also referred to as *themes*).

8.4 Results

Each of the experts modelled the problems in the formalism they deemed best, resulting in a total of 16 unique modellings split over five formalisms. Table 8.4 shows an overview of which formalism the experts used for each problem. While FO(·) is clearly the most represented formalism, DMN/cDMN/Feature Modelling/CNL have all been used at least once as well. This skew towards

FO(·) is partially due to the fact that the problem descriptions were merely "dummy" problems, as will be explained in our Limitations (Section 8.5). The low usage of "alternative" formalisms does not pose a problem for this work fortunately, as the point of the modelling exercises was not necessarily to receive diverse modellings but to stimulate the experts to think about their decision and design processes. In this way, they were better prepared to explain their reasoning during the interviews.

These interviews were quite information-packed: each of them resulted in transcripts of 3000 to 3500 words, as shown in Table 8.5. Through analysis of these transcripts, we identified a total of 25 codes. Similarly to the qualitative interview in the previous chapter, we have included the *code distribution*, as this is an important indicator to see if *data saturation* is reached. Unfortunately, we did not reach the threshold of $< 5\%$ new information as there was an 8% increase, meaning that the discovered codes might not be complete. However, we feel like this is not a big problem, due to the great quality of the codes we *did* get. A more detailed overview of all the codes per interview can be found in Appendix C.3, together with the *code book* explaining each code.

To group the codes together, we divided them into five *themes*: Expressiveness, Modeller, Naturalness, Tooling and User-friendliness. Fig. 8.1 shows a graph connecting these themes to their codes. In the upcoming sections, we will now discuss each of these themes and distil the most important information from the interviews.

8.4.1 Expressiveness

As introduced in Chapter 1, the expressiveness of a formalism is an important property: a good formalism should be sufficiently expressive to correctly model a problem. Therefore, the applicability of a formalism (i.e., which problems it can and cannot model) is heavily influenced by its expressiveness. In fact, this property is one of the key aspects the experts appreciate about FO(·).

> "[FO(·)] was my first choice, for a few reasons. One is that it is the most expressive one."

> "I like working in FO(·) because it can reflect many complex situations, while still remaining relatively readable."

Because FO(·) can correctly model so many problems, it is a prime candidate for any application domain. It seems this expressiveness is a double-edged sword

Table 8.4: The formalisms used by each expert.

Modeller	FO(·)	DMN	cDMN	FM	CNL
M1	P1-4				
M2	P1, P3				P2, P4
M3	P1, P4		P3	P2	
M4	P1, P4	P3		P2	

Table 8.5: Information on the interviews.

Modeller	Interview Duration (min)	Words	Total codes	Code Distribution
M1	67	3211	11	64%
M2	56	3423	12	88%
M3	40	3026	13	92%
M4	47	3323	15	100%

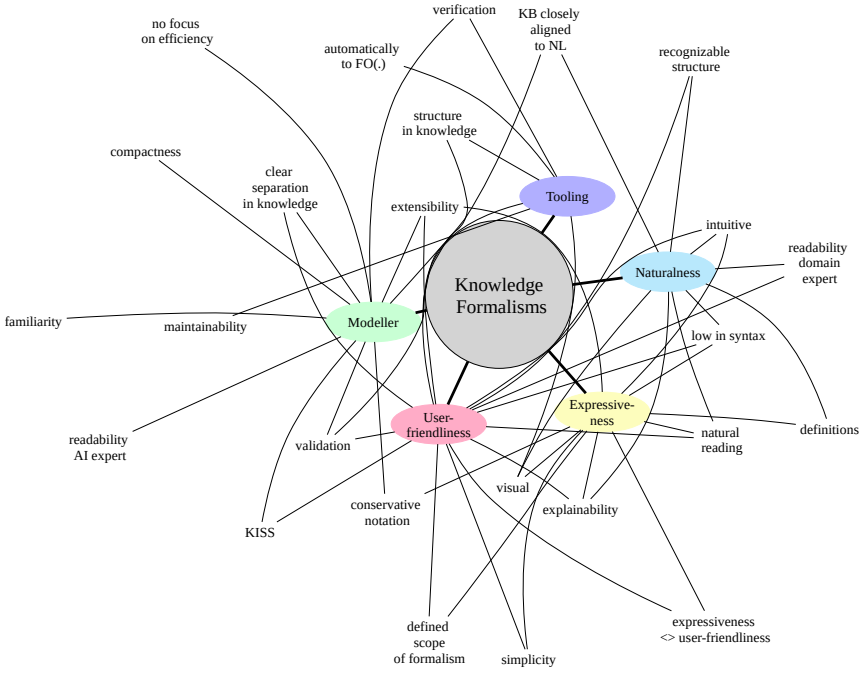


Figure 8.1: Graph showing the connections between the themes and their codes.

however, as the experts do not agree on whether FO(·) is user-friendly. This will be discussed in more details in the sections to come. However, it certainly is not well suited for Joint Interactive Modelling, unless the domain expert has been extensively trained.

> "For FO(·) they should know a lot of syntax, which is more scary than the tabular format [of (c)DMN] in which there is less expressiveness."

> "If you have a notation which can model *everything*, you will have *too much* expressiveness for many problems, which can lead to unintuitive models."

Based on the interviews, it seems like the experts have an implicit notion of "sufficient expressiveness". They use this notion to gauge whether or not a formalism can model a specific problem:

> "If the most simple notation contains all expressiveness which you need, then that will always be the one I choose."

This approach shifts the issue however: how do you *know* if a formalism is sufficiently expressive, based on the problem description? It seems like this is mostly based on modeller intuition, which they have built up through their previous experiences. While there are some clear indicators to decide a formalism's eligibility (e.g., if it contains constraints, you cannot use DMN), it in general does not seem feasible to make sound guidelines for what formalism to pick.

Some experts describe a step-wise approach, where they first started in formalism x but then shifted to formalism y after hitting an expressiveness limitation. To this end, it is useful to know which notations *subsume* each other. Take DMN for example: if you unexpectedly need to model a constraint, you can painlessly shift over to cDMN. If you then discover that cDMN also does not cut it, you can have it converted into FO(·) and continue from there.

8.4.2 Modeller

In many cases, selecting a formalism is not an exact science but rather comes down to intuition. This intuition is modeller-specific: different modellers might choose different formalisms for the same problem for different reasons. The only way to gain this intuition is to train it, for instance by implementing various

use cases or solving modelling challenges. This also means that familiarity with a language plays a big role: a modeller cannot use formalisms which they are not familiar with. While this might seem like pointing out the obvious, it is still worth mentioning explicitly. For instance, when asked why a modeller chose FO(·), they replied

> "Pragmatism, it is the paradigm I am most familiar with"

The more you train with a formalism, the easier your modelling process will be. In this regard, three out of four experts responded that "ease of use" is very important to them, and modelling should be easy and straightforward. They lean towards the KISS design principle: Keep It Simple, Stupid. When developing a knowledge base for a domain expert, they are for instance not worried about the computational performance – thanks to the KBP, this is off-loaded to IDP-Z3[3]. They are also not too worried about extensibility initially, but rather follow the Agile approach of "dealing with it when it comes up". They all agree that modelling should be as uncomplicated as possible.

> "I want to make something that is easy to make."

As the Joint Interactive Modelling approach also adds domain experts in the mix, this "ease of use" becomes even more important.

8.4.3 Naturalness

"Naturalness" was a frequently-mentioned topic in all four of the interviews. The concept came up in two ways: the naturalness of the problem representation, and the naturalness of the formalism.

The first type of naturalness is about "straightforwardly translating the problem description to the KB", in the most "pure" way possible. Here, the experts want the KB to mirror the original problem description as closely as they can, so that the original knowledge is reflected clearly.

> "I'm trying to translate the sentences as closely as possible in formal language, so that the informal reading of the sentences matches the language."

[3]Or alternative FO(·) reasoning engines, such as FOLASP [121] and ManyWorlds [49]

> "The point is really to try to make the KB as close to the NL description as possible."

> "What I usually pay attention to is a close alignment to the description. If you get a text describing a problem domain, stick to it."

This, in a sense, makes the KB more *natural*, and thus easier to interpret and work with. Moreover, it also results in a better maintainability and extensibility of the KB, as the changes to the NL description of the problem can be directly applied to the KB as well. It also loops back to the principle of KISS: if you can represent the knowledge without having to add auxiliary variables or complex statements, creating a KB will be effortless.

The second type of naturalness concerns that of the formalisms itself, i.e., how naturally a formalism lends itself to a specific type of problem.

> "Choose the formalism that is closest to the intention of the problem description."

> "If your formalism has been specifically designed to model the type of knowledge you want to model, of course it's going to be easy."

Formalisms with a pre-defined scope, such as DMN and feature modelling, have been specifically designed with a problem domain in mind (respectively business processes and configuration problems). Therefore, it makes sense that these are a great choice for those specific types of problems. In contrast, by not limiting themselves to specific problem domains, FO(·) and cDMN are more general in nature. There is a clear link with expressiveness here: a language with a limited scope will be more natural for a specific type of problem at the cost of expressiveness. Interestingly, this is the same trade-off as was described for domain-specific versus general CNL.

8.4.4 Tooling

The fourth theme discovered in this research is *Tooling*. Throughout the interviews, the importance of proper tooling became evident. For instance, the experts really appreciate the Interactive Consultant, because it allows them to efficiently validate their KB:

> "What I really like about the IC is that it is very easy to check if you are on the right path"

> "For the concrete selector I first used an equivalence instead of an implication. Then I tried three or four problems in the IC and immediately noticed that something was wrong. In that sense, the environments of the editor and the tooling is extremely useful."

These testimonies further reinforce the motivation of the Joint Interactive Modelling approach to validate the KB by frequently building prototypes, for instance by use of the IC.

Next to validation, the experts also appreciate tools for verification, such as our DMN verification (Chapter 6). Another tool for verification is FOLint [138], a linter for FO(·), developed by a master student under our supervision. This tool statically analyses FO(·) to warn about three types of errors (syntax errors, type errors & common mistakes), and gives suggestions on the style of the KB. Since IDP-Z3 v0.10.0, FOLint has also been integrated into the editor of the IC.

> "I noticed that, when I was modelling IDP-Z3, interesting suggestions [by the linter] were showing up."

While these types of tools are already useful on their own, they become even more effective when combined with others. Therefore, it is important to have such software integrated in an accessible editor. In this way, verification and validation techniques are readily available, and can effortlessly complement each other.

A good editor can also leverage the structure of a formalism to simplify modelling. For instance, both DMN-IDP (Chapter 2.3) and FM-IDP (Chapter 5.4) automatically take care of the formatting, so that the modeller can focus on the logic rather than its form.

> "A structured editor [such as in DMN-IDP and FM-IDP] guides you in modelling, which makes the process more pleasant."

In short, tooling makes the life of the modeller easier by streamlining the modelling process. In fact, proper tooling is one of the most crucial factors when selecting a notation:

> "Tooling is probably the deciding factor. It's less about "what syntax does your language have, how readable is it" or "how efficient is your engine". In the end, there will be other ways to solve your problems, and if those have better tooling, you will want to use them instead."

8.4.5 User-friendliness

The fifth and final theme from the interviews is, unsurprisingly, *user-friendliness*. This is a deceptively tricky concept, as it is difficult to state exactly what constitutes it. Though the dictionary entry for the word is quite straightforward[4], it is not obvious to see how this applies to modelling formalisms. When asked about user-friendliness, the experts typically respond with terms such as "Readability", "Easy to write", "Intuitive", and others. There does not seem to be clear consensus, however.

Take FO(·) as an example: the experts do not agree on whether or not it is user-friendly.

> "I think that if someone can learn Excel, they should be able to learn [FO(·)]."

> "FO(·) is front-loaded, you have to learn and describe everything before you can get started. There's many good ideas in it, but sometimes they do not shine because people are required to do a lot of work before they can start."

Through the interviews, we got some very mixed opinions on this topic. This again confirms that FO(·) is ill-suited for Joint Interactive Modelling, as ideally we would use a formalism that is easy to learn for laypeople. However, there is also the notion that the background of the domain expert heavily influences how quickly they can learn a formalism. It is for instance expected that a programmer, who likely has extensive experience with computational thinking, can pick up FO(·) faster than, e.g., a notary.

One thing the experts do seem to agree on is that formalisms that are visual in nature are more user-friendly for non-experts.

[4] "The fact or quality of being simple for people to use" – Cambridge Dictionary [2]

> "I found it easy to display it visually; the fact that the base knowledge was then converted for me, so I only had to add two constraints."

> "If you can easily visualise something in a structured manner, I think it will be user-friendly."

Moreover, it is also a big plus when "common structures" are used to represent knowledge. Think for instance of the way (c)DMN uses tables and feature modelling uses the tree structure. Most people will be able to recognize these structures from other domains, and will therefore already have developed a basic intuition on how to interpret them.

> "The tree-like structure is very recognizable and intuitive to understand. [(c)DMN] tables also have such a recognizable structure."

> "Basically, DMN and cDMN are just Excel."

> "If you follow the structure, and you understand it, it should be easy to learn"

Such visual notations come at a cost however, as they might become too unwieldy to use when modelling large or complex problem domains. Indeed, it can be difficult to get a clear overview of all the knowledge when there are, e.g., many tables or tree-nodes present.

8.5 Limitations

This research is not without its limitations. Firstly, all interviewees work/have worked in the same DTAI-KR research group of KU Leuven. This is unavoidable however, given our selection requirements for experts.

Secondly, the interviewees are not familiar with all five of the presupposed formalisms, but rather only with a few of them. Still, each of the interviewees has full confidence with at least two formalisms. Moreover, as pointed out earlier, the point of the dummy problems was not to get "diverse" modellings w.r.t. the formalism used, but to prepare the experts to efficiently explain their reasoning when modelling.

Lastly, the dummy cases might have been too small to really be representative of real-life problems. While this is not necessarily an issue, two out of four experts have said that this lead them to select a different formalism.

> "I found the problem so trivial that it was easier to write it in FO(·) than to use cDMN."

However, both experts took the time to elaborate on which design choices they would have made for a real-life version of the problems, which proved an equally valuable source of information for the interviews.

8.6 Conclusion

In this chapter, we have elicited the important aspects of the modelling process, based on interviews with experts. This information can guide further research on existing formalisms, the design of new formalism, modelling techniques, and more. We have categorised this information in five main themes: Expressiveness, Modeller, Naturalness, Tooling, and User-friendliness.

The *Expressiveness* of a formalism is of high importance, as it indicates which problem domains it's capable of modelling. There exists however a trade-off with user-friendliness, where a very expressive formalism might become more difficult to use. To counter this, experts have the notion of "sufficient expressiveness": based on trained intuition, they select the most user-friendly formalism that is still sufficiently expressive for the problem.

In this sense, the modelling process is very dependant on the *Modeller*. For one, they need to train to be confident enough in their knowledge of the formalism(s). Moreover, a modelling expert also needs to make certain design choices. Through the interviews, we have for instance learned that the experts take a KISS approach, where they do not worry about computational performance or extensibility but rather consider these as extras.

One of the main design focusses of the experts is *Naturalness*. Here, they wish to translate the problem description in the KB as "purely" as possible, so that the KB closely mirrors the original description. Their main motivation is that this results in a more interpretable, maintainable and extendable KB. We also discussed a formalism-specific naturalness, where a formalism with a pre-defined scope (e.g., DMN and Feature Modelling) will be well-suited for their targeted problem types.

The experts also greatly emphasized the importance of *Tooling*. Indeed, verification and validation of the KB play a crucial role in its conception. When integrated in a good editor, tools such as the IC, our DMN verification (Chapter 6) and FOLint [138] allow the experts to frequently check whether their KB is valid and correct. Additionally, such an editor can also leverage the structure of a formalism to simplify modelling, by for example taking care of the structure of decision tables or feature graphs.

The final theme is *User-friendliness*. While on the surface this seems like a straightforward concept, the experts were unable to precisely specify what makes a formalism user-friendly. For instance, they do not agree on whether or not FO($\cdot$) is user-friendly. Two properties stick out as being user-friendly however: formalisms that are visual in nature, and formalisms that leverage "common structures" to represent knowledge. For instance, Feature Modelling is very visual and uses a graph-like structure which most people can intuitively interpret correctly.

In future work, we would like to use this chapter as a springboard to more qualitative research on the topic of user-friendliness of formalisms. For instance, while this chapter considers the viewpoint of modelling experts, the viewpoint of non-AI experts w.r.t. modelling would also provide a valuable source of information. This type of research however leans more closely to the field of cognitive sciences, and could benefit from a collaboration with cognitive psychology researchers.

Chapter 9

FOLL-E: Teaching First Order Logic to Children

There is a rising awareness that we should teach computational thinking and other computer science aspects to children, e.g., through block-based programming environments. Yet, tools and support for teaching logic, a cornerstone of fields such as mathematics, computer science and philosophy, curiously remain absent. The goal of this chapter is to find possible explanations for this discrepancy, and to design an application that overcomes it.

This chapter is based on work presented at the Thirteenth AAAI Symposium on Educational Advances in Artificial Intelligence, January 2023 [132]. We thank Kaat De Bock (`https://www.kaatdebock.be`) for their robot illustrations, Davy of Makerspace De Nayer for assisting us with lasercutting, Walter Schaeken for his help with preparing the questionnaire, and Maarten Vandersteegen and Kristof Van Beeck for their technical suggestions.

9.1 Introduction

First-order logic (FOL) is an important foundation of mathematics, philosophy and computer science. Zooming in on computer science, many curricula offer students an introductory logic course, which helps students to, e.g., reason about the correctness of programs, prove properties such as termination of an algorithm, analyse requirements, etc. Also in the domain of AI, the importance of logic can hardly be exaggerated. Indeed, many of the first AI systems were based on FOL, and to this day there is a vibrant community that uses logic for AI, in fields such as Knowledge Representation, Logic Programming, Theorem Proving and SAT solving.

Many key insights from computer science and computational thinking have been successfully taught to children of various ages [15]. Probably the most spectacular example are block-based programming environments, such as Scratch (Junior) [110, 87] and Blockly [56], which are used around the world to introduce even very young children to key concepts of algorithmic thinking. Learning computational thinking at an early age may help children in making more informed decisions about their future field of study, and may provide them with useful skills, either for a career in CS or some other domain.

While Scratch is a very successful example, there exist also numerous other efforts to make important ideas from computer science accessible to children. For instance, in the field of AI, research has been done into helping children understand concept of neural networks and deep learning [107, 81]. One topic which appears to be curiously understudied, however, is the teaching of FOL to children. On the face of it, logic would seem to be a prime candidate: it is not only important in computer science and AI, but it can also help children in a variety of other disciplines. Moreover, it nicely complements the Scratch-like tools: these tools teach children to think about how algorithms work and how computers perform tasks, while logic could help children to think about requirements and about properties of problem domains. In other words, Scratch can teach children the "how", while logic could help them to figure out the "what".

The main research question of this chapter is therefore:

> Is it possible to use a "Scratch-like approach" to teach first-order logic to children?

At first sight, there seems to be a number of obvious objections, such as:

- Logic has a steep learning curve. Maybe its syntax and semantics are too complex for children to understand.
- Logic does not "do" anything. When children learn to program, especially in a graphical environment, they can immediately observe what a program does, thereby getting valuable direct feedback. Such feedback may be impossible to obtain when learning logic.
- Logic is not fun. Animations, sounds and graphics can all help to get children engaged in learning how to program. By contrast, logic may be too boring to learn.

In this chapter, we will investigate whether these objections can be overcome by designing an appropriate learning environment. First, we will clarify the concrete goals of this environment. Then, we describe our approach to achieving these goals, both conceptually and technically. Afterwards, we present FOLL-E (First-Order Logic Learning Environment), a concrete implementation of a blocks-based tool for teaching FOL to children. We also discuss our experiences and observations of using FOLL-E at two events centred around introducing CS to children. Finally, we finish by concluding, and lay out potential future work.

9.2 Concrete Goals

When designing the tool, we will attempt to achieve a number of concrete goals. The tool will have children write logic formulas in order to perform certain tasks, with the goal of teaching them certain skills. Some of our goals have to do with the format in which the logic is written, while others have to do with the task that is to be performed, and still others with the skills that the children should learn.

9.2.1 Skills

So far, we have been talking about "teaching children first-order logic". Since FOL can be used in different ways and for different purposes, we should start by specifying what it is precisely that we want children to learn. In this work, our goal is to make children understand which piece of knowledge is expressed by a FOL formula, and conversely, to be able to formalise a given piece of knowledge by means of a FOL formula. In other words, we want children to understand that a formula such as

$$\forall x : Human(x) \Rightarrow Mortal(x) \tag{9.1}$$

means that "all humans are mortal", and we want them to be able to come up with this formula as a representation of this knowledge.

Our tool will therefore have a model-theoretic rather that proof-theoretic view on FOL: we will focus on teaching the relation between a formula and its models, rather than on constructing proofs or deducing formulas. We make this choice because we view this as the more foundational skill: if the children do not understand what a formula means or how they can correctly formalise a certain statement, then deduction will not be of much use to them anyway, because they will not understand what it is that they have deduced.

9.2.2 Representation of FOL

The traditional way of writing logic in mathematics—that is, the use of formulas such as (1)—is not well-suited for our purposes. Our goal is therefore to design a way of writing FOL formulas that is more suited for getting children to understand the essence of FOL, while avoiding needless struggles with syntax. In particular, we have the following goals for our notation.

- *Avoid discouraging the children with finicky syntax.* The traditional notation requires learning several new symbols, and how to combine them correctly. Forgetting the meaning of a symbol or hitting syntax errors may discourage children, so we want to avoid this as much as possible.
- *Make the structure of formulas clear.* To know what a formula means, it is necessary to understand its structure. In traditional notation, this requires knowing the rules of operator precedence, which introduces an additional layer of complexity.
- *Encourage experimentation.* Building different formulas and figuring out their meaning is an excellent way of learning. We want a notation that encourages children to try things out.
- *Encourage collaboration.* Talking about formulas is also an excellent way of learning: trying to explain the meaning of a formula to someone else is typically a great way of increasing not only the other person's but also your own understanding.

9.2.3 Task to Solve

Our tool will ask children to construct formulas in order to solve certain tasks. In designing the tasks to be solved, we have the following goals.

- *The task should be focused on the $\models$-relation between formulas and their models.* In other words, the tool should actually be teaching the skills that we want to teach.
- *The task should allow clear and immediate feedback.* A tight feedback loop is crucial for the learning process, so if the user makes a mistake, we should be able to tell them as quickly as possible what they actually did wrong.
- *The task should allow a gradual increase in difficulty.* To create a succesful learning trajectory, we should be able to start from very simple exercises and gradually build up to more complex ones.
- *The tasks should be engaging and fun.* A fun task should help to keep children motivated throughout the learning process.

9.3 Conceptual Approach

To achieve the goals identified in the previous section, we opt for the following approach. The task that we ask the user to perform is to construct a formula that distinguishes a given set of "good" examples from a given set of "bad" examples. For instance, we could present the user with a red square, a blue square and a green square as good examples and a red circle, a blue circle and a green circle as bad examples, and the solution could then be the formula $Shape = Square$. Here, the vocabulary in which the formula is to be written (e.g., the symbols $Shape$ and $Square$), is part of the assignment. Such a task has the following benefits:

- It is focused on the model-theoretic semantics of formulas, corresponding to our goals.
- It avoids the use of natural language: if we were to ask the user to formalise a given natural language statement ("the shape is square"), the user would first have to correctly understand the statement, which may introduce problems that are related to reading comprehension.

- It introduces a puzzle element. By asking the user to figure out by themselves what the piece of knowledge is that has to be formalised, they are presented with an additional challenge, which they hopefully perceive as fun. Moreover, the puzzle element is not simply "tacked on", but it focuses on the difference between the models and non-models of a formula, which goes to the essence of what the user should learn.
- It allows gradually increasing difficulty. This general framework has enough flexibility to allow a wide range of difficulty.
- We can present the different examples as pictures to the user. By choosing a topic that is more whimsical than just coloured shapes, we can introduce a visually appealing and fun setting.
- The setting allows for an intuitive way of providing feedback to the user: for an incorrect solution, we can highlight precisely those examples that are misclassified (i.e., those that should be models of the formula but are not, and those that should not be models of the formula but are). This tells the user what they did wrong and may immediately point them in the right direction for fixing their solution.

The notation that we use for FOL-formulas is a visual, block-based notation. The success of Scratch-like tools suggests that this is a good approach, which achieves the goal of being syntax free and engages the users' visual intuitions. By carefully designing the blocks, we ensure that it is impossible for the user to create syntax or typing errors. To visualise the structure of the formulas, we lay-out the blocks in the style of a parse-tree, placing subformulas below the formulas in which they appear. Finally, the blocks can be shaped like pieces of a puzzle, which encourages users to try to fit different pieces together, thereby encouraging experimentation.

The last design choice that we make is to allow the user to build formulas with actual wooden blocks, rather than by just using a GUI on a computer. By providing the user with a tactile, hands-on experience, we hope to further encourage both experimentation and collaboration in a playful way. Manipulating the building blocks together with classmates seems more easy and natural when they are real-world objects, instead of pictures on a computer screen.

9.4 Concrete Implementation

This section introduces FOLL-E (First-Order Logic Learning Environment), the concrete implementation of our proposed approach. For this tool, we want

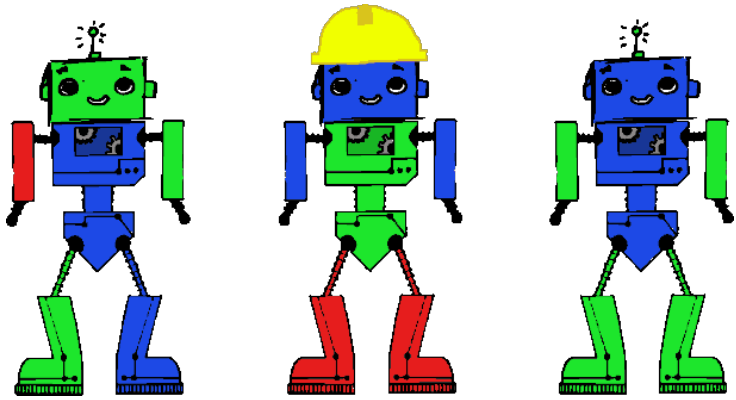


Figure 9.1: Example robot designs.

a specific problem domain which is (a) enticing and captivating for children and (b) simple enough for them to experiment with. While this is a fine line to tread, it may have a substantial effect on the attractiveness of the tool, and the children's engagement with it.

For the application domain, we have chosen the design of simple robots, as shown in Fig. 9.1. The robots have been hand-drawn in a cartoonish style with vibrant colours to create an inviting visual. A robot consists of six body parts in total, called *components*, which can be individually coloured in one of three colours. Specifically, a robot consists of two arms, two legs, a head and a body, which can each be coloured either red, blue or green. Additionally, a robot may also wear a hard hat.

To express constraints on an application domain, we devised a generic blocks-based formalism for FOL. It was designed with a *pegs-and-slots* approach in mind, in which blocks physically fit together if the connection is syntactically correct and the formulas are correctly typed. Fig. 9.2 shows an overview of the specific blocks. Note that our application uses only predicates and functions with arity ≤ 1, but the notation can of course be extended to higher arities if needed.

These generic blocks can be instantiated to specific blocks for a chosen vocabulary. For our tool, we prepared a library of blocks specific to the robot domain. In total, 16 different expression blocks are available in the tool. An overview of the specific blocks is shown in Table 9.1, together with the generic block type that they belong to. We represent all terms of type *Component* by circles and terms of type *Color* by squares. Future applications could use more types mapped to more such basic shapes. Variables are represented by the shape of their type

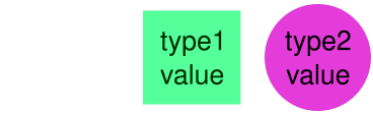
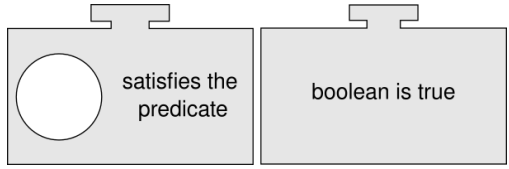
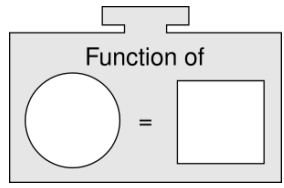
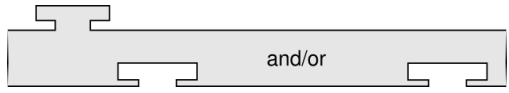
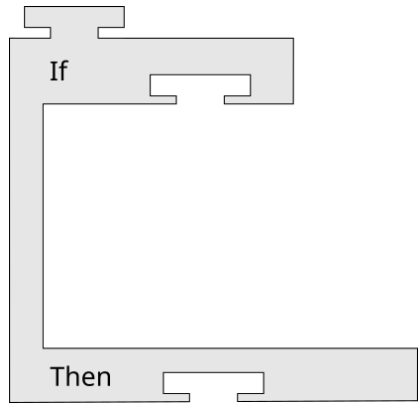
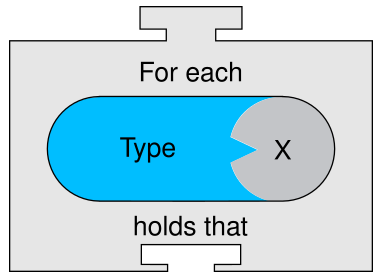
Symbol type	Block
Constant $c/0$	
Unary or nullary Predicate $p/1$ or $p/0$	
Unary Function $f(\cdot) = \cdot$	
$\wedge/\vee$	
$\Rightarrow$	
$\forall x[Type]$	

Figure 9.2: Overview of all generic blocks

Block type	Specific Block Meaning
Constant	Blue, Red, Green, Right Arm, Left Arm, Right Leg, Left Leg
Proposition	Robot wears a helmet
Predicate	... is an Arm, ... is a Leg
Function	Color of ... = ...
Implication	If ... Then ...
Con-/Disjunction	... And/Or ...
Type	Component
Variable	c
Quantifier	For each ... holds ...

Table 9.1: Overview of the blocks specific to the robot application, and their general block type. "..." signifies an open slot.

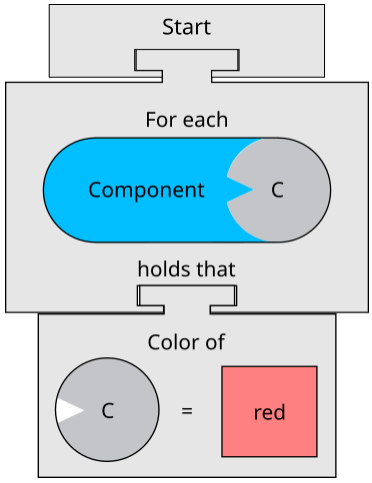


Figure 9.3: Blocks expressing "$\forall c : Color(c) = red$"

with an additional cut-out triangle in their left side. This allows a variable to be used wherever another term of that type could be used, but restricts the terms that can be quantified over to variables. Fig. 9.3 shows a simple example.

The blocks are laser cut out of 3mm plywood and engraved with their intended meaning in text and, when applicable, also with an icon such as an arm or a helmet. Blocks representing a colour were also painted in that colour. In this way, the notation is low on text, with much visual stimulation. As an added bonus, this also leads to better visibility from a distance, making it easier for

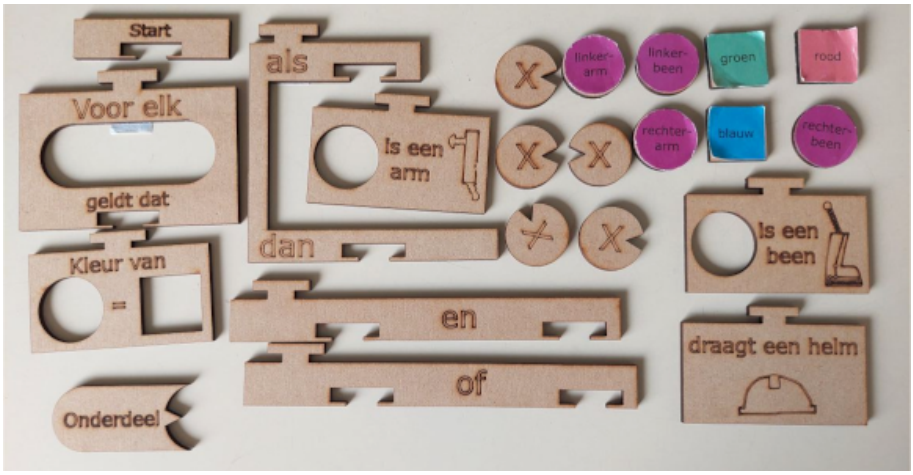


Figure 9.4: The laser-cut blocks included in each FOLL-E box. (Text engraving in Dutch)

the teachers to assist. Fig. 9.4 shows the end result.

In the application, the children are shown three "good" robots and three "bad" robots, and must devise a rule that distinguishes between them. By puzzling together blocks, the children can express sentences of increasing difficulty such as "The left leg is blue", "If the robot wears a helmet, it has a green body" and "Every component that is a limb must be coloured green.". Similar to Scratch, the tool is "always live". A Raspberry Pi mini-computer continually scans the playing area and, as soon as it detects a complete sentence (i.e., all the slots of the connected blocks have been filled), it converts this into FOL. The IDP-Z3 reasoning engine then checks whether the "good" robots indeed satisfy the formula, in which case a green check mark pops up next to them. Similarly, the "bad robots" are checked for unsatisfiability, in which case a green cross is shown. Finally, IDP-Z3 performs propagation, i.e., computes those properties that hold in all models of the formula. These are then shown as a partially coloured robot, where for example only the left leg is coloured blue. This robot is displayed in the centre of the screen, to help the children understand which property they actually expressed.

In this way, FOLL-E gives immediate feedback which results in high interactivity, allowing the children to quickly explore the effects of specific blocks. Moreover, the feedback is purely visual in the form of the check/cross marks and the centre robot, which we feel makes the application more intuitive and enticing. In total, we hand-crafted ten levels of increasing complexity for the children to play with.

The complete tool consists of the laser cut blocks, a box with a clear panel on top and a Raspberry Pi 3 B+ with a Raspicam inside, a computer monitor and a keyboard. The Pi is able to detect the individual blocks by the fiducial markers glued to them. Specifically, we use ArUco markers [60]. To detect complete sentences, we chain blocks together by starting at the *start* block and recursively looking for attached blocks. This approach has multiple advantages: it is invariant to the blocks' rotation, the designs can be changed easily by replacing the markers and occlusions are not possible due to the camera placement.

The application is written entirely in Python 3, with the GUI created using the Pygame [106] library. All source code is available online[1]. To ensure smooth communication to and from the IDP-Z3 system, the application makes use of *IDP-Z3Py*, an as of yet unreleased Python API which aims at facilitating the embedding of IDP-Z3 in Python applications. A preliminary version of this API is contained in the code repository. The total cost of one FOLL-E box, including the Raspberry Pi and the laser cutting but excluding the monitor and keyboard, comes down to about €70.

Figure 9.5 shows a few photos of children playing with FOLL-E.

9.5 Results

As a preliminary evaluation, we tested FOLL-E at two separate events aimed at different age groups. The first event was a "STEM-workshop" organised for high school students aged 16-18, typically with a technical background. The students were split into two groups of 15 and 9 students, which both partook in our workshop on Logics, but at different times of the day. During this four hour workshop, we aim to make the students aware of human reasoning by analysing our thinking behaviour when solving simple puzzles such as Sudoku. By gradually increasing the level of abstraction used to solve the puzzles, the students implicitly start solving SAT-encodings of the logic puzzles manually. During the last 30 minutes of the workshop, we briefly introduce the role of reasoning engines such as IDP-Z3 in solving such problems automatically, after which the students work through the FOLL-E puzzles in pairs.

The second event, the "Children's University", is an annual event for children between the age of 8 and 13. Both the morning and afternoon consist of a lecture of two hours and a workshop of 1.5 hours. In total, 28 children (13 morning + 15 afternoon) registered for our workshop "Teach the computer". During the workshop, we first spend 30 minutes on an interactive exercise in which we

[1] `https://gitlab.com/Vadevesi/foll-e`

(a) (b)

(c)

(d)

Figure 9.5: Photos taken during the Children's University workshop showing children playing with FOLL-E.

illustrate that computers are by themselves incapable of solving problems, and should be "taught" instead. Afterwards, they are given a 5 minute introduction to FOLL-E after which they are free to play through the levels in pairs.

We observed the children throughout both events, and will now summarize our observations. Firstly, the blocks-based notation successfully eliminates both syntax errors and type errors. The errors that the children did make fall in one of two categories: *reasoning errors* and *formalization errors*. This first category contains the errors made when trying to deduce the rule that distinguishes good from bad robots. The second category of errors happen during formalization, where children incorrectly translate their own understanding of the rule into the blocks-based notation.

Regardless of their age, all student pairs typically follow the same approach. First, they examine the robots and discuss what they think the distinguishing rule is. Once they agree on a rule, they try to represent it using the blocks. If their solution is not correct, they try to see if the error is in their formalization or their reasoning, and correct it accordingly. If they really can not figure it out, they ask for help.

All students in all groups were able to complete all ten levels, but the time required varied with age. The 16-18 year olds completed all levels in 20 minutes or less, whereas the 8-13 year olds generally needed 40-50 minutes. A significant time difference was also noticeable within the younger groups, where the children near the upper age limit (13) finished about 15 minutes ahead of the others.

One of the co-authors of this work has extensive experience with teaching Scratch to children aged 8-13, and reports the following differences:

- The use of physical blocks makes the tool more inviting than Scratch, and makes collaboration easier.
- Students are more focused. In Scratch, they are easily distracted by all of its bells and whistles.

Of course, our scope differs from Scratch's, not just in the skills we teach, but also in the fact that we fix the vocabulary that is to be used up front, while Scratch allows children to define their own functions.

In both age groups, children who had previous experience with Scratch initially had some difficulty adjusting to the declarative paradigm. Instead of the intended declarative solution such as "robot wears a hat", they initially tried to express an imperative statement such as "if the robot wears a hat, then it is correct".

The children reported that in general (a) they enjoyed their time with FOLL-E, (b) found the puzzle-like nature inviting and (c) liked the assignment. One of the older Children's University students said: "Being able to see how it works is very impressive", with regards to the visualization of the aruco marker detection. Another mentioned afterwards that "It's satisfying to puzzle the pieces together and see the result, especially if it's correct."

Many of the younger children reported some difficulty spikes between the levels. Among others, the transition to levels using quantification was difficult. A possible explanation for this is that some children solved the levels that were meant to initially introduce quantifiers in an untended way, without actually using quantification, which lead to a difficulty spike later on. The tests also revealed a few minor bugs. Syntax errors are technically still possible by placing an incorrect block *on top* of another block's slot instead of fitting it inside. In these situations, the application simply crashes. Also, sentences without models (such as "left arm is a leg"), crash the application, which should not happen.

To measure the effect of the tool on the logical reasoning of the children, we surveyed both Children's University groups (aged 8-13) using a questionnaire that gauges logical thinking. This questionnaire consists of four questions on spatial reasoning, and four questions on propositional reasoning. Examples of these questions are as follows:

> "The pen lies to the left of the book, and the book lies to the right of the pencil. Is the pencil definitely to the left of the pen?" (spatial)

and

> "If there is an L on this page, there is also a 7. There is no 7 on this page. Is there definitely an L? " (propositional)

The students in the morning session took this test before using the tool, as control group, and the afternoon students took the test after using the tool. This resulted in an average score of 5.46/8 for the control group and 4.6/8 for the intervention group, which therefore performed worse than the control group. A possible explanation might be that the control group took the test at 11.30h, while the intervention group took it at 16.15h, at the end of a busy day. In future work, a test with larger groups in more controlled circumstances will be done.

9.6 Related Work

Interest in teaching logic to children has been around for a long time. For example, already in 1982 did Kowalski [77] report on how they teach logic programming to children using Prolog, with the aim of learning how to apply logic to other subjects, and to spark interest in programming. While this work has the same target audience as ours, its goal is different, since they aim to teach (logic) programming whereas we are focused on the declarative meaning of FOL formulas. Most tools for teaching logic are geared towards university-level students [85, 91, 71]. We briefly discuss some of the tools that could also be used to teach to K-12.

An overview of early work is provided by [61]. Typically, these tools were focused on creating proofs and verifying them. The most well known of these is *Hyperproof* [11], a descendant of *Tarski's World* [10], which implements a graphical environment to teach FOL syntax and semantics. The user is tasked with expressing formulae on labelled geographical shapes in a 3D-world. For instance, if cube A is in front of tetrahedron B, they could write "*Cube(A)*", "*Tet(B)*" and "*FrontOf(A,B)*" If all cubes are in front of a tetrahedron, they could write "$\forall x : \mathit{Cube}(x) \Rightarrow \exists y : \mathit{Tet}(y) \wedge \mathit{FrontOf}(x, y)$". A survey by Fung et al. [58] on students in a logics CS course reported the benefits of Tarski's World (and by extension, Hyperproof) as encouraging exploration, removing difficulty spikes and being able to visualise quantification.

Reeves [109] notes that expressing logic specifically on geometrical shapes is not sufficiently *captivating* for students, as there seems to be no connection between the "subject matter" and CS. Therefore, the author proposes an extension of Tarski's World in which students reason over graphical user interfaces instead, stating that this is more relevant to them and thus more motivational.

Mauco et al. [92] present two tools, *FOST* and *LogicChess*, which like Hyperproof allows students to express FOL statements about objects in a graphical environment. The tools can check the statements for syntax errors, and evaluate their truth values. In the paper, the authors also remark that both these tools are domain-specific with pre-coded relations and functions. They regard this as a major downside, as "FOLs expressiveness lies in the possibility of working with arbitrary domains", and not just the predetermined domains. To overcome this, the authors propose a framework for didactical FOL tools focusing on extendability.

Our approach differentiates itself from these previously mentioned works in multiple ways. Firstly, it has a model-theoretic view on FOL rather than a proof-theoretic view, which we regard as a more foundational skill for children. Secondly, we use a novel blocks-based notation for FOL, in which children do

not need to pay attention to FOL's syntax and can thus focus more on creating sentences with a correct meaning. They also do not interact with the notation through a GUI, but instead write formulas with wooden blocks, resulting in a type of CS Unplugged [15]. For these reasons, we feel that our approach results in a more captivating experience, in which children remain engaged longer in their tasks.

9.7 Conclusion and Future Work

First-order logic has always been a cornerstone of computer science and AI. Yet, it is difficult to teach in general due to its steep learning curve. In this chapter, we present a novel blocks-based notation for FOL aimed at children, which is designed to be syntaxless, structured and playful. Using physical blocks further enforces the playful nature while stimulating experimentation and team work.

Based on these blocks, we created FOLL-E, a First-Order Logic Learning Environment with a focus on high interactivity, visual feedback, and fun. Children are asked to express knowledge that distinguishes "good" examples from "bad" ones. These blocks are then scanned and converted to FOL, after which a logical reasoning engine verifies the (in)correctness of the examples. This result is then shown, together with a visualisation of the consequences of their expression.

We tested the tool with around 50 children aged between 8-13 and 16-18. They were all enthusiastic, stating that the notation is intuitive and inviting, and that they overall enjoyed their time.

In future work, we will be teaming up with cognitive psychology researchers to correctly measure the benefit of the tool. Furthermore, we will be looking at what changes can be made to make the tool more effective, such as introducing more levels, experimenting with different applications besides robot design, and adding automated error explanation.

Chapter 10

Research Valorisation

In this chapter, we will take a closer look at some possible ways the research of this thesis can be *valorised*. Research valorisation is defined as follows [122]: "Valorisation is the process of creating value from knowledge, by making knowledge suitable and/or available for economic and/or societal use." In other words, it is about taking the knowledge acquired through the PhD, and putting it to use.

This chapter is split up into two sections: we first elaborate on the general economic benefit in Section 10.1, followed by a description of a more specific plan in Section 10.2.

10.1 Economic Benefit

Throughout this thesis, we primarily looked at how to make knowledge formalization more accessible. Using the Joint Interactive Modelling methodology as a framework, we looked into novel and user-friendly knowledge formalisms, improved verification techniques, and elucidated how experts select the most suitable formalism, among other things. In this way, our research serves as a good basis to make the development of knowledge bases (and their applications) easier for companies.

As evidenced by our past experiences in use cases (see later), embracing knowledge-based techniques entails quite a few advantages from a company's perspective:

A1 Declarative KBs are highly **adaptable**. As time goes on, traditional code bases tend to turn into rigid behemoths that are difficult to update. Updating a knowledge-based tool is easier, as we typically only need to modify (a part of) the KB. Moreover, due to its natural format, modifying the knowledge is usually straightforward.

A2 Knowledge-based tools leverage the knowledge already contained within an organisation. Data-based AI systems require expensive training to "learn" how to perform an operation, whereas knowledge-based systems can use a company's own knowledge. This is especially meaningful in data-poor environments, where it might be expensive to generate new data.

A3 A KB is **multi-purpose**: as the knowledge is formalized with no specific task in mind, a single KB can be applied effortlessly to solve multiple problems within the same domain.

A4 As all the knowledge used to derive specific decisions is explicitly present, declarative AI systems are more **explainable**. This is especially important in fields dealing with sensitive information, where it is important to be able to sufficiently explain why a value was derived.

A5 The knowledge in the organisation becomes **persistent**: if an employee leaves, their knowledge stays within the company by way of the knowledge base. As an additional benefit, this same KB can be used to train new members of the team.

A6 Processes will be more **consistent**: where different employees might have different approaches to solve a problem (such as rules-of-thumb or "guestimates"), a KB provides a way to standardize these approaches. Not only does this ensure that all employees will reach the same outcomes, but it also aids in aligning knowledge across multiple (geographical) branches of an organisation.

A7 Processes will also be more **efficient** and more **accurate**, requiring less manpower and overall less time.

To substantiate these statements, we will briefly go over four past use cases and highlight the advantages that were reported.

In the component designer [4], Aerts et al. describe how they built a knowledge-based tool to assist design experts within a certain multi-national company. To formalize the KB, they performed multiple knowledge workshops including engineers from different locations worldwide, in which they used a mix of DMN and cDMN. During an evaluation session, the engineers reported that they

can design components roughly 10% faster and with less trial-and-error (A7) and that their designs are now consistent (A6) between the different engineers globally. Moreover, they also appreciated that the tool is multi-purpose (A3): besides assisting in designing a component, it can also help explain why an unsuitable component design failed (A4), and it can serve as a way to teach newer members of the team (A5).

In a use case concerning a piece of Belgian legislation [48], Deryck et al. successfully built a knowledge-based tool to support notaries in calculating registration duties. Using DMN as a base formalism, they collaborated with a notary to formalize the KB based on the relevant law texts (A2). Here too, the resulting tool is multi-functional (A3). Besides calculating registration duties, it can also be used to reason in "reverse" by selecting the lowest possible tax first and then answering the other information. Moreover, it is also highly explainable (A4), which is an important property for tools in the legal field. Perhaps most interestingly however, the KB has also proven to be highly adaptable (A1): when the law surrounding the registration duties was significantly changed in 2018, updating the KB required only 0.5 person-days.

Together with Intelli-Select, Deryck et al. [45] built a knowledge-based tool for financial asset management. Through a CNL interface, users can express rules over financial assets (A2). The resulting KBs can then be used to perform many operations (A3): check eligibility of profiles, explain why an asset is (in)eligible (A4), compare against others, generate specific documents, and more. For one specific operation, they also report an incredible time reduction of multiple months to mere seconds (A7)! Intelli-Select further developed this tool into OSCAR, which has now been integrated in the toolkit of one of the global leading financial institutes [1].

In the Adhesive Selection use case (Chapter 7), we built a tool to support experts in selecting suitable glues. Here, the KB was formalized during multiple workshops with the experts (A2) in which we used a mix of DMN and cDMN. The resulting tool helps them find adhesives faster and with less wasteful testing (A7). By all reasoning on the same set of adhesives, the selection process has also become more consistent (A6) across different experts. The tool also offers a way to challenge an expert's assumptions, because it can explain why a selection is incorrect (A4). Lastly, the team lead especially appreciates that the knowledge of the experts is now captured in the KB (A5), and can be used to train newer members of the lab.

To summarize: by lowering the threshold for building KR applications, more organizations are able to take advantage of the aforementioned benefits by efficiently leveraging the knowledge in their organisation.

10.2 Building a knowledge-based system

We see three main ways in which a company could turn internal knowledge into a knowledge-base system: they could develop the entire application in-house, they could hire a consultant to help create the KB, or they could outsource the complete development of the tool to an external company. We will now briefly describe each possibility's pros and cons, followed by a closer look at one of the specific options.

In-house development. Developing a knowledge-based system in-house allows a company to have complete freedom over its design, use and capabilities. However, this also means that they have to bear the full burden of KB creation, a task which is not straightforward. It is likely in this case that one of the company's employees should be trained in knowledge formalization, which is a lengthy and thereby expensive process requiring a lot of experience (see Chapter 8). To help in this regard, we have developed online resources such as *IDP-Z3 by Example*[1], an interactive IDP-Z3 tutorial website, and the *cDMN documentation pages*[2], which go in great detail on how to use DMN, cDMN and pDMN. Though always having a knowledge expert at your disposal has obvious advantages, such as being able to quickly update knowledge bases if needed, this option is most likely only worth it if the company wants to develop multiple knowledge-based tools for in-house use.

Outsourcing the KB creation. Another possibility is to develop a tool internally for the most part, but to outsource the creation of the KB[3]. For instance, a company could hire a knowledge expert as a consultant, who develops the KB together with domain experts by way of Joint Interactive Modelling. In this way, the company still has complete control over the resulting tool with regards to functionality, UI, and others, without requiring an in-house knowledge expert. One downside of this approach is that this external person might not be well-versed in the problem domain, and might therefore require some time to fully understand it. Additionally, each time the company wants to make drastic changes to the KB, they will have to hire the consultant again.

Outsourcing the entire tool. The third approach to acquire a knowledge-based tool is by outsourcing its development entirely. In this way, the company does not need to be concerned about the technical aspects of its development, as long

[1] `https://interactive-idp.gitlab.io/`
[2] `www.cdmn.be`
[3] This is possible thanks to the modular nature of KB systems.

as the delivered product fits the bill. Though it is the costliest option of the three, it is most likely ideal for organisations that do not consider developing IT applications as their core business.

10.2.1 Valorisation plan

To complete our section on the economic valorisation of knowledge-based systems, we will now describe the third possibility in more detail. We believe that this approach to development can offer the most value to the largest number of companies. Instead of describing this approach from the viewpoint of the company however, we will describe it from the viewpoint of the external knowledge expert, "as if we ourselves wanted to start such a service".

What is involved in the valorisation? We would offer a service to build knowledge-based applications for companies, in which we:

1. capture their expert knowledge in a user-friendly formalism, through Joint Interactive Modelling;
2. build a knowledge-based tool based on the knowledge (either using the IC or by designing a custom interface);
3. train their staff on how to interpret and maintain the knowledge;
4. possibly offer support when needed.

This approach has already proven effective in use cases in the past[4]. The main difference is that we would like to build "finished products", instead of prototypes.

At the same time, we would maintain and extend our toolstack: cDMN solver, pDMN solver, DMN-IDP, FM-IDP, and others.

Target group. We believe that there is a sufficiently large market for such services, as throughout the PhD we have already been contacted multiple times by interested parties. For example, companies have reached out to us for the following.

- Build full-fledged chat bots based on (c)DMN models.

[4]See Adhesive Selector (Chapter 7), but also [3, 4, 6, 45, 48].

- Use pDMN to model Bayesian belief networks in a user-friendly manner, to assist legal experts
- Build ASP support for the cDMN solver, to model planning problems.
- Use cDMN to model Dutch government regulations, to increase explainability and transparency.
- Develop software to verify architectural designs w.r.t. building codes based on (c)DMN and IDP-Z3.
- Construct a KB-based tool to support the design of custom kitchen cabinets.
- Evaluate analytics data using cDMN models.
- ...

Moreover, as the graph in Figure 10.1 shows, our website detailing information on DMN-IDP(y), cDMN and pDMN has a rising number of pageviews, indicating interest in the projects. Based on this information, we see the most potential in two specific types of applications: support systems for design/configuration and decision process management.

Design/configuration problems, such as the Adhesive Selector and the component design problem of [4], are ubiquitous in industry. We believe that we have an excellent set of tools for modelling such problems, by way of cDMN and Feature Modelling, and an equally excellent set of tools for solving them, by way of IDP-Z3 and the IC. One challenge, however, is that companies are sometimes unaware that there exists tooling to support their staff in this process, which might make it more difficult to find new projects.

Decision process management is one of the rising trends in industry, and is by some even being called "the new AI" [21]. Here too, we belief that we have a unique and attractive toolset to offer. Firstly, we could assist companies without any prior decision intelligence experience with setting up DMN, cDMN or pDMN models, and equip them with proper tooling. Secondly, companies that already use decision intelligence can plug in their DMN models into our tools and get the added functionality offered by our tools without any extra effort. Thirdly, if those companies bump into the limits of DMN for their purposes, they can be further helped by transitioning to cDMN or pDMN.

Validation. As mentioned earlier, the approach of IDP-Z3 and the IC has been successfully applied to a number of use cases in the past. Moreover, with

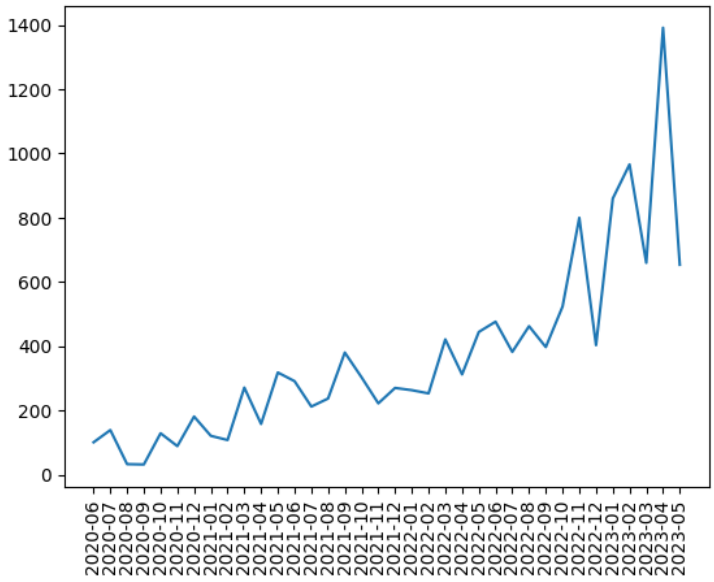


Figure 10.1: The pageviews for the cDMN website (`www.cdmn.be`)

IDP-Z3 already running in production at one of the global leading financial institutions [86], we are more confident than ever on IDP-Z3's usefulness. Similarly, as they are tried-and-tested industry standards, DMN and feature modelling should both be ready for use in production.

For cDMN and pDMN on the other hand, we do not have such extensive use cases yet. While we are convinced that they are already effective in a multitude of real-life applications, we would require to build more concrete prototypes to be sure. However, both of these languages are still quite young, and we can change aspects of them (such as adding new operators) when we feel it would be beneficial.

Intellectual property rights / freedom to operate. We will now give a brief overview of the required software for this valorization and their commercial licenses. In total, we encounter two license types: MIT, and LGPLv3. The former license indicates that we may use the software as we please, even in commercial applications. Software with the latter license may also be used commercially, as long as we publicize any changes we make to it.

- IDP-Z3, DTAI, LGPLv3.

- Z3, Microsoft Corp., MIT.
- Interactive Consultant, DTAI, LGPLv3.
- DMN-IDP, DTAI, LGPLv3.
- cDMN solver, DTAI, LGPLv3.
- FM-IDP, DTAI, LGPLv3.

10.3 Societal Benefit

Besides economic benefit, we believe that the output of this thesis can also have great societal benefits. Indeed, as our work on FOLL-E has shown, our technology stack could help prepare an entire generation of students on the ever-growing digitization of the workplace. To this end, we would like to continue development on FOLL-E by, among other things, expanding on the exercises that it offers and experimenting with different novel approaches.

Moreover, we would would like to make it broadly accessible to schools and other organizations that are interested in sharpening the computational thinking skills of children. We believe FOLL-E offers an engaging mix of interactivity, fun, and high didactic potential. FOLL-E's main social value is sharpening the reasoning skills of the next generation: critical thinking and reasoning are becoming evermore important, and we feel FOLL-E could be a great asset in the toolkit of many organisations to help children develop these competences. Therefore, we do not want to sell it as a product, but instead we want to launch an open-source building kit.

The kit would include building instructions, open source code, and a complete bill of materials, to be published online for free. In this way, anyone with access to the materials and a laser cutter (e.g., through a Makerspace) will be able to construct their own FOLL-E set-ups. By making it open-source, FOLL-E will be broadly accessible: the total cost of a FOLL-E box is about €70, making it much cheaper than most other tools for computational thinking. For people that do not have access to a Makerspace or that do not want to spend time building, we can also provide puzzle pieces or fully pre-built boxes at a slight extra cost.

We feel that there are many organisations worldwide that could benefit from a tool such as FOLL-E. Schools are increasingly focussing on their STEM curriculum, and FOLL-E could form a fun introduction to logic. Organisations teaching programming courses to K-12 (such as CoderDojo) can switch up

imperative programming with logic, while still keeping the children entertained. Universities can build their own FOLL-E set-ups to raise awareness about the importance of logic through workshops, similar to what we do at KU Leuven. In other words: there are many possibilities.

As of right now, FOLL-E has already been tested in multiple events: two "Children University" events and, three STEM workshops on our campus, and five workshops at primary schools throughout the country. By way of these events, FOLL-E has already proven itself as being an enjoyable, captivating and intuitive tool for children. While the initial results w.r.t. increasing the children's logical thinking were not significant (see Chapter 9), the test was flawed in its methodology. To rectify this, we are in active collaboration with pedagogical researchers to perform more thorough tests, and expect the results to be published by July of 2024.

Chapter 11

Conclusion

The research goal of this thesis was to investigate how to effectively simplify the KB creation process for domain experts. The main motivation for this goal is that, while knowledge-based AI systems offer many potential benefits, the knowledge acquisition bottleneck remains a hurdle in their practical application. Though the methodology of Joint Interactive Modelling is already a big step forwards, there is quite a lot more ground to cover to support the process. We will now conclude our work towards this goal by briefly discussing each chapter, followed by a wrap-up and a list of potential future work.

Throughout this PhD, we have looked into different ways to simplify the modelling process. Our main approach was to look into alternative formalisms for knowledge representation. While the standard formalism for modelling knowledge in IDP-Z3, FO($\cdot$), is very expressive, it is ill-suited for non-AI experts due to its highly complex nature. At the same time, we wanted to avoid reinventing the wheel by making up our own formalism. Instead, we looked at two existing industry standards and evaluated them on their KR potential. We also created two extensions of such an existing standard to further improve on their expressiveness. Moreover, we also worked on other aspects of the joint interactive modelling approach, such as a tool for verification, and performed two interviews to gauge the opinions of experts.

In Chapter 2, we looked into using the Decision Model and Notation standard for KRR. DMN is widely used to represent (deterministic) decision processes, and aims to be as user-friendly as possible so that anyone involved in the decision process (business experts, IT staff, ...) can play an active role in modelling. While DMN models are intended to be evaluated from the bottom to the top, we showed that the knowledge inside the models can actually be put

to many more uses. Indeed, by supporting other reasoning mechanisms such as reasoning on partial information, evaluating sub-decisions and "reasoning backwards", DMN models can serve as the basis for complex decision support software. We build two tools to support this approach: an integration of DMN and IDP-Z3's Interactive Consultant interface, and an IDP-Z3-based API for a DMN reasoning engine. To demonstrate the viability of DMN for KR, we have used this latter tool to build a chat bot to interact with DMN models. Based on these findings, we conclude that DMN is a viable and user-friendly option to build KR tools based on decision logic.

One downside of DMN is its relatively low expressiveness, which severely limits its number of potential applications. To overcome this, Chapter 3 introduces an extension of DMN called Constraint Decision Model and Notation (cDMN). This new notation enhances the expressiveness of the original by introducing concepts such as constraint tables, n-ary predicates and functions, quantification, and more. At the same time, by holding on to DMN's tabular notation, it aims to maintain the intuitive nature of the original formalism. Through implementing and comparing decision management challenges, we have shown that cDMN is effective at modelling more types of problems, and that in general it results in more compact solutions. These results suggest that cDMN is indeed typically quite readable and maintainable.

Another type of problem that is frequently encountered in real-life problems is probabilistic reasoning. While DMN's deterministic nature is one of its main advantages, there are still cases in which it is useful to include uncertainties. To this end, Chapter 4 presents a probabilistic extension of DMN, called Probabilistic Decision Model and Notation (pDMN). pDMN extends DMN with probabilities, predicates, quantification, and a new hit policy to represent annotated disjunctions. We demonstrate pDMN by implementing classic PLP examples, which we feel result in more user-friendly models.

The second industry standard that we have looked at in this work is feature modelling, a notation for modelling a product's variability. As it models variability, it actually lends itself excellently towards modelling configuration problems as well, such as the design of industrial components. Yet, when looking at the state-of-the-art, there is not much research on using this formalism in this way. In Chapter 5, we investigate this discrepancy and find that this is due to feature modelling's inability to express *background knowledge*. To overcome this, we present the idea of combining feature models with background knowledge in (propositional) FO($\cdot$), allowing a modeller to, e.g., express environmental knowledge. By integrating a feature modelling tool in the Interactive Consultant, we demonstrate that our approach is indeed capable of modelling complex configuration problems while maintaining a user-friendly feature model as basis.

To further support the Joint Interactive Modelling approach, in which frequent verification and validation plays a big role, Chapter 6 describes a new technique for the verification of DMN tables. Instead of verifying tables in isolation, our verification method is *context-aware*: it always verifies a table within the context of the other tables and user-defined background knowledge. As demonstrated by a comparison to the other state-of-the-art techniques, our approach results in a more thorough verification. Our context-aware verification has also been integrated into our DMN-IDP tool, making it accessible to all.

Throughout this PhD, we have collaborated with the Flanders Make Joining Materials Lab on a use case to build a decision support tool for adhesive selection. This use case proved to be an excellent opportunity to test our approaches, as is described in Chapter 7. To extract the adhesive knowledge, we have performed multiple Joint Interactive Modelling workshops with adhesive experts, using a combination of DMN and cDMN as base notations. Through qualitative interviews, we have gained a better view on the opinions of the domain experts on our knowledge-based approach. The experts appreciate both the interactive and explainable nature of the tool, and the fact that they can see, interpret and possible update the KB themselves. They have also expressed that the current interface, a slightly adapted version of the IC, is rather unorganized and somewhat unintuitive. This output of the interviews will be used to guide further work on our approaches. Overall, everyone involved in the project is very positive about the progress thus far.

In Chapter 8, we presented the results of four qualitative interviews in which we asked a modelling expert about their modelling experiences. Here, the main goal was to elucidate the modelling intuitions and instincts of experts, and to discuss concepts such as the user-friendliness of formalisms. Based on the interviews, we discussed five main themes: expressiveness, modeller, naturalness, tooling, and user-friendliness. By among others discussing FO($\cdot$)'s user-friendliness, indicating the main design focusses of modellers and describing the crucial role of tooling, the output of this interview is a valuable source of information to guide future research w.r.t. modelling formalisms.

Lastly, we presented an "extreme" version of a user-friendly formalism in Chapter 9 for teaching FOL to children aged 8-14. While FOL plays an important role in fields such as CS, mathematics and AI, there is little to no research around teaching it to children. To address this gap, we created FOLL-E, a fun and engaging environment in which children can freely experiment with FOL formulas by expressing knowledge on robot designs. Through multiple workshops with children, we have evaluated FOLL-E in a real-life setting and found that the children enjoy their experience, stating that it is intuitive and inviting.

To conclude the thesis, simplifying the KB creation process for domain experts is a non-trivial task, due to a few reasons. Firstly, domain experts come in many shapes and forms, each with different backgrounds and experiences. It is possible that what is considered user-friendly for one, might be very difficult to use for the other and vice versa. Secondly, when looking into alternative formalisms, it is difficult to define or quantise terms such as "user-friendliness", "readability" and "easy to write", as they are ambiguous and heavily depend on the people involved. Thirdly, besides the formalism, there are many more aspects of the modelling process to take into consideration, such as the available tooling and the nature of the knowledge. By using tried and tested notations such as DMN and feature modelling for KR, we know up-front that they are considered user-friendly. Moreover, even if they are not sufficiently expressive for some applications, they still form an excellent foundation to build off when developing extensions. However, when we are looking into new user-friendly formalisms for KR, we should also keep other factors in mind such as the tooling available.

This thesis has given a lot of food for thought about future research directions. For instance, future work could focus on quantifying terms such as "readability" and "user-friendliness", by performing studies on domain experts. Another direction consists of expanding the tooling for the formalisms discussed in this work: for instance, they could build a structured editor for cDMN or examine verification techniques for feature modelling. Lastly, it is worth investigating if other AI techniques such as machine learning could play a role in modelling knowledge, by for example automatically building KBs based on positive examples or by using large language models to generate FOL.

Appendix A

Software Overview

This appendix contains an overview of all the relevant software which we developed or contributed to in the course of this thesis. Fig. A.1 shows a graphical diagram containing the relations between these software projects. Accompanying it, Table A.1 contains references to online resources for each piece of software.

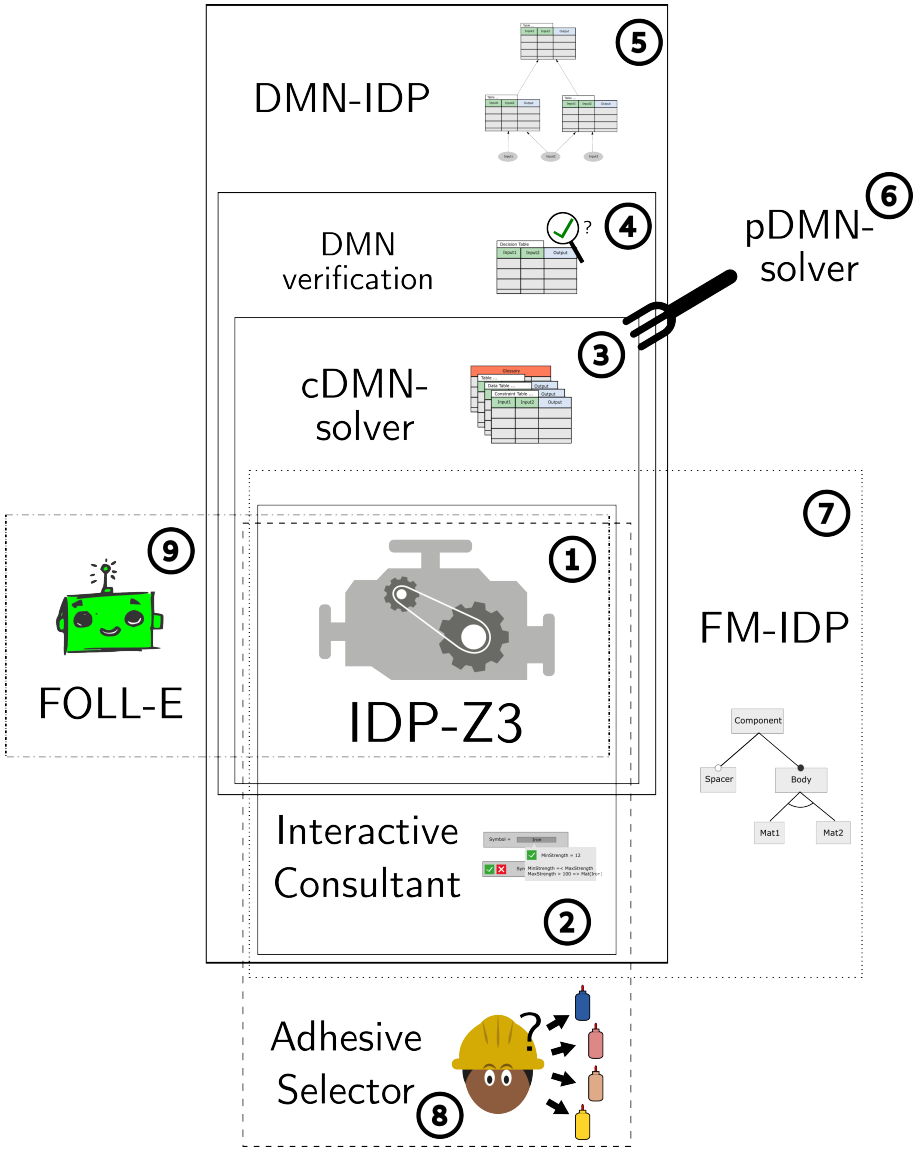


Figure A.1: Graphical overview of all software relevant to this thesis

	Name	Chapter	URL
1.	IDP-Z3	Chapter 1	https://IDP-Z3.be https://gitlab.com/krr/IDP-Z3
2.	Interactive Consultant	Chapter 1	https://www.idp-z3.be/Interactive-Consultant/ https://gitlab.com/krr/IDP-Z3
3.	cDMN solver	Chapter 3	https://www.cdmn.be https://gitlab.com/EAVISE/cdmn/cdmn-solver
4.	DMN verification	Chapter 6	https://gitlab.com/EAVISE/cdmn/DMN-IDP
5.	DMN-IDP	Chapter 2	https://gitlab.com/EAVISE/cdmn/DMN-IDP
6.	pDMN solver	Chapter 4	https://gitlab.com/EAVISE/cdmn/pdmn
7.	FM-IDP	Chapter 5	https://gitlab.com/EAVISE/featuremodel/feature-model-IDP
8.	Adhesive Selector	Chapter 7	Not public.
9.	FOLL-E	Chapter 9	https://FOLL-E.com https://gitlab.com/Vadevesi/foll-e

Table A.1: Detailed list of the software relevant to this thesis.

Appendix B

Adhesive Selector Codes

This appendix contains a more detailed description of the Adhesive Selector interview codes presented in Chapter 7. Table B.1 shows an overview of each code and the interviews in which they appeared. Each code is also explained in our "code book", included in Table B.2.

Table B.1: Discovered codes per participant, and the data saturation.

Code	1	2	3	4
Cluttered Interface	✓	✓	✓	✓
Interactive Counter	✓	✓	✓	
Time Gain	✓	✓	✓	✓
Avoiding Mistakes	✓			
Playing Around	✓		✓	
More Confident	✓		✓	✓
Understandable	✓			
Trust in the System	✓	✓	✓	✓
Verification of the Result	✓	✓	✓	✓
Adequate Performance	✓	✓	✓	✓
Broader Scope	✓		✓	✓
Understanding Knowledge	✓			✓
Teaching Tool	✓		✓	✓
Explainability	✓	✓	✓	
Difficult Explanations	✓	✓	✓	✓
Extended DB Required		✓	✓	
Select Specific Adhesives		✓	✓	
Help in Design		✓		
Expert in the Loop		✓	✓	✓
Interactivity		✓	✓	
Relevancy		✓		
More Detail		✓	✓	✓
User-added Knowledge		✓		
Flexibility			✓	
Immediate Feedback			✓	
Central Place for Knowledge				✓
Total codes	15	16	19	13
Data saturation	58%	88%	96%	100%

Table B.2: A code book elaborating on the meaning of each code of the interviews for the Adhesive Selector.

Code	Explanation
Cluttered Interface	The interface's structure seems chaotic.
Interactive Counter	The counter denoting the number of suitable adhesives.
Time Gain	Experts can more efficiently select an adhesive.
Avoiding Mistakes	The tool prevents making mistakes.
Playing Around	Interacting with the knowledge allows the expert to "play around" with it.
More Confident	The experts are more sure of their selection using the tool.
Understandable	The interface presents information in a sufficiently understandable way.
Trust in the System	The experts trust the choices of the system.
Verification of the Result	The resulting short-list of adhesives will always need to be verified.
Adequate Performance	The loading time of the tool is within reason.
Broader Scope	The experts discover more adhesives.
Understanding Knowledge	The tool helps understand the knowledge.
Teaching Tool	The tool is well-suited for teaching newer members of the lab.
Explainability	Explainability is an important factor.
Difficult Explanations	The automatically generated explanations are difficult.
Extended DB Required	More adhesives should be added to the system.
Select Specific Adhesives	The AS allows selecting specific adhesives, whereas the other tools only select families.
Help in Design	The tool can also play a role in joint design.
Expert in the Loop	An expert is always required to stay in the loop.
Interactivity	Interactivity is a crucial aspect of the tool.
Relevancy	Detecting which parameters are still relevant aids in selecting adhesives.
More Detail	The tool is more detailed than any other in terms of possible parameters.
User-added Knowledge	The experts would like to add their own knowledge to the system.
Flexibility	The tool adapts well to the expert's way of working.
Immediate Feedback	The experts appreciate that the interface updates automatically.
Central Place for Knowledge	All knowledge for adhesive selection is stored in one location.

Appendix C

Modelling Interview Appendix

This appendix contains supplementary information for Chapter 8. Section C.1 contains the document that was sent out to the modelling experts, consisting of a short introduction for the interview and four problem descriptions. Section C.2 discusses some of the implementations made by the expert modellers in more detail by comparing them and pointing out interesting differences. Section C.3 contains an overview of all the codes discovered in the interviews.

C.1 Qualitative Expert interview: use cases

As part of a qualitative study into modelling formalisms, this document details four small dummy use cases. You can consider these descriptions "as if given by a domain expert with the goal of building an interactive knowledge-based tool", and you should implement them in the way you feel fits best w.r.t. formalism and design focus. Afterwards, we will hold an interview to attempt "extracting" your reasoning when modelling.

C.1.1 Concrete Selector

> A construction consultancy company is specialised in finding the best type of concrete for a job. Selecting concrete is done on the basis of three parameters: strength, highest working temperature, and maximum viscosity. The selection typically goes as follows: the company has a range of "standard mixes'' for concrete (c1 through

c3), for which the values of the above three parameters are mostly known. These standard mixes can be further tweaked to meet specific needs, like adding a bit less water to decrease viscosity at the cost of some strength.

As these tweaked mixes are empirically validated, the experts do not always know up-front what their parameter values will be. So, they take the following approach: if a parameter of a specific mix is unknown, assume the parameter value of the default mix on which it is based as an approximation. If that value is also unknown, then do not apply any constraints that would exclude the mix.

Mix	Strength	High_T	Max_Vis
c1	50	75	30
c2	60	85	??
c3	80	??	25

Mix	Strength	High_T	Max_Vis
c1.1	55	73	30
c1.2	60	65	??
c1.3	47	??	20
c2.1	??	95	12
c2.2	65	85	??
c3.1	??	??	18
c3.2	90	??	??
c3.3	75	100	??

C.1.2 Phone Configuration

A producer of phones builds custom, personalized phones for their customers. To make this design process more straightforward, they employ a modular design: phones consist of multiple components, which can require or exclude others. All phones must include the call module, and must also contain a screen. This screen is one of three types: it can be basic, colour, or high resolution. The phone may also optionally contain a GPS module, but only if the screen is not basic. Moreover, the phone can also support two types of media: a camera, and an MP3 player (possibly at the same time). There are no special requirements for the MP3 player. However, the camera may only be selected if the screen supports a high resolution.

C.1.3 Vacation Days

The number of vacation days depends on age and years of service. Every employee receives at least 22 days. Additional days are provided according to the following criteria:

1. Only employees younger than 18 or at least 60 years, or employees with at least 30 years of service will receive 5 days.
2. Employees with at least 30 years of service and also employees of age 60 or more, receive 3 extra days, on top of possible additional days already given.
3. If an employee has at least 15 but less than 30 years of service, 2 extra days are given. These 2 days are also provided for employees of age 45 or more. These 2 extra days can not be combined with the 5 extra days.

C.1.4 Planning problem

To ensure that a person capable of applying first-aid is always present in a school, a weekly permanence schedule is made. Days are split into a morning segment and an afternoon segment. For each segment, someone must be assigned as first-aid officer, with another person being assigned back-up. In total, there are five people that can be assigned permanence: William, Julie, Alex, Emma, Sophie. Each of these people have their own availabilities. A planning must be made according to these rules:

1. A person cannot be assigned both first-aid and back-up at the same time.
2. If a person is not available, they cannot be scheduled.
3. Emma must be assigned first-aid on Wednesday afternoon.
4. William must be back-up on Mondag morning.
5. Sophie may only be planned as back-up.

C.2 Expert Models

C.2.1 Concrete Selection

In the implementations that we received for the concrete selection problem, two stand out. Listing C.2.1 shows an implementation by M1 which uses FO(·) definitions, and Listing C.2.2 shows an implementation using IDP's CNL by M2. Both modellers motivated their design choice using the same argument: they wanted to stick as closely as possible to the problem descriptions to ensure naturalness (see Chapter 8.4.3).

M1's reasoning for definitions was that the problem description, in their opinion, *defined* the assumed values for strength, temperature, and viscosity. Note that besides using definitions, they also included some CNL-like statements in the form of "if-then-else" operators – combined, these two lead to a compact and relatively readable knowledge base. One design choice that seems a bit "off" however is their usage of predicates (such as $known\_strength : Concrete \times Num \to \mathbb{B}$) instead of functions ($known\_strength : Concrete \to Num$).

Listing C.2.1: FO(·) solution by **M1**
(structure left out to reduce clutter)

```
vocabulary {
    type Concrete := {c1,c2,c3,c1_1,c1_2,c1_3,c2_1,c2_2,c2_3,c3_1,c3_2,c3_3}
    type Num := {0..100}

    Tweaked : Concrete → 𝔹
    Base : Concrete → 𝔹
    base : Tweaked → Base

    known_strength : Concrete × Num → 𝔹
    known_temp : Concrete × Num → 𝔹
    known_vis : Concrete × Num → 𝔹

    assumed_strength : Tweaked × Num → 𝔹
    assumed_temp : Tweaked × Num → 𝔹
    assumed_vis : Tweaked × Num → 𝔹

    strength : () → Int
    temp : () → Int
    vis : () → Int
}

theory {
    {
        ∀c in Concrete, x in Num: assumed_strength(c,x) ← if known_strength(c,x) then
     known_strength(c,x) else known_strength(base(c),x).
    }
    {
        ∀c in Concrete, x in Num: assumed_temp(c,x) ← if known_temp(c,x) then known_temp(c,
     x) else known_temp(base(c),x).
    }
```

```
    {
        ∀c in Concrete, x in Num: assumed_vis(c,x) ← if known_vis(c,x) then known_vis(c,x)
      else known_vis(base(c),x).
    }
}
```

In contrast, Modeller 2's entire theory consists of nested "if-then-else" CNL statements. Here, *is enumerated* is a special operator that evaluates to *true* if and only if an applied symbol is interpreted in the structure, i.e., the structure of the knowledge base contains an entry for that symbol applied to its arguments. As an example, *strengthOf(mix())* is *true* when the structure contains a strength for the value of *mix()*.

When looking at the vocabulary, we can see that M2 used a special operator for type *Mix*. Indeed, this is a *constructor* type, which allows a modeller to build types using other types. Here, *Mix* represents every possible combination of *Nat* and *StandardMix*, thus combining both standard and modified concretes in one type. In general, this will make the theory more compact, and thus more maintainable. At the same time, it will also make it more complex, thereby showcasing a possible trade-off between expressiveness and user-friendliness.

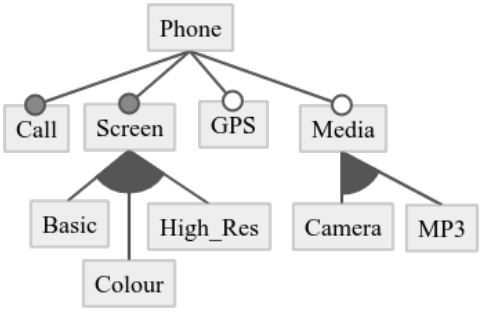
Listing C.2.2: Controlled Natural Language solution by **M2**
(structure left out to reduce clutter)

```
vocabulary V {
    type Nat := {0..3}
    type StandardMix := {C1, C2, C3}
    type Mix := constructed from {c(standardOf:StandardMix, Nat)}

    strengthOf, temperatureOf, viscosityOf : Mix → Int
    strength, temperature, viscosity : () → Int // requested
    mix : () → Mix // choice
}

theory T:V {
    if strengthOf(mix()) is enumerated
    then strength() > strengthOf(mix())
    else (if strengthOf(c(standardOf(mix()),0)) is enumerated
        then strength() > strengthOf(c(standardOf(mix()),0))
        else true).

    if temperatureOf(mix()) is enumerated
    then temperature() > temperatureOf(mix())
    else (if temperatureOf(c(standardOf(mix()),0)) is enumerated
        then temperature() > temperatureOf(c(standardOf(mix()),0))
        else true).

    if viscosityOf(mix()) is enumerated
    then viscosity() > viscosityOf(mix())
    else (if viscosityOf(c(standardOf(mix()),0)) is enumerated
        then viscosity() > viscosityOf(c(standardOf(mix()),0))
        else true).

}
```

C.2.2 Phone Configuration

For the phone configuration problem, the formalism choice was evenly split: we received two FO(·) implementations, and two feature model implementations. Below, in Listing C.2.3, M4's solution using a feature model is shown. As can be seen, using a feature model for this problem leads to quite an intuitive representation of a phone, by giving a visual overview of what components are possible. Together with a small theory block, it fully encapsulates the problem domain[1].

Listing C.2.3: Feature Modelling solution by **M4**




```
theory {
    GPS() ⇒ ¬Basic().
    Camera() ⇒ High_Res().
}
```

Comparing the above implementation to one in FO(·) (Listing C.2.4) shows off another major advantage of feature models: creating an ontology for the problem (= deciding how to represent the domain) is mostly done implicitly in feature models, whereas FO(·) always requires an explicit vocabulary. This takes a huge burden away from the modeller, and reduces the potential of errors.

At the same time, feature models are more prone to exploding in size when the problem domain is larger. Imagine, for instance, that the phone could be configured with 100 different types of media: this would require adding 100 children to the *Media* node! It goes without saying that this would negatively impact readability, maintainability, and general usability of the feature model.

[1]Note however that the modeller made a slight mistake: *Screen*'s children should be in an "alternative" relation instead of an "or" relation.

By contrast, in the FO(·) shown below, it would suffice to simply expand the interpretation of the *Component* type.

Listing C.2.4: FO(·) solution by **M2**

```
vocabulary V {
    type Component := {Call, Screen, GPS, Camera, MP3}
    type ScreenType := {Basic, Colour, HighRes}
    has : Component → 𝔹
    screenType: () → ScreenType
}

theory T:V {
    has(Call).
    has(Screen).
    screenType() ≠ Basic ⇐ has(GPS).
    screenType() = HighRes ⇐ has(Camera).
}
```

C.2.3 Vacation Days

For the vacation days problem, we have received one cDMN model and three FO(·) models. The cDMN model is as follows:

Listing C.2.5: cDMN solution by **M3**

Type		
Name	**DataType**	**Possible Values**
Employee	String	person_a

Function	
Name	**Datatype**
vacation days of Employee	Int
age of Employee	Int
service years of Employee	Int

Vacation days				
C+	Employee called e	age of e	service years of e	vacation days of e
1	-	-	-	22
2	-	< 18	-	5
3	-	≥ 60	-	5
4	-	$[18, 60)$	≥ 30	5
5	-	-	≥ 30	3
6	-	< 60	< 30	3
7	-	< 45	$[15, 30)$	2
8	-	$[45, 60)$	< 30	2

The decision table is relatively easy to interpret. The *C+* hit policy means that it sums the output value of each applicable row, with every criterion modelled in separate rows:

- Criterion 1 (5 extra days) is split over rows 2-4
- Criterion 2 (3 extra days) is split over rows 5-6
- Criterion 3 (2 extra days) is split over rows 7-8

The first rule of the table represents the default number of vacation days that every employee gets, regardless of the other criteria. This makes the decision table intuitive to interpret, as there is a direct connection between specification and model.

One downside of this model is that the modelling for criterion 3 does not correspond 100% to the original rule, because of the constraint that "the 2 extra days cannot be combined with the extra 5 days". To see this difference, consider row 8: while the description states that "these 2 days are also provided for employees of age 45 or more", the row also contains an explicit upper bound of 60 years, to ensure it does not overlap with row 3. This can be solved by splitting the table in three separate tables, as demonstrated in the cDMN documentation[2].

In the FO(·) solution by M1 (Listing C.2.6), the criteria are encoded in a similar fashion to the *C+* table. Here, the vacation days are calculated as the sum of multiple *if-then-else* statements, with each if-then-else representing one criterion. Note that this representation does not suffer from the aforementioned issue, as the modeller cleverly *nested* criterion 3 in criterion 1. In other words, the criterion for the two extra days is only considered if they are explicitly ineligible for the extra five days.

Listing C.2.6: FO(·) solution by **M1**

```
vocabulary {
    age : () → Int
    service_years : () → Int
    vacation_days : () → Int
}

theory {
    vacation_days() = 22
    + (if age() <18 ∨ age() ≥ 60 ∨ service_years() ≥ 30 then 5 else
       if age() ≥ 45 ∨ service_years() ≥ 15 then 2 else 0)
    + (if age() ≥60 ∨ service_years() ≥ 30 then 3 else 0).
}
```

[2] https://cdmn.readthedocs.io/en/stable/Examples/vacation_days.html

C.2.4 Planning Problem

For the fourth problem, all modellers chose to use FO(·). While their solutions are roughly the same in many ways, we did see three very different ways to represent the time points of the planning.

- M2 created a type *Slot*, constructed from types *Day* and *Segment* (Listing C.2.7)
- M3 created one type *days*, containing elements to represent both day and day half (Listing C.2.8)
- M4 created *Days* and *Segment*, and uses both of them together in the other symbol declarations (Listing C.2.9)

This difference can be seen in the listings below. To save on space, we only include two rules in the theories, namely "If a person is not available, they cannot be scheduled" and "Emma must be assigned first-aid on Wednesday afternoon". These shorter theories are already good examples of how there are typically multiple ways to model a problem, each with their own pros and cons:

1. Constructed type (M2, Listing C.2.7)

   Pros. If a new segment is added, the formulas stay the same.
   Requires only one quantification to reason over time points.

   Cons. Arguably more complex to declare and understand for non-experts.
   Less natural when describing formulas over specific time points.

2. Single type (M3, Listing C.2.8)

   Pro: Very straightforward and easy to understand.
   Requires only one quantification to reason over time points.

   Con: Less natural when distinction between day and segment is required.
   If a new segment is added, the entire *days* type needs updating.

3. Separate *Day* and *Segment* (M4, Listing C.2.9)

   Pro: Easy to use and flexible.
   Relatively natural.

   Con: Quantifying over segments requires two quantifications.
   Other symbol declarations require two types as domain instead of one.

Knowing the impact of each design decision and which trade-offs to make requires intuition, which can only truly be gained through experience. This relates back to the Modeller theme (Chapter 8.4.2).

Listing C.2.7: partial FO(·) solution by **M2**

```
vocabulary V {
    type Day := {Mon, Tue, Wedn, Thurs, Fri, Sat, Sun}
    type Segment := {AM, PM}
    type Slot := constructed from {slot(Day, Segment)}
    type Person := {William, Julie, Alex, Emma, Sophie}

    availableOn : Person × Slot → 𝔹
    firstAid, backup: Slot → Person
}

theory T:V {
    ∀p in Person, ∀s in Slot: availableOn(p, s) ⇐ firstAid(s) = p ∨
    backup(s) = p.
    firstAid(slot(Wedn, PM)) = Emma.
}
```

Listing C.2.8: partial FO(·) solution by **M3**

```
vocabulary {
    type person := {Emma, William, Sophie, Julie, Alex}
    type days := {MondayAM,  MondayPM, TuesdayAM, TuesdayPM, WednesdayAM
    , WednesdayPM, ThursdayAM, ThursdayPM, FridayAM, FridayPM}
    FirstAid, Backup : (days) → person
    Available : (days × person) → 𝔹
}
theory {
    ∀p in person, d in days: ¬Available(d,p) ⇒ ¬(FirstAid(d) = p ∨
    Backup(d) = p).
    FirstAid(WednesdayPM) = Emma.
}
```

Listing C.2.9: partial FO(·) solution by **M4**

```
vocabulary {
    type Person := {William, Julie, Alex, Emma, Sophie}
    type Days := {Monday, Tuesday, Wednesday, Thursday, Friday}
    type Segment := {Morning, Afternoon}
    first_aid: Days × Segment → Person
    back_up: Days × Segment → Person
    available: Person × Days × Segment → 𝔹
}
theory {
```

```
    ∀p in Person, d in Days, s in Segment: (first_aid(d, s) = p ∨
    back_up(d, s) = p) ⇒ available(p, d, s).
    first_aid(Wednesday, Afternoon) = Emma.
}
```

C.3 Modelling Interview Codes

Table C.3: Discovered codes per expert modeller, and the data saturation.

Code	M1	M2	M3	M4
KB closely aligned to NL	✓	✓		✓
Compactness	✓	✓		✓
Definitions	✓	✓		
Familiarity	✓	✓	✓	
KISS	✓		✓	✓
Readability AI expert	✓	✓	✓	
Readability domain expert	✓	✓	✓	
No focus on efficiency	✓			✓
Expressiveness <> user-friendliness	✓	✓	✓	✓
Validation	✓		✓	✓
Verification	✓			✓
Explainability		✓		
Extensibility		✓		
Maintainability		✓		
Natural reading		✓		
Simplicity		✓		
Automatically to FO(·)			✓	✓
Conservative notation			✓	
Recognizable structure			✓	✓
Clear separation in knowledge			✓	✓
Structure in knowledge			✓	✓
Low in syntax			✓	✓
Visual			✓	✓
Intuitive				✓
Defined scope of formalism				✓
Total codes	11	12	13	15
Data saturation	64%	88%	92%	100%

Table C.4: A "code book" elaborating on the meaning of each code found during the expert modeller interviews.

Code	Explanation
KB closely aligned to NL	The formalised knowledge corresponds clearly to the original description.
Compactness	The compactness of the representation.
Definitions	Usage of logical definitions in the KB.
Familiarity	The familiarity of an expert w.r.t. a formalism.
KISS	Keep It Simple, Stupid
Readability AI expert	How readable is the KB for AI experts?
Readability domain expert	How readable is the KB for non-AI experts?
No focus on efficiency	Efficient execution is not a modelling focus.
Expressiveness <> user-friendliness	The expressiveness vs user-friendliness trade-off.
Validation	Validation of the KB.
Verification	Verification of the KB.
Explainability	Explainability of the KB.
Extensibility	Extensibility of the KB.
Maintainability	Maintainability of the KB.
Natural reading	The notation has a "natural" interpretation.
Simplicity	Simplicity of the modelling process.
Automatically to FO(·)	A notation can automatically be translated into FO(·).
Conservative notation	The notation does not change much.
Recognizable structure	The notation uses a common structure.
Clear separation in knowledge	The KB is separated into smaller parts, each with their own role.
Structure in knowledge	The KB is well-structured.
Low in syntax	The formalism does not have much syntax.
Visual	The formalism is visual in nature.
Intuitive	The formalism is intuitive.
Defined scope of formalism	The formalism was designed for one or more specific type of problems.

Bibliography

[1] Thriving in a changing world. *Securities Finance Times*, 319 (Jan. 2022), 36–38.

[2] User-friendliness. In *Cambridge Dictionary*. Cambridge University Press, Cambridge, 2023.

[3] Aerts, B. *Advanced Applications of Knowledge Representation Technology*. PhD thesis, KU Leuven, 2020.

[4] Aerts, B., Deryck, M., and Vennekens, J. Knowledge-based decision support for machine component design: A case study. *Expert Systems with Applications 187* (Jan. 2022), 115869.

[5] Aerts, B., Vandevelde, S., and Vennekens, J. Tackling the DMN challenges with cDMN: A tight integration of DMN and constraint reasoning. In *Rules and Reasoning*, Lecture Notes in Computer Science. Springer International Publishing, Cham, 2020, pp. 23–38.

[6] Aerts, B., and Vennekens, J. An application of logic-based methods to machine component design. vol. 64 of *OASIcs : OpenAccess Series in Informatics*, Palù, Alessandro Dal, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, pp. 13:1–13:15.

[7] Antkiewicz, M., Bąk, K., Murashkin, A., Olaechea, R., Liang, J., and Czarnecki, K. Clafer tools for product line engineering. In *Software Product Line Conference* (Tokyo, Japan, 2013).

[8] Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., and Wasowski, A. Clafer: Unifying Class and Feature Modeling. *Software and Systems Modeling* (2014).

[9] Barrett, C., Stump, A., and Tinelli, C. The SMT-LIB Standard: Version 2.0. In *Proceedings of the 8th International Workshop on*

Satisfiability Modulo Theories (Edinburgh, UK) (2010), A. Gupta and D. Kroening, Eds.

[10] Barwise, J., and Etchemendy, J. The Language of First-Order Logic. In *Center for the Study of Language and Information* (1991).

[11] Barwise, J., and Etchemendy, J. Hyperproof: Logical reasoning with diagrams. In *Working Notes of the AAAI Spring Symposium on Reasoning with Diagrammatic Representations* (1992), pp. 80–84.

[12] Batory, D. Feature models, grammars, and propositional formulas. In *Software Product Lines* (Berlin, Heidelberg, 2005), Springer Berlin Heidelberg, pp. 7–20.

[13] Batoulis, K., and Weske, M. A tool for checking soundness of decision-aware business processes. In *BPM (demos)* (2017), pp. 1–5.

[14] Bazhenova, E., Zerbato, F., Oliboni, B., and Weske, M. From BPMN process models to DMN decision models. *Information Systems 83* (2019), 69–88. Publisher: Elsevier.

[15] Bell, T., and Vahrenhold, J. CS Unplugged—How Is It Used, and Does It Work? In *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, H.-J. Böckenhauer, D. Komm, and W. Unger, Eds. Springer International Publishing, Cham, 2018, pp. 497–521.

[16] Benavides, D., Martín-Arroyo, P. T., and Cortés, A. R. Using Constraint Programming to Reason on Feature Models. In *SEKE* (2005).

[17] Benavides, D., Segura, S., and Ruiz-Cortés, A. Automated analysis of feature models 20 years later: A literature review. *Information Systems 35*, 6 (Sept. 2010), 615–636.

[18] Benavides, D., Trinidad, P., and Ruiz-Cortés, A. Automated reasoning on feature models. In *Advanced Information Systems Engineering* (Berlin, Heidelberg, 2005), Springer Berlin Heidelberg, pp. 491–503.

[19] Biard, T., Le Mauff, A., Bigand, M., and Bourey, J.-P. 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* (Cham, 2015), L. M. Camarinha-Matos, F. Bénaben, and W. Picard, Eds., Springer International Publishing, pp. 489–496.

[20] Boehm, B. W. Verifying and Validating Software Requirements and Design Specifications. *IEEE Software 1*, 1 (Jan/Feb 1984), 75–88.

[21] Bornet, P. Is decision intelligence the new ai?

[22] Botterweck, G., Janota, M., and Schneeweiss, D. A design of a configurable feature model configurator. In *Proceedings, Third International Workshop on Variability Modelling of Software-intensive Systems, held from January 28 - 30, at the University of Sevilla, Spain* (2009), pp. 165 – 168.

[23] Botterweck, G., Thiel, S., Nestor, D., bin Abid, S., and Cawley, C. Visual tool support for configuring and understanding software product lines. In *2008 12th International Software Product Line Conference* (2008), pp. 77–86.

[24] bpmn.io. Dmn viewer and editor, 2015.

[25] Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., and Schaub, T. Asp-core-2: Input language format. *ASP Standardization Working Group* (2012).

[26] Calvanese, D., Dumas, M., Laurson, Ü., Maggi, F. M., Montali, M., and Teinemaa, I. Semantics and analysis of dmn decision tables. In *Business Process Management* (Cham, 2016), M. La Rosa, P. Loos, and O. Pastor, Eds., Springer International Publishing, pp. 217–233.

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

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

[29] Calvanese, D., Montali, M., Dumas, M., and Maggi, F. M. Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge. *Theory and practice of logic programming 19*, 4 (2019), 536–573.

[30] Camunda Services GmbH. Camunda DMN Decision Engine. `https://camunda.com/`, 2013 – 2021.

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

[32] Carbonnelle, P., Aerts, B., Deryck, M., Vennekens, J., and Denecker, M. An interactive consultant. In *Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (BENELEARN 2019), Brussels, Belgium, November 6-8, 2019* (2019), vol. 2491 of *CEUR workshop proceedings*, CEUR-WS.org.

[33] Carbonnelle, P., and Denecker, M. FO-dot, 2022.

[34] Carbonnelle, P., and Vandevelde, S. IDP-Z3's Reference Manual, 2023.

[35] Carbonnelle, P., Vandevelde, S., Vennekens, J., and Denecker, M. Interactive configurator with FO(.) and IDP-Z3, 2023.

[36] Corbin, J. M., and Strauss, A. Grounded theory research: Procedures, canons, and evaluative criteria. 3–21.

[37] Corea, C., Blatt, J., and Delfmann, P. A tool for decision logic verification in dmn decision tables. In *BPM (PhD/Demos)* (2019), pp. 169–173.

[38] Czarnecki, K., Helsen, S., and Eisenecker, U. Staged Configuration Using Feature Models. In *Software Product Lines* (Berlin, Heidelberg, 2004), Springer Berlin Heidelberg, pp. 266–283.

[39] da Silva, L. F. M., Öchsner, A., and Adams, R. D. *Handbook of adhesion technology*, vol. 1. Springer, 2018.

[40] Dasseville, I., Janssens, L., Janssens, G., Vanthienen, J., and Denecker, M. Combining DMN and the knowledge base paradigm for flexible decision enactment. vol. 1620 of *Supplementary Proceedings of the RuleML 2016 Challenge*, Athan, Tara, CEUR-WS.org.

[41] Dauxais, Y., Gautrais, C., Dries, A., Jain, A., Kolb, S., Kumar, M., Teso, S., Van Wolputte, E., Verbruggen, G., and De Raedt, L. Synthlog: A language for synthesising inductive data models (extended abstract). In *Machine Learning and Knowledge Discovery in Databases* (Cham, 2020), P. Cellier and K. Driessens, Eds., Springer International Publishing, pp. 102–110.

[42] De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., and Denecker, M. Predicate logic as a modeling language: the IDP system. In *Declarative Logic Programming: Theory, Systems, and Applications*, M. Kifer and Y. A. Liu, Eds. ACM, Sept. 2018, pp. 279–323.

[43] De Raedt, L., Kimmig, A., and Toivonen, H. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI* (2007), vol. 7, Hyderabad, pp. 2462–2467.

[44] Denecker, M., and Vennekens, J. Building a Knowledge Base System for an Integration of Logic Programming and Classical Logic. In *Logic Programming*, M. Garcia de la Banda and E. Pontelli, Eds., vol. 5366. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 71–76. Series Title: Lecture Notes in Computer Science.

[45] Deryck, M. *Knowledge Base Systems in Practice: Approaches, Application Areas and Limitations*. PhD thesis, KU Leuven, Leuven, 2022.

[46] Deryck, M., Aerts, B., and Vennekens, J. Adding constraint tables to the dmn standard: Preliminary results. In *Rules and Reasoning: Third International Joint Conference, RuleML+RR 2019, Bolzano, Italy, September 16–19, 2019, Proceedings* (2019), vol. 11784, Springer, pp. 171–179.

[47] Deryck, M., Comenda, N., Coppens, B., and Vennekens, J. Combining logic and natural language processing to support investment management. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* (2021), vol. 18, pp. 666–670.

[48] Deryck, M., Devriendt, J., Marynissen, S., and Vennekens, J. Legislation in the knowledge base paradigm: interactive decision enactment for registration duties. In *Proceedings of the 13th IEEE Conference on Semantic Computing* (2019), IEEE, pp. 174–177.

[49] Devriendt, J. ManyWorlds: A Combinatorial Programming Language. Nonfiction Software.

[50] Dos Martires, P. Z., Dries, A., and De Raedt, L. Exact and approximate weighted model integration with probability density functions using knowledge compilation. *Proceedings of the AAAI Conference on Artificial Intelligence 33*, 01 (Jul. 2019), 7825–7833.

[51] Elfaki, A. O., Phon-Amnuaisuk, S., and Ho, C. K. Using first order logic to validate feature model. In *VaMoS* (2009), pp. 169–172.

[52] Estrada-Torres, B., del-Río-Ortega, A., and Resinas, M. DemaBot: A Tool to Automatically Generate Decision-Support Chatbots. In *BPM 2021 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track* (Rome, Italy, Oct. 2021), vol. 2973, CEUR Workshop Proceedings, pp. 1–5.

[53] Etikala, V., Goossens, A., Van Veldhoven, Z., and Vanthienen, J. Automatic generation of intelligent chatbots from DMN decision models. In *Rules and Reasoning* (Cham, 2021), S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, and D. Roman, Eds., Springer International Publishing, pp. 142–157.

[54] Ewen J.C., K. Key issues in selecting the right adhesive. In *Advances in structural adhesive bonding*, D. A. Dillard, Ed., Woodhead publishing in materials. Woodhead Publishing, 2010, pp. 3–19.

[55] Fan, S., and Zhang, N. Feature model based on description logics. In *Knowledge-Based Intelligent Information and Engineering Systems* (Berlin, Heidelberg, 2006), Springer Berlin Heidelberg, pp. 1144–1151.

[56] Fraser, N. Ten things we've learned from Blockly. In *2015 IEEE Blocks and beyond Workshop (Blocks and Beyond)* (2015), pp. 49–50.

[57] Fuchs, N. E., Kaljurand, K., and Kuhn, T. Attempto Controlled English for Knowledge Representation. In *Reasoning Web: 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*, C. Baroglio, P. A. Bonatti, J. Małuszyński, M. Marchiori, A. Polleres, and S. Schaffert, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 104–124.

[58] Fung, P., O'Shea, T., Goldson, D., Reeves, S., and Bornat, R. Computer tools to teach formal reasoning. *Computers & Education 27*, 1 (Aug. 1996), 59–69.

[59] Galindo, J. A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A.-M., and Ruiz-Cortés, A. Automated analysis of feature models: Quo vadis? *Computing. Archives for Scientific Computing 101*, 5 (2019), 387–433. Publisher: Springer.

[60] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Marín-Jiménez, M. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition 47*, 6 (June 2014), 2280–2292.

[61] Goldson, D., Reeves, S., and Bornat, R. A Review of Several Programs for the Teaching of Logic. *The Computer Journal 36*, 4 (Jan. 1993), 373–386.

[62] Gonzalez, A. J., and Barr, V. Validation and verification of intelligent systems - what are they and how are they different? *Journal of Experimental & Theoretical Artificial Intelligence 12*, 4 (Oct. 2000), 407–420.

[63] Goossens, A., Maes, U., Timmermans, Y., and Vanthienen, J. Automated intelligent assistance with explainable decision models in knowledge-intensive processes. In *Business Process Management Workshops* (Cham, 2023), C. Cabanillas, N. F. Garmann-Johnsen, and A. Koschmider, Eds., Springer International Publishing, pp. 25–36.

[64] Gross, A., Kracher, B., Kraus, J. M., Kühlwein, S. D., Pfister, A. S., Wiese, S., Luckert, K., Pötz, O., Joos, T., Van Daele, D., De Raedt, L., Kühl, M., and Kestler, H. A. Representing dynamic biological networks with multi-scale probabilistic models. *Communications Biology 2*, 1 (Jan 2019), 21.

[65] Guest, G., Namey, E., and Chen, M. A simple method to assess and report thematic saturation in qualitative research. e0232076.

[66] Gutmann, B., Thon, I., and De Raedt, L. Learning the parameters of probabilistic logic programs from interpretations. In *Machine Learning and Knowledge Discovery in Databases* (Berlin, Heidelberg, 2011), D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, Eds., Springer Berlin Heidelberg, pp. 581–596.

[67] Hasic, F., Corea, C., Blatt, J., Delfmann, P., and Serral Asensio, e. A Tool for the Verification of Decision Model and Notation (DMN) Models. In *Proceedings of the 2020 14th International Conference on Research Challenges in Information Science (RCIS)* (2020), Springer, pp. 1–6.

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

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

[70] Hasić, F., De Smedt, J., and Vanthienen, J. Augmenting processes with decision intelligence: Principles for integrated modelling. *Decision Support Systems 107* (2018), 1–12. Publisher: Elsevier.

[71] Hatzilygeroudis, I., and Perikos, I. A Web-Based Interactive System for Learning NL to FOL Conversion. vol. 226. July 2009, pp. 297–307.

[72] Jordens, J., Vandevelde, S., Van Doninck, B., Witters, M., and Vennekens, J. Adhesive selection via an interactive, user-friendly system based on symbolic AI. In *Proceedings of CIRP Design 2022* (2022).

[73] Jordens, J., Vandevelde, S., Van Doninck, B., Witters, M., and Vennekens, J. Adhesive selection via an interactive, user-friendly system based on symbolic AI. In *Proceedings of CIRP DESIGN 2022* (2022), Procedia CIRP.

[74] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Inst, 1990.

[75] Kannan, T., and Prabu, S. S. Expert system for selection of adhesives. In *Proceedings of the Recent Developments in Materials Processing Conference* (2004).

[76] Karataş, A. S., Oğuztüzün, H., and Doğru, A. From extended feature models to constraint logic programming. *Science of Computer Programming 78*, 12 (2013), 2295–2312. Publisher: Elsevier.

[77] Kowalski, R. A. Logic as a computer language for children. In *ECAI* (1982), pp. 2–10.

[78] Kuhn, T. *Controlled English for Knowledge Representation*. PhD thesis, University of Zurich, 2010.

[79] Kuhn, T. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics 40*, 1 (Mar. 2014), 121–170.

[80] Lammel, C., and Dilger, K. Software for a rule-based adhesive-selection system. 42–43. Publisher: Business News Publishing Company.

[81] Lane, D. *Machine Learning for Kids: A Project-Based Introduction to Artificial Intelligence*. No Starch Press, 2021.

[82] Laurson, U., and Maggi, F. M. A Tool for the Analysis of DMN Decision Tables. In *BPM (Demos)* (2016), pp. 56–60.

[83] Le, V.-M., Felfernig, A., Uta, M., Benavides, D., Galindo, J., and Tran, T. N. T. DIRECTDEBUG: Automated testing and debugging of feature models. In *2021 IEEE/ACM 43rd international conference on software engineering: New ideas and emerging results (ICSE-NIER)* (2021), pp. 81–85.

[84] Lees, W., and Selby, P. The PAL program mark II. 120–125. Publisher: Elsevier.

[85] Lodder, J., Jeuring, J., and Passier, H. An interactive tool for manipulating logical formulae. *Technical Report UU-CS 2006* (2006).

[86] Lomax, J., Haswell, C., and Carter, L. Clearstream and Intelli-Select target Q1 23 for collateral management automation release. *Securities Finance Times*, 314 (Oct. 2022), 16–18.

[87] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. The scratch programming language and environment. *ACM Transactions on Computing Education 10*, 4 (Nov. 2010).

[88] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems* (2018), S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc.

[89] Mannion, M. Using First-Order Logic for Product Line Model Validation. In *Software Product Lines* (Berlin, Heidelberg, 2002), Springer, pp. 176–187.

[90] Marković, Đ., Vandevelde, S., Vennekens, J., and 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* (Berlin, Germany, Mar. 2022), vol. 3229, CEUR Workshop Proceedings, pp. 1–15.

[91] Mauco, M. V., Ferrante, E., et al. Clausula: A Didactic Tool to Teach First Order Logic. *Proceedings de ISECON* (2009).

[92] Mauco, V., Ferrante, E., and Felice, L. Educational software for first order logic semantics in introductory logic courses. *Information Systems Education Journal 12*, 6 (2014), 15.

[93] Mazo, R., Lopez-Herrejon, R. E., Salinesi, C., Diaz, D., and Egyed, A. Conformance checking with constraint logic programming: The case of feature models. In *2011 IEEE 35th Annual Computer Software and Applications Conference* (2011), IEEE, pp. 456–465.

[94] McCarney, R., Warner, J., Iliffe, S., Van Haselen, R., Griffin, M., and Fisher, P. The Hawthorne Effect: a randomised, controlled trial. 1–8.

[95] Meyler, K. L., and Brescia, J. A. Design of a computer expert system for adhesive selection using artificial intelligence techniques. Tech. rep., Army armament research development and engineering center Picatinny Arsenal, 1993.

[96] Mishra, S. Product configuration in answer set programming. In *Proceedings 37th International Conference on Logic Programming, ICLP Technical Communications 2021, Porto (Virtual Event), 20-27th September 2021* (2021), vol. 345 of *EPTCS*, pp. 296–304.

[97] Moseley, L., and Cartwright, M. The development of an expert system for operational use in the selection of industrial adhesives. 319–328. Publisher: Elsevier.

[98] Moura, L. d., and Bjørner, N. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems* (2008), Springer, pp. 337–340.

[99] Myllärniemi, V., Tiihonen, J., Raatikainen, M., and Felfernig, A. Using Answer Set Programming for Feature Model Representation and Configuration. In *Configuration Workshop* (2014), pp. 1–8.

[100] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings* (2007), C. Bessiere, Ed., vol. 4741 of *Lecture Notes in Computer Science*, Springer, pp. 529–543.

[101] Northrop, L. Software product lines essentials. Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE INST, 2008.

[102] Object Management Group. Decision model and notation, 2020.

[103] OpenRules, Inc. Openrules, 2017.

[104] Persson, A., Zuidberg Dos Martires, P., De Raedt, L., and Loutfi, A. Semantic relational object tracking. *IEEE Transactions on Cognitive and Developmental Systems 12*, 1 (2020), 84–97.

[105] Progress. Corticon, 2019.

[106] Pygame Project. Pygame 2.1.2, dec 2021.

[107] Rauber, M. F., and Gresse von Wangenheim, C. Assessing the Learning of Machine Learning in K-12: A Ten-Year Systematic Mapping. *Informatics in Education* (2022).

[108] RB, A., and HH, V. Expert-system selects adhesives for composite-material joints. 16–19. Publisher: Intertec Publishing Corp. PRIMEDIA Co., Chicago, USA.

[109] Reeves, S. A teaching and support tool for building formal models of graphical user-interfaces. In *Proceedings 1996 International Conference Software Engineering: Education and Practice* (1996), IEEE, pp. 98–105.

[110] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch: Programming for all. *Communications of The Acm 52*, 11 (Nov. 2009), 60–67.

[111] Richards, K. A. R., and Hemphill, M. A. A Practical Guide to Collaborative Qualitative Data Analysis.

[112] Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., and Bontemps, Y. Generic semantics of feature diagrams. *Computer Networks 51*, 2 (2007), 456–479.

[113] Schwitter, R. The jobs puzzle: Taking on the challenge via controlled natural language processing. *Theory and Practice of Logic Programming 13*, 4-5 (2013), 487–501.

[114] Signavio GmbH. Signavio Process Manager. `https://www.signavio.com/`, 2009 – 2021.

[115] Smit, K., Zoet, M., and Berkhout, M. Verification capabilities for business rules management in the Dutch governmental context. In *2017 international conference on research and innovation in information systems (ICRIIS)* (2017), pp. 1–6. tex.organization: IEEE.

[116] Sooter, L. J., Hasley, S., Lario, R., Rubin, K. S., and Hasić, F. Modeling a clinical pathway for contraception. *Applied clinical informatics 10*, 5 (October 2019), 935—943.

[117] Sree-Kumar, A., Planas, E., and Clarisó, R. Validating feature models with respect to textual product line specifications. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems* (New York, NY, USA, 2021), VaMoS'21, Association for Computing Machinery.

[118] Su, Y., Srihari, K., and Adriance, J. A knowledge update mechanism for an adhesive advisor. 111–114. Publisher: Elsevier.

[119] Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. FeatureIDE: An extensible framework for feature-oriented software development. *Experimental Software and Toolkits (EST 4) 79* (Jan. 2014), 70–85.

[120] Van den Broeck, G., Thon, I., van Otterlo, M., and De Raedt, L. Dtproblog: A decision-theoretic probabilistic prolog. *Proceedings of the AAAI Conference on Artificial Intelligence 24*, 1 (Jul. 2010), 1217–1222.

[121] Van Dessel, K., Devriendt, J., and Vennekens, J. FOLASP: FO(·) as Input Language for Answer Set Solvers. *Theory and practice of logic programming 21*, 6 (2021), 785–801.

[122] van Drooge, L., and de Jong, S. Valorisation: researchers already do much more than they realise.

[123] Vandevelde, S. User-friendly probabilistic decision logic modeling, 2023.

[124] Vandevelde, S., Aerts, B., and Vennekens, J. cDMN: Combining DMN with constraint reasoning, 2020.

[125] Vandevelde, S., Aerts, B., and Vennekens, J. Tackling the DM challenges with cDMN: A tight integration of DMN and constraint reasoning. Publisher: Cambridge University Press (CUP).

[126] Vandevelde, S., Callewaert, B., and Vennekens, J. Context-aware verification of DMN. Proceedings of the 55th Hawaii International Conference on System Sciences.

[127] Vandevelde, S., Callewaert, B., and 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* (2022), pp. 209–216.

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

[129] Vandevelde, S., Jordens, J., Van Doninck, B., Witters, M., and Vennekens, J. Knowledge-based support for adhesive selection. In *Logic Programming and Nonmonotonic Reasoning* (Cham, 2022), G. Gottlob, D. Inclezan, and M. Maratea, Eds., Springer International Publishing, pp. 445–455.

[130] Vandevelde, S., and Vennekens, J. A multifunctional, interactive DMN decision modelling tool. In *Proceedings of BNAIC/BeneLearn 2020* (2020), pp. 399–400.

[131] Vandevelde, S., and Vennekens, J. Unlocking the full potential of DMN, Sept. 2021.

[132] Vandevelde, S., and Vennekens, J. FOLL-E: Teaching First Order Logic to Children. *Proceedings of the AAAI Conference on Artificial Intelligence 37*, 13 (June 2023), 15869–15876.

[133] Vandevelde, S., Verreet, V., De Raedt, L., and Vennekens, J. A table-based representation for probabilistic logic: Preliminary results. *arXiv preprint arXiv:2110.01909* (2021).

[134] Vanthienen, J. Challenge Jan-2016: Decision Table for Vacation Days Calculation, Jan. 2016.

[135] Vanthienen, J., and Dries, E. Illustration of a decision table tool for specifying and implementing knowledge based systems. *International Journal on Artificial Intelligence Tools 3* (1994), 267–288.

[136] Vennekens, J. Lowering the learning curve for declarative programming: A python API for the IDP system. vol. 10137 of *Practical Aspects Of Declarative Languages (PADL 2017)*, Springer Verlag, pp. 86–102.

[137] Vennekens, J., Denecker, M., and Bruynooghe, M. FO(ID) as an extension of DL with rules. *Annals of Mathematics and Artificial Intelligence 58*, 1 (2010), 85–115.

[138] Vermeulen, L. Statische Code Analyse voor FO(.). Master's thesis, KU Leuven, Sint-Katelijne-Waver, Belgium, 2022.

[139] White, C., and Schwitter, R. An update on PENG light. In *Proceedings of the Australasian Language Technology Association Workshop 2009* (2009), pp. 80–88.

Publications

Journal Articles

[1] Vandevelde, S., Aerts, B., Vennekens, J. (2021). Tackling the DM Challenges with cDMN: A Tight Integration of DMN and Constraint Reasoning. Theory And Practice Of Logic Programming, 1-24. doi: 10.1017/S1471068421000491

Journal Articles (Accepted)

[1] Vandevelde, S., Jordens, J., Van Doninck, B., Witters, M., Vennekens, J. (2023). Knowledge-Based Support for Adhesive Selection: Will it Stick?. Theory And Practice Of Logic Programming.

Conference Proceedings

[1] Goossens, A., Vandevelde, S., Vanthienen, J., Vennekens, J. with Goossens, A. (joint first author), Vandevelde, S. (joint first author) (2023). GPT-3 for Decision Logic Modeling. In: Proceedings of the 17th International Rule Challenge and 7th Doctoral Consortium @ RuleML+RR 2023 co-located with 19th Reasoning Web Summer School (RW 2023) and 15th DecisionCAMP 2023 as part of Declarative AI 2023: vol. 3485, (1-14). Presented at the RuleML+RR Challenge and Doctoral Consortium, Oslo, Norway, 18 Sep 2023-20 Sep 2023.

[2] Vandevelde, S., Vennekens, J. (2022). FOLL-E: Teaching First-Order Logic to Children. In: Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence. Presented at

the Thirteenth AAAI Symposium on Educational Advances in Artificial Intelligence, Washington, D.C., USA, 11 Feb 2023-12 Feb 2023.

[3] Markovic, D., Vandevelde, S., Vennekens, J., Denecker, M., Bruynooghe, M. (contr.) (2022). On the semantics of "null" in DMN: Undefined is not unknown. In: Proceedings of RuleML+RR 2022. Virtual, 26 Sep 2022-30 Sep 2022. Open Access

[4] Vandevelde, S., Callewaert, B., Vennekens, J. (2022). Interactive Feature Modeling with Background Knowledge for Validation and Configuration. In: SPLC '22: Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B, (209-216). Presented at the International Workshop on Configuration, Graz, Austria, 11 Sep 2022-16 Sep 2022. ISBN: 978-1-4503-9206-8. doi: 10.1145/3503229.3547039

[5] Carbonnelle, P., Vandevelde, S., Vennekens, J., Denecker, M. (2022). IDP-Z3: a reasoning engine for FO(.). arXiv. doi: 10.48550/arXiv.2202.00343

[6] Vandevelde, S., Jordens, J., Van Doninck, B., Witters, M., Vennekens, J. with Vandevelde, S. (corresp. author) (2022). Knowledge-Based Support for Adhesive Selection. In: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Logic programming and nonmonotonic reasoning, (445-455). Presented at the International Conference on Logic Programming and Nonmonotonic Reasoning, Genova, Italy, 05 Sep 2022-08 Sep 2022. ISBN: 978-3-031-15706-6. doi: 10.1007/978-3-031-15707-3_34

[7] Jordens, J., Vandevelde, S., Van Doninck, B., Witters, M., Vennekens, J. (2022). Adhesive Selection via an Interactive, User-friendly System based on Symbolic AI. In: Procedia CIRP: vol. 109, (161-166). Presented at the CIRP DESIGN 2022, Paris Saclay, 28 Mar 2022-30 Mar 2022. doi: 10.1016/j.procir.2022.05.230.

[8] Vandevelde, S., Callewaert, B., Vennekens, J. (2022). Context-Aware verification of DMN. In: Proceedings of the 55th Hawaii International Conference on System Sciences, (1-8). Presented at the Hawaii International Conference on System Sciences, Manoa, Hawaii, 04 Jan 2022-07 Jan 2022. ISBN: 978-0-9981331-5-7.

[9] Vandevelde, S., Etikala, V., Vanthienen, J., Vennekens, J. (2021). Leveraging the Power of IDP with the Flexibility of DMN: a Multifunctional API. In: Rules and Reasoning, (1-14). Presented at the RuleML+RR 2021, Leuven, Belgium, 13 Sep 2021-15 Sep 2021. ISBN: 978-3-030-91166-9.

[10] Vandevelde, S., Vennekens, J. (2021). ProbLife: a Probabilistic Game of Life. arXiv. doi: 10.48550/arXiv.2201.09521. Presented at BNAIC 2021, Esch-Sur-Alzette, Luxembourg, 10 Nov 2021.

[10] Vandevelde, S., Verreet, V., De Raedt, L., Vennekens, J. (2021). A Table-Based Representation for Probabilistic Logic: Preliminary Results. arXiv. doi: 10.48550/arXiv.2110.01909

[11] Aerts, B., Vandevelde, S., Vennekens, J. (2020). Tackling the DMN Challenges with cDMN: A Tight Integration of DMN and Constraint Reasoning. In: V. Gutiérrez Basulto, T. Kliegr, A. Soylu, M. Giese, D. Roman (Eds.), Proceedings of RuleML+RR 2020, (23-38). Presented at the RuleML+RR 2020, Virtual, 29 Jun 2020-01 Jul 2020. ISBN: 978-3-030-57977-7.

[12] Vandevelde, S., Van Dessel, K., Crauwels, H. (2019). An Interactive Knowledge Base Application for Group Assignment. In: http://ceur-ws.org/Vol-2491/: vol. 2491, (1-2). Presented at the Benelux Conference on Artificial Intelligence, Brussels, 06 Nov 2019-08 Nov 2019.

Conference Proceedings (Accepted)

[1] Markovic, D., Vandevelde, S., Vanbesien, L., Vennekens, J., Denecker, M. (2024) An epistemic logic for modeling decisions in the context of incomplete knowledge. Accepted at SAC 2024.

[2] Callewaert, B., Vandevelde, S., Comenda, N., Coppens, B., Decleyre, N., Vennekens, J. (2024) Extending Feature Models with Types. Accepted at HICSS 2024.

[3] Callewaert, B., Decleyre, N. Vandevelde, S., Comenda, N., Coppens, B., Vennekens, J. (2023) Facilitating Investment Strategy Negotiations through Logic. Accepted at IEEE CIFER 2023.

Other (Extended abstracts, demos, . . .)

[4] Vandevelde, S. (2023). User-friendly Probabilistic Decision Logic Modeling. Presented at the DecisionCAMP2023, Virtual, 18 Sep 2023-20 Sep 2023.

[1] Vandevelde, S., Vennekens, J. (2023). FOLL-E: Teaching First Order Logic to Children (Extended Abstract). Presented at ICLP 2023.

[2] Vandevelde, S., Aerts, B., Vennekens, J. (2023). Tackling the DM Challenges with cDMN: A Tight Integration of DMN and Constraint Reasoning (Extended Abstract). Presented at ICLP 2023.

[3] Vandevelde, S., Vennekens, J. (2022). FOLL-E: Teaching First Order Logic to Children (demo abstract). Demo given at BNAIC 2023.

[4] Vandevelde, S., Vennekens, J. (2021). Unlocking the Full Potential of DMN. Presented at the DecisionCAMP2021, Virtual, 13 Sep 2021-15 Sep 2021.

[5] Vandevelde, S., Vennekens, J. (2020). A Multifunctional, Interactive DMN Decision Modelling Tool. Presented at the BNAIC 2020, Virtual, 19 Nov 2020-20 Nov 2020.

[6] Vandevelde, S., Aerts, B., Vennekens, J. (2020). cDMN: Combining DMN with Constraint Reasoning. Presented at the DecisionCAMP 2020, Virtual, 29 Jun 2020-01 Jul 2020.

Awards

2022 | *Best Demo Award*, BNAIC 2022
For our demo *"FOLL-E: Teaching First-Order Logic to Children."*

2022 | *Best PhD Paper Award*, ConfWS 2022
For our paper *"Interactive Feature Modeling with Background Knowledge for Validation and Configuration."*

2020 | *Best Paper Award*, RuleML+RR 2020
For our paper *"Tackling the DMN Challenges with cDMN: A Tight Integration of DMN and Constraint Reasoning."*

FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
DTAI-EAVISE
Jan Pieter Denayerlaan 5
2860 Sint-Katelijne-Waver
s.vandevelde@kuleuven.be
https://www.simonvandevelde.be