Towards RIF-OWL Combination: An Effective Reasoning Technique in Integrating OWL and Negation-free Rules

MASTERARBEIT
zur Erlangung des akademischen Grades

Master of Science
im Rahmen des Studiums

DDP Computational Logic
ausgeführt von

Mohammad Sadnan Al-Manir
Matrikelnummer 0727244

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: O. Univ. Prof. Dr. Thomas Eiter
Externer Mitbetreuer: Dr. Jos de Bruijn (Freie Universität Bozen)
Mitwirkung: Univ.-Ass. MSc. Maria Magdalena Ortiz de la Fuente

Wien, 09.04.2009

(Unterschrift Verfasser/in)  (Unterschrift Betreuer/in)

Technische Universität Wien
A-1040 Wien • Karlsplatz 13 • Tel. +43(0)1/58801-0  http://www.tuwien.ac.at
Abstract

For the development of Semantic Web the World Wide Web Consortium has focussed on the future Web Ontology Language standard OWL 2 and rule languages in Rule Interchange Format which are required to interoperate. A lot of researches have been carried out towards the integration of various Description Logics and rules and RIF-OWL combination is a step forward in this direction. Most of the Description logics are a decidable subset of classical First-Order logic while rules originate from logic programming. In this thesis, we discuss the combination of Description Logics and negation-free rules, both of which are expressed in the standard First-Order logic semantics. We propose an algorithm for reasoning which is sound but not complete for these combinations. Our algorithm uses existing standard reasoning tool for retrieving facts from the Description Logic Knowledge Base with which rules are put together to form a logic program. Finally, rule reasoner is used for answering conjunctive queries on this logic program. We identify the reasons behind the incompleteness and chose a subset of these combinations which combines restricted Description Logic and rules. This restricted subset consists of Description Logic Horn-$SHIQ$ and rules such that Description Logic predicates are allowed only in the rule bodies and all the rules are DL-safe. A prototype implementation of the reasoning process is also presented which uses Pellet Description Logic reasoner and XSB rule reasoner showing the effectiveness of using existing tools with our algorithm. The combination chosen in this thesis can express strictly more information in the Description Logic component compared to the combination of DLP extended using rules.
Acknowledgements

At first, I would like to thank my supervisor in TU WienProf. Thomas Eiter for his remarks and valuable suggestions. I specially thank my external supervisor Jos de Bruijn in Free University of Bozen-Bolzano for his endless support throughout my project and thesis. He explained and answered all relevant questions with great patience and clarity. His sound knowledge in Description Logics and logic programming was very helpful in guiding me in various stages. Most of all, despite his busy schedule he managed to make time for me whenever I needed.

In addition, I would like to thank Thomas Krennwalner for his initial support. Also my thank goes to my good friend Domenico Carbotta for his suggestions and continuous help. My hearty gratitude is also to Nabil. Moreover, I would like to thank the participating universities for providing such an opportunity to study this unique masters program.

Finally, I would like to thank my parents who have given me unconditional love and support throughout my whole life.
# Table of Contents

1 Introduction .................................................................................................................... 1
   1.1 Motivation ................................................................................................................. 1
   1.2 Background .............................................................................................................. 2
   1.3 Contribution ............................................................................................................. 4
   1.4 Outline ..................................................................................................................... 6

2 Preliminaries .................................................................................................................. 7
   2.1 First-Order Logic ..................................................................................................... 7
       2.1.1 Syntax ............................................................................................................. 7
       2.1.2 Semantics ..................................................................................................... 10
   2.2 Description Logics ................................................................................................... 12
       2.2.1 Syntax .......................................................................................................... 12
       2.2.2 Semantics ..................................................................................................... 14

3 Statement of Problems for Combining Description Logics and Rules .................. 17
   3.1 Combination of DLs and rules ............................................................................... 17
       3.1.1 Syntax ........................................................................................................... 17
       3.1.2 Semantics ..................................................................................................... 19
   3.2 Reasoning in Combined Knowledge Base ............................................................. 19
   3.3 Undecidability of reasoning ................................................................................. 20
   3.4 Existing combination: DLP extended with rules .................................................. 21
       3.4.1 DLP syntax .................................................................................................. 21
       3.4.2 Reasoning in this combination .................................................................... 22
       3.4.3 Expressive limitation ................................................................................. 24
   3.5 Problem Statement ............................................................................................... 25

4 An Effective Reasoning Procedure for Query Answering .................................... 27
   4.1 Description of Algorithm ...................................................................................... 27
   4.2 Properties of the Algorithm ................................................................................ 28
   4.3 Computational Complexity of the algorithm ...................................................... 30

5 Imposing Restrictions: Regaining Completeness .................................................... 33
   5.1 Restrictions on Knowledge Base Components .................................................. 33
   5.2 Incompleteness: Motivations, Analysis and Solutions ....................................... 33
      5.2.1 Disjunction .................................................................................................. 34
# TABLE OF CONTENTS

5.2.2 Modular Reasoning .................................................. 35  
5.2.3 Named/Unnamed Objects ......................................... 36  
5.2.4 Problems with equality (inequality) ......................... 38  
5.3 Effects of Restrictions: Comparison with DLP extended with rules ................................................................. 42  

6 Implementation and Experiments ................................. 45  
6.1 Reasoning Tools ......................................................... 45  
6.1.1 Pellet ................................................................. 45  
6.1.2 The XSB System ...................................................... 47  
6.1.3 Syntax of Rules in XSB ........................................... 47  
6.2 Input Format ............................................................. 48  
6.2.1 Description Logic Knowledge Base ......................... 48  
6.2.2 RIF rules ............................................................. 49  
6.3 Reasoning Process: Architecture ............................... 52  
6.4 Experiments ............................................................. 54  
6.4.1 Example 1 ............................................................ 54  
6.4.2 Example 2 ............................................................ 57  
6.4.3 Scaling of various inputs ...................................... 58  

7 Conclusion ................................................................. 61  
7.1 Summary ................................................................. 61  
7.2 Relationship with other combination approaches .......... 62  
7.3 Future work .............................................................. 64

A Appendix ................................................................. 65  
A.1 courses.owl Ontology ............................................... 65

B Appendix ................................................................. 69  
B.1 airtravel.owl Ontology ................................................ 69
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Terminological and assertional axioms</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Translation of $SHOIQ$ into FOL</td>
<td>15</td>
</tr>
<tr>
<td>6.1</td>
<td>Architectures of Reasoning Process</td>
<td>52</td>
</tr>
<tr>
<td>6.2</td>
<td>Reasoning Time Vs Number of Instances for Example 1</td>
<td>59</td>
</tr>
<tr>
<td>6.3</td>
<td>Reasoning Time Vs Number of Instances for Example 2</td>
<td>59</td>
</tr>
</tbody>
</table>
1

Introduction

1.1 Motivation

In recent years, the Semantic Web research focussed on introducing a suitable ontology language for modeling. This resulted in the introduction of Web Ontology Language (OWL) that is based on Description Logics - a family of Knowledge Representation (KR) formalisms. It is known that most of the Description Logics is a subset of function-free First-Order Logic with equality. Due to significant amount of research, Description Logics have gained enough maturity and more efforts are underway by enriching this to build next generation Web Ontology Language standard OWL 2 [8, 9]. The computational properties of different Description Logic languages have been studied extensively. Various reasoning techniques and tools have been implemented and their applications have been proven effective. Examples of some of the widely used and popular reasoning tools are Racer [1], Pellet [2], KAON2 [3], FaCT++ [4].

In Description Logics (DLs) the axioms are able to model information of the objects even if they are not required to be explicitly present in the Knowledge Base. The axioms can also provide disjunctive information and classes appearing in axioms need not be atomic. They can be complex and compound expressions can be built from atomic classes both in the axiom antecedents and consequents using a variety of constructors. But during the research and application building stages several shortcomings of OWL have also been identified. For example, in general ontology modeling in Description Logics are close to having a tree shape and hence hierarchical modeling is more convenient in it. The concepts and roles in the language can only express information corresponding to unary and binary predicate respectively, but not with an arity more that two.

Negation-free rules, on the other hand are based on Horn logic fragment of First-Order logic and can be proposed as a possible solution even though they have limitations. For example, they are not able to express existential information and in their heads disjunctive information are also not allowed to occur. They also disallow negation and complex expressions are difficult to build. But unlike Description Logics, rules can model information in a
CHAPTER 1. INTRODUCTION

non-tree structure and is able to express queries with arbitrary arity, not only unary and binary.

Bringing these two KR formalisms in a common framework could enhance the expressive capability of information because knowledge that cannot be modeled in either Description Logics or rules, can be modeled in their integration. Hence, interoperability between these two KR formalisms is seen as a central task in the development of the Semantic Web language stack [5]. The Rule Interchange Format (RIF) Working Group of World Wide Web Consortium is currently working on developing such rule language RIF-BLD [6] and a detailed technical specification towards the integration of RIF-OWL combination [7] is being thoroughly investigated. Therefore, the combination of Description Logics and rules is of great interest now-a-days and is treated as a very important task in the development of future Semantic Web.

But there are challenges in the integration of Description Logics and rules. It has already been shown in [10] that reasoning is undecidable in the integration of recursive Horn rules and even moderately expressive Description Logic language. So, in a Description Logic Knowledge Base where reasoning is decidable and a rule Knowledge Base where reasoning is decidable, reasoning in their integration is not decidable. It turns out that expressivity of the integration is related to the decidability of reasoning.

1.2 Background

A number of reasoning techniques for integrating Description Logics and rules have been investigated which cover both monotonic and non-monotonic approaches. Both of them can be categorized based on the reasoning techniques they follow namely hybrid and homogenous. While hybrid reasoning combines both Description Logic reasoning and rule reasoning together, homogenous approach uses only one reasoner. Hybrid approach is further categorized based on how tightly the components are integrated.

Hybrid approach in non-monotonic direction is described here where integrations adopts different semantics than in monotonic directions. Based on how strongly Description Logics and rules are coupled, the integration techniques can be loose, in some cases tight or fully integrated.

The Description Logics and rules are said to be loosely integrated if each of them act as if they were separate and independent component. The integration is implied by an interface mechanism based on introducing special safety of the variables in rules. The loose integration techniques are demonstrated in non-monotonic dl-programs [11, 12], defeasible logic integrated with Description Logics [13] as well as in probabilistic dl-programs [14].

In tight semantic integration, although the vocabularies are distinct for the predicates of the components, but are integrated in a larger extent.
1.2. BACKGROUND

The semantics of this integration is based on single model such that it is composed of two separate models who share the same domain and each of which should satisfy the respective Description Logic and rule component. Examples of this sort of integrations are r-Hybrid KBs [15, 16, 17], DL+log [18].

Unlike tight integration approach, full integration of Description Logics and rules use the same vocabularies and in a sense the universes under the two components adopt homogenous characteristics using required axiomatizations. Full integration is evident in the approaches as Hybrid MKNF Knowledge Bases [19], First-Order Autoepistemic Logic [20] and open Answer Set Programs [21]. Terminological Default Logic [22], Description Logics of Minimal Knowledge [23] are some predecessors in this direction.

In addition to these non-monotonic approaches, the possibility of combining monotonic rules with Description Logics have been extensively studied as well in $\mathcal{AL}$-log [24], CARIN [10], SWRL [25, 26, 27], DLP [28] extension using rules, DL-safe SWRL rules [29]. Different types of restrictions have been applied on the components to make the integration as much expressive as possible while maintaining the reasoning decidable thus introducing a trade-off between expressivity and decidability. These restriction sometimes are on the shape of rules, sometimes on the Description Logics and sometimes on both on them.

In CARIN and DLP extension with rules approach Description Logic component is restricted. In CARIN, It was found that certain combinations of Description Logic constructors were responsible for undecidability of reasoning. Hence to achieve decidable reasoning these constructors are disallowed in the combination with rules. DLP which is a fragment of DL $SHIQ$, falls essentially in Horn fragment of First-Order logic. Here the DL is restricted by mainly disallowing disjunction and existential information in the consequents of the DL axioms. The transformation of DLP is then straightforward and they can be easily extended with rules resulting the combination of DLs and rules. Decidable reasoning is possible in this combination. DLP is different in nature from the DLs in $\mathcal{AL}$-log, CARIN and SWRL because of its correspondence to Horn fragment. While DLP falls essentially in Horn fragment from the very beginning requiring no more restrictions on its DL, each of the DLs in $\mathcal{AL}$-log, CARIN and SWRL needs further restrictions for decidable reasoning in their combination with rules.

Different reasoning techniques mentioned above use rule restrictions in syntactic forms. Except for DLP, all three of $\mathcal{AL}$-log, CARIN and DL-safe SWRL use rule restrictions for decidable reasoning. All of the restrictions on rules are on the variables that belong to DL-atoms and are based on the idea of typical rule safeness. While $\mathcal{AL}$-log and DL-safe SWRL use DL-safety restriction, CARIN uses role-safety. The difference in this safeness is twofold. First, while DL-safety considers both concept ($\mathcal{AL}$-log rules allow only concepts) and role atoms, role-safety only considers roles. Second, in
role-safety restriction, only one variable in role is required to appear in non-DL-atom unless it already appears so. The DL-safety restriction requires all the variables instead.

All these restrictions put limitations on the expressivity of the combined language. Among the four approaches discussed above, $\mathcal{AL}$-log rules do not allow roles. Approaches in CARIN, SWRL and DLP extended with rules allow roles in their rules. DLP ontologies disallow disjunction and existential restriction in the consequents of their axioms. Although DLP cannot express existential information, because of the simple representation and correspondence to Horn logic fragment, it is popularly being used in practice. Rule restrictions in $\mathcal{AL}$-log, CARIN and DL-safe SWRL also limit the expressivity. For example, DL-safety restriction in $\mathcal{AL}$-log and DL-safe SWRL imposes restriction on the use of conjunctive query. Also role-safety in CARIN is a weaker version of the DL-safety as it only considers roles and not all the variables in them. This makes it easier to use conjunctive query in CARIN than in $\mathcal{AL}$-log and DL-safe SWRL.

The expressivity of these approaches are also visible from their ability to use the DL-atoms in their rules. In $\mathcal{AL}$-log and CARIN, rules do not allow DL-atoms in their heads but only in bodies. On the other hand, SWRL and DLP extended with rules allow them both in heads and bodies. This shows that information can flow in both directions, to and from DLs and rules thus making $\mathcal{AL}$-log and CARIN less expressive than the other two approaches.

The integration of Description Logics and negation-free rules in CARIN and SWRL is tight because it is based on single model of standard First-Order semantics. Interestingly, RIF-OWL combination in some sense adopts this single model theory which is essentially the same as in SWRL and we are only interested in this kind of semantics.

1.3 Contribution

In this thesis, we are going to focus on the integration of Description Logics and negation-free rules that corresponds to the RIF-OWL combinations that is essentially based on First-Order semantics. The combination proposed here is strictly more expressive in the Description Logic component when combined with rules than in the combination of DLP extended using rules.

For reasoning in this combination, we propose an algorithm that is sound. This algorithm follows a modular reasoning approach which incorporates Description Logic reasoning and rule reasoning using their respective reasoners. In this algorithm, the Description Logic reasoner takes a Description Logic Knowledge Base $\mathcal{O}$ as input and retrieves the entailed facts $\mathcal{F}$ from the ontology. Then the set of rules in $\mathcal{P}$ is put together with these facts forming a logic program $\mathcal{P}' = \mathcal{F} \cup \mathcal{P}$. Now answering conjunctive queries is possible using a rule reasoner on this logic program.
Although this algorithm is sound, it is incomplete for every combination of Description Logics and rules. We identify the reasons behind this incompleteness and propose a subset of this type of combination for which the algorithm is complete. Based on the reasons found, this particular subset imposes certain restrictions on both the Description Logic and rules component. Description Logic Horn-$SHIQ$ [31] is chosen because of its ability to express existential information in its axiom consequents and avoid disjunctive information. It is already known that data complexity in Horn-$SHIQ$ is low in general and is an attractive choice for reasoning with large number of individuals. Standard reasoner like KAON2 also facilitates reasoning in Horn-$SHIQ$ and it captures most of the functionalities of Description Logics. Due to having qualified number restriction it is not a fragment of OWL DL. As $SROIQ$ is a strong choice for upcoming standard OWL 2 which allows this restriction, it would be interesting to explore the possibility of using Horn-$SROIQ$ as an extension of Horn-$SHIQ$. On the other hand, rule restrictions are twofold. Because of the modular approach followed by the algorithm which does not facilitate feedback from one reasoner to another (here rule reasoner to Description Logic reasoner), DL-atoms are allowed only in rule bodies. Another rule restriction is to introduce DL-safety which are syntactic restrictions on rule variables, ensuring that only named objects are considered during rule reasoning as well as the logic program should be partially constant-free.

Our algorithm is simple and easy to implement. We can use any existing standard reasoner for Description Logics and rules. This is shown in our implementation by using Pellet Description Logic reasoner and XSB rule reasoner.

In support of this, we present an implementation that proves the functionality of our algorithm towards RIF-OWL combinations. To retrieve facts, we use Description Logic Horn-$SHIQ$ ontology. Another input is a set of rules in RIF format, simply called RIF rules. RIF corresponds to the language of definite Horn rules with equality and thus serves the purpose of negation-free rules. Using a rewriting tool these RIF rules are rewritten into the input format compatible with rule reasoner. Finally, conjunctive queries are answered in this combination.

In short, in this thesis we discuss a combination of Description Logics and rules corresponding towards the integration of RIF-OWL and development of next generation Semantic Web architecture. We propose a simple algorithm for reasoning in this combination which is sound but not complete for all such combinations. To regain completeness we investigate one subset of such combination by imposing required restrictions on Description Logics and rules. We implemented one prototype of the reasoning approach and showed that our algorithm is capable of using existing reasoning tools. Finally, it is clear that our choice of combination is strictly more expressive with respect to the ontology information it can express compared to the well
known combination DLP extended using rules.

1.4 Outline

The organization of the thesis consists of the following chapters in order:

2. Preliminaries In this chapter we define the basic terminologies to be used in the rest of this thesis. We recall First-Order logic syntax and semantics. Then Description Logics and rules are defined accordingly.

3. Statement of Problems for Combining Description Logics and Rules We start this chapter by describing the syntax and semantics of the combination. Conjunctive query answering is chosen as a reasoning task for this combination and hence defined thereafter. Then we recall how reasoning is undecidable in general in this combination. We describe an existing combination approach where reasoning is performed in DLP extended using rules. The expressive limitation of this approach is pointed out and the problem statements regarding our combination is stated at the end.

4. An Effective Reasoning Procedure for Query Answering We provide an algorithm for query answering in our combination in this chapter. We also talk about the properties of the algorithm: modularity, soundness and completeness. We also analyze the computational properties.

5. Imposing Restrictions: Regaining Completeness Here we analyze the reasons of incompleteness of our algorithm and provide solutions for each problem. We prove that our algorithm is complete with our proposed restrictions. The effects of these restrictions are discussed and compared with the DLP approach.

6. Implementation and Experiments The implementation of our approach is described in this chapter. For the implementation, we use Pellet reasoner to retrieve the facts from the ontology. For rules, we use the so called RIF syntax. These RIF rules are rewritten into XSB rules. Both the rules and facts together form a logic program. Finally, query answering is performed in this logic program using this rule reasoner and shows the effectiveness of our algorithm. Two examples are added in this purpose at the end of this chapter.

7. Conclusion In this chapter, we summarize the contents of our combination approach. We discuss the relationships to similar approaches and compare the expressivity. We also try to point out some possible directions for future research.
In this chapter we formally introduce all terminologies required for the rest of the thesis. We start by defining what Description Logic is, then move on to the classical First-Order Logic. We state the logical formalism by which Description Logic can be easily expressed in First-Order logic. We also define rules in Horn Logic Programming. In each of these cases, the syntax and semantics have been discussed as well.

2.1 First-Order Logic

We recall here the classical First-Order predicate Logic (FOL).

2.1.1 Syntax

Definition 2.1 (Signature) The signature $\Sigma = \langle P, F, C \rangle$ of a first-order language $\mathcal{L}$ consists of:

1. A finite set $P$ of predicate symbols, each of which has a positive integer called arity associated with it.

2. A finite set $F$ of function symbols, each of which has an arity associated with it.

3. A finite set $C$ of constant symbols.

The language of FOL is defined as $\mathcal{L}(P,F,C)$.

Along with the signature $\mathcal{L}$ also contains:

- Set of variables $V$ e.g., $x, y, \ldots$

- Set of connectives And($\land$), Or($\lor$), Implies($\rightarrow$), if and only if($\leftrightarrow$) and not($\lnot$)

- Universal quantifier $\forall$ and Existential quantifiers $\exists$
CHAPTER 2. PRELIMINARIES

Two types of expressions are of utmost importance: terms which are intended to be description of objects and formulas that are for describing statements.

Definition 2.2 (Terms) Terms are

- All variables and constant symbols of \( L \),
- All \( f(t_1, \ldots, t_n) \) in which \( f \in F \) is an \( n \)-ary function symbol and \( t_1, \ldots, t_n \) are terms (formed previously).

Example 2.1 Let \( f \) and \( s \) are binary (2-ary) and unary (1-ary) function symbols respectively, \( c \) is a constant symbol and \( y \) is a variable. Then \( f(s(c), y) \) is a term.

Definition 2.3 (Atomic Formula) Atomic Formulas or atoms are

- All \( p(t_1, \ldots, t_n) \) in which \( p \in P \) is an \( n \)-ary predicate symbol and \( t_1, \ldots, t_n \) are terms
- \( \top \) and \( \bot \)
- All \( t_1 = t_2 \) where \( t_1, t_2 \) are terms

Definition 2.4 (Horn Clause) A Horn clause is a clause with at most one positive atom. Generally, it is presented as one of the three forms below:

- A Horn rule consisting of one positive atom and at least one negative atom.
  For atoms \( b_1, \ldots, b_n, h \in P \) a Horn clause
  \[
  \forall \bar{x}, \bar{y_1}, \ldots, \bar{y_n}. (\neg b_1(\bar{y_1}) \lor \ldots \lor \neg b_n(\bar{y_n}) \lor h(\bar{x}))
  \]
  is logically equivalent to
  \[
  \forall \bar{x}, \bar{y_1}, \ldots, \bar{y_n}. b_1(\bar{y_1}) \wedge \ldots \wedge b_n(\bar{y_n}) \rightarrow h(\bar{x})
  \]
  such that \( n \geq 0 \), \( h(\bar{x}) \), and each \( b_i(\bar{y_i}) \) for \( 0 \leq i \leq n \) is an atom and
  - \( h \), \( b_1, \ldots, b_n \) are positive atoms of arity \( n \)
  - \( H(r) = \{h\} \) is the set of head atoms
  - \( B(r) = \{b_1, \ldots, b_n\} \) is the set of body atoms
  - \( \bar{x} \), each \( \bar{y_i} \) is \( n \) - tuple of variables and constants
2.1. FIRST-ORDER LOGIC

The sign ← is read as if, hence the rule becomes head if body or if body then head, with universal quantification on the outer level.

From the presentation of Horn rules it is clear that they are negation-free. In the rest of this thesis, we consider only Horn rules.

- A fact consisting of only one positive atom

  \( e.g., \text{Male}(\text{John}) \)

- A negated goal consisting of at least one negative atom. The goal to find out if there is a male who has a father named James is written as

  \( \neg\text{Male}(x) \lor \neg\text{hasFather}(x, \text{James}) \)

**Definition 2.5** (Rule safety) A rule is safe when each variable occurring in head also appears in the positive atom in the rule body.

**Example 2.2** A person is good in sports if he is a sportsPerson and he has got a prize in Olympics event.

\[
\forall x, y. \text{sportsPerson}(x) \land \text{sprotsEvent}(y) \\
\land \text{gotPrizeIn}(x, \text{Olimpics}) \rightarrow \text{goodInSports}(x, y)
\]

This rule is safe as both \( x, y \) appear in the rule body in positive atoms.

In this survey, whenever we mention a rule, we mean that they are safe.

**Definition 2.6** (Logic Program) A logic program (simply program) \( P \) is a finite set of rules with respect to \( \Sigma \). A logic program is safe if all the rules are safe.

**Definition 2.7** (Formula) If each of \( \varphi, \psi \) is a formula of \( \mathcal{L} \), then Formula in \( \mathcal{L} \) is defined as follows:

- Any atomic formula is a formula
- \( (\neg \varphi) \) is a formula (Negation)
- \( (\varphi \land \psi) \) is a formula (Conjunction)
- \( (\varphi \lor \psi) \) is a formula (Disjunction)
- \( (\varphi \rightarrow \psi) \) is a formula (Implication)
- \( (\varphi \equiv \psi) \) is a formula (Equivalence)
- \( \forall x \varphi \) is a formula, for variable \( x \)
• \( \exists x \varphi \) is a formula, for variable \( x \)

**Definition 2.8** (Scope of quantifiers) The quantifier combinations \( \forall x \) and \( \exists x \) that begin the quantifications \( \forall x \varphi \) respectively \( \exists x \varphi \) have \( \varphi \) as their scope.

Quantifiers \( \forall x, \exists x \) bind every occurrence of \( x \) in their scope – at least, if such an occurrence is not already in a smaller scope of such a quantifier occurring inside in the scope of the first one. Occurrences of variables that are not bound are called free.

A sentence is a formula without free variables.

An expression without variables and quantifiers is called ground. A ground formula is called closed. A formula which is not closed is open.

**Example 2.3** Only the first occurrence of \( x \) in \( (p(x) \rightarrow \exists x q(x, y)) \) is free in this formula. Adding one universal quantifier \( \forall x, \forall x (p(x) \rightarrow \exists x q(x, y)) \) has no free occurrences of \( x \). The quantifier \( \forall x \) only binds the occurrence of \( x \) in \( p(x) \); it does not bind the one in the \( q \)-atom (bound already by \( \exists x \)).

**Definition 2.9** (FOL Knowledge Base) An FOL Knowledge Base \( \Phi \) in \( \mathcal{L} \) can be written denoted as \( \Phi \subseteq \mathcal{L} \) where \( \Phi \) is a set of formulas mentioned above. The KB \( \Phi \) is also named as FOL Theory.

### 2.1.2 Semantics

**Definition 2.10** (Interpretation) The interpretation of \( \mathcal{L} \) is a tuple \( J = \langle U, \mathcal{I} \rangle \), where \( U \) is a non-empty set (domain) and \( \mathcal{I} \) is a mapping called interpretation function that associates:

- To every constant symbol \( c \in C \), some member \( c^J \in U \)
- To every \( n \)-place function symbol \( f \in F \), some \( n \)-ary function \( f^J : U^n \rightarrow U \)
- To every \( n \)-place relation symbol \( p \in P \), some \( n \)-ary relation \( p^J \subseteq U^n \).

**Definition 2.11** (Assignment) An assignment is a mapping \( A \) from the set of variables to the set \( U \). The image of the variable \( v \) is denoted as \( v^A \) under an assignment \( A \).

**Definition 2.12** (Interpretation) Let \( J = \langle U, \mathcal{I} \rangle \) be the interpretation for the language \( \mathcal{L}(P, F, C) \) and let \( A \) be an assignment in this interpretation. To each term \( t \) of \( \mathcal{L}(P, F, C) \), we associate a value \( t^J,A \) in \( U \) as follows:

1. For a constant symbol \( c \), \( c^J,A = c^J \).
2. For a variable \( v \), \( v^J,A = v^A \).
2.1. FIRST-ORDER LOGIC

3. For a function symbol \( f \), \([f(t_1, \ldots, t_n)]^I, A = f^I(t_1^I, \ldots, t_n^I, A)\)

This definition associates a value in \( U \) with each term of the language. If the term is closed i.e. has no variables, its value does not depend on the assignment \( A \). For closed terms we often write \( t^I \) instead of \( t^I, A \).

**Definition 2.13 (Satisfiability)** Let \( J = \langle U, I \rangle \) be an interpretation of the language \( L(P, F, C) \), \( A \) a variable assignment and \( \phi \) a formula in \( L \). Given \( A \) and \( \phi \), \( J \) satisfies \( \phi \) iff \( J \models_A \phi \) which is inductively defined as follows:

- \( J \models_A \top \) and \( J \not\models_A \bot \)
- \( J \models_A t_1 = t_2 \iff t_1^I, A = t_2^I, A \)
- \( J \models_A \neg \phi \iff J \not\models \phi \)
- \( J \models_A (\phi \land \psi) \iff J \models_A \phi \) and \( J \models_A \psi \)
- \( J \models_A (\phi \lor \psi) \iff J \models_A \phi \) or \( J \models_A \psi \)
- \( J \models_A (\phi \rightarrow \psi) \iff J \not\models \phi \) or \( J \models_A \psi \)
- \( J \models_A (\phi \equiv \psi) \iff J \not\models \phi \) or \( J \models_A \psi \) and \( J \models_A \phi \) or \( J \not\models \phi \)
- \( J \models_A \forall x \phi \) iff for every \( x \)-variant \( A' \) of \( A \), \( J \models_{A'} \phi \)
- \( J \models_A \exists x \phi \) iff for some \( x \)-variant \( A' \) of \( A \), \( J \models_{A'} \phi \)

An interpretation \( J \) satisfies a formula \( \phi \) i.e. \( \phi \) is true in \( J \), written \( J \models \phi \) iff \( J \models_A \phi \) for all variable assignments \( A \). A formula \( \phi \) is valid iff \( J \models \phi \) for all interpretations \( J \). A formula \( \phi \) is satisfiable iff \( J \models \phi \) for for some interpretation \( J \).

**Definition 2.14 (Model)** Let \( J = \langle U, I \rangle \) be an interpretation of the language \( L(P, F, C) \). We say, \( J \) is a model of an FOL KB \( \Phi \) iff \( J \models \phi \) for every formula \( \phi \in \Phi \).

**Definition 2.15 (Entailment)** An FOL KB \( \Phi \) entails a formula \( \phi \in L \) i.e. \( \Phi \models \phi \) iff for every model \( J \) in \( L \) for which \( J \models \Phi \), also \( J \models \phi \).
2.2 Description Logics

Description Logics (DLs) [32] is a family of Knowledge Representation formalisms equipped with a formal logic-based semantics, which captures the structural knowledge portion of the application domain by first defining the relevant concepts of the domain, and then using these concepts to specify properties of objects and individuals occurring in the domain. A lot of emphasis is also implied on the reasoning in DLs which allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the Knowledge Base.

In this section, we formally define the expressive Description Logic $\mathcal{SHOIQ}$ which is a logical counterpart of OWL DL. To describe the syntax and semantics of $\mathcal{SHOIQ}$, we start by defining the constituents concept and role as well as axioms.

2.2.1 Syntax

Elementary descriptions are composed of atomic concepts (concept names) and atomic roles (role names). All DL systems ([32], Chapter 8) allow their users to build complex descriptions inductively using a number of constructors.

The basic building blocks of DL $\mathcal{SHOIQ}$ are as follows:

- Concept names
- Role names
- Individuals
- Operators

Let $\mathbf{A}$ be the set of concept names with a subset $\mathbf{I} \in \mathbf{A}$ of individuals (also known as nominals). These concept names are unary atoms and individuals are constants.

Let the set of role names $\mathbf{R}$ is such that $\mathbf{R} = R_+ \cup R_S$ and $R_+ \cap R_S = \emptyset$ where $R_+$ is a transitive role names and $R_S$ contains simple role names. Then $\mathcal{SHOIQ}$ roles are defined as $\mathbf{R} \cup \{R^{-} \mid R \in \mathbf{R}\}$. $R^{-}$ denotes the inverse of $R$ and for notational convenience Inv function is used. Role names are binary atoms.

As the inverse relation on roles is symmetric, hence it is possible to express $\text{Inv}(R) = R^{-}$ and $\{\text{Inv}(R^{-})\} = R$. For transitive rule, the function $\text{Trans}$ returns true if $R$ is a transitive role regardless of being a role name, inverse of a role name, transitive role name or inverse of a transitive role name.

Let $A \in \mathbf{A}$ is an atomic concept, $R \in \mathbf{R}$ is an simple atomic role, $o_1, \ldots, o_n$ are individuals and $n \in \mathbb{N}$ is a non-negative integer.
2.2. DESCRIPTION LOGICS

The operators used in DLs is a set of constructors which are $\equiv, \subseteq, \neg, \sqcap, \sqcup, \forall, \exists$.

Then concept description $C, D$ can be inductive defined as follows:

$\begin{align*}
C, D & \rightarrow & A & \text{ (atomic concept)} \\
& & \top & \text{ (universal concept)} \\
& & \bot & \text{ (bottom concept)} \\
& & \neg C & \text{ (complement)} \\
& & C \sqcap D & \text{ (conjunction)} \\
& & C \sqcup D & \text{ (disjunction)} \\
& & \forall R.C & \text{ (value restriction)} \\
& & \exists R.C & \text{ (existential restriction)} \\
& & \geq nR.C & \text{ (Qualified number restriction-atleast)} \\
& & \leq nR.C & \text{ (Qualified number restriction-atmost)} \\
& & \{ o_1, \ldots, o_n \} & \text{ (oneOf)} \\
& & \exists R.\{ o \} & \text{ (hasValue)}
\end{align*}$

We now introduce the axioms denoted as $F$.

**Definition 2.16 (Axiom)** Let $C, D$ be concepts, $R, S$ be roles and $a, b$ be the individuals. An axiom is of the following form as shown in Figure 2.1:

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept inclusion</td>
<td>$C \subseteq D$</td>
</tr>
<tr>
<td>Role inclusion</td>
<td>$R \subseteq S$</td>
</tr>
<tr>
<td>Concept equivalence</td>
<td>$C \equiv D$</td>
</tr>
<tr>
<td>Role equivalence</td>
<td>$R \equiv S$</td>
</tr>
<tr>
<td>Role transitivity</td>
<td>$\text{Trans}(R)$</td>
</tr>
<tr>
<td>Concept membership</td>
<td>$C(a)$</td>
</tr>
<tr>
<td>Role membership</td>
<td>$R(b, c)$</td>
</tr>
<tr>
<td>Equality (Inequality)</td>
<td>$a = b \ (a \neq b)$</td>
</tr>
</tbody>
</table>

Figure 2.1: Terminological and assertional axioms

These axioms can be divided into two categories: **terminological axioms** and **assertional axioms**.

Assertional axioms tell us about the individuals. This can be expressed using **concept assertion**, **role assertion** and **equality** or **inequality**.

Concept assertions and role assertions together are known as **membership assertions**. As they are simply atoms and come from DLs, they are also called **DL-atoms**. Atoms which are not from DLs, hence called **non-DL-atoms**. These two types of atoms have been used frequently in the rest of this thesis.
The rest of the axioms in Figure 2.1 are known as terminological axioms. They refer to the relationships and properties of different concepts and roles.

The most general form of terminological axioms have the form

\[ C \subseteq D \ (R \subseteq S) \]

\[ C \equiv D \ (R \equiv S) \]

Axioms with the \( \subseteq \) constructor are called (general) inclusion axioms (GCI) while axioms with the \( \equiv \) constructor are called equivalence or also definition. Concept definition can be expressed by two Inclusion Axioms.

A simple example in this regard is shown below:

**Example 2.4** Let an inclusion axiom

\[ Robot \equiv Machine \sqcap \forall \text{performs.Task} \]

Here Robot, Machine, Task \( \in \mathbf{A} \) are concepts, performs \( \in \mathbf{R} \) is a role, \( \equiv \) and \( \sqcap, \forall \) are operators. Let us suppose that \( Tron \in \mathbf{I} \) is a Robot and it performs the task of Lifting. Then Robot(Tron), Machine(Tron) and Task(Lifting) are concept assertions while performs(Tron, Lifting) is a role assertion. Together they are called membership assertions.

**Definition 2.17** (Knowledge Base) A DL Knowledge Base \( \mathcal{O} \) comprises two components, the TBox \( \mathcal{T} \) and the ABox \( \mathcal{A} \) and written as \( \mathcal{O}(\mathcal{T}, \mathcal{A}) \). The TBox introduces the terminology, i.e., the vocabulary of an application domain, while the ABox contains assertions about named individuals in terms of this vocabulary.

### 2.2.2 Semantics

We know that most of the DLs is a fragment of function-free FOL. The language of DL can be seen as a restricted FOL language over unary and binary predicates and with a controlled form of quantification. As a result, the semantics of DL can also be expressed by translating into FOL. This translation preserves satisfiability, model and entailment of DLs.

A DL KB \( \mathcal{O} \) is equivalent to an FOL theory in which each axiom in the DL KB is expressed in a First-Order sentence. For instance, the TBox inclusion axiom

\[ A_1 \sqcap \exists R_1. A_2 \subseteq (\forall R_2. A_3) \sqcup \neg A_4 \]

is equivalent to the First-Order sentence

\[ \forall x. A_1(x) \land (\exists y. R_1(x, y) \land A_2(y)) \rightarrow (\forall z. R_2(x, z) \rightarrow A_3(z)) \lor \neg A_4(x) \]

The translation is given by two mappings \( \pi_x \) and \( \pi_y \) from SHOIQ-concepts into \( \mathcal{L}^2 \) formulas. Here \( \mathcal{L}^k \) denotes the first order predicate logic
over unary and binary predicates with \( k \) variables. Each concept name \( A \) is viewed as a unary predicate symbol and each role name \( R \) is viewed as a binary predicate symbol.

For DL \( SHOIQ \) KB \( \mathcal{O} \) the translation \( \pi(\mathcal{O}) \) is inductively defined in Figure 2.2 as follows:

\[
\begin{align*}
\pi_x(A) &= A(x) & \pi_y(A) &= A(y) \\
\pi_x(C \sqcap D) &= \pi_x(C) \land \pi_x(D) & \pi_y(C \sqcap D) &= \pi_y(C) \land \pi_y(D) \\
\pi_x(C \sqcup D) &= \pi_x(C) \lor \pi_x(D) & \pi_y(C \sqcup D) &= \pi_y(C) \lor \pi_y(D) \\
\pi_x(\exists R.C) &= \exists y : R(x, y) \land \pi_y(C) & \pi_y(\exists R.C) &= \exists x : R(y, x) \land \pi_x(C) \\
\pi_x(\forall R.C) &= \forall y : R(x, y) \rightarrow \pi_y(C) & \pi_y(\forall R.C) &= \forall x : R(y, x) \rightarrow \pi_x(C) \\
\pi_x(\geq nR) &= \exists y \in \mathbb{N} : R(x, y) = \exists y_1, \ldots, y_n : \bigwedge_{i \neq j} y_i \neq y_j \land \bigwedge_{i} R(x, y_i) \\
\pi_x(\leq nR) &= \forall y \in \mathbb{N} : R(x, y) = \forall y_1, \ldots, y_{n+1} : \bigwedge_{i \neq j} y_i \neq y_j \rightarrow \bigvee_{i} \neg R(x, y_i) \\
\pi_x(\leq nR.C) &= \forall y_1, \ldots, y_{n+1} : \bigwedge_{i} R(x, y_i) \land \bigwedge \pi_x(C) \rightarrow \bigvee y_i = y_j \\
\pi_x(\geq nR.C) &= \exists y_1, \ldots, y_n : \bigwedge_{i} R(x, y_i) \land \bigwedge \pi_x(C) \land \bigwedge y_i \neq y_j \\
\end{align*}
\]

Mapping axioms into FOL
\[
\begin{align*}
\pi(C \subseteq D) &= \forall x : \pi_y(C) \rightarrow \pi_y(D) \\
\pi(R \subseteq S) &= \forall x, y : R(x, y) \rightarrow S(x, y) \\
\pi(C(a)) &= C(a) \\
\pi(R(a, b)) &= R(a, b) \\
\end{align*}
\]

Mapping \( \mathcal{O} \) into FOL
\[
\begin{align*}
\pi(R) &= \forall x, y : R(x, y) \leftrightarrow R^-(y, x) \\
\pi(\mathcal{A}) &= \bigwedge_{a \in A} \pi(a) \\
\pi(\mathcal{T}) &= \bigwedge_{a \in T} \pi(a) \\
\pi(\mathcal{O}) &= \pi(\mathcal{A}) \land \pi(\mathcal{T}) \\
\end{align*}
\]

Figure 2.2: Translation of \( SHOIQ \) into FOL

In the following chapter we use these terminologies defined above to describe the chosen syntax and semantics of our combination.
In this chapter we define the statement of problems of our combination. The syntax of the combination is defined along with the chosen semantics. We mention what kind of reasoning task we are interested in for this combination. Before stating the problems, we discuss in brief why reasoning is generally undecidable in this type of combination. Finally, after the problem statement we include an existing combination where DLP as a Description Logic component is extended using rules and how reasoning is performed in this combination. At the end of this approach, we also mention the expressive limitation as well.

### 3.1 Combination of DLs and rules

Here we define the syntax and semantics of combinations of DLs and rules in terms of standard First-Order logic.

#### 3.1.1 Syntax

To define the combined KB, we need to fix the alphabets for predicate symbols.

**Definition 3.1 (Alphabets of predicates)** Let \( \mathcal{A} = \mathcal{A}_\mathcal{O} \cup \mathcal{A}_\mathcal{P} \) be a set of alphabets such that

- \( \mathcal{A}_\mathcal{O} \) and \( \mathcal{A}_\mathcal{P} \) are disjoint \( \mathcal{A}_\mathcal{O} \cap \mathcal{A}_\mathcal{P} = \emptyset \)
- \( \mathcal{A}_\mathcal{O} \) consists of only unary and binary predicate symbols from DLs known as DL-predicates and
- \( \mathcal{A}_\mathcal{P} \) contains \( n \)-ary predicate symbols that are not DL-predicates

**Definition 3.2 (Combined Knowledge Base)** A Combined Knowledge Base \( \mathcal{C} \) is a pair written as \((\mathcal{O}, \mathcal{P})\) where
CHAPTER 3. STATEMENT OF PROBLEMS FOR COMBINING DESCRIPTION LOGICS AND RULES

- $\mathcal{O}$ is a DL KB
- $\mathcal{P}$ is a program with rules over the predicate alphabets $\mathcal{A}$ and the constants $\mathcal{C}$, i.e., a set of rules where each rule $r$ in $\mathcal{P}$ is of the form

\[
p(X) \leftarrow q_1(Y_1), \ldots, q_l(Y_l), s_1(Z_1), \ldots, s_m(Z_m) \tag{3.1}
\]

such that $l \geq 0, m \geq 0$, $p(X)$, and $p \in \mathcal{A}$, each $q_i(Y_i)$, $s_j(Z_j)$ for $0 \leq i \leq l$, $0 \leq j \leq m$ is an atom where each $q_i \in \mathcal{A}_\mathcal{P}$ and each $s_j \in \mathcal{A}_\mathcal{O}$.

**Example 3.1** An example of combined KB $\mathcal{C} = \langle \mathcal{O}, \mathcal{P} \rangle$ is as follows.

The DL KB contains some facts and axioms about the university education domain as follows

```
MScStudent(John)
Professor(Jones)
teaches(Jones, AdvDatabase)
AdvancedCourse(AdvDatabase)
attends(John, AdvDatabase)
```

```
Professor $\sqcap$ Lecturer $\sqsubseteq$ FacultyMember
MScCourse $\sqcap$ BScCourse $\sqsubseteq$ Course
AdvancedCourse $\sqsubseteq$ MScCourse
FoundationCourse $\sqsubseteq$ BScCourse
Professor $\sqsubseteq$ $\forall$teaches.AdvancedCourse
Lecturer $\sqsubseteq$ $\forall$teaches.FoundationCourse
MScStudent $\sqsubseteq$ Student $\sqcap$ $\exists$attends.AdvancedCourse
MScStudent $\sqsubseteq$ $\forall$hasSupervisor.Professor
```

With DL axioms it is possible mention the existence of something that is not explicitly mentioned in the DL KB. But DL KB generally presents information in tree structure.

Now let us consider a set of rules

```
engagedIn(John, WebProject).
PaidProject(WebProject).
EngagedInStJob(x) $\leftarrow$ Student(x), engagedIn(x, y),
          PaidProject(y).
DoMScThesis(x) $\leftarrow$ Student(x), AdvancedCourse(y),
               hasSupervisor(x, z), Professor(z),
               attends(x, y), teaches(z, y).
hasWorkLoad(x, High) $\leftarrow$ DoMScThesis(x), EngagedInStJob(x).
```
3.2. REASONING IN COMBINED KNOWLEDGE BASE

Now we want to check if John has a *High* workload. This checking cannot be done in either DLs or rules. But in the combined KB it is easy to show that \( C \models \text{hasWorkLoad}(x, \text{High}) \). DL axioms cannot model information like DoMScThesis(x) can because of having a close tree shape modeling structure. On the other hand, reasoning in rules needs the objects explicitly mentioned in the KB, which in DL is easily possible. Axioms of the MSc-Student \( \sqsubseteq \) Student \( \exists \text{attends}. \)AdvancedCourse form can model information of the objects even if they are not explicitly mentioned in DL KB.

Thus, it is possible to express information in the combined Knowledge Base which cannot be expressed by either of the component DL KB or rules KB separately.

### 3.1.2 Semantics

Our proposed combination semantics relies on the standard First-Order interpretation of both the DLs and the rule component of the combined KB \( C \) over predicates in \( A \) and constants in \( C \).

The First-Order representation of a rule in Equation 3.1 is as follows

\[
\forall \bar{x}, \bar{y}_1, \ldots, \bar{y}_l, \bar{z}_1, \ldots, \bar{z}_m.q_1(\bar{y}_1) \land \ldots \land q_l(\bar{y}_l) \land s_1(\bar{z}_1) \land \ldots \land s_m(\bar{z}_m) \rightarrow p(\bar{x})
\]

**Definition 3.3** (Model and Satisfiability) A *model* of a combined KB \( C = \langle O, P \rangle \) is an interpretation \( \mathcal{N} \) such that \( \mathcal{N} \) satisfies \( \pi(O) \cup P \). \( C \) is called *satisfiable* if it has at least a model. A sentence \( \alpha \) is called satisfiable in \( C \) iff \( \alpha \models C \) i.e. \( \alpha \models \pi(O) \cup P \).

**Definition 3.4** (Entailment) A sentence \( \alpha \) is *entailed* by \( C \) denoted as \( C \models \alpha \) iff for each model \( \mathcal{N} \) of \( C \), \( \mathcal{N} \) satisfies \( \alpha \).

### 3.2 Reasoning in Combined Knowledge Base

Query answering is a reasoning problem in the integration of DLs and rules. As conjunctive queries are known to be useful in various practical applications, hence it is practical to use them as an expressive formalism for querying our combined KB.

**Definition 3.5** (Conjunctive Query) A *Conjunctive Query* (CQ) \( q \) is an expression of the form \( q(\bar{x}) \leftarrow \exists \bar{y} \varphi(\bar{x}, \bar{y}) \) where \( \varphi \) is a formula built from the alphabets of atoms in \( A \) using the conjunction (\( \land \)) connective with *free* variables \( \bar{x} \) and \( \bar{y} \). We call the variables in \( \bar{x} \) the *distinguished* variables and those in \( \bar{y} \) the *non-distinguished* variables. A Conjunctive Query \( q \) is *boolean* if the arity of \( q \) is 0.
CHAPTER 3. STATEMENT OF PROBLEMS FOR COMBINING DESCRIPTION LOGICS AND RULES

The answer to a Conjunctive Query \( q \) over a KB \( \mathcal{C} = (\mathcal{O}, \mathcal{P}) \) is a tuple of constants \( \bar{c} \) as a result of the substitution of the distinguished variables \( \bar{x} \) with the set of constants in \( \mathcal{C} \).

The answer \( \text{ans}(q, \mathcal{C}) \) to a CQ \( q \) over \( \mathcal{C} \) is represented as follows:

\[
\text{ans}(q, \mathcal{C}) = \{ \bar{c} \mid \exists \bar{y} \varphi(\bar{c}, \bar{y}) \}
\]

3.3 Undecidability of reasoning

As we know, decidability is a crucial issue for reasoning in the integration of DLs and rules. Starting from a DL KB in which reasoning is decidable and rule KB for which reasoning is decidable, reasoning in the integrated KB may not be a decidable problem. We try to illustrate some of the reasons here.

Most of the Description Logics enjoys tree model property. This property states that every satisfiable concept has a model that has the structure of a (possibly infinite) tree with branching degree linearly bounded by the size of the concept.

For example, a DL KB consists of an inclusion axiom

\[
\text{Machinery} \sqsubseteq \exists \text{partOf}.\text{Machinery} \\
\text{Machinery(Transistor)}
\]

This KB represents the infinite chain of Machineries: since a Transistor is a machinery which is a part of another machinery \( m_1 \), which is also part of another machinery, and it goes so on. If we want to check if Book is a machinery in our KB, we have to check if \( KB \cup \{ \neg \text{Machinery(Book)} \} \) is unsatisfiable. Checking this can be done by building a model which is of the tree shape.

While building this tree model, the expansion of the axiom creates infinite tree starting from Transistor. This simple example shows us the non-termination of model building. This kind of problem is usually handled by concentrating on building nice models which are derived by a strategy called blocking. This blocking is a cycle-detection mechanism which states that if the labels of a node \( x \) and one of its ancestors coincide, then the application of rules to \( x \) is blocked. So reasoning is possible even if the objects mentioned in the axioms are not explicitly introduced in the KB. This is clear from our example, as we don’t know the object of whom the Transistor is part of.

Rules on the other hand, is not concerned with this kind of restriction of Tree-model. Unlike DLs they can represent cycles. An example would clarify this.

**Example 3.2** A lazy student eats at the place close to where he lives at.
3.4. EXISTING COMBINATION: DLP EXTENDED WITH RULES

This can be represented using rules as follows

\[ \text{lazyStudent}(x) \leftarrow \text{student}(x), \text{livesAt}(x, y), \text{eatsAt}(x, z), \text{closeTo}(z, y) \]

If we carefully look at the above rule we find that it forms a cycle. This is not possible to represent using DLs because of the tree model property.

As a result when Description Logics is extended with rules tree model property is not preserved anymore. Thus reasoning in this combination leads to undecidability and the loss of tree model property gives a clue in this regard. In fact, in SWRL undecidability was shown by a reduction to the reasoning in combination from a known undecidable domino problem.

The next section talks about one such combination of Description Logic and rules.

3.4 Existing combination: DLP extended with rules

Basically in this DLP with rule extension, the DL part is restricted such that DL axioms can only be written in Horn logic fragment. No restrictions are imposed on the rules and LP reasoners correctly and completely implements this semantics of combinations. The restrictions on the language becomes such that they become unstable even with very simple DL is beyond this DLP.

This section discusses the reasoning in the combination where DLP is extended with rules. Description Logic Program (DLP) [28] is a fragment of $SHIQ$. Hence its syntax and semantics come from OWL itself and DLP KB $\mathcal{O}$ is an OWL KB. Moreover, each DLP KB $\mathcal{O}$, which is disjunction and existential-free in their axiom consequents, can be considered as a Horn logic fragment under FOL and can therefore correspond syntactically to logic program. This DLP can easily be extended with rules forming logic program $\mathcal{P}$ for which any rule reasoner can be used. The DLP extension with rules thus form a combination of DLs and rules and presented as $\mathcal{C} = \langle \mathcal{O}, \mathcal{P} \rangle$. This combination allows DL predicates both in bodies and heads of rules thus allowing information flow in both directions. As DLP is existential-free, reasoning in their rule extension only considers the objects explicitly mentioned in KB. This means no specialized restrictions are required for reasoning in rules. Reasoning is this rule extension is decidable and the data complexity of DLP is of polynomial time. Because of the inability to allow DL constructors in the axioms, DLP has limited expressivity compared to other DLs. But due to the polynomial complexity and simplicity, DLP is also an inspiration to the OWL 2 RL [33] and standard reasoning tools are available facilitating reasoning services.

3.4.1 DLP syntax

A DLP ontology $\mathcal{O}(\mathcal{A}, \mathcal{T})$ is defined as follows:
CHAPTER 3. STATEMENT OF PROBLEMS FOR COMBINING
DESCRIPTION LOGICS AND RULES

- ABox:
  C(a) (individual assertion)
  R(a, b) (property assertion)

- Characteristics of Property:
  $C \equiv D$ (Class equivalence)
  $R \equiv S$ (Property equivalence)
  $R \subseteq S$ (Subproperty)
  $T \sqsubseteq \forall R.C$ ($C \neq \bot$) (domain)
  $T \sqsubseteq \forall R^-.C$ ($C \neq \bot$) (range)
  $R \equiv S^-$ (inverse)
  Trans($R$) (transitive)

- The expression in TBox is of the form

$$C_B \subseteq D_H$$

where $C_B$ is the body of the DLP inclusion axioms where they allow only DL constructors conjunction, disjunction and existential restriction while $D_H$ denoting axiom head allows only conjunction and universal restriction.

Example 3.3 The example of a DLP is given below. We see that in the axiom consequents there is no existential or disjunctive information.

CLMastersStudent(John) John is a computational logic masters student
taughtBy(AdvancedLogic, Jones) advanced logic course is taught by Prof. Jones
(advancedCourse $\sqcap$ CLCourse)(AdvancedLogic) Advanced logic is an advanced and computational logic course
AdvancedCourse $\subseteq$ MastersCourse Advanced course is masters course
MastersCourse $\sqcap$ CLCourse $\sqsubseteq \forall$taughtBy.inEnglish All the masters and computational logic courses are advanced courses and taught only in English by the professors

3.4.2 Reasoning in this combination

The DL KB is transformed into rules and then reasoning is done with rule extension. The extension of these DLP axioms with rules is derived simply by translation $\pi$. The translation of these axioms follows the standard mapping of DL into FOL described in Figure 2.2.

DLP restricts the presence of class constructors namely disjunction, universal and existential restriction.
3.4. EXISTING COMBINATION: DLP EXTENDED WITH RULES

When a disjunction occurs in the axiom antecedents it poses no problems, but when it is on the consequents of an inclusion axiom it becomes a disjunction in the head of the corresponding rule. This cannot be handled within the rules as they provide more that one minimal model.

Again, when a universal restriction appears on the antecedents of an inclusion axiom it becomes an implication in the body of the corresponding rule which can’t be mapped into our rule as it would require negation in the rule body.

In DLP the existential DL constructor of \( \exists R.C \) can only occur on the left hand side of an inclusion axiom, it allows the form \( \exists R.C \subseteq D \) but disallows \( D \subseteq \exists R.C \) as they deal with unnamed objects not explicitly mentioned in the KB.

The rule reasoning tasks include the following queries:

- ground atomic query: To determine whether a ground atom \( A \) is entailed by the logic program
- non-ground atomic query : To determine whether a given atom \( A \) with variables is entailed after the substitution with constants

Reasoning in the Description Logic fragment DLP is reduced via the translation and to reasoning in rules. For example, \( a \) is an instance of a class \( C \) if and only if \( P \) entails \( \pi(C(a)) \). Another example is class subsumption where \( C \) is a subclass of \( D \) if and only if \( P \cup \pi(C(a)) \) entails \( \pi(D(a)) \).

The whole reasoning process is thus modified and summarized as follows:

The DLP ontology \( O \) is transformed using translation function \( \pi \) and then rewritten into rules resulting \( \mathcal{P}_O \)

\[
O \rightarrow \pi(O) \rightarrow \mathcal{P}_O
\]

An example of this process follows next.

**Example 3.4** Let us consider the following DLP axiom

\[
A \sqcap \exists R.C \sqsubseteq B \sqcap \forall P.D
\]

After the translation the axiom is transformed as follows:

\[
B(x) \land (D(z) \leftarrow P(x, z)) \leftarrow A(x) \land R(x, y) \land C(x)
\]

which is then rewritten into rules as follows:

\[
B(x) \leftarrow A(x) \land R(x, y) \land C(x)\\
D(x) \leftarrow A(x) \land R(x, y) \land C(x) \land P(x, z)
\]
CHAPTER 3. STATEMENT OF PROBLEMS FOR COMBINING DESCRIPTION LOGICS AND RULES

The logic program is formed as $P_O \cup P$.

Finally, to check if a ground query $q(\bar{c})$ holds in combined Knowledge Base $C$ it will enough to check if the logic program entails $q(\bar{c})$. This statement corresponds to the proposition below:

**Proposition 3.1** Given a ground query $q(\bar{c})$, combined Knowledge Base $C \models q(\bar{c})$ if and only if the logic program $P_O \cup P \models q(\bar{c})$.

**Example 3.5** The reasoning example follows from above. After the translation $\pi(O)$ and rewriting we get $P_O$

\begin{verbatim}
CLMastersStudent(John).
taughtBy(AdvancedLogic, Jones).
MastersCourse(AdvancedLogic).
CLCourse(AdvancedLogic).
MastersCourse(x) ← AdvancedCourse(x).
inEnglish(x) ← MastersCourse(x), CLCourse(x), taughtBy(x, y).
\end{verbatim}

We would to find out if John can attend the AdvancedLogic course. This can be easily verified by checking if

$P_O \cup P \models canAttendCourse(John, AdvancedLogic)$

The logic program $P_O \cup P$ becomes

\begin{verbatim}
CLMastersStudent(John).
taughtBy(AdvancedLogic, Jones).
MastersCourse(AdvancedLogic).
CLCourse(AdvancedLogic).
MastersCourse(x) ← AdvancedCourse(x).
inEnglish(x) ← MastersCourse(x), CLCourse(x), taughtBy(x, y).
canAttendCourse(x, y) ← CLMastersStudent(x), inEnglish(y).
\end{verbatim}

By resolving this program it can be concluded that John can indeed attend AdvancedLogic course.

3.4.3 Expressive limitation

We have seen that reasoning in DLP extended using rules is decidable. As the translation of DLP ontologies directly map into the LP rules, any LP reasoners can be used for reasoning. Moreover no restrictions are required on rules for decidable reasoning.

Even if there is no restriction on rules, restriction on DL constructors makes this combination less expressive. This is because DLP limits the
use of existential restriction, disjunction and universal restriction in their axioms. As a result, simple axioms that are expressible in other DLs, is inexpressible in DLP.

This can be demonstrated comparing other simple DL as DL-Lite [34]. DL-Lite is a basic ontology language that captures conceptual data modeling paradigms such as entity relationship, Object-Oriented formalisms such as basic UML \(^1\) class diagrams.

The following example shows a simple DL-Lite axiom

**Example 3.6** Every person has a child

\[
\text{Person} \sqsubseteq \exists \text{hasChild}.	op
\]

The expressive limitation of DLP is evident from this simple DL-Lite example above. Even this simple axiom cannot be express by DLP because DLP is not able to capture existential restriction in the axiom consequents.

### 3.5 Problem Statement

To derive a more expressive language than only DLs or only rules, the combination of these two has been studied extensively. From these studies it is clear that without putting restrictions on the components, reasoning is not decidable. For example, without restrictions reasoning (e.g., SWRL) becomes undecidable while with simple kind of restrictions (e.g., DL-safeness) reasoning is still hard. In addition, just taking the intersection of DLs and rules (e.g., DLP) components become virtually useless because of the lack of expressivity.

For this reason, we would like to introduce a simple and easily implementable reasoning procedure for the combination where reasoning will be *sound*. This procedure should have the advantage of using standard out of the box reasoning tools rather than inventing new ones.

Let this procedure say “YES” whenever it answers our query, otherwise it says “Unknown”.

By *sound* we mean that whenever the reasoning procedure returns “YES” (written as \( \vdash \) indicating that a procedure derives) to our query then the entailment holds according to the combination semantics. This is shown as follows:

\[
\mathcal{C} \vdash q(\bar{c}) \Rightarrow \mathcal{C} \models q(\bar{c})
\]

More importantly, we try to investigate one subset for which reasoning would be complete with respect to the combination semantics i.e. whenever

---

\(^1\)http://www.omg.org/uml/
the semantics says that an entailment should hold, the reasoning procedure answers “Yes”

\[ C \models q(\bar{c}) \Rightarrow C \vdash q(\bar{c}) \]

As mentioned at the beginning of this thesis, our goal is not only to look for such a combination of Description Logics and rules for which the algorithm is sound and complete but also is capable of expressing strictly more information in the Description Logic component than in the combination of DLP extended using rules.

The various aspects of this procedure will be our main agenda throughout the next chapters.
In this chapter, we introduce the reasoning algorithm for answering conjunctive queries in the combined Knowledge Base. This algorithm is effective in the sense that any kind of ontologies and rules can be used and still we get an answer that is supported by the integration semantics. The algorithm is easily implementable too because we don’t need new reasoners, rather use standard out of the box reasoners. We also show that our algorithm is not complete for all combinations with an example. We conclude this chapter by analyzing the computational property of the algorithm.

4.1 Description of Algorithm

We get all kinds of facts $\mathcal{F}$ from the Description Logic Knowledge Base $\mathcal{O}$ using a standard DL reasoners. Pellet, Racer, KAON2 are some well known DL reasoners. We then form a new logic program $\mathcal{P}' = \mathcal{P} \cup \mathcal{F}$ where $\mathcal{P}$ contains a set of rules. To answer the query $q$ over the logic program $\mathcal{P}'$ we use a rule reasoner (e.g. XSB). The algorithm returns “YES” if the query $q$ is entailed by the logic program $\mathcal{P}'$, otherwise it says “Unknown”.

CHAPTER 4. AN EFFECTIVE REASONING PROCEDURE FOR QUERY ANSWERING

The pseudocode of the algorithm is as follows:

**Algorithm 1 CombinedKBEntails \((q, \mathcal{C}, \bar{c})\)**

**Input:** Combined KB \(\mathcal{C} = (\mathcal{O}, \mathcal{P})\)
- Conjunctive Query \(q\) as \(q(\bar{x}) \leftarrow \exists y \varphi(\bar{x}, \bar{y})\)
- Answer tuple \(\bar{c}\) as a tuple of constants

**Output:** CombinedKBEntails \((q, \mathcal{C}, \bar{c})\) says YES if modified logic program entails \(q(\bar{c})\), otherwise says Unknown

**Begin**
1: Derive the set \(\mathcal{F}\) of all facts from Ontology \(\mathcal{O}\) such that
2: \[\mathcal{F} = \bigcup_{C \in \mathcal{A}_O} \{C(a) \mid \pi(\mathcal{O}) \models C(a)\} \cup \bigcup_{R \in \mathcal{A}_O} \{R(b, c) \mid \pi(\mathcal{O}) \models R(b, c)\}\]
3: Form a logic program \(\mathcal{P}'\) such that
4: \(\mathcal{P}' = \mathcal{P} \cup \mathcal{F}\)
5: if \(\mathcal{P}' \models q(\bar{c})\) then
6: \quad return YES
7: else
8: \quad return Unknown
9: end if
**End**

4.2 Properties of the Algorithm

Our algorithm is modular as it can be divided into two modules: DL reasoner is used in first step and then rule reasoner is used to answer query in the logic program.

It follows immediately from the modular property that standard tools can be used for reasoning, no new techniques or new tools are required. For DL reasoners Pellet, Racer or KAON2 can be used while for rule reasoner XSB can be used.

Our algorithm is sound. This intuition of soundness comes from the observation informally as follows. In this combination we have the ontology and logic program. Following the algorithm, we replace the ontology with all the facts. A set of these facts are entailed by the ontology which means they are true in the ontology model(s). From the property of monotonicity it is immediate that if more rules or axioms are added to these facts, there
4.2. PROPERTIES OF THE ALGORITHM

will be more entailments. So whenever we take an entailment of the part of
the KB, this is also an entailment of the whole KB.

The soundness can be formally expressed as a theorem as follows:

**Theorem 1:** Given a Knowledge Base $\mathcal{C} = \langle \mathcal{O}, \mathcal{P} \rangle$ with ontology $\mathcal{O}$ and
logic program $\mathcal{P}$, a conjunctive query $q$ and a tuple of constants $\tilde{c}$, if
CombinedKBEntails($q$, $\mathcal{C}$, $\tilde{c}$) returns “YES” then $\mathcal{C} \models q(\tilde{c})$.

**Proof:** To prove that the algorithm is sound we have to prove that if
$\mathcal{P}' \models q(\tilde{c})$ then $\mathcal{C} \models q(\tilde{c})$. It will be enough to show that for function $\mathcal{M}$
that returns a set of models, $\mathcal{M}(\mathcal{C}) \subseteq \mathcal{M}(\mathcal{P}')$.
The model $\mathcal{M}$ for the KB $\mathcal{C} = \langle \mathcal{O}, \mathcal{P} \rangle$ is the model of its components

$$\mathcal{M}(\mathcal{C}) = \mathcal{M}(\mathcal{O}) \cap \mathcal{M}(\mathcal{P}) \quad \text{(4.1)}$$

We have the set of facts $\mathcal{F}$ that are retrieved from the ontology. Naturally
this $\mathcal{F}$ is entailed by the ontology $\mathcal{O}$. So we can write

$$\mathcal{M}(\mathcal{O}) \subseteq \mathcal{M}(\mathcal{F}) \quad \text{(4.2)}$$

Again the model $\mathcal{M}$ for the modified logic program $\mathcal{P}'$ is the intersection of
the models of logic program $\mathcal{P}$ and $\mathcal{F}$. Using 4.1 and 4.2 we get

$$\mathcal{M}(\mathcal{P}') = \mathcal{M}(\mathcal{F}) \cap \mathcal{M}(\mathcal{P})$$

As the algorithm answers “YES”, we can write

$$\mathcal{M}(\mathcal{P}') \subseteq q(\tilde{c}) \quad \text{(4.3)}$$

Clearly the model in $\mathcal{C}$ lies also in the model in $\mathcal{P}'$

$$\mathcal{M}(\mathcal{C}) \subseteq \mathcal{M}(\mathcal{P}') \quad \text{(4.4)}$$

As the two inclusions in 4.3 and 4.4 hold, the following inclusion also holds

$$\mathcal{M}(\mathcal{C}) \subseteq \mathcal{M}(q(\tilde{c}))$$

and from the definition of entailment

$$\mathcal{C} \models q(\tilde{c})$$

In conclusion, the soundness of this algorithm states that we can use
any kind of Ontologies and rules in our combination and any decision we
get from the procedure is supported by the semantics of the combinations.

The algorithm is not complete.

To show that the algorithm is not complete, we look at the example
below:
CHAPTER 4. AN EFFECTIVE REASONING PROCEDURE FOR QUERY ANSWERING

Example 4.1 Let an ontology in Description Logic KB $O$ contains

$$B \subseteq C$$

and a program $P$ contains a fact

$$B(a)$$

Here $B$ subsumes $C$ in $O$ and $a$ is a member of $B$ in $P$. If we combine this $O$ and $P$, because of the axiom in $O$, $a$ should as well be a member of $C$. So, according to the semantics it is the case that $C \models C(a)$.

Now for the KB $C = \langle O, P \rangle$ using the procedure we retrieve the facts as follows:

$$F = fact(O) = \{ \alpha \text{ is a fact } | \ O \models \alpha \} = \emptyset$$

The new logic program $P'$ is formed as follows:

$$P' = F \cup P = \{ B(a) \}$$

Clearly

$$P' \not\models C(a)$$

hence the decision procedure CombinedKBEntails($q, C, (a)$) returns “Unknown” even though $C \models C(a)$.

Hence we can conclude that even though the combination semantics says $C \models q$, the algorithm says “Unknown”. Therefore the algorithm is not complete. From this it is clear that to achieve completeness of our algorithm, we cannot use any kind of ontologies and rules in our combination. There should be some kind of restrictions on the participating components. In the next chapter, we try to find out what kind of restrictions are necessary for proving that our algorithm will be complete.

4.3 Computational Complexity of the algorithm

Our algorithm works in two steps:

1. Retrieving the entailed facts from the Description Logic Knowledge Base

2. Query answering in the positive logic program

The first step of our algorithm states that we retrieve all the entailed facts from the ontology. Although the size of the set of facts is polynomial in the size of the whole ontology, retrieving does not depend only on this data.
4.3. COMPUTATIONAL COMPLEXITY OF THE ALGORITHM

Because we also retrieve the inferred facts, it also depends on the TBox as TBox is no more static during this process. Hence it makes sense to talk about the whole ontology. Thus, the set of inferred facts \( \mathcal{F} \) is polynomial in the ontology (both ABox and TBox). In Description Logic Horn-SHIQ these entailments of instances is exponential because of non-static TBox even if we have polynomial number of facts, deriving polynomial number of facts is still exponential.

Finally, rule reasoning is performed. In general, if the rules contain function symbols then reasoning is undecidable. In the absence of functions in rules, the complexity of rule reasoning is exponential.

Therefore, the complexity of our algorithm is exponential in the size of the combined Knowledge Base.
5

Imposing Restrictions: Regaining Completeness

In the previous chapter, we have already shown that our algorithm is not complete for the combination of every Description Logics and rules. In this chapter we are going to identify the reasons behind this incompleteness. Based on this findings, we impose restrictions on the Description Logic and rule components and choose a subset of this sort of combinations. We prove that with the introduction of these restrictions, the completeness of the algorithm can be regained. Finally, we discuss various aspects of these restrictions and find out what effect they have on the expressive power of our approach compared to the combination of DLP extended using rules.

5.1 Restrictions on Knowledge Base Components

We have already seen with a simple example in the previous chapter that for every ontologies and rules in the combinations, the algorithm is sound but not complete. For this reason, we need to put restrictions on the components of our Knowledge Base. In the following sections, we provide the reasons behind incompleteness, analyze them and motivate what kind of restrictions we need to regain completeness.

5.2 Incompleteness: Motivations, Analysis and Solutions

The following subsections focusses on the motivations to find ways to make the algorithm complete. We analyze every case for which the algorithm results incompleteness. Based on the analysis we propose what kind of measures we need to take to regain complete algorithm. These measure are of the form of restrictions on Description Logics and rules.
CHAPTER 5. IMPOSING RESTRICTIONS: REGAINING COMPLETENESS

5.2.1 Disjunction

Incompleteness due to Disjunct Construct

The type of restriction depends on the characteristics of the combination semantics. It has been already shown in the reasoning of DLP extension with rules that the presence of disjunction in the axioms in $\mathcal{O}$ gives unexpected results. Unfortunately, in our case disjunction in $\mathcal{O}$ shows a proof of incompleteness of our algorithm. An example regarding this is as follows:

Example 5.1 Let an ontology in Description Logic KB $\mathcal{O}$ contains

$$A \subseteq B \sqcup C$$

$$A(a)$$

and a logic program $\mathcal{P}$ contains a set of rules

$$r(x) \leftarrow B(x)$$

$$r(x) \leftarrow C(x)$$

Here $A$ subsumes $B$ or $C$ and $a$ is a member of $A$ in $\mathcal{O}$, on the other hand in $\mathcal{P}$, all the elements under $B$ and $C$ are also elements under $r$.

For the combination of $\mathcal{O}$ and $\mathcal{P}$ and for axiom in $\mathcal{O}$, $a$ should as well be a member of either $B$ or $C$ and hence $a$ must be a member of $r$ as well. So, according to the semantics it is the case that $C \models r(a)$.

Now for the KB $\mathcal{C} = \langle \mathcal{O}, \mathcal{P} \rangle$

$$\mathcal{F} = \{A(a)\}$$

The new logic program $\mathcal{P}'$:

$$\mathcal{P}' = \mathcal{F} \cup \mathcal{P}$$

$$\mathcal{P}' = \begin{cases} 
A(a) \\
  r(x) \leftarrow B(x) \\
r(x) \leftarrow C(x) 
\end{cases}$$

Clearly

$$\mathcal{P}' \not\models r(a)$$

hence the algorithm $CombinedKBEntails(q, C, (a))$ returns “Unknown” even though $\mathcal{C} \models r(a)$.
5.2. INCOMPLETENESS: MOTIVATIONS, ANALYSIS AND SOLUTIONS

From this example it is obvious that if the problem of disjunction is taken care of, then one obstacle to the way of achieving completeness is removed. So, we disallow all situations that are equivalent to disjunction. Following this idea, we will consider the class of ontologies in $\mathcal{O}$ which is essentially disjunction free. One example of such an expressive ontology language which does not have disjunction is Horn-$\mathcal{SHIQ}$.

Disallowing Disjunction

The Description Logic Horn-$\mathcal{SHIQ}$ was introduced in [31] as a fragment of $\mathcal{SHIQ}$. The basic idea is not to allow disjunction $\sqcup$ in their expressions by putting syntactic restrictions on $\mathcal{SHIQ}$. This way, there is a correspondence of Horn-$\mathcal{SHIQ}$ expressions and Horn fragment of First-Order logic with equality. Following [35] without loss of generality, the normal form of Horn-$\mathcal{SHIQ}$ is rewritten in [36] and described here.

**Definition 5.1** (Horn-$\mathcal{SHIQ}$) A (normal) Horn-$\mathcal{SHIQ}$ KB contains only General Inclusion axioms (GCI) of the forms

$$
A \cap B \subseteq C \quad A \subseteq \forall R.B \quad A \subseteq nS.B
$$

$$
\exists R.A \subseteq B \quad A \subseteq \exists R.B \quad A \subseteq 1S.B
$$

where $A, B, C$ are concept names including special concepts $\top$ and $\bot$ and $R$ is a role, $S$ is a simple role and $n \geq 1$

The advantage of Horn-$\mathcal{SHIQ}$ as ontologies is that it is simple and it provides a local criterion for checking Hornness by investigating the structure of single axioms. Also data complexity in Horn-$\mathcal{SHIQ}$ is PTIME. This enables reasoning with larger ABoxes in situations where the TBox is static.

5.2.2 Modular Reasoning

Incompleteness due to modular reasoning

The modular property of our algorithm states that reasoning is done in two stages. For retrieving facts we use DL reasoner and query answering uses rule reasoner. As these two reasonings are separate, there is no interaction. Therefore, in rule reasoning if we use DL atoms in the head of the rules of the logic program, there are certain inferences we don’t get from this algorithm that agree with the combination semantics. The fact is we don’t have any feedback from the program back to the ontology as DL reasoner is no longer part of rule reasoning process any more.
CHAPTER 5. IMPOSING RESTRICTIONS: REGAINING COMPLETENESS

DL Predicates only in Rule Body

To avoid incompleteness we have to cope with this modular reasoning procedure. Hence the restriction on rules is defined such that DL (ontology) predicates can only be used in the rule bodies. This means our combination does not allow concepts and roles to appear in the heads of the rules.

As a result of this rule restriction, the flow of inferred information in only in one direction, i.e. from DLs to rules.

5.2.3 Named/Unnamed Objects

Incompleteness due to Unnamed objects

We now observe what happens in the presence of unnamed objects in our reasoning algorithm. From the concept of open-domain we know that unnamed objects are expressed using existential quantification in Description Logic KB \( \mathcal{O} \). On the other hand, logic program \( \mathcal{P} \) only deals with objects that are explicitly mentioned in their KB. So in their combination, while reasoning, the logic program cannot handle those unnamed objects introduced in \( \mathcal{O} \) thus resulting the incompleteness of our algorithm. An example of this incompleteness is shown below:

**Example 5.2** Let \( \mathcal{O} \) contains

\[
C \subseteq \exists R. \top
\]

\[C(a)\]

and \( \mathcal{P} \) contains a rule

\[r(x) \leftarrow R(x, y)\]

Here \( C \subseteq \exists R. \top \) and \( a \) is a member of \( C \) in \( \mathcal{O} \). And in \( \mathcal{P} \), \( x \) is a member of predicate \( r \) if \( x \) is also a member of a relation \( R \). If we combine this \( \mathcal{O} \) and \( \mathcal{P} \), according to the semantics it is the case that \( \mathcal{C} \models r(a) \).

Now for the KB \( \mathcal{C} = (\mathcal{O}, \mathcal{P}) \)

\[
\mathcal{F} = \{ C(a) \}
\]

The new logic program \( \mathcal{P}' \) stands:

\[
\mathcal{P}' = \mathcal{F} \cup \mathcal{P}
\]

\[
\mathcal{P}' = \left\{ \begin{array}{l}
C(a) \\
r(x) \leftarrow R(x, y)
\end{array} \right\}
\]
5.2. INCOMPLETENESS: MOTIVATIONS, ANALYSIS AND SOLUTIONS

Clearly

\[ P' \not\models r(a) \]

hence the algorithm \textit{CombinedKBEntails}(q,\mathcal{C}, (a)) returns “Unknown” even though \( \mathcal{C} \models r(a) \).

To remedy this problem of unnamed objects in the logic program, we make sure that only named objects are considered during the evaluation in logic program. This is done by restricting the rules such that they only consider named objects. A syntactic restriction on the variables called \textit{DL-safeness} is introduced for this purpose.

**Only named objects are considered**

Before introducing DL-safeness, we recall what a safe rule is.

In a safe rule, each variable occurs in a positive atom in a body and can therefore be bound only to constants explicitly mentioned in the database. DL-safeness is also inspired by this rule safeness.

**Definition 5.2**(DL-safe rule) A rule \( r \) is \textit{DL-safe} if each variable occurring in \( r \) also occurs in a non-DL atom in the body of \( r \). A logic program is DL-safe if all of its rules are DL-safe.

Hence, DL-safety ensures that each variable is bound only to individuals explicitly mentioned in the DL KB. For example, \textit{Student}, \textit{livesAt} and \textit{eatsAt} are concepts and roles from \( \mathcal{O} \), the following rule is not DL-safe, because both \( x, y \) and \( z \) occur in DL-atom, but not in an atom with a predicate outside \( \mathcal{O} \).

\[
\text{Lazystudent}(x) \leftarrow \text{Student}(x), \text{livesAt}(x,y), \text{eatsAt}(x,z)
\]

The previous rule can be made DL-safe by adding special non-DL atoms \( \tilde{\mathcal{O}}(x), \tilde{\mathcal{O}}(y) \) and \( \tilde{\mathcal{O}}(z) \) to the body of the rule and by adding a \textit{fact} \( \tilde{\mathcal{O}}(a) \) for each individual \( a \) occurring in \( \mathcal{O} \) or \( \mathcal{P} \). Thus the above rule can be made DL-safe as follows:

\[
\text{Lazystudent}(x) \leftarrow \text{Student}(x), \text{livesAt}(x,y), \text{eatsAt}(x,z), \tilde{\mathcal{O}}(x), \tilde{\mathcal{O}}(y), \tilde{\mathcal{O}}(z)
\]

**Limited Expressivity of DL-safe Conjunctive Query**

**Definition 5.3**(DL-safe Conjunctive Query) A Conjunctive Query is called DL-safe Conjunctive Query if after the (syntactic) transformation it acts as if it were a DL-safe rule.
CHAPTER 5. IMPOSING RESTRICTIONS: REGAINING
COMPLETENESS

DL-safe rules are less expressive than conjunctive queries. Let us consider
an example $O = \{(\exists \text{hasChild}. \text{Person})(Peter)\}$. Here the ontology states
that Peter has a child but does not disclose who it is which means the child
is not explicitly present in the ABox.

Now for the conjunctive query retrieving all objects having a child is
written as $q(x, y) = \exists \text{hasChild}(x, y)$. Clearly $O \models \exists \text{hasChild}(Peter, y)$
giving an answer $\text{Peter}$ to the CQ $q(x, y)$ for the KB $O$.

When we transform CQ to a rule that is DL-safe, this results in a DL-safe
rule $q(x) \leftarrow \text{hasChild}(x, y), \bar{O}(x), \bar{O}(y)$ where $\bar{O}(x), \bar{O}(y)$ are introduced for
DL-safeness making sure of the explicit existence of object $y$ in the KB.

The logic program $P \not\models Q(\text{Peter})$ because the child object is unknown
and $\bar{O}(y)$ cannot find anything to substitute with $y$. So, it is clear that
the presence of non-distinguished variables poses problems for the CQ when
DL-safety restriction is applied.

To avoid this problem, rolling-up technique introduced in [37] can be
used. The intuition is that the queries which have a tree structure with
non-distinguished variables can be transformed into queries without non-
distinguished variables. For example, we can add an axiom $\exists R.C \subseteq D$ to
the ontology $O$. As a result, the CQ stands $q(x) \leftarrow D(x), \bar{O}(x)$.

5.2.4 Problems with equality (inequality)

Incompleteness due to equality (inequality)

The Description Logic KB $O$ contains equality/inequality. Due to the presence
of this property the algorithm shows incompleteness as it does not agree with the combination semantics. The following example shows how
this happens:

Example 5.3 Let $O$ contains

\[ a = b \]

and $P$ contains a fact

\[ p(a) \]

If we combine this $O$ and $P$, according to the semantics it is the case that
$C \models p(b)$.

Now for the KB $C = \langle O, P \rangle$

\[ F = \emptyset \]

The new logic program $P'$ stands:
\[ \mathcal{P}' = \mathcal{F} \cup \mathcal{P} = \{p(a)\} \]

Clearly

\[ \mathcal{P}' \not\models p(b) \]

hence the algorithm CombinedKBEntails(q, C, (b)) returns “Unknown” even though C \models p(b).

Thus equality (also for example cardinality restriction) acts also as a reason for incompleteness of our algorithm.

**Constants are not allowed in logic program**

Thus to avoid incompleteness due to presence of equality in Description Logic KB we require the program in general to be partially constant-free. The logic program is not entirely constant-free as facts of the form \( \mathcal{O}(a) \) can be present for \( a \) a named individual. But if we would have only equality-free Description Logic KB which also does not have functionality and cardinality restriction in that case we could use constants in the program.

Thus, we come up with the restrictions which states that DL KB \( \mathcal{O} \) is such that it contains only the classes of ontologies in DLs called Horn-\( SHIQ \) and for the rules, DL (ontology) predicates can only be used in the rule bodies and all the rules be DL-safe. In addition, the logic program should be partially constant-free.

**Theorem 2** Given a Knowledge Base \( C = (\mathcal{O}, \mathcal{P}) \) with DL KB \( \mathcal{O} \) and partially constant-free logic program \( P \) containing facts of the form \( \mathcal{O}(a) \) where \( a \) is a named individual, a conjunctive query \( q \) and a tuple of constants \( \bar{c} \). If \( \mathcal{O} \) is Horn-\( SHIQ \) and \( P \) and \( P \) is DL-safe and \( P \) has no DL-predicates in the head, then it is the case that if \( C \models q(\bar{c}) \) then CombinedKBEntails(q, C, \( \bar{c} \)) returns “YES”.

**Proof:** We start by recalling the required definitions to formulate the proof.

**Definition 5.4(Homomorphism)** A homomorphism \( h : \mathcal{I} \rightarrow \mathcal{J} \) between two interpretations \( \mathcal{I} \) and \( \mathcal{J} \) is a mapping \( h \) from the domain \( \Delta^\mathcal{I} \) to the domain \( \Delta^\mathcal{J} \) such that for every constant \( c \), \( h(c^\mathcal{I}) = c^\mathcal{J} \) and for every predicate symbol \( p \) and every \( \bar{a} \in \Delta^\mathcal{I} \), if \( \bar{a} \in p^\mathcal{I} \), then \( h(\bar{a}) \in p^\mathcal{J} \).

**Definition 5.5(Canonical Model)** An interpretation \( \mathcal{I} \) is a canonical model of a theory \( \varphi \) if \( \mathcal{I} \models \varphi \) and for every interpretation \( \mathcal{I}' \in Mod(\varphi) \), there exists a homomorphism \( h : \mathcal{I} \rightarrow \mathcal{I}' \).
CHAPTER 5. IMPOSING RESTRICTIONS: REGAINING Completeness

**Definition 5.5** (Canonical Model Property) A theory \( \varphi \) has the canonical model property if it holds that whenever \( \varphi \) is satisfiable, it has a canonical model.

**Definition 5.7** (Minimal Model) Let \( \mathcal{P} \) be a positive logic program. A Herbrand interpretation \( M \) of \( \mathcal{P} \) is a model of \( \mathcal{P} \) if for every rule \( r \in gr(\mathcal{P}) \), \( B^+(r) \subseteq M \) implies \( H(r) \cap M \neq \emptyset \). A Herbrand model \( M \) of a logic program \( \mathcal{P} \) is minimal iff for every model \( M' \) such that \( M' \subseteq M \), \( M' = M \).

Every positive normal logic program has a single minimal Herbrand model, which is the intersection of all Herbrand models.

**(Proposition 5.1)** The Description Logic Horn-\( \mathcal{SHIQ} \) has a canonical model property.

It follows from the above proposition that Horn-\( \mathcal{SHIQ} \) ontology \( \mathcal{O} \) has a canonical model \( \mathcal{I} \).

An interpretation \( \mathcal{I}' = (\Delta', \mathcal{I}') \) extends an interpretation \( \mathcal{I} = (\Delta, \mathcal{I}) \) of signature \( \mathcal{C} = (\mathcal{O}, \mathcal{P}) \) if \( \Delta' \subseteq \Delta \), for every constant symbol \( c \in C \), \( c^\mathcal{I} \subseteq c^\mathcal{I}' \), and for every predicate symbol \( p \in A \), \( p^\mathcal{I} \subseteq p^{\mathcal{I}'} \).

We take canonical model and extend it using the rules using a fixpoint computation. As ontology predicates don’t occur in rule heads, fixpoint computations does not add anything to the extension of the ontology predicates. This is obvious from the construction of fixpoint computation.

During fixpoint computation, in first step we add named elements to \( \mathcal{O} \). After that, in each next step, for every tuple of elements in rules we assign only named elements such that rule body is going to be satisfied. Thus we add these tuple to the head predicate extension. Obviously the fixpoint operator is monotonic and this fixpoint is going to result in in interpretation \( \mathcal{I}' \) which is necessarily a canonical model of \( \mathcal{C} \).

Thus we derive an extended interpretation \( \mathcal{I}' \) that is a model of the combination such that \( \mathcal{I}' \models \mathcal{C} \). We observe that this \( \mathcal{I}' \) is a canonical model of \( \mathcal{C} \). It follows that whenever the ontology is satisfiable the combination is also satisfiable.

Because \( \mathcal{I}' \) is a canonical model we can take the named part of this interpretation over the alphabet \( A_{\mathcal{O}} \), let’s call this \( \mathcal{I} \rvert_{\text{named}} \) which is defined as follows:

**(Definition 5.8)** The interpretation of the named part of the extended canonical model is defined as \( \mathcal{I}' \rvert_{\text{named}} = (\Delta^{\mathcal{I}'}, \mathcal{I}' \rvert_{\text{named}}) \) where

\[
\begin{align*}
    a^{\mathcal{I}' \rvert_{\text{named}}} &= a^{\mathcal{I}'} \quad \text{for each term } a \\
    p^{\mathcal{I}' \rvert_{\text{named}}} &= \{ \bar{u} \mid \bar{u} \in p^{\mathcal{I}'} \text{ and } \exists a_1, \ldots, a_n \text{ such that } a_1^{\mathcal{I}'} = u_1, \ldots, a_n^{\mathcal{I}'} = u_n \}
\end{align*}
\]
5.2. INCOMPATIBILITY: MOTIVATIONS, ANALYSIS AND SOLUTIONS

The set of entailed ontology facts are denoted as $\mathcal{F}_O$ which contains the named part of the ontology the predicate $p$ of which are such that $p \in \mathcal{A}_O$.

Clearly for a named entailed fact $p(\bar{a}) \in \mathcal{F}_O$ it is the case that

$$\mathcal{I} \mid_{named} = p(\bar{a}) \iff \mathcal{C} \models p(\bar{a}) \iff \mathcal{O} \models p(\bar{a})$$

On the other hand, the rule parts of the extended interpretation $\mathcal{I}'$ over the alphabets $\mathcal{A}_\mathcal{P}$, lets call this $\mathcal{I} \mid_{Rule}$ satisfy the rules

$$\mathcal{I} \mid_{Rule} = \mathcal{F}_\mathcal{P}$$

where $\mathcal{F}_\mathcal{P}$ is the set of entailed rule facts for which the predicate are over the set of alphabets $\mathcal{A}_\mathcal{P}$.

Hence after the extension using fixpoint computation the extended interpretation $\mathcal{I}'$ consisting of named part and rule part over the predicate alphabets $\mathcal{A}$ is a model of the modified logic program $\mathcal{P}'$

$$\mathcal{I}' \models \mathcal{F} \cup \mathcal{P}$$

$$\mathcal{I}' \models \mathcal{P}'$$

The Herbrand Universe $U_H$ is the set of all ground terms over function and predicates. The Herbrand Base $B_H$ is the set of all atomic formulas which can be formed using the predicate symbols of $\mathcal{A}$ and the terms in $U_H$. A Herbrand Interpretation $M$ is a subset of $B_H$.

We define Herbrand Interpretation $M$ as a set of ground atomic formulas satisfied by $\mathcal{I}' \mid_{named}$ and by $\mathcal{I}' \mid_{Rule}$.

The Herbrand Interpretation $M$ is a model for $\mathcal{P}'$. $\mathcal{I}' \mid_{Rule}$ contains only named objects because the rules are DL-safe and $\mathcal{F}$ is simply represented by named part from ontology as $\mathcal{I}' \mid_{named}$.

Finally, to prove our theorem we have to show that this Herbrand model is the minimal Herbrand model.

The grounding of logic program $P$, denoted as $gr(P)$ is the union of all possible ground instantiations of $P$, obtained by replacing each variable in $r$ with a term in $U_H$, for each rule $r \in P$.

Let us suppose that $M$ is not minimal model of $\mathcal{P}'$. Then $M$ satisfies some ground atomic formula that is not entailed.

So, $M$ is not $MM$ of $\mathcal{P}'$ iff $\exists \alpha \in M. \mathcal{P}' \not\models \alpha$

As $\alpha$ was satisfied in the canonical model then either it is represented in the named DL part in which case it is obviously entailed by $\mathcal{P}'$ or it is in rule part which is just an extension of the DL part by fixpoint procedure.

Therefore, no such $\alpha$ exists and we can conclude that $M$ is indeed the minimal model $\mathcal{P}'$.
CHAPTER 5. IMPOSING RESTRICTIONS: REGAINING COMPLETENESS

5.3 Effects of Restrictions: Comparison with DLP extended with rules

We mentioned before that our goal is to integrate negation-free rules and Description Logic such that the Description Logic component has necessarily more expressivity than that of DLP extension with rules. Regarding this we now compare case by case the effects of restrictions on our integration approach and in DLP extended with rules.

At first, let us consider the rule restrictions. One of our rule restrictions says that we are not allowed to use DL-atoms in rule heads mainly because of the modular reasoning approach. This allows information flow only in one direction - Descriptions Logics to rules. DLP extended using rules does not have this restriction. Hence, DL-atoms are allowed in both rule heads and bodies and information can flow in both directions - Descriptions Logics to rules as well as rules to Descriptions Logics. As a result, our approach supports one direction of the interoperability of rules and ontologies, while DLP extension using rules supports both.

Another rule restriction is DL-safety. DLP extended using rules doesn’t apply DL-safety restriction on its rules. Even if this restriction is imposed, it does not change anything semantically. The reason is that DLP axioms disallow existential restriction in their heads. Therefore, after the translation of DLP axioms, only known objects are present in the rules making them already DL-safe. As a result, adding special predicates \( \mathcal{O} \) to the rules won’t change the semantics because it does not change the entailed facts.

Moreover, the logic program in our combination is partially constant-free. This is not the case for DLP approach.

From the discussion above for restriction on rules, it is clear that our combination is not as expressive as DLP extension using rules. But in terms of ontology restriction, we are able to express more knowledge from a syntactic point of view. As DLP is based on Horn framework, it cannot express existential information on the right hand side of the inclusion axioms. Horn-SHIZ on the other hand as our choice, is able to express such axioms. Hence, we can still have a reasonably expressive ontology language which can capture some practical needs, able to be used in combination with rules although some syntactic restrictions are required. In addition, we can add rules on top of this ontology and still use existing reasoning tools for computing entailments.

Example 5.4 Consider the Example 3.1 before. Among the axioms in the Description Logic Knowledge Base one is

\[
MScStudent \sqsubseteq \text{Student} \land \exists \text{attends}.\text{AdvancedCourse}
\]
5.3. EFFECTS OF RESTRICTIONS: COMPARISON WITH DLP EXTENDED WITH RULES

With this axiom it is possible to express the existence of an object even if it is not explicitly mentioned in the Knowledge Base. This kind of information is not possible to express with an axiom in DLP ontology as they don’t allow existential information in their axiom heads. Despite the inability of using DL-atoms in rule heads in our approach, this shows how our combination allows this kind of axioms which are widely used in Description Logics which DLP is unable to express.
6

Implementation and Experiments

In this chapter, we describe the whole implementation process in order. First, we become familiar with the reasoning tools used. Here, we describe the main features of Description Logic reasoning tool Pellet and rule reasoner XSB in brief. We use java interface to access the functionalities of Pellet and hence only the relevant functions have been mentioned. The syntax of rules in XSB is also included in this section. The input formats for the Description Logic Knowledge Base and rules are stated next. The input of Description Logic ontology is written here in human readable syntax and rules are written in RIF format. A short description of RIF syntax is added with examples for this purpose. Then the architecture of the reasoning process is given in details. Here the interface is described in details how using Pellet reasoning in Description Logic is performed and RIF rules are rewritten into XSB compatible format and finally rule reasoning is performed. Finally, we add two examples of the whole reasoning process based on our algorithm one of which shows the success and one shows the failure to answer query.

6.1 Reasoning Tools

In this section we give overview of the two reasoning tools which we have used for DL reasoning and rule reasoning. Pellet: a reasoner for Description Logic OWL DL reasoner.

6.1.1 Pellet

Pellet is a sound and complete OWL DL reasoner written in Java. It provides an extensive support for reasoning with individuals, user-defined datatypes and ontology debugging. It supports nominals and conjunctive query over assertions. The implementation of Pellet consists of several extensions to OWL DL, a non-monotonic operator and support for OWL/rule hybrid reasoning.

Pellet facilitates the following services:

- Standard reasoning services
Multiple inferences to the reasoner

SPARQL-DL conjunctive query answering

Datatype reasoning

SWRL rules support

Ontology analysis and repair

Ontology debugging

Incremental reasoning

CHAPTER 6. IMPLEMENTATION AND EXPERIMENTS

For our implementation, the only concern is standard reasoning services. Standard reasoning services are possible with expressive Description Logic SHOIQ. The standard reasoning services are:

Consistency Checking: Makes sure that the ontology does not have any contradictory information

Concept satisfiability: Checks if an instance belongs to a concept.

Classification: Computes the hierarchy of the classes in the ontology.

Realization: Finds the most specific classes that an individual belongs to.

In addition to providing standard reasoning services mentioned above, Pellet includes the functionality of testing arbitrary entailments and conjunctive query answering recommended by W3C recommendation. All these functionalities can be accessed using a programmatic API written in Java that can be used in a standalone application.

The core of the Pellet reasoner is the tableaux reasoner that checks the consistency of a knowledge base. TBox axioms go through the standard preprocessing of DL reasoners, e.g., normalization, absorption and internalization, before they are fed to the tableaux reasoner. Thus tableaux reasoner is coupled with other pre-processing and post-processing components. For details of the architecture interested reader can check [2].

Java Interface to Pellet

Pellet provides a java based API for accessing Pellet’s functionality. It consists of different packages for different tasks. For retrieving facts using Pellet API, we only need the package org.mindswap.pellet.owlapi. This package includes 8 packages among which one is Reasoner.

For retrieving facts, we can use the methods in the Reasoner class. But the elements retrieved here are not with IRI. To get all the elements with IRI, we go inside the KB by invoking getKB() method. This denotes the KB class. Some methods relevant to our implementation are as follows:
6.1. REASONING TOOLS

- `getIndividuals()` method retrieves all the individuals as `ind` in KB
- `getTypes(ind)` method retrieves all the named class individual `ind` belongs to
- `getProperties()` method retrieves all the property names as `propName`
- `getPropertyValues(propName, ind)` method retrieves the property values for each individual `ind` against a property `propName`
Example 6.1 Below are some examples of compound terms

\[
\text{foo}(\text{bar}) \quad 123(\text{John, 200})
\]

\[\text{engagedIn}(\text{John, WebProject}) \quad \text{‘PaidProject’}('\text{WebProject}')\]

Example 6.2 The examples of rules following from Example 3.1 are represented as usual. It is noticeable that \(\leftarrow\) is replaced by the symbol :-

\begin{align*}
\text{EngagedInStJob}(X) : & \quad \neg \text{Student}(X), \text{engagedIn}(X, Y), \\
& \quad \text{PaidProject}(Y).
\end{align*}

\begin{align*}
\text{DoMScThesis}(X) : & \quad \neg \text{Student}(X), \text{AdvancedCourse}(Y), \\
& \quad \text{hasSupervisor}(X, Z), \text{Professor}(Z), \\
& \quad \text{attends}(X, Y), \text{teaches}(Z, Y).
\end{align*}

\begin{align*}
\text{hasWorkLoad}(X, 'High') : & \quad \neg \text{DoMScThesis}(X), \text{EngagedInStJob}(X).
\end{align*}

Query \text{hasWorkLoad}(X, 'High') \quad \text{tries to retrieve the answer for } X \quad \text{who has a high workload.}

6.2 Input Format

There are two kinds of inputs: the Description Logic Knowledge Base \(\mathcal{O}\) and RIF rules. In this section, we discuss them in brief and show how they are represented in the system with appropriate examples.

6.2.1 Description Logic Knowledge Base

One of the inputs is the Description Logic Knowledge Base \(\mathcal{O}\). Pellet takes OWL ontology as input. OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. As OWL ontology is represented in RDF/XML and RDF-Schema, this is not readable to human. Hence, we provide only human syntax of OWL ontology. Example 3.1 can be used for this purpose.

Example 6.3 The university ontology in DL KB \(\mathcal{O}\) contains the facts and axioms as follows

\begin{align*}
\text{MScStudent}(\text{John}) \\
\text{Professor}(\text{Jones}) \\
\text{teaches}(\text{Jones, AdvDatabase})
\end{align*}
AdvancedCourse(AdvDatabase)
attends(John, AdvDatabase)

Professor □ Lecturer □ FacultyMember
MScCourse □ BScCourse □ Course
AdvancedCourse □ MScCourse
FoundationCourse □ BScCourse
Professor □ ∀teaches. AdvancedCourse
Lecturer □ ∀teaches. FoundationCourse
MScStudent □ Student □ ∀attends. AdvancedCourse
MScStudent □ ∀hasSupervisor. Professor

6.2.2 RIF rules

Rule Interchange Format Basic Logic Dialect (RIF-BLD) [6] is a Web Rule Language based on Horn rules with equality and not based RDF or OWL. RIF-BLD corresponds to the language of definite Horn rules with equality and a standard First-Order semantics. The format of RIF-BLD allows logic rules to be exchanged between rule systems. Interested readers can consult [6] for detailed specification. In this thesis we ignore equality for avoiding complexity in implementation.

The concrete syntax for rules is presented as RIF rules.

Syntax of RIF rules

Here we give a short description of the syntax of the RIF rules. We are only concerned with the part of syntax required for our implementation and hence any irrelevant term mentioned in the original specification in [6] is avoided.

RIF alphabets consist of the following:

- a countably infinite set of constants Const
- a countably infinite set of variables Var
- connective symbols And, Or, :-
- Quantifiers Exists, Forall
- Auxiliary symbols ‘(’, ‘)’, ‘[’, ‘]’, ‘<’, ‘>’, ‘:’ and ‘^\wedge’
Just like rules defined in the preliminaries in chapter 2, RIF rules building blocks are terms. Constant, variable, positional terms or frame terms are terms. Variables are preceded with a ? mark examples of which are ?X, ?Y. Constants are preceded by a prefix which is an IRI and colon (:) works as a separator in this case. For example, ex:John is a constant where ex prefix denotes the IRI http://www.example.org.

Positional term is written as engagedIn(ex:John ex:WebProject) where inside the parenthesis () constants or variables can occur and no comma ‘,’ is required as a separator, space is used instead. Examples of frame are X[rdf:type -> ex:Student] which is a class membership, X[ex:hasSupervisor -> Z] which is role-value.

RIF has its own semantics but to be appear at the combination with OWL DL defined in [7], frame formula X[rdf:type -> ex:Student] is effectively interpreted as if it were a unary predicate ex:Student(X) and X[ex:hasSupervisor -> Z] is interpreted as if it were a binary predicate ex:hasSupervisor(X, Z). This is different from the standard RIF semantics and only applicable in RIF-OWL DL combinations as seen in these two predicates ex:Student_ and ex:hasSupervisor_ with an underscore at the end to clarify the modification.

Next comes the formula which is a term or collection of terms. Formula can be of different types: atomic, conditional, rule implication, universal rule, universal fact or a group of formulas. Condition formulas can contain conjunction or disjunction of formulas or can be existentially quantified. Rule implication is a rule with bodies and heads and a symbol ‘:-’ between them. The rule heads can have atomic formulas or conjunction of atomic formulas and rule bodies can be any condition formula. A universal rule is a rule implication with all the variables universally quantified using quantifier Forall. An atomic formula with all the variables universally quantified is called universal fact.

All kinds of formulas mentioned above are put together in a group under Group symbol. A group can also be nested. Finally, we introduce Document which contains the group. Hence, everything regarding RIF has document as root.

A RIF Document is expressed in the form

\[
\text{Document}(\text{directive}_1 \ldots \text{directive}_n, \Gamma)
\]

which is a document formula where \( \Gamma \) is a well-formed group formula. The directive is a prefix directive and has the form Prefix(p v), where p is an alphanumeric string that serves as the prefix name and v is a macro-expansion for p – a string that forms an Internationalized Resource Identifier(IRI). This directive is used to allow more concise representation of IRI constants.

In Example 3.1 we described the abstract syntax of rules. Here, we represent them in RIF notations. All these facts and rules mentioned in that
example are put together in a group that is the contained by the document. Like the rules in combination in abstract syntax, RIF rules also contain DL-atoms and non-DL-atoms. These atoms can be easily recognized as DL-atoms are effective represented as frame formulas.

**Example 6.4** Here, we consider again the Example 3.1 and present the RIF syntax of the same rules

```xml
Document(
    Prefix(ex http://www.example.org)
    Prefix(rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#type)

Group(
    'ex:engagedIn'('ex:John' 'ex:WebProject')

    'ex:PaidProject'('ex:WebProject')

    Forall ?X ?Y ( 'ex:EngagedInStJob'(?X) :- And(?X[rdf:type -> 'ex:Student']
        'ex:engagedIn'(?X ?Y)
        'ex:PaidProject'(?Y))

    )

        ?Y[rdf:type ->'ex:AdvancedCourse']
        ?X['ex:hasSupervisor' -> ?Z]
        ?Z[rdf:type ->'ex:Professor']
        ?X['ex:attends' -> ?Y]
        ?Z['ex:teaches' -> ?Y])

    )

    Forall ?X ( 'ex:hasWorkLoad'(?X 'ex:High') :- And('ex:DoMScThesis'(?X)
        'ex:EngagedInStJob'(?X))

    )

)
```

The above example shows a RIF document structure. The document consists of prefix `ex` directives and a group of formulas. The first two atomic formulas are denoted by the predicate name `engagedIn`, `PaidProject`. The rest of the three formulas are universal rules all of which have condition formulas depicting the conjunction in their bodies and atomic formulas in their heads.
6.3 Reasoning Process: Architecture

Here we present the architecture of the prototype of our implementation system with two reasoning tools. The architecture depicts only two kinds of elements: elements inside box denoting processing units taking inputs and providing outputs and elements outside box as input or output. In the following we present these units in brief along with their functionalities:

![Architectures of Reasoning Process](image)

**Pellet OWL DL Reasoner** The Pellet OWL DL Reasoner takes a DL Knowledge Base $\mathcal{O}$ and retrieves all the facts $\mathcal{F}$ from that Knowledge Base. We use the Java API of Pellet for accessing the functionalities. The process is as follows in brief.

To get the facts, we retrieve all the individuals and property names of the ontology first using these two methods `getKB().getIndividuals()` and `getKB().getProperties()` respectively.

- **Getting class assertions**: Using the method `getKB().getTypes(ind)` we get all the named classes individuals are member of

- **Getting property assertions**: For each individual and each property we retrieve the property value of that individual using the method
getKB().getPropertyValues(prop, ind)

The facts in F can only be of the following forms: class assertions as unary predicate and property assertions as binary predicates.

ClassName(IndividualName).
PropertyName(IndividualName, PropertyValue).

We consider the ontology in Example 6.3 above. The ontology O contains some facts and axioms about the university education domain. Using Pellet reasoner we retrieve all facts. It is interesting to see that some new facts have been included after DL reasoning. The retrieved facts are as follows:

‘ex:MScStudent_’(‘ex:John’).
‘ex:Student_’(‘ex:John’).
‘ex:Professor_’(‘ex:Jones’).
‘ex:teaches_’(‘ex:Jones’, ‘ex:AdvDatabase’).
‘ex:AdvancedCourse_’(‘ex:AdvDatabase’).
‘ex:MScCourse_’(‘ex:AdvDatabase’).
‘ex:hasSupervisor_’(‘ex:John’, ‘ex:Jones’).

All the predicate names in the facts have been modified with a underscore (_) at the end to correspond to the modified semantics of frame formulas.

Rewriting Tool The rules in P are taken in the form of RIF Rules. RIF rules correspond to the language of definite Horn rules with equality and a standard First-Order semantics. We have already seen the abstract syntax of rules. RIF syntax are presented in concrete representation which correspond to the abstract form. As this concrete syntax is different from the one compatible in XSB, we use a rewriting tool that transforms these RIF rules understandable to rules in XSB.

RIF has its own semantics but to be appear at the combination with OWL DL defined in [7], frame formula X[rdf:type -> ex:Student] is effectively interpreted as if it were a unary predicate ex:Student(X) and X[ex:hasSupervisor -> Z] is interpreted as if it were a binary predicate ex:hasSupervisor(X, Z). This is different from the standard RIF semantics and only applicable in RIF-OWL DL combinations as seen in these two predicates ex:Student_ and ex:hasSupervisor_ with an underscore at the end to clarify the modification.

We mentioned before in this chapter that during the combination of RIF and OWL DL, the semantics of frame formulas are modified. The
frame formulas are effectively interpreted as if they pose as unary and predicates. The modification is performed as follows:

\[ t_1[t_2 \rightarrow t_3] \approx t'_2(t_1, t_3) \]

\[ t_1[\text{rdf:type} \rightarrow t_2] \approx t'_2(t_1) \]

where \( t_1, t_2, t_1 \) are frame terms and \( t'_2 \) is a fresh term modifying \( t_2 \).

As a result, we can show that after rewriting RIF rules in Example 6.4, the results of the translation become

‘ex:engagedIn(‘ex:John’, ‘ex:WebProject’).
‘ex:PaidProject(‘ex:WebProject’).
‘ex:EngagedInStJob’(X) :- ‘ex:Student’(X), ‘ex:engagedIn’(X, Y),
‘ex:PaidProject’(Y).
‘ex:DoMScThesis’(X) :- ‘ex:Student’(X), ‘ex:AdvancedCourse’(Y),
‘ex:hasSupervisor’(X, Z), ‘ex:Professor’(Z),
‘ex:hasWorkLoad’(X, ‘ex:High’) :- ‘ex:DoMScThesis’(X),
‘ex:EngagedInStJob’(X).

**XSB Rule Reasoner** Finally, XSB rule reasoner performs the rule reasoning. The facts and the rule are put together in a logic program \( P' \). The logic program is loaded in XSB reasoner and it gives an answer to the query. In the following sections, examples will be included to demonstrate the query answering.

Answering the query \(-\ hasWorkLoad(X, ‘ex:High’)\) is performed in XSB reasoner and it returns ‘John’ deducing as answer that ‘John’ has a high workload.

### 6.4 Experiments

In this section, we provide two reasoning examples based on our algorithm. The first example shows the success of the query answering while the second shows failure. For both the examples we use Horn-\(\mathcal{SHIQ}\) ontologies which are in Description Logic which are in human readable syntax and recursive Horn rules.

#### 6.4.1 Example 1

For this example, the Horn-\(\mathcal{SHIQ}\) ontology talks about university course domain. The Description Logic KB \( \mathcal{O} \) consists of the following facts and axioms
6.4. EXPERIMENTS

FullProfessor(John)
AdvancedCourse(AI)
Topic(KR)
Topic(LP)
Topic(DL)
(FullProfessor \(\cap\) (\(\forall\) teachCourse. AdvancedCourse))(Mary)
Student(Paul)

FullProfessor \(\subseteq\) FacultyMember
NonTeachingFullProfessor \(\equiv\) FullProfessor \(\cap\) \(\forall\) teachCourse. \(\neg\) Course
AdvancedCourse \(\subseteq\) Course
BasicCourse \(\subseteq\) Course
AdvancedCourse \(\cap\) BasicCourse \(\equiv\) \(\bot\)
AdvancedCourse \(\equiv\) \(\exists\) hasPrerequisite.BasicCourse

Below are the set of RIF rules

Document(
  Prefix(ex http://www.example.org)
  Prefix(rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#type)

Group
  (‘ex:passedExam’(‘ex:Paul’ ‘ex:AI’)
   ‘ex:subjectOf’(‘ex:AI’ ‘ex:LP’)
   ‘ex:expertIn’(‘ex:John’ ‘ex:KR’)
   ‘ex:expertIn’(‘ex:Mary’ ‘ex:DL’)

  ?Y[rdf:type ->‘ex:Course’]
  ?Z[rdf:type ->‘ex:Topic’]
  ‘ex:hasKnowledgeOf’(?X ?Y)
  ‘ex:subjectOf’(?Y ?Z))
)

  ‘ex:subTopicOf’(?Z ?Y))
)

CHAPTER 6. IMPLEMENTATION AND EXPERIMENTS

?Y[rdf:type ->'ex:FacultyMember']
?Z[rdf:type ->'ex:Topic']
'ex:hasKnowledgeOf'(?X ?Z)
'ex:expertIn'(?Y ?Z))
)
)

After retrieving facts \( \mathcal{F} \) and rewriting the RIF rules into LP rules in \( \mathcal{P} \), they are put together in a modified logic program \( \mathcal{P}' \) which is now recognizable in XSB rule reasoner.

'ex:FullProfessor'('ex:John').
'ex:AdvancedCourse'('ex:AI').
'ex:Course'('ex:AI').
'ex:Topic'('ex:KR').
'ex:Topic'('ex:LP').
'ex:Topic'('ex:DL').
'ex:FullProfessor'('ex:Mary').
'ex:FacultyMember'('ex:Mary').
'ex:FacultyMember'('ex:John').
'ex:Student'('ex:Paul').
'ex:passedExam'('ex:Paul' 'ex:AI').
'ex:subjectOf'('ex:AI' 'ex:KR').
'ex:subjectOf'('ex:AI' 'ex:LP').
'ex:expertIn'('ex:John' 'ex:KR').
'ex:expertIn'('ex:Mary' 'ex:DL').
'ex:subTopicOf'('ex:DL' 'ex:KR').
'ex:hasKnowledgeOf'(?X ?Z) :-
  'ex:Student'(X),
  'ex:Course'(Y),
  'ex:Topic'(Z),
  'ex:passedExam'(X, Y),
  'ex:subjectOf'(Y, Z).
'ex:hasKnowledgeOf'(X, Z) :-
  'ex:hasKnowledgeOf'(X, Z),
  'ex:subTopicOf'(Z, Y).
'ex:mayDoThesisIn'(X, Y, Z) :-
  'ex:Student'(X),
  'ex:Topic'(Z),
  'ex:FacultyMember'(Y),
  'ex:hasKnowledgeOf'(X, Y),
  'ex:expertIn'(Y, Z).

To answer the query : - 'ex:mayDoThesisIn'(X, Y, Z) using the XSB rule reasoner we get that student Paul can do his thesis on topic DL under the supervision of faculty member Mary who is an expert in that area. This shows that the rule reasoner is able to successfully answer our query.
6.4. EXPERIMENTS

6.4.2 Example 2

This example, which is on the airport-flight domain, shows a case where the rule reasoner fails to answer our query. Similar to the program in the above example, it starts with the ontology below:

Hub ⊆ Airport
Airline ∈ ∃operates.(∃startsFrom.Hub /
   ∃arrivesAt.Hub) ≡ MajorAirline
Airplane ∈ ∃uses−.∃operates−.MajorAirline ≡ NewAirPlane
Flight ∈ ∃uses.Airplane
Hub(Heathrow)
Hub(CDG)
Flight(F101)
startsFrom(F101, Heathrow)
arrivesAt(F101, CDG)
Airline(BA)
operates(BA, F101)

The RIF rules are as follows:

Document(
   Prefix(ex http://www.example.org)
   Prefix(rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#type)

Group
      ?Z[ex:arrivesAt' -> ?Y]
      ?Z[ex:uses' -> ?W]
      ?W[rdf:type ->'ex:NewAirplane'])))

      ?Z[ex:arrivesAt' -> ?V]
      ?Z[ex:uses' -> ?W]
      ?W[rdf:type ->'ex:NewAirplane']
      'ex:easilyReachable'(?V ?Y))
)
)
)

57
CHAPTER 6. IMPLEMENTATION AND EXPERIMENTS

The modified logic program \( P' \) consists of the facts and rules as follows

\[
\begin{align*}
\text{ex:Hub} &\left( \text{ex:Heathrow} \right). \\
\text{ex:Airport} &\left( \text{ex:Heathrow} \right). \\
\text{ex:Hub} &\left( \text{ex:CDG} \right). \\
\text{ex:Airport} &\left( \text{ex:CDG} \right). \\
\text{ex:Flight} &\left( \text{ex:F101} \right). \\
\text{ex:startsFrom} &\left( \text{ex:F101}, \text{ex:Heathrow} \right). \\
\text{ex:arrivesAt} &\left( \text{ex:F101}, \text{ex:CDG} \right). \\
\text{ex:Airline} &\left( \text{ex:BA} \right). \\
\text{ex:operates} &\left( \text{ex:BA}, \text{ex:F101} \right). \\
\text{ex:easilyReachable} &\left( X, Y \right) := \text{ex:Flight}\left( Z \right), \text{ex:startsFrom}\left( Z, X \right), \\
& \quad \text{ex:arrivesAt}\left( Z, Y \right), \text{ex:uses}\left( W \right), \\
& \quad \text{ex:NewAirplane}\left( W \right). \\
\text{ex:easilyReachable} &\left( X, Y \right) := \text{ex:Flight}\left( Z \right), \text{ex:startsFrom}\left( Z, X \right), \\
& \quad \text{ex:arrivesAt}\left( Z, V \right), \text{ex:uses}\left( Z, W \right), \\
& \quad \text{ex:NewAirplane}\left( W \right), \\
& \quad \text{ex:easilyReachable}\left( V, Y \right). 
\end{align*}
\]

During reasoning process XSB rule reasoner is unable to answer the query \( \lnot \text{ex:easilyReachable}\left( X, Y \right) \). This is because none of the above two rules has any information about the airplane that is used in flight F101. Thus the query answering fails.

6.4.3 Scaling of various inputs

We test the our reasoning algorithm with an increasing number of instances and try to measure the time it takes to process. The results of this experiments are depicted in the following figures. All these experiments were conducted on a machine running under Mac OS X operating system with 2 GHz Intel processor and 1GB of memory.

In both cases, they are tested with the same number of instances ranging from 10 to 10,000. The ontologies are necessarily Horn-\( SHIQ \) and recursive rules are used for reasoning. Number of instances are plotted on X-axis and for each run the accumulated time spend in Pellet Description Logic reasoner and XSB rule reasoner are plotted on Y-axis.
6.4. EXPERIMENTS

Figure 6.2: Reasoning Time Vs Number of Instances for Example 1

Figure 6.3: Reasoning Time Vs Number of Instances for Example 2
CHAPTER 6. IMPLEMENTATION AND EXPERIMENTS

For both the cases with Example 1 and Example 2, the scaling shows a linear characteristics.

The implementations performed with two standard reasoners Pellet and XSB on the examples above show that reasoning of the combination of Description Logics and rules can be performed using existing reasoning tools. Thus, our algorithm proves that any such standard reasoning tools can be used for reasoning. During the implementation we particularly concentrated on building the modified logic program. As it contains rules where DL-atoms can occur in their body and need to be matched with the fact retrieved from the ontology, manipulation is required. This problem is solved while retrieving facts from ontology and using rewriting tool on RIF rules. As a young rule language, tools able to process RIF syntax are yet to mature. Hence, performance could be increased with more sophisticated tools on RIF rules in future.
The conclusion section is summarized as follows. At first, we summarize the content of this thesis. We recall some previous contributions in this area and compare the different aspects of our approach along with the results with those. We present some scopes for future work in this area and finish this thesis with some concluding remarks.

7.1 Summary

In this thesis we have described a combination of Description Logics and negation-free rules corresponding to the wake of research in this area and in conformance with RIF-OWL combination semantics based on single models. We have chosen the reasoning task of answering conjunctive queries in this combination. In this regard, an algorithm is proposed for which reasoning is sound but not complete for all such combinations. With a view to regaining completeness, the reasons behind this have been identified. In this regard, we have shown that by imposing restrictions on the Description Logic and rules components, a particular combination is possible to find for which the algorithm regains completeness.

We have chosen Horn-$SHIQ$ as Description Logic component because it has the expressivity that is common in most of the Description Logic languages. For example, unlike Description Logic DLP, our choice of Horn-$SHIQ$ can express existential information in its axiom consequents. It has also low data complexity.

The restrictions on rules include: disallowing DL-atoms and imposing syntactic restrictions on rules called DL-safety. As the reasoning algorithm follows modular approach which performs reasoning in Description Logic and rules in isolation, feedback from one reasoner to the other is not facilitated. Hence, DL-atoms are not allowed in rule heads. On the other hand, DL-safety restrictions were imposed on the variables of rules that makes sure that only explicitly mentioned objects in the Knowledge Base are taken into account during evaluation process.

In addition to the above restrictions, the logic program in our approach is partially constant-free to avoid incompleteness of our algorithm regarding
equality (inequality) in Description Logic.

The algorithm we propose here is the heart of this thesis. It is sound and can be implemented easily. The algorithm retrieves the facts from the Description Logic Knowledge Base. These facts and rules are put together in a logic program. Answering conjunctive queries is performed in a rule reasoner on this logic program. More importantly, this algorithm can be used with any existing standard Description Logic and rule reasoners that are available now-a-days. No new reasoning tools are required to invent. For example, during the prototype implementation we used Pellet OWL Description Logic reasoner and XSB rule reasoner.

To support our theory presented in this thesis, we provided a prototype implementation. To retrieve facts, we use Description Logic Horn-$SHIQ$ ontology. As negation-free rules, we use RIF rule format as the rule component. Using a rewriting tool these RIF rules are rewritten such that they are compatible in XSB rule reasoner. All these facts and rules form a logic program. Finally, conjunctive queries are answered on this logic program using the XSB rule reasoner.

### 7.2 Relationship with other combination approaches

Our thesis is a step followed from the previous contributions in this kind of combinations. The notable previous integration approaches include $\mathcal{AL}$-log, CARIN, DL-safe SWRL and DLP extended with rules.

To achieve decidable reasoning restrictions were imposed on the DL and rule components in these combination approaches. Here we mention some of these restrictions and compare the expressivity of these approaches with our approach. We start with restrictions on Description Logics in this part. The combinations of Description Logic constructors are among the well known sources for undecidable reasoning. Just like our approach, in CARIN and DLP extension with rules the Description Logic component is restricted to disallow certain constructors in their axioms. For example, DLP does not allow existential restriction and disjunctive constructors in their axiom consequents while Horn-$SHIQ$ in our case does allow existential restriction.

Unlike DLP extended with rules approach all the previous approaches including the combination in this thesis use restrictions. These restrictions are syntactically imposed on the variables occurring in rules and can vary. For example, CARIN uses role-safety which apply to only one variable in roles while (strong) DL-safety restriction used in $\mathcal{AL}$-log and DL-safe SWRL apply to all the variables in concepts and roles. We also use DL-safety restriction here. This restrictions on rules, in turn, poses problems for using arbitrary conjunctive query.

Both $\mathcal{AL}$-log and CARIN uses modular reasoning approach where Description Logic reasoning and rule reasoning is performed in isolation and
there is no feedback from one reasoner to the other. As our reasoning adopts
the same technique of modular reasoning, as a result just like $\mathcal{AL}$-log and
CARIN approach DL-atoms are allowed to occur only in rule bodies, not
in heads. This is not the case for DLP extension with rules and SWRL
approach as they use only one reasoner, namely rule reasoner. Therefore,
while SWRL and DLP extension with rules support the interoperability of
Description Logics and rules in Semantic Web stack, our approach doesn’t.

From the discussion above based on restriction on rules, it is clear that
our combination is not as expressive as DLP extension with rules in terms
of interoperability. But in terms of restriction on Description Logic compo-
nent, we are able to express more knowledge from syntactic point of view.
As DLP is based on Horn framework, it cannot express existential restriction
on the right hand side of the inclusion axioms. Horn-$\mathcal{SHIQ}$ on the other
hand, is able to express such axioms. Hence, we can still have a reasonably
expressive ontology language which can capture some practical needs. Al-
though modular reasoning approach used in CARIN and $\mathcal{AL}$-log limits us
to express DL-atoms in rules bodies only, we have significant advantage over
these two approaches. The algorithm used in this thesis enables us to use
any existing standard reasoning tools which by the way is not possible in
$\mathcal{AL}$-log and CARIN. Because of the algorithms used in $\mathcal{AL}$-log and CARIN,
new reasoning tools are required to implement in those cases.

Combining Description Logics and rules is currently a very important
topic and an important step in the development of Semantic Web architec-
ture. In this respect, upcoming OWL standard OWL 2 and rule format RIF
from World Wide Web Consortium are getting a lot of attentions and the
RIF-OWL combination [7] reflects a strong focus in this direction. In this
thesis, such a combination of Description Logics and rules has been studied
that follows the direction towards this RIF-OWL integration. We have in-
troduced and implemented a new reasoning algorithm for such integration.
Our algorithm is sound and easily implementable with existing reasoning
tools. But the algorithm is not complete for all such integrations. To re-
gain completeness, we chose a particular subset of such combinations where
the components Description Logic and rules are restricted. In this subset,
the Description Logic is chosen as Horn-$\mathcal{SHIQ}$ and rules are DL-safe and
DL-atoms are allowed to occur only in rule bodies, not in heads. The imple-
mentation is experimented with standard Description Logic reasoner Pellet
and rule reasoner XSB. The choice of Horn-$\mathcal{SHIQ}$ is justified because of
its expressivity and prospect of its superset Horn-$\mathcal{SROIQ}$’s being included
in the upcoming OWL 2. We used the rules in RIF format. Being a re-
cent language format, sophisticated tools for recognizing RIF are yet to be
implemented. It would also be interesting to use a more sophisticated al-
gorithm in future. In essence, our intension was to find a combination of
rules and Description Logics that follows the path of RIF-OWL combina-
tion where the Description Logic component is strictly more expressive than
CHAPTER 7. CONCLUSION

DLP ontologies and sound and complete reasoning is possible using existing reasoning tools. From the discussions and comparisons above, it can be concluded that this goal is fairly achieved.

7.3 Future work

The content of this thesis can be further investigated and extended in different manners. Here we try to explore some of these possibilities of extending our approach with particular directions.

We showed that the completeness of our algorithm is regained for a particular subset of the integration of Description Logics and rules. As the upcoming owl standard OWL 2 considers expressive Description Logic $SROIQ$ [39], it would be interesting to see if the Description Logic component in our approach can be extended to capture as much as expressive power as $SROIQ$. We think that Horn-$SROIQ$ can be used used in this direction.

The rules in our combination are negation-free and disjunction is not allowed in rule heads. The combination attains much expressivity if these two are allowed. Therefore, more research in this direction would be interesting as this will also help capture disjunction in the Description Logic component.

Due to straightforward DL-safety rule restriction, it is inconvenient to use arbitrary conjunctive query. As conjunctive query is widely used in practice, it is important to overcome such problem. Weak DL-safe rules have been proposed to overcome this sort of problem and hence could be investigated in future.

Our algorithm follows a modular reasoning approach which does reasoning in Description Logics and rules separately. Due to this technique, DL-atoms are allowed only in rule bodies and not in heads. This limits the expressivity significantly as information flow is in one direction only, from Description Logics to rules. We would like to investigate other algorithms in this regard so that DL-atoms can be used both in rules and heads. This will greatly support the future Semantic Web architecture where rules and Description Logics stand side-by-side and are required to interoperate.
A.1 courses.owl Ontology

```xml
<?xml version="1.0"?>

<!DOCTYPE Ontology [ 
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" > 
]>

<Ontology xmlns="http://www.w3.org/2006/12/owl2-xml#" 
  xml:base="http://www.w3.org/2006/12/owl2-xml#" 
  xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#" 
  courses.owl#" 
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#" 
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" 
  xmlns:rdfl="http://www.w3.org/1999/02/22-rdf-syntax-ns#" 
  xmlns:owl="http://www.w3.org/2002/07/owl#" 

<EquivalentClasses> 
  <SubClassOf> 
    <Class URI="& courses;AdvancedCourse" /> 
    <Class URI="& owl;Thing" /> 
  </SubClassOf> 
</EquivalentClasses> 

<DisjointClasses> 
  <Class URI="& courses;AdvancedCourse" /> 
  <Class URI="& courses;BasicCourse" /> 
</DisjointClasses> 

<Declaration> 
  <Class URI="& courses;AdvancedCourse" /> 
</Declaration>
```

65
<SubClassOf>
<Class URI="& courses;BasicCourse"/>
<Class URI="& courses;Course"/>
</SubClassOf>
<SubClassOf>
<Class URI="& courses;BasicCourse"/>
<Class URI="& owl;Thing"/>
</SubClassOf>
<Declaration>
<Class URI="& courses;BasicCourse"/>
</Declaration>
<SubClassOf>
<Class URI="& courses;Course"/>
<Class URI="& owl;Thing"/>
</SubClassOf>
<Declaration>
<Class URI="& courses;Course"/>
</Declaration>
<SubClassOf>
<Class URI="& courses;FacultyMember"/>
<Class URI="& owl;Thing"/>
</SubClassOf>
<Declaration>
<Class URI="& courses;FacultyMember"/>
</Declaration>
<SubClassOf>
<Class URI="& courses;FullProfessor"/>
<Class URI="& courses;FacultyMember"/>
</SubClassOf>
<SubClassOf>
<Class URI="& courses;FullProfessor"/>
<Class URI="& owl;Thing"/>
</SubClassOf>
<Declaration>
<Class URI="& courses;FullProfessor"/>
</Declaration>
<EquivalentClasses>
<Class URI="& courses;NonTeachingFullProfessor"/>
<ObjectIntersectionOf>
<Class URI="& courses;FullProfessor"/>
<ObjectAllValuesFrom>
<ObjectProperty URI="& courses;teachCourse"/>
<ObjectComplementOf>
<Class URI="& courses;Course"/>
</ObjectComplementOf>
A.1. COURSES.OWL ONTOLOGY

</ObjectComplementOf>
</ObjectAllValuesFrom>
</ObjectIntersectionOf>
</EquivalentClasses>
<SubClassOf>
<Class URI="& courses;NonTeachingFullProfessor"/>
<Class URI="& owl;Thing"/>
</SubClassOf>
<Declaration>
<Class URI="& courses;NonTeachingFullProfessor"/>
</Declaration>
<SubClassOf>
<Class URI="& courses;Student"/>
<Class URI="& owl;Thing"/>
</SubClassOf>
<Declaration>
<Class URI="& courses;Student"/>
</Declaration>
<SubClassOf>
<Class URI="& courses;Topic"/>
<Class URI="& owl;Thing"/>
</SubClassOf>
<Declaration>
<Class URI="& courses;Topic"/>
</Declaration>
<Declaration>
<ObjectProperty URI="& courses;hasPrerequisite"/>
</Declaration>
<Declaration>
<ObjectProperty URI="& courses;teachCourse"/>
</Declaration>
<ClassAssertion>
<Class URI="& courses;AdvancedCourse"/>
<Individual URI="& courses;AI"/>
</ClassAssertion>
<Declaration>
<Individual URI="& courses;AI"/>
</Declaration>
<ClassAssertion>
<Class URI="& courses;Topic"/>
<Individual URI="& courses;DL"/>
</ClassAssertion>
<Declaration>
<Individual URI="& courses;DL"/>
</Declaration>
<ClassAssertion>
<Class URI="& courses;FullProfessor"/>
<Individual URI="& courses;John"/>
</ClassAssertion>
<Declaration>
<Individual URI="& courses;John"/>
</Declaration>
</Ontology>
B.1 airtravel.owl Ontology

<?xml version="1.0"?>
<!DOCTYPE Ontology [ 
<!ENTITY owl “http://www.w3.org/2002/07/owl#” > 
<!ENTITY xsd “http://www.w3.org/2001/XMLSchema#” > 
<!ENTITY rdfs “http://www.w3.org/2000/01/rdf-schema#” > 
<!ENTITY rdf “http://www.w3.org/1999/02/22-rdf-syntax-ns#” > 
<!ENTITY airtravel 
]> 

<Ontology xmlns="http://www.w3.org/2006/12/owl2-xml#" 
 xml:base="http://www.w3.org/2006/12/owl2-xml#" 
 xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#" 
 airtravel.owl#" 
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#" 
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" 
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" 
 xmlns:owl="http://www.w3.org/2002/07/owl#" 
 URI="http://www.semanticweb.org/ontologies/2009/2/airtravel.owl” > 
<SubClassOf> 
<Class URI="&airtravel;Airline” /> 
<Class URI="&owl;Thing” /> 
</SubClassOf> 
<Declaration> 
<Class URI="&airtravel;Airline” /> 
</Declaration> 
<SubClassOf> 
<Class URI="&airtravel;Airplane” /> 
<Class URI="&owl;Thing” /> 
</SubClassOf>
</SubClassOf>  
</Declaration>  
<Class URI="&airtravel;Airplane"/>
</Declaration>  
</SubClassOf>  
<Class URI="&airtravel;Airport"/>  
<Class URI="&owl;Thing"/>
</SubClassOf>  
</Declaration>  
<Class URI="&airtravel;Airport"/>  
</SubClassOf>  
</Class URI="&airtravel;Flight"/>  
<ObjectSomeValuesFrom>  
<ObjectProperty URI="&airtravel;uses"/>  
<Class URI="&airtravel;Airplane"/>  
</ObjectSomeValuesFrom>  
</SubClassOf>  
</Declaration>  
<Class URI="&airtravel;Flight"/>  
</Declaration>  
</SubClassOf>  
<Class URI="&airtravel;Hub"/>  
<Class URI="&airtravel;Airport"/>  
</SubClassOf>  
</SubClassOf>  
<Class URI="&airtravel;Hub"/>  
<Class URI="&owl;Thing"/>  
</SubClassOf>  
</Declaration>  
<Class URI="&airtravel;Hub"/>  
</Declaration>  
<EquivalentClasses>  
<Class URI="&airtravel;MajorAirline"/>  
<ObjectIntersectionOf>  
<Class URI="&airtravel;Airline"/>  
<ObjectSomeValuesFrom>  
<ObjectProperty URI="&airtravel;operates"/>  
<ObjectIntersectionOf>  
<ObjectSomeValuesFrom>
<ObjectProperty URI="&airtravel;arrivesAt"/>
<Class URI="&airtravel;Hub"/>
</ObjectSomeValuesFrom>
<ObjectSomeValuesFrom>
<ObjectProperty URI="&airtravel;startsFrom"/>
<Class URI="&airtravel;Hub"/>
</ObjectSomeValuesFrom>
<ObjectIntersectionOf>
</ObjectSomeValuesFrom>
<ObjectIntersectionOf>
</EquivalentClasses>
</SubClassOf>
<Class URI="&airtravel;MajorAirline"/>
</SubClassOf>
</Declaration>
<Class URI="&airtravel;NewAirplane"/>
<ObjectIntersectionOf>
<Class URI="&airtravel;Airplane"/>
<ObjectSomeValuesFrom>
<ObjectProperty URI="&airtravel;usedBy"/>
<ObjectSomeValuesFrom>
<ObjectProperty URI="&airtravel;operatedBy"/>
<Class URI="&airtravel;MajorAirline"/>
</ObjectSomeValuesFrom>
<ObjectSomeValuesFrom>
</ObjectIntersectionOf>
</EquivalentClasses>
</SubClassOf>
<Class URI="&airtravel;NewAirplane"/>
<Class URI="&owl;Thing"/>
</SubClassOf>
</Declaration>
<Class URI="&airtravel;NewAirplane"/>
</SubClassOf>
</Declaration>
<ObjectProperty URI="&airtravel;arrivesAt"/>
</Declaration>
<ObjectProperty URI="&airtravel;operatedBy"/>
</Declaration>
APPENDIX B. APPENDIX

<InverseObjectProperties>
<ObjectProperty URI="&airtravel:operates"/>
<ObjectProperty URI="&airtravel:operatedBy"/>
</InverseObjectProperties>
<Declaration>
<ObjectProperty URI="&airtravel:operates"/>
</Declaration>
<ObjectProperty URI="&airtravel:operates"/>
</Declaration>
<ObjectProperty URI="&airtravel:operates"/>
</ObjectProperty>
<ObjectProperty URI="&airtravel:uses"/>
<ObjectProperty URI="&airtravel:usedBy"/>
</InverseObjectProperties>
<Declaration>
<ObjectProperty URI="&airtravel:uses"/>
</Declaration>
<ObjectPropertyURI="&airtravel:uses"/>
</ObjectPropertyAssertion>
<Declaration>
<Individual URI=&airtravel:BA"/>
<Individual URI=&airtravel:FL583"/>
</ObjectPropertyAssertion>
<Declaration>
<Individual URI=&airtravel:BA"/>
</Declaration>
<ClassAssertion>
<Class URI=&airtravel:Hub"/>
<Individual URI=&airtravel:CDGaulle"/>
</ClassAssertion>
<Declaration>
<Individual URI=&airtravel:CDGaulle"/>
</Declaration>
<ClassAssertion>
<Class URI=&airtravel:Flight"/>
<Individual URI=&airtravel:FL583"/>
</ClassAssertion>
<ObjectPropertyAssertion>
<ObjectProperty URI="&airtravel:startsFrom"/>
<Individual URI=&airtravel:FL583"/>
<Individual URI=&airtravel:Heathrow"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
<ObjectProperty URI="&airtravel;startsFrom"/>
<Individual URI="&airtravel;FL583"/>
</ObjectPropertyAssertion>
</Declaration>
</Ontology>

<ObjectPropertyAssertion>
<ObjectProperty URI="&airtravel;startsFrom"/>
<Individual URI="&airtravel;CDGaulle"/>
</ObjectPropertyAssertion>
</Declaration>
</Ontology>

<ObjectPropertyAssertion>
<ObjectProperty URI="&airtravel;startsFrom"/>
<Individual URI="&airtravel;FL583"/>
</ObjectPropertyAssertion>
</Declaration>
</Ontology>

<ClassAssertion>
<Class URI="&airtravel;Hub"/>
<Individual URI="&airtravel;Heathrow"/>
</ClassAssertion>
</Declaration>
</Ontology>

<ObjectPropertyAssertion>
<ObjectProperty URI="&airtravel;startsFrom"/>
<Individual URI="&airtravel;Heathrow"/>
</ObjectPropertyAssertion>
</Declaration>
</Ontology>
Bibliography


description logics. volume 1705 of Lecture Notes in Artificial

[27] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet,
Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule
language combining owl and ruleml. W3c member submission, World

Description logic programs: combining logic programs with

[29] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for

[30] Bernardo Cuena Grau (eds.). Owl 1.1 web ontology language
tractable fragments, w3c member submission, 19 December 2006.

[31] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of
reasoning in very expressive description logics. In Leslie Pack
Kaelbling and Alessandro Saffiotti, editors, IJCAI, pages 466–471.
Professional Book Center, 2005.

[32] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele
Nardi, and Peter F. Patel-Schneider, editors. The Description Logic

[33] Ian Horrocks Zhe Wu Achille Fokoue Carsten Lutz (eds.) Boris Motik,
Bernardo Cuena Grau. Owl 2 web ontology language, profiles, 22
September 2008.

[34] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo
Rosati, and Guido Vetere. Dl-lite: Practical reasoning for rich dlgs. In
Proc. of the 2004 Description Logic Workshop (DL 2004), volume 104
of CEUR Electronic Workshop Proceedings,


Query answering in the description logic horn-. In Steffen Hilboldt,
Carsten Lutz, and Heinrich Wansing, editors, JELIA, volume 5293 of
