IDP-Z3: a reasoning engine for FO(·)

Pierre Carbonnelle[1,3], Simon Vandevelde[2,3,4], Joost Vennekens[2,3,4], and Marc Denecker[1,3]

[1] KU Leuven, Dept. of Computer Science, Celestijnenlaan 200, 3001 Heverlee, Belgium
[2] KU Leuven, De Nayer Campus, Dept. of Computer Science
J.-P.-De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium
[3] Leuven.AI — KU Leuven Institute for AI, B-3000, Leuven, Belgium
[4] Flanders Make — DTAI-FET
{pierre.carbonnelle, s.vandevelde, joost.vennekens, marc.denecker}@kuleuven.be

Abstract. An important sign of intelligence is the capacity to apply a body of knowledge to a particular situation in order to not only derive new knowledge, but also to determine relevant questions or provide explanations. Developing interactive systems capable of performing such a variety of reasoning tasks for the benefits of their users has proved difficult, notably for performance and/or development cost reasons. Still, recently, a reasoning engine, called IDP3, has been used to build such systems, but it lacked support for arithmetic operations, seriously limiting its usefulness. We have developed a new reasoning engine, IDP-Z3, that removes this limitation, and we put it to the test in four knowledge-intensive industrial use cases.
This paper describes FO(·) (aka FO-dot), the language used to represent knowledge in the IDP3 and IDP-Z3 system. It then describes the generic reasoning tasks that IDP-Z3 can perform, and how we used them to build a generic user interface, called the Interactive Consultant. Finally, it reports on the four use cases.
In these four use cases, the interactive applications based on IDP-Z3 were capable of intelligent behavior of value to users, while having a low development cost (typically 10 days spread over involved parties) and an acceptable response time (typically below 3 seconds). Performance could be further improved, in particular for problems on larger domains.

1 Introduction

In [27], McCarthy presents four possible levels of use of logic in Artificial Intelligence. Intelligent machines at the first level, such as neural networks, do not use logic sentences at all. At the second level, machines use logic sentences to represent facts from which they reach conclusions using ad-hoc procedures, typically written in imperative programming languages, without the generality of ordinary logical inference.

Machines at the third level use logical deduction to reach conclusions. He cites Prolog as one of the languages used to program them. Such machines are rather specialized: "the facts of one program usually cannot be used in a database for other programs." This is a result of their fixed deduction strategy: because Algorithm = Logic + Control [26], one has to use a new set of logic rules to create an algorithm for a new task in the same problem domain.

By contrast, machines of the fourth and most advanced level do not have a fixed deduction strategy. The fourth level "involves representing general facts about the world as logical sentences. [..] The facts would have the neutrality of purpose characteristic of much human information. [..] A key problem for achieving the fourth level is to develop a language for a general common-sense database.".

The IDP-Z3 system seeks to address that challenge: it is designed so that a) one can express knowledge about possible worlds using logical sentences, and b) this knowledge can be used for many different computational tasks. To distinguish it from inference engines at the third level, we call it a "reasoning engine". Reasoning engines enable the Knowledge Base paradigm [18], in which systems store declarative domain knowledge, and use it to solve a variety of problems. This approach can significantly reduce the development and maintenance costs of intelligent machines [21].

In this paper, we present IDP-Z3 and various tools and extensions built around it. In particular, we demonstrate that IDP-Z3 allows users to leverage their domain knowledge to produce flexible interactive systems that offer all the necessary functionality and computational performance to handle real-world problems. There are several aspects to this:

- The FO(·) language is important to allow complex knowledge to be represented in a natural and elaboration-tolerant knowledge base.
- The modular and classical nature of FO(·) make it easy to extend a KB with parts that are written in more user-friendly notations such as DMN or Controlled Natural Language.
- The range and performance of the generic reasoning algorithms offered by IDP-Z3 suffices to implement a large class of interactive applications in an efficient way.

We present four knowledge-intensive use cases as evidence for our claims: they show that (1) real users are indeed able to participate in the construction of the KB, (2) that IDP-Z3 delivers significant value to these users, at low development costs, (3) that the IDP-Z3 system, while not the most efficient solver that exists, is able to deliver performance that suffices to handle the real-world instances that the users want to tackle.

We begin by elaborating on FO(·) and alternative formalisms in Section 2.1. Next, we present the IDP-Z3 engine and its features in Section 3, followed by Section 4 in which we expand on the Interactive Consultant, a generic, user-friendly interface to solve real-world problem using the reasoning power of IDP-Z3. As an empirical evaluation of the system, we report on four knowledge-intensive industrial use cases in Section 5, and demonstrate the benefits of creating interactive applications using IDP-Z3. Finally, we compare IDP-Z3 to other reasoning engines for model-based KR languages in Section 6, and conclude in Section 7.

In short, the contributions of this paper are:

- an overview of FO(·) and related formalisms;
- the presentation of IDP-Z3, a new reasoning engine;
- a summary of case studies involving IDP-Z3, to support our claims;
- a qualitative comparison between IDP-Z3 and other reasoning engines.

2 Knowledge representation

This section presents the languages used to represent knowledge in IDP-Z3-based systems: FO(·) and related formalisms.

2.1 FO(·)

FO(·) (aka FO-dot) is the Knowledge Representation language used by the IDP-Z3 reasoning engine. It was introduced in [15]. It is based on first-order logic (FOL) for its constructs ($\wedge, \vee, \neg, \Rightarrow, \forall, \exists$) and its model semantics. FO(·) extends FOL with a few language constructs to express complex information such as non-inductive, inductive and recursive definitions and aggregates. The syntax of the concrete logic used in IDP-Z3 is documented online[5].

An FO(·) Knowledge Base minimally consists of a vocabulary and a theory. The *vocabulary* describes the domain-specific ontology symbols that can be used in the theory. A *theory* is a collection of assertions about possible state of affairs. There are three classes of assertions: axioms, definitions and enumerations.

A Knowledge Base written in FO(·) cannot be *run*: it is just a "bag of information" formally describing models in a problem domain. This is a consequence of the FO(·) design goal to be *task-agnostic*. A corollary is that such a KB does not distinguish inputs from outputs, and allows reasoning in any direction.

A key advantage of the model-theoretic semantics is that it allows reasoning with incomplete knowledge of the state of affairs. When not much is known, many states of affairs are possible, and the theory has many models representing them. As more information is obtained, the set of models is reduced. This reduced set of models can be used to perform various forms of reasoning, e.g., to derive the consequences of what is known, or to find the model that maximizes a utility function.

As a very simple example, consider the voting law that states: "You have to vote in an election if you are at least 18 years old at election time (otherwise you can not)". The formula in FO(·) is:

```
vote() ⇔ 18 ≤ age().
```

If the age is known, the obligation to vote can be inferred; if the obligation to vote is known to be true instead, the age is known to be 18 or more, in any model.

We highlight the main features of FO(·) below.

Types Besides boolean, integer and real types, FO(·) allows the creation of custom types, e.g., `Person`. Predicates and functions are declared in the vocabulary, with a *type signature*, e.g., `weight: Person → Real`. In the quantified formula `∀x in T : p(x)`, `x` ranges over the extension of type `T`. Types are used to syntactically verify that formulae are *well-typed*, helping detect common errors. Types are also called "sorts" [38] in the literature.

[5] http://docs.idp-z3.be/en/latest/FO-dot.html

Axioms The first class of assertions in a FO(·) theory is the class of axioms. Axioms are logic sentences that are true in any possible state of affairs.

The voting law above is an example of axiom. A voting law that does not make voting mandatory can be expressed in another axiom using material implication:

```
vote() ⇒ 18 ≤ age().
```

(Notice that the voting obligations and permissions are represented without any modal operator, unlike formulations in deontic logic [37].)

(Inductive) definitions The second class of assertions in FO(·) is the class of (possibly inductive) definitions. Definitions are very useful forms of knowledge: they specify a unique interpretation of a defined symbol, given an interpretation of its parameters. For example, the transitive closure of a graph is uniquely defined for every graph.

It is well known that FOL cannot represent inductive definitions such as the transitive closure of a graph. By contrast, FO(·) can represent such definitions based on an extension of FOL for inductive definitions, called FO(ID) [15]. (Inductive) definitions in FO(ID) define a defined predicate P in terms of the parameters of its definition by specifying an iterative process to construct the interpretation of P from the interpretation of its parameters. However, consistent with the model-based approach, such definitions allow reasoning with any partial knowledge, in any direction. For example, it allows finding all the graphs that have a given transitive closure.

Definitions are often formulated in natural languages as a set of "rules" specifying necessary and sufficient conditions for the definiendum to hold. FO(·) definitions are also of this form, as illustrated in Listing 1.1. The definiens, i.e., the body of a rule, can be any FOL formula. The formalism of definitions in FO(·) is elaboration tolerant in the sense that one can easily add a rule to a definition.

Listing 1.1: Multi-rule definition

```
{can_drive() ← has_license() ∧ age() ≤ 85.
 can_drive() ← has_license() ∧ tested().
}
```

Because rules are part of definitions in FO(·), the head of a rule must be a single atom (in contrast to ASP which allows a disjunction in the head of so-called choice rules). Unlike in default logic [32], exceptions to a rule must be explicitly stated in the rule (possibly in the form of a predicate defined separately). Unlike in defeasible logic [29], rules do not have any priority ordering.

Data theory FOL is not well suited to express simple data about a concrete state of affairs. Unique Names (UNA), Domain Closure (DCA), and Completion (CA) Axioms are needed, but stating them explicitly is cumbersome. For example, the UNA consists of a number of axioms quadratic in the number of constant symbols.

To address this issue, FO(·) offers a third class of assertions, called enumerations, that specify the set of identifiers interpreting a type (under UNA and DCA), and the sets and functions interpreting predicate and function symbols (under DCA and CA).

For example, `Person := {Bob, Alice}` and `weight := {Bob → 80, Alice → 80}` specify the interpretation of type $Person$ and function $weight$. A collection of enumerations is called a `structure` in FO(·). A `structure` is a particular type of theory: a "data" theory.

Arithmetic FO(·) supports formulae with the 4 arithmetic operations on integers and reals (addition, substraction, multiplication and division), as well as the comparison operators (equality, disequality and inequalities).

Cardinality and Aggregates The number of elements of type T that satisfy a property `p` is formulated in FO(·) as:

```
#{el ∈ T: p(t)}
```

Similarly, `sum(lambda x in T: f(x))` denotes the sum of `f(x)` for each `x` in `T`. The minimum and maximum aggregates are denoted `min`, `max`. Cardinality and aggregate expressions can occur in axioms and in the body of rules.

`Concept` as a type Sometimes, it is necessary to reason about the concepts in an ontology. Informally, the concept behind an ontology symbol is its informal meaning. FO(·) introduces the type `Concept` (whose extension is the set of concepts in the ontology of the problem domain), and the "`$`" operator that maps a concept to its interpretation [8]. For example, one might want to count the number of symptoms that a person `p` has. This can be formulated as:

```
#{x ∈ Concept[Person→Bool]:
    Symptom(x) ∧ $(x)(p)}
```

This eliminates the need to reify the `Symptom` predicates, a technique that is not elaboration tolerant [28] because it requires rewriting every formula involving symptoms.

2.2 Related formalisms

Although FO(·) uses common mathematical notations, it is typically only usable by trained knowledge engineers. Several approaches have been developed to allow domain experts without such background to build their own KB, when the full power of FO(·) is not required. Knowledge in the simpler formalisms is then converted to FO(·) to allow reasoning using IDP-Z3.

Decision Model and Notation and cDMN The Decision Model and Notation (DMN) standard [30] is a notation for decision logic. Its goal is to be user-friendly, readable for everyone involved in the decision process (e.g., business people, IT experts, ...), and executable. In DMN, all logic is contained in *decision tables*: these represent an input-output relation between the input variables (left, in green) and the output variables (right, in blue). As an example, consider the table shown in Fig. 1, which defines a patient's BMILevel based on their BMI value. Each row of the table expresses a *decision rule*, which fires if the value of the input variable(s) matches the condition in the input cell(s).

BMILevel		
U	BMI	BMILevel
1	< 18.5	Underweight
2	$[18.5..25)$	Normal
3	$[25..30)$	Overweight
4	≥ 30	Obese

Fig. 1: DMN tables express knowledge in a user-friendly way.

In [10], decision tables were used as a way to represent the knowledge in an FO(·) KB. Indeed, decision tables can be seen as syntactic sugar for FO(·), allowing a user-friendly representation of definitions. For example, the decision table shown in Fig. 1 can be translated into FO(·) as follows:

```
{ BMILevel() = Underweight ← BMI() < 18.5.
  BMILevel() = Normal ← 18.5 ≤ BMI() < 25.
  ... }
```

This approach was further explored in [22] to formalize knowledge together with a domain expert. Vandevelde and Vennekens [36] present a tool capable of, among other things, automatically translating DMN into FO(·), thus further increasing the user-friendliness of DMN as an alternative modeling language for FO(·).

One downside of this approach however, is the limited expressiveness of decision tables. While sufficient for typical decision modeling, DMN is ill-suited to express more complex problems, e.g. that require constraints. As an attempt to overcome this issue, Vandevelde et al. [35] presents Constraint Decision Model and Notation (cDMN), which extends DMN with the ability to express constraints and related concepts, such as types, quantification, and more, while retaining the user-friendly tabular format. cDMN tables are also translatable to FO(·).

Controlled Natural Language Computational semantics [3] studies the translation of expressions in natural language into formal representations that allow reasoning. Often, a subset of natural language is considered, with a limited lexicon and grammar.

Claes et al. [9] uses this approach to build ZebraTutor, a semi-automated tool that solves logic grid puzzles given the clues in a simple natural language. The clues are translated to FO(·) using a typed version of the semantical framework described in [4].

The "Intelli-Select" use case, described in more detail in Section 5.3, is another example of this approach. Here, a tree-based grammar is created in advance to define the valid CNL sentences. An ad-hoc mechanism is used to translate paths (spanning from the root node to a leaf) to FO(·) sentences.

3 IDP-Z3

IDP-Z3 is a reasoning engine that can perform a variety of reasoning tasks on knowledge bases in the FO(·) language. It is the successor of IDP3 [12], another reasoning engine for FO(·). IDP3 used a custom SAT solver, called minisat(ID) [11]: hence, its support for arithmetic was limited. By contrast, IDP-Z3 uses an off-the-shelf SMT solver,

Z3 [13], which supports reasoning over linear arithmetic. FOLASP [34] is another reasoning engine for FO(·), which uses ASP-Core-2 solvers as back-end. However, its reasoning capabilities are limited to model expansion.

IDP-Z3 can be run at the command line, or integrated in a Python application as a Python package downloadable from pypi[6]. Computations can also be run online via a webIDE[7]. It is open source[8] under the LGPL 3 license.

A challenge in writing IDP-Z3 was to re-implement the custom functionality of minisat(ID) around Z3. In particular, minisat(ID) used custom procedures to handle inductive definitions. In IDP-Z3, inductive definitions are reduced to formulae acceptable by Z3, using level mapping, as explained in [31]. Another challenge was the re-implementation of a custom procedure to determine relevance [24].

The following generic computations are supported by IDP-Z3:

- *Model checking* Verifies that a theory is satisfiable, i.e., that it has at least one model.
- *Model expansion* Takes a theory T and a partial structure S, and computes a model of T that expands S, if one exists.
- *Propagation* Takes a theory T and a partial structure S, and computes all their logical consequences, i.e., all the ground literals that are true in every model of T and S.
- *Explanation* Takes a theory T, a partial structure S and a literal L obtained by propagation, and computes an explanation for L in the form of a minimal set of axioms in $T \cup S \cup \{\neg L\}$ that is inconsistent. This explanation is not necessarily subset-minimal.
- *Optimisation* Takes a theory T, a partial structure S and a term, and computes the minimal value of the term in the set of all model expansions of T and S.
- *Relevance* Takes a theory T and a partial structure S, and determines the atoms that are irrelevant (or "do-not-care") in the sense that, if one of their value were changed in any model M of T expanding S, the resulting M' structure would still be a model of T.
- *Other reasoning tasks* While not natively supported in IDP-Z3, other reasoning tasks can be developed around IDP-Z3. For example, one could compare two FO(·) formulations, and show models where they differ, as in the Intelli-Select use case described in Section 5.3. One could also verify the completeness of definitions (or of DMN tables), or generate test cases for a KB.

4 Interactive Consultant

IDP-Z3 comes with a demo web application, called the Interactive Consultant [6], that helps users make decision in accordance with an FO(·) knowledge base, using the reasoning abilities of IDP-Z3. It is the successor of AutoConfig [10], which was based on IDP3. The Interactive Consultant is used in three of the four case studies described in the next section. It is generic in the sense that it can be reconfigured by simply changing the FO(·) knowledge base. The user interface is automatically generated based on the

[6] https://pypi.org/project/idp-engine/
[7] https://interactive-consultant.idp-z3.be/IDE
[8] https://gitlab.com/krr/IDP-Z3

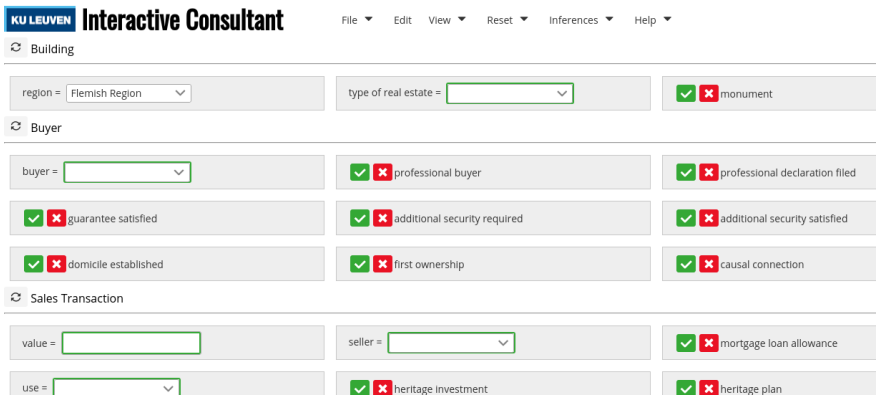

Fig. 2: The Interactive Consultant asks relevant questions to its user, from which it draws conclusions that it can explain.

vocabulary of the knowledge base: this helps reduce the cost of developing applications significantly [21]. It is available online[9].

The Interactive Consultant (IC) allows the user to enter data in any order. This data is stored in a data theory that is combined with the knowledge base for reasoning. The IC enables a safe exploration of the decision search space, without the possibility of making decisions leading to dead ends. This is achieved by continuously computing the consequences of the data theory, using **propagation**. If the user is unsure why the IC propagated a specific choice, they can ask for an **explanation**. Additionally, while the user fills in what they know, the interface determines which parameters remain **relevant**, avoiding unnecessary work for the user. After having input all values that they deem necessary, the user can ask the IC to show *an optimal* decision according to what is known, using **optimization**.

The response time of the system after the user asserts or retracts a fact depends on the speed of the propagation reasoning task. Propagation is often performed by iterative satisfiability testing [23], i.e., by checking every ground atom to see if it is a consequence of the theory and user input, i.e., if it has only one possible interpretation. We improve speed of propagation in the Interactive Consultant by reducing the number of ground atoms to consider:

- When new facts are asserted by the user, previously propagated atoms do not need to be considered again: indeed, they will remain consequences of the theory and user input;
- When facts are retracted by the user, atoms that were not consequences of the theory and user input already will still not be, and do not need to be considered again.

Because decisions are made by a user *in a context*, it is often important to separate the ontology describing the context from the one describing the decision and its consequences: while the user has control over his decision, they do not have control over their context. The inferences described in Section 3 have been adapted to accommodate this split ontology [7].

[9] https://interactive-consultant.idp-z3.be/

5 Case studies

5.1 Machine Component Designer

In [1], the authors describe the creation of an IDP-based knowledge base system for the design of machine components, implemented in collaboration with a multinational company. This company employs engineers worldwide to conceive "design-to-order" components. Before the collaboration, each engineer followed their own *ad hoc* design process, mostly based on their own experience and preferences. This approach has multiple downsides: (a) designing a component consumes a large amount of time, (b) engineers may choose sub-optimal designs, and (c) if a senior engineer leaves, a great deal of knowledge is lost by the company.

To overcome these issues, the design knowledge was formalized through a series of knowledge extraction workshops. In such a workshop, both knowledge engineers and domain experts are involved in modeling the knowledge used when designing machine components. Each workshop spanned a few days, and was performed in geologically different branches of the company, to ensure diversity in the knowledge.

Initially, the DMN standard was used to create a model of the experts' knowledge. While DMN was found to be intuitive, it was unable to represent all knowledge in a straightforward manner. Indeed, DMN is well suited for rule-based, hierarchical decision procedures with one unique output, but it is not suitable for reasoning when several choices are possible. For example, to find the optimal design of the component, the user had to make tentative design choices, determine the resulting cost, backtrack, and subsequently consider alternative designs.

In FO(·) parlance, we would say that DMN can represent definitions, but not axioms. Axioms are used to exclude designs that are not feasible. To allow the addition of axioms on top of the rule-based logic, the DMN model was converted into an FO(·) KB. Additionally, some *preferences* were added: e.g., "always use the cheapest material possible." Using a weighted sum, these preferences can then be used to automatically determine the optimal design for any given circumstances.

In total, the KB contains 10 parameters describing 60 different materials (such as a maximum temperature of steel) and 27 parameters for 31 components (such as torque and maximum pressure).

The Interactive Consultant (described in the previous section) is used to allow interaction with the KB. It is configured for this application by simply changing the FO(·) KB. Besides the standard interactions, the interface was further extended to fit the company's specific needs. Examples of such extensions are a functionality to compare two designs, an extended version of explanations in which not only the set of choices are shown but also the underlying constraints (which has since been added to the IC), a way to deactivate and reactivate certain axioms, and an integration with a Machine Learning algorithm that suggests designs based on historic data.

Overall, the company and its engineers are very positive about the tool. Besides a reported daily time-save of up to 30 minutes for each engineer, they report other benefits to its usage. First and foremost, it leads to more "first-time right" designs, which lowers production time and cost. Secondly, for new engineers the tool serves as an excellent learning tool, allowing them to indirectly learn from the knowledge of the

more experienced engineers. For more experienced engineers on the other hand, the tool is used to challenge their assumptions: when in doubt, they can swiftly verify if their initial ideas are correct. Lastly, with their knowledge captured in a KB, engineers leaving will not result in loss of knowledge for the company.

The functionality of the original IDP3-based tool was somewhat limited because IDP3 is not able to perform floating point calculations. To overcome this, it has recently been ported over into IDP-Z3, to benefit from its support of arithmetic.

5.2 Adhesive Selection Tool

Together with the Flanders Make Joining & Materials Lab we have been working on a case study concerning adhesive selection [25].

In industry, the usage of glues is rising in popularity due to their favorable characteristics. However, besides some superficial, vendor-locked websites, there is no tooling available to support selecting the right adhesive for the right use case. Adhesives come in a wide range of options, categorised into 18 different adhesive families, with none suitable for all applications. The selection of an adhesive is based on which substrates are used (e.g., steel, wood, plastics, ...) and on bond requirements such as minimum strength, maximum elongation, operating temperature range and more.

To begin, we held multiple knowledge workshops in order to create a KB. Instead of using DMN to create the initial model, as in the previous case, we used cDMN due to the constraint-heavy nature of the problem domain. In total, we identified and formalised 21 adhesive parameters (such as bond strength, adhesion and temperature range) and 11 substrate parameters (such as solvent resistance, maximum temperature and magnetic type). The current version of the KB contains 55 individual adhesives, and 31 substrates.

An interesting aspect of this case is how missing data is handled: if a parameter value is not known (because it was, e.g., not listed in the adhesive's data sheet), the tool assumes the value of the adhesive's family. In this way, the tool uses a reasonable estimate of the real value, similar to what the experts do. However, if the family's value is also unknown, the tool warns the user that the value is unknown and it does not apply any constraints to that parameter, instead asking the expert to verify it manually.

The adhesive experts interact with the KB through the Interactive Consultant, allowing them to benefit from all of its features. In particular, it allows reasoning in any direction. While generally, the goal is to select an adhesive, in other cases, the adhesive is known, as well as one of the substrates, and the goal is to find a suitable second substrate. This "substrate selection" task is performed without modifying the KB in any way.

Overall, the first impressions by the experts are positive. In an initial test, the tool has reduced the selection process from 3 hours to 5 minutes for one especially difficult case. While it seemed that the tool would be most useful for newer members of the J&ML lab, the most experienced member has indicated that they can also benefit from it. Indeed, this member typically chooses from a (limited) set of adhesives of which they know most properties by heart. Using the tool, they can find out if there are any other adhesives which might be better suited for a job.

5.3 Intelli-Select

In [20], the authors present their work together with Intelli-Select[10], on a tool created for international financial institutions to support investment management. This tool combines Constrained Natural Language (CNL) and the IDP-Z3 system, to offer a user-friendly way for a customer to define his *investment profile*. An investment profile is a set of rules that specify the financial assets that they consider eligible for investment. This investment profile is created in CNL, and later converted to FO(·) for processing.

For example, a user can construct the CNL sentence "Equities issued in Germany are eligible" to allow all German equities. Additionally, a free-form Natural Language (NL) interface is also available, which suggests CNL equivalents. Here, an NL sentence such as "I would like to invest in German equities" would be translated into the CNL statement shown earlier.

To create the KB, the CNL statements are represented in FO(·) in the form of two definitions: one for eligibility, and one for ineligibility. In the application, the IDP-Z3 system is then used to perform several reasoning tasks. Firstly, each time a user adds new (in)eligibility rules, the system performs **propagation** to show the effect of the new rules. If a rule's effect is unexpected, or the user is unsure why it happened, they can invoke the **explanation** inference. When a profile is considered finished, **model expansion** can be performed to identify the eligible assets, i.e., those that satisfy the *eligible* definition. Moreover, using **optimization**, it is possible to calculate minimal cost combinations of these eligible assets.

Besides these standard reasoning tasks, the collaborating company requested a way to automatically convert the financial profile into a long and complex document with a specific format. This document, called *Appendix A*, is used both as an appendix to the contract with the client, and as the formal input to one of the systems in place at the company. Previously, the company created such a document manually, a process which typically took a few months to complete. However, because the required knowledge of a financial profile is already present in the KB, we were able to add a specific method to automatically generate these documents in a few seconds, as described in [19].

As lessons learned, the authors outline two things. Firstly, they mention that compared to standard applications in the field of financial technology, their tool is low in maintenance due to the separation between domain knowledge and reasoning tasks. Secondly, while the creation of a KB is typically a challenging task, the use of CNL is a good way of lowering the effort.

5.4 Notary / Legislation

The fourth case study deals with registration duties on property purchases in Belgium. It was originally presented in [22] and later extended in [21]. In Belgium, registration duties depend on many parameters, such as the location, the type of property, the characteristics of the buyer and the seller, and more. At the time of the first publication, these duties were determined by 11 articles of law. For this case, a collaboration with a notary was set up to ensure the correctness of the knowledge.

Initially, the knowledge engineer and domain expert worked together to create a DMN model of the legislation. Here, the user-friendliness of DMN meant that it could

[10] https://www.intelli-select.com/

be used as a "common language" between the two, leading to less formalization errors. This is an important point, as the law field contains a great deal of complex jargon. After the DMN model was finished, the knowledge engineer converted the tables into FO(·), ready to be used by the IDP system. Together with the Interactive Consultant, this formed an easy-to-use application. Note that standard DMN tools would not have been sufficient, as the notary required a tool that could reason with partial information and could optimise the cost of the duties. In total, the formalization process took 10 person-days.

Later in 2018, the relevant law was simplified by the Belgian government. In [21], the KB is updated to reflect these changes, together with improvements to the Interactive Consultant as requested by the notary office. While the change was the most significant change to the real estate sales law to have ever happened, updating the KB required only 0.5 person-days, due to the KB's modular nature.

This application has been ported to IDP-Z3, and is available online[11]. It takes advantage of the arithmetic capabilities of IDP-Z3 to calculate the tax amount due.

6 Evaluation

We evaluate IDP-Z3 by comparing its functional capabilities to those of other systems, and by comparing its performance against user expectations.

6.1 Functional comparison to other systems

Table 1 compares FO(·) to two other model-based languages on the basis of their documentation: SMT-LIB-2 [2] and ASP-Core-2 [5]. Table 2 compares IDP-Z3 to reasoning engines for SMT and ASP. The features in the comparison tables are described in Sections 2.1 and 3.

None of the languages or systems are complete. Hence, they are under further development to bring more expressivity to the languages, and more reasoning capability to the reasoning engine.

Because they share many concepts, some researchers have investigated the possibility to transform a KB in one language into a KB in another, e.g., to improve performance:

- IDP-Z3 itself transforms FO(·) KBs into SMT-LIB-2 KBs;
- FOLASP transforms FO(·) KBs into ASP-Core-2, allowing performance comparisons [34]; the semantics correspondence between FO(ID) and ASP is explored in [16];
- and several ASP-Core-2 solvers are based on SMT solvers (e.g., [33]).

6.2 Performance evaluation

Deryck et al. [20] already reported response time below 3 seconds for the Intelli-Select application, for a typical investment profile.

We now discuss the performance of IDP-Z3 in the other case studies. We believe that these results are representative for other interactive applications based on IDP-Z3 and the Interactive Consultant.

[11] https://interactive-consultant.idp-z3.be/?file=registratie.idp

Language	FO(·)	SMT	ASP
Classical syntax	✓	✓	
Types, quantification	✓	✓	
Uninterpreted types		✓	
(Inductive) definitions	✓		✓
Disjunction in head			✓
Integer arithmetic	✓	✓	✓
Real arithmetic	✓	✓	-**
Non-linear arithmetic		✓	
Transcendental		-*	
Aggregates	✓	-*	✓
Vector, Array		✓	
Concept as a type	✓		

Table 1: Comparison of model-based languages. *Not part of the standard, but provided by, e.g., Z3 or dReal. **Not part of the standard, but provided by Constraint-ASP solvers.

Reasoning	IDP-Z3	SMT	ASP
Model checking	✓	✓	
Model expansion	✓	✓	✓
Propagation	✓	-**	
Explanation (unsat)	✓	✓	
Step-wise explanation	-*		
Optimisation	✓	-**	✓
Relevance	✓		

Table 2: Comparison of model-based reasoning systems.
*Provided by some solvers, e.g., IDP3.
**Provided by some solvers, e.g., Z3.

Use case	# symb	model size	# sentences	load time (sec)	response time (sec)
Component designer	92	227286	42	32.9	2.8
Adhesive selection	136	2061	154	8.3	2.5
Registration	31	31	15	0.3	0.1

Table 3: Average response time is below 3 seconds, and load time acceptable.

Table 3 shows various metrics and performance indicators for each use case. The column headings are:

- # symb: the number of symbols in the vocabulary;
- model size: the size of a model, i.e., the sum of the cardinality of the domain of each predicate and function symbol (they all have a finite domain);
- # sentences: the number of axioms and definitional rules in the theory;
- load time: the number of seconds needed to load the knowledge base in the Interactive Consultant;
- resp. time: the number of seconds needed to process the assertion or retraction of a fact (triggering a propagation).

The load and response times are measured on a Intel® Core™ i7-8850H CPU @ 2.60GHz × 12 machine, with 16 GB of memory, using Ubuntu 20.04.03 and CPython 3.9.

The table shows that the load time can exceed 10 seconds for KB with large models. This is not ideal, but still acceptable in the use case presented. The delay is due to the transformation of the FO(·) KB into the equivalent grounded FO formula that is submitted to Z3: the transformation is performed at every load, by code written in Python, a language not known for its speed. Load time could be improved by rewriting the transformation in a faster language, or by loading the pre-computed result of the transformation directly from storage.

The response time of the Interactive Consultant, on the other hand, is below 3 seconds, an adequate delay for interactive applications. This is achieved by retaining the internal state of IDP-Z3 between interactions. Our experience indicates that such retention is important in interactive applications based on IDP-Z3. Performance could be further improved by parallelizing propagation, i.e., by running the iterative satisfiability testing method over multiple instances of Z3.

7 Conclusions

Thirty years ago, McCarthy introduced the concept of intelligent machines of the fourth kind, capable of performing a variety of computational tasks by applying task-independent knowledge. Our work shows that such machines are now feasible, and that interactive applications that deliver real value to their users can be developed at low costs (typically 10 days of effort split over involved parties) and with acceptable performance (response time typically below 3 seconds).

The following elements have contributed to this success:

- Our use cases are knowledge-rich but data-poor, making it possible to use computationally complex forms of reasoning. IDP-Z3 brings value by having knowledge that the user may not have, and by reasoning faster and more rigorously with it than an expert can.
- IDP-Z3 goes beyond inference engines of the third kind by allowing reasoning not only with deterministic rule-based definitions, but also with non-deterministic axioms describing possible worlds. Our result further justifies the revival of interest

in the seminal papers on the integration of rule-based languages and classical logic: [15] won 20-year Test-of-Time award at ICLP 2020, and [17] won 20-year Test-of-Time award at ICLP 2021).
- The use of generic reasoning methods (as recommended in the Knowledge Base paradigm) and the automatic generation of the user interface of the Interactive Consultant significantly reduce the development costs of intelligent applications. User-friendly notations like DMN or Controlled Natural Languages can further empower users to encode their knowledge.

We believe that machines of the fourth kind merits further research. The interaction between the user and the Interactive Consultant has many similarities with the conversation in a Turing test. Here, the interaction is not conducted in a natural language, but the machine shows signs of intelligence that would be tested in a Turing test, such as the capability to ask relevant questions or provide explanations.

Additional research would expand the expressiveness of the knowledge representation languages and/or improve the performance of reasoning engine. The standardization of the concrete syntax of FO(·) could facilitate such research.

Acknowledgements

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

The authors thank Ingmar Dasseville and Jo Devriendt for their contributions to the development of IDP-Z3. They also thank the contributors to the many open source projects IDP-Z3 is built on, such as textx [14].

References

1. Aerts, B., Deryck, M., Vennekens, J.: Knowledge-based decision support for machine component design: A case study. Expert Systems with Applications **187** (Jan 2022)
2. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (2010)
3. Blackburn, P., Bos, J.: Computational semantics. Theoria (2003)
4. Blackburn, P., Bos, J.: Working with discourse representation theory. An Advanced Course in Computational Semantics (2006)
5. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: Asp-core-2: Input language format. ASP Standardization WG (2012)
6. Carbonnelle, P., Aerts, B., Deryck, M., Vennekens, J., Denecker, M.: An interactive consultant. In: Proceedings of BNAIC 2019. vol. 2491 (2019)
7. Carbonnelle, P., Bogaerts, B., Vennekens, J., Denecker, M.: Interactive advisor for lax legislation and observable situations. In: Workshop on Models of Legal Reasoning (2020)
8. Carbonnelle, P., Van der Hallen, M., Vennekens, J., Denecker, M.: Quantification and aggregation over concepts of the ontology. In: KR 2022 (submitted) (2022)
9. Claes, J., Bogaerts, B., Canoy, R., Guns, T.: User-oriented solving and explaining of nl logic grid puzzles. In: The Third Workshop on Progress Towards the Holy Grail (2019)
10. Dasseville, I., Janssens, L., Janssens, G., Vanthienen, J., Denecker, M.: Combining DMN and the knowledge base paradigm for flexible decision enactment. CEUR Proceedings (2016)
11. De Cat, B., Bogaerts, B., Denecker, M.: Minisat (id) for satisfiability checking and constraint solving. ALP Newsletter (2014)

12. De Cat, B., Jansen, J., Janssens, G.: Idp3: Combining symbolic and ground reasoning for model generation. In: 2nd Workshop on Grounding and Transformations for Theories With Variables (2013)
13. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer (2008)
14. Dejanovic, I., Vaderna, R., Milosavljevic, G., Vukovic, Z.: Textx: A python tool for domain-specific languages implementation. Knowledge-Based Systems **115** (2017)
15. Denecker, M.: Extending classical logic with inductive definitions. In: Computational Logic - CL 2000. Lecture Notes in Computer Science, vol. 1861. Springer (2000)
16. Denecker, M., Lierler, Y., Truszczynski, M., Vennekens, J.: A tarskian informal semantics for answer set programming. In: Technical Communications of the ICLP 2012
17. Denecker, M., Pelov, N., Bruynooghe, M.: Ultimate well-founded and stable semantics for logic programs with aggregates. In: Logic Programming, ICLP 2001. Springer (2001)
18. Denecker, M., Vennekens, J.: Building a knowledge base system for an integration of logic programming and classical logic. In: Logic Programming, 24th International Conference, ICLP 2008. Lecture Notes in Computer Science, vol. 5366. Springer (2008)
19. Deryck, M.: Knowledge Base Systems in practice: Approaches, application areas and limitations. Ph.D. thesis, KU Leuven (2022)
20. Deryck, M., Comenda, N., Coppens, B., 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)
21. Deryck, M., Devriendt, J., Marynissen, S., Vennekens, J.: Legislation in the knowledge base paradigm: interactive decision enactment for registration duties. In: Proceedings of the 13th IEEE Conference on Semantic Computing. IEEE (2019), iSSN: 2325-6516
22. Deryck, M., Hasić, F., Vanthienen, J., Vennekens, J.: A case-based inquiry into the decision model and notation (DMN) and the knowledge base (KB) paradigm. In: R&R (2018)
23. Janota, M., Lynce, I., Marques-Silva, J.: Algorithms for computing backbones of propositional formulae. Ai Communications **28**(2) (2015)
24. Jansen, J., Bogaerts, B., Devriendt, J., Janssens, G., Denecker, M.: Relevance for SAT(ID). In: Proceedings of IJCAI 2016. IJCAI/AAAI Press (2016)
25. Jordens, J., Vandevelde, S., Van Doninck, B., Witters, M., Vennekens, J.: Adhesive selection via an interactive, user-friendly system based on symbolic ai. Procedia CIRP (2022)
26. Kowalski, R.: Algorithm= logic+control. Communications of the ACM **22**(7) (1979)
27. McCarthy, J.: Artificial intelligence, logic and formalizing common sense. In: Philosophical logic and artificial intelligence. Springer (1989)
28. McCarthy, J.: Elaboration tolerance. In: Common Sense (1998)
29. Nute, D.: Defeasible logic. In: Proceedings of the 14th INAP 2001 (2001)
30. Object Modelling Group: Decision model and notation v1.3 (2021)
31. Pelov, N., Ternovska, E.: Reducing inductive definitions to propositional satisfiability. In: International Conference on Logic Programming (2005)
32. Reiter, R.: A logic for default reasoning. Artif. Intell. **13** (1980)
33. Shen, D., Lierler, Y.: Smt-based answer set solver CMODELS(DIFF). In: 34th ICLP. vol. 64. Schloss Dagstuhl (2018)
34. Van Dessel, K., Devriendt, J., Vennekens, J.: FOLASP: FO(·) as Input Language for Answer Set Solvers. Theory and practice of logic programming **21**(6) (2021)
35. Vandevelde, S., Aerts, B., Vennekens, J.: Tackling the DM challenges with cDMN: A tight integration of DMN and constraint reasoning. TPLP (2021)
36. Vandevelde, S., Vennekens, J.: A multifunctional, interactive DMN decision modelling tool. In: Proceedings of BNAIC/BeneLearn 2020. Leiden University (2020)
37. Von Wright, G.H.: Deontic logic. Mind **60**(237) (1951)
38. Wang, H.: Logic of many-sorted theories. The Journal of Symbolic Logic **17**(2) (1952)