Combining representation formalisms for reasoning upon mathematical knowledge

Mathieu D'aquin, Renata Bunoiu, Horatiu Cirstea, Michel Lenczner, Jean Lieber, Frédéric Zamkotsian

▶ **To cite this version:**

Mathieu D'aquin, Renata Bunoiu, Horatiu Cirstea, Michel Lenczner, Jean Lieber, et al.. Combining representation formalisms for reasoning upon mathematical knowledge. K-CAP '23: Knowledge Capture Conference 2023, Dec 2023, Pensacola FL USA, United States. pp.180-187, 10.1145/3587259.3627549 . hal-04315073

HAL Id: hal-04315073
https://hal.science/hal-04315073v1

Submitted on 30 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Combining Representation Formalisms for Reasoning upon Mathematical Knowledge

Mathieu d'Aquin
mathieu.daquin@loria.fr
LORIA - Université de Lorraine, CNRS, Inria
Vandœuvre-lès-Nancy, France

Renata Bunoiu
renata.bunoiu@univ-lorraine.fr
Université de Lorraine, CNRS, IECL
Metz, France

Horatiu Cirstea
horatiu.cirstea@loria.fr
LORIA - Université de Lorraine, CNRS, Inria
Vandœuvre-lès-Nancy, France

Michel Lenczner
michel.lenczner@univ-fcomte.fr
FEMTO-ST - UTBM, ENSMM, UFC, CNRS
Besançon, France

Jean Lieber
jean.lieber@loria.fr
LORIA - Université de Lorraine, CNRS, Inria
Vandœuvre-lès-Nancy, France

Frédéric Zamkotsian
frederic.zamkotsian@lam.fr
LAM - Aix Marseille Université, CNRS, CNES
Marseille, France

ABSTRACT

Knowledge in mathematics (definitions, theorems, proofs, etc.) is usually expressed in a way that combines natural language and mathematical expressions (e.g. equations). Using an ontology formalism such as OWL DL is well-suited for formalizing the natural language part, but complex mathematical expressions can be better handled by symbolic computation systems. We examine this representation issue and propose an original extension of OWL DL by call formulas, i.e., formulas from which assertions can be drawn thanks to calls to external functions. Using this formalism makes it possible to classify a mathematical problem defined by its relations to instances and classes and by some mathematical expressions: if a theorem for solving this problem is represented in the knowledge base, it can be retrieved, and thus, the problem can be solved by applying this theorem. We describe an inference algorithm and discuss its properties as well as its limitations. Indeed, the proposed extension, algorithm, and implementation represent a first step towards a combined formalism for representing mathematical knowledge, with some open issues regarding the representation of more complex problems: the resolution of multiscale, multiphysics cases in physics are foreseen.

KEYWORDS

knowledge modeling, knowledge representation, mathematical knowledge

ACM Reference Format:
Mathieu d'Aquin, Renata Bunoiu, Horatiu Cirstea, Michel Lenczner, Jean Lieber, and Frédéric Zamkotsian. 2023. Combining Representation Formalisms for Reasoning upon Mathematical Knowledge . In *Proceedings of The Twelfth International Conference on Knowledge Capture (KCAP 2023).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KCAP 2023, December 5 - 7, 2023, Pensacola, Florida, USA
© 2023 Association for Computing Machinery.
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Knowledge representation usually starts with deciding on a formalism to be used. Different languages and paradigms might indeed be better suited for certain tasks or certain types of knowledge [4]. Description logics and ontologies, for example, are more often used to represent conceptual knowledge, for reasoning tasks such as classification, and in contexts where the exchange of knowledge and interoperability are important. In other cases, other logics or approaches focused on computational aspects can be favored. However, in relatively complex domains, multiple representation formalisms might be required to fully capture the knowledge needed for a given task.

This work originates from a project in such a domain, as a collaboration between physicists, mathematicians, and computer scientists in modeling the physical properties and behavior of arrays of micro-mirrors (see, for example, [5]). Arrays of micro-mirrors are optical devices permitting the development of breakthrough instruments for Earth and Universe Observation. These components are made of a high number of micron-scale tiltable mirrors and are controlled electrostatically. Predicting the actual position of each mirror with respect to the behavior of the neighboring mirrors is a complex case involving multiscale, multiphysics systems. These issues have been addressed thanks to the mathematical principles of asymptotic analysis of partial differential equations, and have been studied through term rewriting tools in order to represent both knowledge of the models and of the computations applicable to them [1, 2, 10].

However, there are a number of aspects for which symbolic computation formalisms are insufficient (or inefficient). These specific aspects relate mostly to knowledge useful to recognize the situations in which specific computations are needed, particular resolution strategies should be applied, or a given theorem holds. The goal of this study is to examine how to organize mathematical knowledge in order to be able to select relevant modules (representing, for example, theorems) of a symbolic computation system, when a mathematical problem is raised (it is noteworthy that the goal is *not* to reason within mathematical knowledge).

To illustrate this, we consider (here and in the rest of this paper) an example simpler than the differential equations eventually

targeted by this approach: the resolution of a second-degree polynomial equation such as $2x^2 + 3x + 1 = 0$. There are different methods to achieve this, which might apply in different cases, and even before starting to solve it, a system aiming to support the resolution of mathematical problems would first have to recognize that the aforementioned expression: 1- is an equation, 2- is a polynomial equation, 3- is a polynomial equation of degree 2. It might then, for example, calculate the discriminant of the equation to determine whether it has one, two, or no real solutions.

Even for such a simple example, the knowledge manipulated can be seen as having some conceptual aspects (a polynomial equation is a kind of equation, a second-degree polynomial equation, a.k.a. a quadratic equation, has a discriminant, etc.) and some computational aspects (how to obtain the degree of a polynomial equation, how to calculate the discriminant of a quadratic equation, etc.).

In this paper, we present a logic (OWL DL$^{\texttt{call}}$) to combine knowledge representation formalisms, which we design as an extension of the description logic underlying OWL DL [9] with calls to external symbolic computation engines. While calls to external functions have been present in knowledge representation systems in the past (see, for example, [3]), our extension is designed to enable an inference algorithm, which is described here with a discussion on its formal properties. We also discuss the proposed implementation of this algorithm, illustrated with a simple example in mathematics using a popular symbolic computation tool (the Sympy Python library[1]). It shows that the current version of OWL DL$^{\texttt{call}}$, as a first attempt to combine multiple representation formalisms for mathematical knowledge, can already handle a large class of problems. We, however, also discuss its limitations and the way in which it will need to evolve in order to support more complex problems.

After a reminder on OWL DL (§2), an example of a representation of a simple mathematical problem is detailed (§3), leading to a new representation formalism (§4). The paper ends with a discussion and a conclusion (§5).

2 BACKGROUND ON OWL DL

This paper uses and extends the description logic $\mathcal{SROIQ}(\mathtt{D})$ that is assimilated to the W3C recommendation OWL DL in its version 2. In this section, only a fragment of this logic is presented, restricted to the needs of the paper.

Throughout the paper, classical notions in mathematics are used. In particular, $\mathbb{N}$ and $\mathbb{R}$ are the sets of natural numbers and real numbers, $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$, $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$, and $[\![a, b]\!] = \{n \in \mathbb{N} \mid a \leq n \leq b\}$ for $a, b \in \mathbb{N}$.

Concrete domains (datatypes). Concrete domains in OWL DL corresponds to XML schema built-in datatypes, e.g. `bool`, `float`, `unsigned int`, etc. With τ one of these datatypes, the concrete domain associated to τ is given by the ordered pair (Δ_τ, Φ_τ) where Δ_τ is the set of values of type τ (e.g. $\Delta_{\texttt{unsigned int}} = \mathbb{N}$) and Φ_τ is a countable set of symbols that represent some subsets of Δ_τ: for $\sigma \in \Phi_\tau$, this subset of Δ_τ is denoted by σ^τ. In particular, if $v \in \Delta_\tau$, $\{v\} \in \Phi_\tau$ represents the singleton $\{v\}^\tau = \{v\}$, and $\top_\tau \in \Phi_\tau$ represents the set of all values of the concrete domain: $(\top_\tau)^\tau = \Delta_\tau$.

[1] https://www.sympy.org

2.1 Syntax

Four pairwise disjoint and countable sets of symbols are assumed to be given: $\mathcal{AC}$ (set of atomic concepts, a.k.a. atomic *classes*), $\mathcal{OP}$ (set of object properties, a.k.a. *roles*), $\mathcal{DP}$ (set of datatype properties), and $\mathcal{I}ns$ (set of instances). Moreover, each datatype property $p \in \mathcal{DP}$ is associated to a datatype τ called *range* of p; p is called a τ-property in the following. Finally, a *property* is an element of $\mathcal{OP} \cup \mathcal{DP}$.

A *concept* is either an atomic concept or an expression of one of the following forms (where C and D are concepts, $r \in \mathcal{OP}$, $a_1, a_2, \ldots, a_n \in \mathcal{I}ns$, p is a τ-property, and $\sigma \in \Phi_\tau$): $\top$, $\bot$, $C \sqcap D$, $\exists r.C$, $(\leq 1\ r)$, $\{a_1, a_2, \ldots, a_n\}$, $\exists p.\sigma$. The notation $(= 1\ r)$ is used as an abbreviation for $(\leq 1\ r) \sqcap \exists r.\top$.

A *formula* of OWL DL is either a *terminological formula* (usually expressing general knowledge) or an *assertion* (usually expressing ground facts). A terminological formula is an expression of one of the following forms (C and D are concepts, p_1 and p_2 are properties): $C \sqsubseteq D$, $C \equiv D$, and $p_1 \sqsubseteq p_2$. An assertion is an expression of one of the following forms (C is a concept, r is an object property, a and b are two instances, p is a τ-property, and $v \in \Delta_\tau$): $C(a)$, $r(a, b)$, $p(a, v)$.

2.2 Semantics

An interpretation $\mathcal{I}$ is an ordered pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$ where $\Delta_\mathcal{I}$ is a nonempty set and $\cdot^\mathcal{I}$ maps:

- An atomic concept A to a subset $A^\mathcal{I}$ of $\Delta_\mathcal{I}$,
- An object property r to a subset $r^\mathcal{I}$ of $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$ (i.e. $r^\mathcal{I}$ is a binary relation on $\Delta_\mathcal{I}$),
- A τ-property p to a partial function $p^\mathcal{I} : \Delta_\mathcal{I} \to \Delta_\tau$, and
- An instance a to an element $a^\mathcal{I}$ of $\Delta_\mathcal{I}$.

$\cdot^\mathcal{I}$ is extended on all concepts as follows:

$$\top^\mathcal{I} = \Delta_\mathcal{I} \qquad \bot^\mathcal{I} = \emptyset$$

$$(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$$

$$(\exists r.C)^\mathcal{I} = \left\{ x \in \Delta_\mathcal{I} \;\middle|\; \begin{array}{l} \text{there exists } y \in \Delta_\mathcal{I} \\ \text{such that } (x, y) \in r^\mathcal{I} \text{ and } y \in C^\mathcal{I} \end{array} \right\}$$

$$(\leq 1\ r)^\mathcal{I} = \left\{ x \in \Delta_\mathcal{I} \;\middle|\; \begin{array}{l} \text{there exists at most one } y \in \Delta_\mathcal{I} \\ \text{such that } (x, y) \in r^\mathcal{I} \end{array} \right\}$$

$$\{a_1, a_2, \ldots, a_n\}^\mathcal{I} = \{a_1^\mathcal{I}, a_2^\mathcal{I}, \ldots, a_n^\mathcal{I}\}$$

$$(\exists p.\sigma)^\mathcal{I} = \left\{ x \in \Delta_\mathcal{I} \;\middle|\; p^\mathcal{I}(x) \text{ is defined and } p^\mathcal{I}(x) \in \sigma^\tau \right\}$$

Given an interpretation $\mathcal{I}$ and a formula φ, $\mathcal{I}$ satisfies φ (denoted by $\mathcal{I} \models \varphi$) is defined as follows:

- $\mathcal{I} \models C \sqsubseteq D$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$,
- $\mathcal{I} \models C \equiv D$ if $C^\mathcal{I} = D^\mathcal{I}$,
- $\mathcal{I} \models p_1 \sqsubseteq p_2$ if $p_1^\mathcal{I} \subseteq p_2^\mathcal{I}$,
- $\mathcal{I} \models C(a)$ if $a^\mathcal{I} \in C^\mathcal{I}$,
- $\mathcal{I} \models r(a, b)$ if $(a^\mathcal{I}, b^\mathcal{I}) \in r^\mathcal{I}$,
- $\mathcal{I} \models p(a, v)$ if $(a^\mathcal{I}, v) \in p^\mathcal{I}$.

A *knowledge base* is a finite set of formulas of the considered logic. For a knowledge base $\mathcal{B}$ and an interpretation $\mathcal{I}$, $\mathcal{I} \models \mathcal{B}$ if $\mathcal{I} \models \alpha$ for each $\alpha \in \mathcal{B}$. $\mathcal{B}$ is satisfiable if there is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{B}$. Given a formula φ of OWL DL, $\mathcal{B}$ entails φ (noted $\mathcal{B} \models \varphi$) if for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{B}$, $\mathcal{I} \models \varphi$.

An *ontology* is a knowledge base containing only terminological formulas.

2.3 Some practical notations

The following notions do not add to the expressivity of the logic but are useful for giving readable notations and are used in the remainder of the paper.

Domains. Let r be an object property, and D be a concept. The sentence "D is a domain of r" expresses informally the formula $\exists r.\top \sqsubseteq D$. Therefore, the fact that an interpretation $\mathcal{I}$ satisfies this formula means that, for $x, y \in \Delta_{\mathcal{I}}$, if $(x, y) \in r^{\mathcal{I}}$ then $x \in D^{\mathcal{I}}$. D is qualified as a *domain* of r. In general, there is no unicity of the domain of a property: if D is a domain of r and if E is a superconcept of D (i.e. $D \sqsubseteq E$ is entailed by the current ontology) then E is also a domain of r.

Similarly, the sentence "D is a domain of p", for a τ-property p expresses informally the formula $\exists p.\top_\tau \sqsubseteq D$, and D is qualified as a domain of p.

Functional object property. Let r be an object property. The sentence "r is functional" expresses informally the formula $\top \sqsubseteq (\leq 1\, r)$. Therefore, the fact that an interpretation $\mathcal{I}$ satisfies this formula means that, for $x, y_1, y_2 \in \Delta_{\mathcal{I}}$, if $(x, y_1) \in r^{\mathcal{I}}$ and $(x, y_2) \in r^{\mathcal{I}}$ then $y_1 = y_2$.

It is recalled that, in this paper, we consider every datatype property as functional.

Property chains. A property chain is an expression of the form $r_1; r_2; \ldots; r_n; p$ (with $n \in \mathbb{N}$) where $r_1, r_2, \ldots, r_n$ are object properties and p is a property. The first purpose of this notion in this paper is to be used as a syntactic abbreviation for some assertions. Let a be an instance and v be an instance if p is an object property, or an element of Δ_τ if p is a τ-property. Then, that a is related to v by the property chain $r_1; r_2; \ldots; r_n; p$ can be expressed as

$$(\exists r_1.\exists r_2.\ldots.\exists r_n.\exists p.\{v\})(a)$$

For the sake of readability, this assertion is abbreviated by

$$(r_1; r_2; \ldots; r_n; p)(a, v)$$

Let $pc = r_1; r_2; \ldots; r_n; p$. A concept D is a *domain* of pc if D is a domain of r_1 if $n \geq 1$, and of p if $n = 0$. If p is a τ-property then the *range* of pc is τ, and, for an interpretation $\mathcal{I}$, $(pc)^{\mathcal{I}}$ denotes the set of $(x, v) \in \Delta_{\mathcal{I}} \times \Delta_\tau$ such that x is related to v by the composition of $r_1^{\mathcal{I}}, r_2^{\mathcal{I}}, \ldots, r_n^{\mathcal{I}}$, and $p^{\mathcal{I}}$.

3 A SIMPLE EXAMPLE

This section shows how to represent the concept corresponding to the problem of solving a second-degree equation, as well as two methods to solve instances of such a problem in our framework. By "represent", what is meant here is to describe it in a way that enables recognizing a particular problem to correspond to the resolution of a second degree equation and infer the method(s) which can be used to solve it. This simple example is here to point out modeling and representation issues, without the need to understand complex mathematics.

3.1 Definitions and theorems related to second degree equations

In this section, the notions to be represented are highlighted by ***bold italics typesetting***.

Definitions. ***Solving an equation*** is a ***problem*** consisting in finding the (possibly empty and possibly infinite) set of bindings of the ***unknowns*** of an ***equation*** that satisfy the ***equality*** given by the equation. A ***second degree equation on*** $\mathbb{R}$ is an equation with one unknown, say x, of the form $P(x) = Q(x)$ where P and Q are two ***polynomials*** such that the degree of the polynomial $Q - P$ is equal to 2.

Here is an example of a second degree equation solving problem:

$$\underbrace{\text{Solve} \quad \overbrace{\underbrace{x^2 + 2 = -3x}_{\text{an equality of the form } P(x) = Q(x)}}^{\text{an equation of second degree}} \quad \text{with unknown } x}_{\text{a problem of solving an equation of second degree}} \tag{1}$$

The solution of this equation is $S = \{\{x = -2\}, \{x = -1\}\}$.

Here is another example of a second degree equation solving problem:

$$\text{Solve} \quad t^2 + 4t + 1 = 0 \text{ with unknown } t \tag{2}$$

The solution of this equation is $S = \{\{t = -2 - \sqrt{3}\}, \{t = -2 + \sqrt{3}\}\}$.

Two methods for solving a second degree equation are presented below. The first one is simpler but is not complete, in the sense that it can fail: it succeeds for the problem (1) but fails for the problem (2). The second one is complete but is (a little bit) less simple. In both methods, the unknown is denoted by x and the equality is of the form $P(x) = Q(x)$. Let $R = Q - P$: the equality is then equivalent to $R(x) = 0$. Let $(a, b, c) \in \mathbb{R}^3$ such that $R = aX^2 + bX + c$ (with $a \neq 0$ since the degree of the equation is 2). A precondition for applying both methods is that it is established that this equation is a second degree equation.

The first method is called ***find an obvious solution***. It consists in the following steps:

(S1) For $x_1 \in \{-2, -1, 0, 1, 2\}$ perform the test $R(x_1) = 0$.
(S2) If all the answers to this test are *false* then the method fails.
(S3) Otherwise, with x_1 the smallest value satisfying this test, let $x_2 = -\frac{b}{a} - x_1$. Then, the solution is $S = \{\{x = x_1\}, \{x = x_2\}\}$.

The theorem associated with this method states that when this method does not fail, its result is the actual solution of the equation.

The second method is called ***use the discriminant***. It consists in the following steps:

(S1) Compute $\Delta = b^2 - 4ac$.
(S2) If $\Delta < 0$ then the solution is $S = \emptyset$.
(S3) Otherwise, $S = \left\{\left\{x = \frac{-b-\sqrt{\Delta}}{2a}\right\}, \left\{x = \frac{-b+\sqrt{\Delta}}{2a}\right\}\right\}$ (if $\Delta = 0$, S contains one binding if $\Delta > 0$, S contains two bindings).

3.2 Partial representation in description logic

In this section, the notions and statements involved in the resolution of a second degree equation are partially represented in the description logic OWL DL. For the sake of simplicity, we use our

own vocabulary, even though we could have used established ontologies for representing mathematical notions [7]. First, the general knowledge expressed by terminological formulas is presented:

- Solving an equation is a problem and it is associated with an equation.

1. $\texttt{EquationSolving} \sqsubseteq \texttt{Problem} \sqcap \exists \texttt{hasEquation}.\texttt{Equation}$

- An equation with one real unknown is an equation that has exactly one unknown such that this unknown is of type real.

2. $\texttt{Equation1RealUnknown} \equiv \texttt{Equation} \sqcap (= 1\ \texttt{hasUnknown}) \sqcap \exists \texttt{hasUnknown}.\exists \texttt{hasType}.\{\texttt{real}\}$

- A polynomial equation is necessarily an equation with one real unknown.

3. $\texttt{PolynomialEquation} \sqsubseteq \texttt{Equation1RealUnknown}$

- An equation of second degree is a polynomial equation of degree 2.[2]

4. $\texttt{Equation2ndDegree} \equiv \texttt{PolynomialEquation} \sqcap \exists \texttt{degree}.\{2\}$

- The method based on finding an obvious solution suited to the equation of second degree can be tried.

5. $\texttt{EquationSolving} \sqcap \exists \texttt{hasEquation}.\texttt{Equation2ndDegree} \sqsubseteq \exists \texttt{hasSolvingMethod}.\{\texttt{findObviousSolEqDeg2}\}$

- The method based on a discriminant for solving an equation of second degree can be tried.

6. $\texttt{EquationSolving} \sqcap \exists \texttt{hasEquation}.\texttt{Equation2ndDegree} \sqsubseteq \exists \texttt{hasSolvingMethod}.\{\texttt{useDiscriminant}\}$

Then, the two problems of the previous section are described by assertions:

- The first problem is a problem of solving equations.

7. $\texttt{EquationSolving}(\texttt{pb1})$

- This problem is associated to an equation with one real unknown.

8. $\texttt{hasEquation}(\texttt{pb1}, \texttt{eq1})$
9. $\texttt{Equation1RealUnknown}(\texttt{eq1})$

- This equality is based on the equality $x^2 + 2 = -3x$ and its unkown's name is "x".

10. $\texttt{hasEquality}(\texttt{eq1}, \texttt{"x ** 2 + 2 = -3 * x"})$
11. $(\texttt{hasUnknown;hasName})(\texttt{eq1}, \texttt{"x"})$

- For the other problem, pb2, the assertions are the same, after substitutions of pb1 with pb2 and eq1 with eq2, with the exception of the two last formulas which become:

12. $\texttt{hasEquality}(\texttt{eq2}, \texttt{"t ** 2 + 4 * t + 1 = 0"})$
$(\texttt{hasUnknown;hasName})(\texttt{eq2}, \texttt{"t"})$

3.3 Remainder of the representation by external functions

The knowledge base $\mathcal{B}_{\S 3.2}$ described in the previous section is insufficient to identify the two problems pb1 and pb2 as problems o solving a second degree equation. In other words, for $\texttt{pb} \in \{\texttt{pb1}, \texttt{pb2}\}$:

$$\mathcal{B}_{\S 3.2} \not\models \left(\begin{array}{l}\texttt{EquationSolving} \sqcap \\ \exists \texttt{hasEquation}.\texttt{Equation2ndDegree}\end{array}\right)(\texttt{pb})$$

[2]The degree of a polynomial equation $P(x) = Q(x)$ is the degree of $Q - P$.

Indeed, the base contains neither the knowledge that enables to identify the equations as being polynomial, nor to obtain the degrees of these equations. These inferences are based on expression handling, for which external computation is better suited.

Two functions are assumed to be implemented in Sympy (i.e. Python with the Sympy library, but it could be in another programming language): `isAPolEquation` and `degreeOfAPolynomialEquation`. The inputs of both functions are strings representing Sympy expressions. The former outputs the Boolean `true` iff this expression is one of a polynomial expression. The latter outputs the degree of a polynomial equation.

3.4 Connecting representation entities of the two languages

Connections between OWL DL and Sympy have to be made since:

(C1) The function `isAPolEquation` written in Sympy can be used to properly classify the equations `eq1` and `eq2` under the concept `PolynomialEquation`, i.e. to be able to infer the assertions $\texttt{PolynomialEquation}(\texttt{pb})$ for $\texttt{pb} \in \{\texttt{pb1}, \texttt{pb2}\}$.

(C2) The function `degreeOfAPolynomialEquation` written in Sympy can be used in order to infer the assertions $\texttt{degree}(\texttt{eq1}, 2)$ and $\texttt{degree}(\texttt{eq2}, 2)$.

From the description logic perspective, `isAPolEquation` and `degreeOfAPolynomialEquation` are considered external functions, regardless of the way they are implemented.

The connection (C1) is represented in the proposed approach by the following formula:

13.
```
isAPolynomialEquation ⊒
 call(function = "isAPolEquation @ Sympy",
      params = [hasEquality,   hasUnknown;hasName],
      domain = Equation1RealUnknown,
      range = bool)
```

The feature `function` of the call indicates the function name and the system under which this function has to be executed (here, Sympy). The feature `params` is associated with a list of property chains that indicate how the parameters of the function can be related to an instance of the domain (here, `Equation1RealUnknown`). The function has two arguments: the first one is the equality expression of the instance equation and the second one is the name of the unknown of the instance equation, both interpreted as strings by Sympy.

Formula 13 belongs to OWL $\text{DL}^{\texttt{call}}$ which is an extension of OWL DL proposed in Section 4. Intuitively, this formula means that if `eq` is inferred to be an instance of `Equation1RealUnknown` for which the equality and the name of the unknown are two identified strings, then the function `isAPolEquation` can be triggered with these strings as parameters and, if v is the Boolean returned by this function, the assertion $\texttt{isAPolynomialEquation}(\texttt{eq}, v)$ is inferred.

The idea is that with the formulas of $\mathcal{B}_{\S 3.2}$, the above `call` formula and the additional OWL DL formula

14.
```
PolynomialEquation ≡ Equation1RealUnknown ⊓
                     ∃isAPolynomialEquation.{true}
```

it can be entailed that eq1 and eq2 are instances of PolynomialEquation (provided that the call to Sympy of the function isAPolynomialEquation returns the expected value[3]).

The connection (C2) is represented in the proposed approach in a similar way:

15. degreePolynomialEquation ⊒ call(function = "degreeOfAPolynomialEquation@Sympy", params = [hasEquality, hasUnknown;hasName], domain = PolynomialEquation, range = unsigned int)

With the two call formulas in this section, the formulas in $\mathcal{B}_{\S 3.2}$, and the following additional formula:

16. Equation2ndDegree ≡ PolynomialEquation ⊓ ∃degreePolynomialEquation.{2}

it can be entailed that eq1 and eq2 are instances of Equation2ndDegree and, therefore, the following assertions

17. hasSolvingMethod(pb1, findObviousSolEqDeg2)
18. hasSolvingMethod(pb1, useDiscriminant)
19. hasSolvingMethod(pb2, findObviousSolEqDeg2)
20. hasSolvingMethod(pb2, useDiscriminant)

are consequences of $\mathcal{B}_{\S 3.4}$, i.e. the union of $\mathcal{B}_{\S 3.2}$ and of the formulas of the current section, under the assumption of the correct execution of the function in Sympy.

4 REPRESENTING MATHEMATICAL KNOWLEDGE IN OWL DL$^{\texttt{call}}$

This section presents the proposed approach for representing mathematical knowledge. First, the logic OWL DL$^{\texttt{call}}$ is presented (§4.1). Then, an algorithm for the inferences in this logic is described (§4.2). Section 4.3 presents the implementation of the algorithm and some discussion on the practical and methodological aspects of using OWL DL$^{\texttt{call}}$. Finally, Section 4.4 studies the issue of more complex examples that require an extension of this formalism.

4.1 OWL DL$^{\texttt{call}}$

The syntax of formulas is the same as for OWL DL, with the following additional terminological formula construct:

$p \sqsupseteq \texttt{call}(\texttt{function} = f, \texttt{params} = LPC, \texttt{domain} = D, \texttt{range} = \tau)$

where

- τ is a datatype and p is a τ-property;
- D is a concept;
- LPC is a list of na property chains, $LPC = [pc_1, pc_2, \ldots, pc_{na}]$ such that, for $k \in [\![1, na]\!]$, D is a domain of pc_k and pc_k is a τ_k-property;
- f is a string providing access to an external function (e.g. the code or a URL to a service), this function having na arguments and being denoted by $\widehat{f}$ in the following;
- The types of the parameters of $\widehat{f}$ are $\tau_1, \tau_2, \ldots, \tau_{na}$ (in this order) and the type of its output is τ;
- The execution of $\widehat{f}(v_1, v_2, \ldots, v_{na})$, for $v_k \in \Delta_{\tau_k}$ ($k \in [\![1, na]\!]$), is assumed to always terminate and may fail to return an answer in which case, this is denoted by $\widehat{f}(v_1, v_2, \ldots, v_{na}) = \texttt{failure}$. Moreover, two calls $\widehat{f}(v_1, v_2, \ldots, v_{na})$ with the same parameter values are assumed to return the same result; in particular, if one call fails, the other one also fails.

In the following, this construct is often given with the following abbreviated form:

$$p \sqsupseteq \texttt{call}(f, LPC, D, \tau)$$

Such a formula is called a *call formula*. Other formulas are qualified as *call-free*.

Let $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation, $\mathcal{I}$ satisfies the formula $p \sqsupseteq \texttt{call}(f, LPC, D, \tau)$ if, for every $(x, v) \in \Delta_{\mathcal{I}} \times \Delta_{\tau}$,

if (a) $x \in D^{\mathcal{I}}$ and (b) for every $k \in [\![1, na]\!]$ there exists v_k such that $(x, v_k) \in (pc_k)^{\mathcal{I}}$, and $\widehat{f}(v_1, v_2, \ldots, v_{na}) = v$ (and is therefore different from failure).

then $p^{\mathcal{I}}(x) = v$.

Now, as an illustration, it is proven, under the assumption that the Sympy function isAPolEquation is correctly implemented and that its implementation never fails, that

$$\mathcal{B}_{\S 3.4} \models \texttt{PolynomialEquation(eq1)}$$

Let $\mathcal{I}$ be an interpretation satisfying all the formulas of $\mathcal{B}_{\S 3.4}$. Let $x = \texttt{eq1}^{\mathcal{I}}$, $v_1 = \texttt{"x ** 2 + 2 = -3 * x"}$, $v_2 = \texttt{"x"}$, and $f = \texttt{isAPolEquation}$. The execution of $\widehat{f}(v_1, v_2)$ returns True and since $\mathcal{I}$ satisfies formulas 9, 10, and 11, it comes that:

(a) $x \in \texttt{Equation1RealUnknown}^{\mathcal{I}}$ and
(b) $(x, v_1) \in \texttt{hasEquality}^{\mathcal{I}}$, $(x, v_2) \in (\texttt{hasUnknown;hasName})^{\mathcal{I}}$.

Therefore, according to the semantics of call formulas, it can be inferred that $(x, \texttt{true}) \in \texttt{isAPolynomialEquation}^{\mathcal{I}}$. Finally, since $\mathcal{I}$ satisfies formula 14, it comes that $x \in \texttt{PolynomialEquation}^{\mathcal{I}}$, i.e. $\mathcal{I} \models \texttt{PolynomialEquation(eq1)}$, which ends the proof.

4.2 An inference algorithm for OWL DL$^{\texttt{call}}$

The inference algorithm presented in this section is based on an inference engine for OWL DL that is assumed to always terminate (e.g. Hermit [6]). This algorithm simply consists in (1) inferring a set of assertions $p(a, v)$ from the formulas based on calls for a given instance a, (2) adding these assumptions to the knowledge base (for all instances a) and making OWL DL inferences on the updated knowledge base. Step (1) is detailed in Algorithm 1. Step (2) is detailed in Algorithm 2.

Properties of the algorithm. The study of the inference algorithm under general assumptions remains to be done. However, a study has been carried out under the following assumptions:

(A1) Satisfiability of the knowledge base $\mathcal{B}$ given as parameter (i.e. existence of an interpretation $\mathcal{I}$ that satisfies all the call-free formulas and all the call formulas);
(A2) Functionality of the object properties involved in the chain properties of a call formula (i.e. for all call formula $p \sqsupseteq \texttt{call}(f, LPC, D, \tau)$), the object properties occurring in each property chain of LPC are functional).

First, it can be noted that under assumption (A1) if two call formulas enable to infer respectively $p(a, v_1)$ and $p(a, v_2)$ then v_1 and v_2

[3] This is shown at the end of Section 4.1 for eq1.

inputs • $\mathcal{B}$: a knowledge base of OWL DL$^{\texttt{call}}$,
- a: an instance,
- cache: a dictionary whose keys are pairs (Γ, b) where $\Gamma = (p \sqsupseteq \texttt{call}(f, LPC, D, \tau))$ is a call formula and b is an instance, and values are assertions of the form $p(b, v)$ (cache stores the value of previous calls and is initially empty).

effect cache is updated.
output A set of assertions $p(a, v)$ entailed by $\mathcal{B}$ thanks to external function calls.

function assertionsEntailedByCalls($\mathcal{B}, a$, cache):
$\mathcal{A} \leftarrow \emptyset$
$\mathcal{B}^{\texttt{call}} \leftarrow$ the set of call formulas of $\mathcal{B}$
$\mathcal{B}^{\texttt{noCall}} \leftarrow \mathcal{B} \setminus \mathcal{B}^{\texttt{call}}$
$C \leftarrow \left\{(p \sqsupseteq \texttt{call}(f, LPC, D, \tau)) \in \mathcal{B}^{\texttt{call}} \mid \mathcal{B}^{\texttt{noCall}} \models D(a)\right\}$
for $\Gamma = (p \sqsupseteq \texttt{call}(f, LPC, D, \tau)) \in C$ **do**
  if *the access to* cache *with key* (Γ, a) *fails* **then**
    $na \leftarrow$ length of LPC
    Let $pc_1, pc_2, \ldots, pc_{na}$ be such that $LPC = [pc_1, pc_2, \ldots, pc_{na}]$
    $\texttt{parameter-tuples} \leftarrow \left\{(v_1, v_2, \ldots, v_{na}) \;\middle|\; \begin{array}{l}\mathcal{B}^{\texttt{noCall}} \models pc_i(a, v_i) \\ \text{for } i \in [\![1, na]\!]\end{array}\right\}$
    newAssertions $\leftarrow \emptyset$
    for $(v_1, v_2, \ldots, v_{na}) \in$ parameter-tuples **do**
      $v \leftarrow \widehat{f}(v_1, v_2, \ldots, v_{na})$
      if $v \neq$ failure **then**
        $\alpha \leftarrow p(a, v)$
        if $\mathcal{B}^{\texttt{noCall}} \not\models \alpha$ **then**
          newAssertions $\leftarrow$ newAssertions $\cup \{\alpha\}$
        end
      end
    end
    $\mathcal{A} \leftarrow \mathcal{A} \cup$ newAssertions
    $\mathcal{B}^{\texttt{noCall}} \leftarrow \mathcal{B}^{\texttt{noCall}} \cup$ newAssertions
    cache$(\Gamma, a) \leftarrow$ newAssertions
    Remark: if newAssertions $= \emptyset$, the cache retains the fact that no new assertion has been entailed from Γ about a.
  end
end
return $\mathcal{A}$

Algorithm 1: Computing assertions entailed by call formulas.

input $\mathcal{B}$: a knowledge base of OWL DL$^{\texttt{call}}$,
effect $\mathcal{B}$ is enriched with all the assertions deduced from the calls.

function enrichByAssertionsInferredFromCalls($\mathcal{B}$):
Ins $\leftarrow$ the set of instances occurring in $\mathcal{B}$
cache $\leftarrow$ empty dictionary
repeat
  $s \leftarrow \text{card}\,\mathcal{B}$
  for *each instance a occurring in* $\mathcal{B}$ **do**
    $\mathcal{B} \leftarrow \mathcal{B} \cup$ assertionsEntailedByCalls($\mathcal{B}, a$, cache)
  end
until *card* $\mathcal{B} = s$;

Algorithm 2: Enriching the knowledge base with assertions coming from calls.

must be equal: in this paper, datatype properties are assumed to be interpreted as partial functions, therefore $v_1 \neq v_2$ would lead to inconsistency. For instance, two methods for finding the degree of a polynomial equation have to lead to the same result when neither of them fail, otherwise, $\mathcal{B}$ would be inconsistent.

Under assumptions (A1) and (A2), the process always terminates. This is a consequence of (1) the fact that there is a finite number of pairs (Γ, a) where Γ is a call formula of $\mathcal{B}$ and a is an instance occurring in $\mathcal{B}$, (2) the OWL DL inference engine and the called functions always terminate.

The algorithm is sound: if $p(a, v)$ is an assertion added to $\mathcal{B}$ by this algorithm then $\mathcal{B}_{\texttt{init}} \models p(a, v)$, where $\mathcal{B}_{\texttt{init}}$ is the value of $\mathcal{B}$ at the start of the algorithm (it does not add assertion that are not logically inferable from the knowledge base). This follows from the definition of the satisfaction of call formulas by interpretations.

Finally, it remains to be proven that the algorithm is complete in the following sense: for any formula φ of OWL DL, if $\mathcal{B}_{\texttt{init}} \models \varphi$ in the logic OWL DL$^{\texttt{call}}$ then $\mathcal{B}_{\texttt{final}} \models \varphi$ in the logic OWL DL (i.e. without taking into account call formulas), where $\mathcal{B}_{\texttt{init}}$ and $\mathcal{B}_{\texttt{final}}$ are the values of $\mathcal{B}$ before and after the application of Algorithm 2. In particular, this means that if an assertion $p(a, v)$ (where p is a τ-property) can be inferred from $\mathcal{B}_{\texttt{init}}$ using one or several call formulas but cannot be inferred from $\mathcal{B}_{\texttt{init}}$ without call formulas, then it can be inferred from $\mathcal{B}_{\texttt{final}}$ without call formula. Therefore, once Algorithm 2 is applied, the call formulas are useless until new formulas are added to the knowledge base.

Towards an improvement of the algorithm. This algorithm is based on an eager evaluation: all the calls to external functions that can be made are actually made. Now, a given query to the inference engine may not necessarily require all these calls. Therefore, triggering calls only when they are needed, according to the principle of lazy evaluation, would be beneficial in terms of computing time.

4.3 Using OWL DL$^{\texttt{call}}$ in practice

A first implementation of the algorithm described above is available online[4], together with the ontology and call formulas for the running example used in this article. This Python implementation relies on the OwlReady2 library[5] to manipulate knowledge in OWL DL and for interactions with the Hermit reasoner [6], and on SymPy to implement external functions based on symbolic computation. In practice, the tool takes as input an ontology that includes the definition of call formulas. It outputs a set of RDF triples corresponding to the assertions generated as results of executing the functions associated with those call formulas whenever relevant (i.e., the assertions that complete the ontology so that all inferences derivable from it based on OWL DL$^{\texttt{call}}$ can be obtained by a standard description logic reasoner). Therefore, taking inspiration from this example, a user might define new concepts of problems, the parts of their definitions that require computation, and the corresponding call formulas, so as to enable inferences combining both description logic reasoning and symbolic computation.

From a knowledge representation point of view, the following question must be raised: Given a mathematical piece of knowledge to be formalized, what parts of it should be represented in OWL DL

[4] https://github.com/mdaquin/OWLDLcall/
[5] https://pypi.org/project/owlready2/

and what part should be represented by external functions? In the example of Section 3, the choice consisted in leaving to the external functions the handling of mathematical expressions and keeping in OWL DL all the conceptual notions, that are usually expressed in natural language using a mathematical terminology. This seems to be a sensible approach to answer this representation question in general but, in some situations, this criterion may not be so simple to apply, for example, when the same theorem is expressed differently (with more or less formal parts) in two textbooks in mathematics. Therefore, this methodological question is likely to require more in-depth studies in the future.

4.4 Towards more complex cases

In the previous sections, it has been explained how some mathematical problems, represented by instances pb of the `Problem` concept can be solved thanks to the use of call-formulas in OWL DL$^{\texttt{call}}$. Some mathematical problems, such as the ones arising in physics for the modeling of micro-mirror arrays, are more complex in the sense that they require the use of several definitions and theorems. Provided that each of these definitions and theorems is triggered by an instance and given a first instance $\texttt{pb}_{\texttt{init}}$, the idea is to use call formulas to reify new instances (instead of datatype values).

Extension of the logic OWL DL$^{\texttt{call}}$. This extension consists in considering the new call formulas

$$r \sqsupseteq \texttt{call}(f, LPC, D, R)$$

with the following differences with the call formulas of the previous sections:

- r is an object property and R is a concept;
- f is interpreted in $\widetilde{f}$ that returns a new instance `inst` (i.e. an instance not occurring in the knowledge base) accompanied by assertions associated to this instance;[6]
- In particular if the call formula is triggered on a instance a of D, then one of the new assertions is always $r(a, \texttt{inst})$ (R is qualified as a *range concept* of r).

In what follows we use the following OWL DL formula, not introduced before:

$$r_1 ; r_2 ; \ldots ; r_n \sqsubseteq s$$

where r_1, r_2, …, r_n and s are object properties. An interpretation $\mathcal{I}$ satisfies this formula if, for every pair of instances (a, b), $\mathcal{I} \models (r_1 ; r_2 ; \ldots ; r_n)(a, b)$ entails $\mathcal{I} \models s(a, b)$.

Example. Let us consider the following problem:

$$\texttt{pb}_{\texttt{init}} = \text{find the eigenvalues of the matrix } M = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

For solving this problem, the following definitions and theorems are used:

(Def. 1) The characteristic polynomial of a square matrix M of order n is the polynomial $\det(X \cdot I_n - M)$ (where I_n is the unit matrix of order n and $\det(A)$ is the determinant of a square matrix A).

(Def. 2) The roots of a polynomial P are the solutions of the polynomial equation $P(t) = 0$ with unknown t.

(Th. 1) The eingenvalues of a square matrix M are the roots of its characteristic polynomial.

— And the definitions and theorems of Section 3.

In the following, the principles for implementing this example in the extended logic are outlined.

The definition of $\texttt{pb}_{\texttt{init}}$ is based on the introduction of an instance `mat` representing the matrix M:

21. `FindingEigenValues` $\sqsubseteq$ `Problem`;
22. `FindingEigenValues`($\texttt{pb}_{\texttt{init}}$);
23. `hasMatrix`($\texttt{pb}_{\texttt{init}}$, `mat`);
24. `hasExpression`(`mat`, `"[[2, 1], [1, 2]]"`).
25. `SquareMatrix`(`mat`);[7]

Now, (Def. 1) can be formalized by:

26. `hasCharacteristicPolynomial` $\sqsupseteq$
    `call("computeCharPol", [hasExpression], SquareMatrix, Polynomial)`.

using the Sympy function `"computeCharPol"`.

From these formulas the following assertions can be inferred:

27. `hasCharacteristicPolynomial`(`mat`, `inst1`);
28. `Polynomial`(`inst1`);
29. `hasExpression`(`inst1`, `"X ** 2 - 4 * X + 3"`).

Now, (Def. 2) can be formalized by:

30. `hasEquation;hasSolutionSet` $\sqsubseteq$ `hasRootSet`;
31. `hasEquation` $\sqsupseteq$
    `call("polToEquation", [hasExpression], Polynomial, PolynomialEquation)`.

where the Sympy function `"polToEquation"` consists in substituting `"X"` by `"t"`, concatenating `" = 0"` to a string and generating a new instance and relevant assertions.

From this, the following assertions can be inferred:

32. `hasEquation`(`inst1`, `inst2`);
33. `PolynomialEquation`(`inst2`);
34. `hasEquality`(`inst2`, `"t ** 2 - 4 * t + 3 = 0"`);
35. `hasUnknown`(`inst2`, `"t"`).

The instance `inst2` represents a second degree equation, which can be solved using the principles described in Section 3: using a call formula based on `findObviousSolEqDeg2` leads to the solution set $\{t = 1, t = 3\}$ represented by the instance `inst3`:

36. `hasSolutionSet`(`inst2`, `inst3`).

Now, (Th. 1) and the fact that the solution of a problem of finding eigenvalues of a matrix is the set of these eingenvalues can be expressed respectively by:

37. `hasCharacteristicPolynomial;hasRootSet` $\sqsubseteq$ `hasSetOfEigenValues`;
38. `hasMatrix;hasRootSet` $\sqsubseteq$ `hasSolution`.

from which it can be deduced that

39. `hasSolution`($\texttt{pb}_{\texttt{init}}$, `inst3`)

which solves the problem.

[6] Technically, adding these new assertions to the knowledge base does not keep the equivalence (the knowledge base after this addition is not equivalent to the one before it). However, the equivalence is recovered if new instances are considered as existential variables (as in a skolemization process).

[7] It is noteworthy that a call formula could be added to the knowledge base to make this assertion inferable from assertion 24 and this call formula.

Algorithmic issues. The application of the algorithm presented in Section 4.2 to this extension of OWL DL$^{\texttt{call}}$ does not always terminate as the example hereafter shows. Consider the formula $r \sqsupseteq \texttt{call}(f, [p], \top, \texttt{unsigned int})$ where p is a τ-property with $\tau = \texttt{unsigned int}$, and $\widehat{f}$ associates to $v \in \mathbb{N}$ an instance inst together with the assertion $p(\texttt{inst}, v+1)$. Then, this call formula and an assertion $p(a, 0)$ generates an infinite sequence of assertions $r(a, \texttt{inst1})$, $p(\texttt{inst1}, 1)$, $r(\texttt{inst1}, \texttt{inst2})$, $p(\texttt{inst2}, 2)$, $r(\texttt{inst2}, \texttt{inst3})$, $p(\texttt{inst3}, 3)$, etc., hence the non-termination of the algorithm.

Two directions of future work are considered in order to address this termination issue. The first one is to examine whether the inference relation in this logic is decidable and, if so, what algorithm can be proposed (intuitively, using lazy evaluation seems to be a good idea). The second one is to consider restrictions to ensure termination, even with the algorithm presented in Section 4.2. A possibility of such a restriction could be to have call formulas such that the set of concepts that are domains or ranges of these formulas are pairwise disjoint, which would avoid, at least, some loops.

Towards the representation and use of theorems in the domain of the asymptotic analysis of partial differential equations. Although the problems presented in this article are simple mathematical problems, the proposed approach has been developed with the goal to be adapted to the complex domain of asymptotic analysis of partial differential equations, a domain in which thousands of theorems have been developed in the literature. At this stage, the authors are convinced that at least a large proportion of such theorems can be represented in OWL DL$^{\texttt{call}}$ (with the extension presented in this section). However, scalability is an issue that might appear when actually dealing with an important number of theorems.

5 DISCUSSION AND CONCLUSION

This paper has presented a novel approach to the representation of mathematical knowledge in a formalism extending OWL DL by "call formulas", i.e. formulas related to calls to external functions (e.g. functions for handling mathematical expressions). The necessity of such a formalism has emerged from the way mathematicians express their knowledge, using both natural language with a mathematical terminology and mathematical expressions.

The use of calls to external functions is not a new issue in the realm of description logics and related formalisms, and it faces two opposing constraints: (1) the theoretical ideal of having a fully specified logic on which theoretical properties can be proven without additional assumptions, (2) the practical expressivity (i.e. the possibility of using all the required external functions).

The use of complex concrete domains as presented in [8] is more on the side of (1) on the (1)–(2) spectrum. For such a concrete domain $\tau = (\Delta_\tau, \Phi_\tau)$, $\sigma \in \Phi_\tau$ represents a relation σ^τ of Δ_τ that can be binary, ternary, etc., and not only unary as in concrete domains of OWL DL (in OWL DL, σ^τ is a subset of Δ_τ hence a unary relation on Δ_τ). We have given up the idea of using complex concrete domains because this approach would make it difficult (if not impossible) to represent even simple examples of mathematical notions.

Our approach is more on the (2) side of the spectrum: priority has been given to expressiveness, and the study of theoretical properties requires assumptions on the external functions, while remaining more nuanced than previous attempts such as in KL-ONE [3]. Indeed, KL-ONE enabled the definition of *interpretative hooks* "in which direct instructions to the interpreter are expressed in the language that implements the interpreter itself" without any restriction, making it impossible to provide any theoretical guarantee regarding the properties of the language.

It appears clearly throughout the paper that a lot of work remains to be done in order to reach our goal: a general-purpose logic for representing mathematical knowledge associated with a scalable inference system. The current prototype can handle simple mathematical problems, but considering complex problems (involving, in our setting, the creation of new instances during the inference process) requires an extension of the logic that raises theoretical and practical issues, as described in Section 4.4.

Another issue to be studied occurs when several paths exist in order to solve a problem. In Section 3, each of the problems pb1 and pb2 can be solved using two methods: findObviousSolEqDeg2 and useDiscriminant, but which one should be tried first? Being able to infer from knowledge about mathematical problem-solving strategies the priority with which to consider each method and to use this priority in the inference algorithm constitutes another direction for future work.

From an application point of view, our objective is to represent and handle numerous theorems and their proofs in our target domain of asymptotic analysis of partial differential equations, in order to solve practical problems in physics, e.g. modeling micro-mirror arrays. Following this line, a (very) long-term future work would be to solve problems of this domain for which no theorem is available, by *adapting* some theorems (and their proofs) to such a problem, according to the principles of case-based reasoning [11].

REFERENCES

[1] W. Belkhir, N. Ratier, D. D. Nguyen, N. B. T. Nguyen, M. Lenczner, and F. Zamkotsian. 2017. A tool for aided multi-scale model derivation and its application to the simulation of a micro mirror array. In *2017 18th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*. IEEE, 1–8.

[2] W. Belkhir, N. Ratier, D. D. Nguyen, B. Yang, M. Lenczner, F. Zamkotsian, and H. Cirstea. 2015. Towards an automatic tool for multi-scale model derivation illustrated with a micro-mirror array. In *17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 47–54.

[3] R. Brachman, E. Ciccarelli, N. Greenfeld, and M. Yonke. 1978. *KL-ONE reference manual.* Technical Report. BBN report.

[4] Ronald J. Brachman and Hector J. Levesque. 2004. *Knowledge Representation and Reasoning.* Morgan Kaufmann.

[5] MD Canonica, F Zamkotsian, P Lanzoni, W Noell, and N De Rooij. 2013. The two-dimensional array of 2048 tilting micromirrors for astronomical spectroscopy. *Journal of Micromechanics and Microengineering* 23, 5 (2013), 055009.

[6] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. 2014. HermiT: an OWL 2 reasoner. *Journal of automated reasoning* 53 (2014), 245–269.

[7] C. Lange. 2013. Ontologies and languages for representing mathematical knowledge on the semantic web. *Semantic Web* 4, 2 (2013), 119–158.

[8] C. Lutz. 2003. Description Logics with Concrete Domains – A Survey. In *Advances in Modal Logics Volume 4.* King's College Publications.

[9] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. 2009. OWL 2 web ontology language profiles. W3C recommendation.

[10] D. D. Nguyen, W. Belkhir, N. Ratier, B. Yang, M. Lenczner, F. Zamkotsian, and H. Cirstea. 2015. A multi-scale model of a micro-mirror array and an automatic model derivation tool. In *16th Int. Conf. on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems*. IEEE, 1–9.

[11] C. K. Riesbeck and R. C. Schank. 1989. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey.