

REWRITING LOGICAL RULES AS *SR**OIQ* AXIOMS:
THEORY AND IMPLEMENTATION

FRANCIS GASSE

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

SEPTEMBER 2008

© FRANCIS GASSE, 2008



Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-45296-7
Our file Notre référence
ISBN: 978-0-494-45296-7

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Abstract

Rewriting Logical Rules as *SROIQ* Axioms: Theory and Implementation

Francis Gasse

Description Logics (DLs) is a family of very expressive logics. But some forms of knowledge are much more intuitive to formulate otherwise, say, as rules. Rules in DLs can be dealt with by two approaches: (i) use rules as they are even though it leads to undecidability. (ii) or make the rules DL-safe, which will restrict their semantic impact and, e.g., loose the nice “car owners are engine owners” inference. Here, we offer a third possibility: we rewrite the rule, if it satisfies certain restrictions, inherited from *SROIQ*, into a set of axioms which preserves the nice inferences. This rewriting also has the benefit of being compatible with existing tools and standards, and to make the most of this, software supporting this approach is provided.

In this thesis, we describe the rewriting technique and prove that it does really preserve the semantics of the rules and also present the software we developed to support the adoption of the proposed technique.

Acknowledgments

This thesis is now a reality thanks to the support I received throughout my academic career. Here are the persons who made it possible.

First and foremost, I'd like to thank my supervisor, Dr Volker Haarslev, for his constructive comments, his time, the opportunities he gave me and the support he gave me for my visit to University of Manchester. This visit allowed me to meet Dr Uli Sattler who has given me much of her time, judicious advice and cheers while I was there. I also have to thank my fellow students, Jiewen, Nasim and Jocelyne, for the pleasant and constructive discussions we had. I have to thank two persons for their support during my undergraduate years, Tony Garneau and Dr Sylvain Delisle.

Last but not least, the most heartfelt thanks to very important persons, my family: my parents, my brother and my daughter Mia, to whom this thesis is dedicated.

C'est pour toi ma belle....♡♡♡

Contents

List of Figures	viii
1 Introduction	1
1.1 Description Logics	3
1.2 Rules	6
1.3 DL and Rule Integration	7
1.4 Research Objectives	8
1.5 Structure of the thesis	9
2 Preliminaries	11
2.1 <i>SROIQ</i>	11
2.2 Web Ontology Language (OWL) 2	15
2.2.1 New Features	16
2.2.2 Semantics	16
2.2.3 Nonstructural Restrictions on Axioms	17
2.3 SWRL	17
2.3.1 Semantics	19
3 Related Work	20
3.1 CARIN	20
3.2 Description Logic Programs (DLPs)	21

3.3	Hybrid MKNF	23
3.4	Datalog-based Approaches	24
3.4.1	$\mathcal{DL}+log$	24
3.4.2	DL-Safety	25
3.5	Using a FOL Theorem Prover for OWL	26
3.6	Other Relevant Work	28
3.6.1	Fire	28
3.6.2	Integration with Tableaux Rules	28
3.7	Overall State of the Art	29
4	DL Rules and their Rewriting	31
4.1	DL Rules	31
4.2	DL Rules as Graph	33
4.2.1	Deriving a Graph from a Rule	34
4.2.2	Notation	34
4.2.3	Skeletonization	35
4.3	Rewriting Technique	37
4.3.1	Admissible Rules	38
4.3.2	Folding a Rule Graph	39
4.3.3	Rule Insertion	40
4.3.4	Generalization to Rule-bases	42
4.4	Walk-through Example	43
4.5	Correctness of the Rewriting	46
5	Prototype Implementation	50
5.1	Architecture of the Common Framework	51
5.1.1	Rule Syntax	51
5.1.2	Persistence	52

5.1.3	Other Tasks	55
5.2	Command-line Program	55
5.2.1	Invocation	56
5.2.2	Report	57
5.3	Protégé plug-in	57
5.3.1	Protégé	58
5.3.2	Overview of the Interface	59
5.3.3	Opening and Saving Operations	66
5.3.4	Design Problem	66
5.3.5	DLRule vs. Other Rule Editors	66
5.4	Practical Reasoning Results	67
6	Conclusion	71
6.1	Summary	71
6.2	Contribution	72
6.2.1	Publications	73
6.2.2	Contribution versus Objectives	74
6.3	Future Work	75
6.3.1	<i>SROIQ</i> ⁺	75
6.3.2	Cover More Rules with Description Graphs	76
A	The University Ontology	78
	Bibliography	85

List of Figures

1	Overview of DL constructs.	3
2	Syntax of SWRL rules.	18
3	Overview of the relation of DLP with other formalisms.	22
4	Mapping from DL to FOL.	27
5	Graph visualization of a rule and its associated RIAs.	36
6	Graphical view of the example's rule.	44
7	The graph of the rule's body with the labels removed.	45
8	Axioms generated by the rewriting and added to the ontology.	46
9	The valid syntax of DL rules in EBNF notation	52
10	Example of an inserted rule with its annotations	54
11	An example of report generated by the command-line application.	58
12	Overview of the DLRule plug-in.	60
13	The Rule List component.	61
14	The text tab of the Rule Pane.	62
15	The OWL tab of the Rule Pane.	63
16	The Graph tab of the Rule Pane.	64
17	The Class and Property pop-ups from the Graph tab.	64
18	The Rule Application Previewer component.	65
19	Graph of the realization time of KBs containing rules.	69
20	Graph of the realization time of KBs not containing rules.	69

21	Statistics of the University Ontology.	78
----	--	----

Chapter 1

Introduction

A few decades back, the massive introduction of the computer greatly improved our capacity to process and stock information. When these computers were networked, it created a platform with the potential to revolutionize the way information was communicated. However, it was not until the introduction of the World Wide Web (Web) that this potential was fully realized and that people outside the corporate and governmental spheres could use it. The Web was successful because it allows typical users to easily navigate and retrieve the information they needed, it masked the complexity of the process. The Web is now, by far, the biggest repository of information on Earth. It contains so much information that it is humanly impossible to efficiently navigate it to harvest the bits we need. Indeed, the time needed to retrieve the interesting bits of information from the available data grows with the size of this same data.

We have computers that can process data much faster than we can, so if a computer could “understand” the information on the Web and identify which parts we need and treat them, it would be much better. Even though specialized systems doing that type of tasks exist for certain precise areas, a versatile framework would optimize the way data can be used and offer endless opportunities. That is, in a nutshell, the promise of what the *Semantic Web* [SBLH06] aims to do, make the information on the Web machine understandable. In order to do that, we need a standard formal representation (in the same way we needed HTML) expressive enough to model any kind

of knowledge and still be understandable by a machine. Researchers in the field of knowledge representation (KR) designed various logics for that purpose, with varying degree of expressivity and complexity. The World Wide Web Consortium (W3C), the governing body of the Internet, has published a recommendation for a language that fulfills these requirements, it is the Web Ontology Language (OWL) [BvHH⁺04]. It is based on *SHOIN*, a logic that is member of a family of logics, mainly a subset of the first-order logic (FOL), called Description Logics (DLs) [BCM⁺03] which we will further introduce in the next section. The main characteristics of this group of logics are that it is very expressive, it is decidable and it has robust computational properties. These features have driven its adoption in many fields that need to formalize their knowledge, most notably in life sciences, to use OWL to build their ontologies. The successor to-be of OWL, OWL 2, is currently being developed by a working group¹ mandated by the W3C, and it is based on another, more expressive, DL called *SROIQ* [HKS06].

Even though modern DLs are very expressive, there are still some forms of knowledge that they cannot represent. So researchers continuously work to add new constructs and relax restrictions on DLs to maximize their expressive power and retain the logic's decidability. Function-free Horn rules are another widely used form of knowledge representation that can roughly be seen as *if... then....* or *.... imply....* statements. These two logics are rather expressive and are orthogonal to each other. There is a consensus in the community that combining rules with DL would be a nice way to provide the extra expressivity needed for some problems. The problem is that it has been shown in [HPS04] that a naive combination of the two is undecidable. This is the essence of the problem which motivated the research presented in this thesis.

We will now introduce more precisely the two formalisms we are working with so that we can fully describe the problem and state our objectives.

¹http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

Name	Syntax	Semantics	Symbol
Top	\top	$\Delta^{\mathcal{I}}$	\mathcal{AL}
Bottom	\perp	\emptyset	\mathcal{AL}
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	\mathcal{AL}
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	\mathcal{U}
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	\mathcal{C}
Value restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b. (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	\mathcal{AL}
Existential quant.	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b. (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	\mathcal{E}
Unqualified number restriction	$\geq n R$ $\leq n R$ $= n R$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} = n\}$	\mathcal{N}
Qualified number restriction	$\geq n R.C$ $\leq n R.C$ $= n R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} = n\}$	\mathcal{Q}
Role-value- map	$R \subseteq S$ $R = S$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b. (a, b) \in R^{\mathcal{I}} \rightarrow (a, b) \in S^{\mathcal{I}}\}$ $\{a \in \Delta^{\mathcal{I}} \mid \forall b. (a, b) \in R^{\mathcal{I}} \leftrightarrow (a, b) \in S^{\mathcal{I}}\}$	
Agreement and disagreement	$u_1 \doteq u_2$ $u_1 \neq u_2$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b \in \Delta^{\mathcal{I}}. u_1^{\mathcal{I}}(a) = b = u_2^{\mathcal{I}}(a)\}$ $\{a \in \Delta^{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta^{\mathcal{I}}. u_1^{\mathcal{I}}(a) = b_1 \neq b_2 = u_2^{\mathcal{I}}(a)\}$	\mathcal{F}
Nominal	I	$I^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ with $ I^{\mathcal{I}} = 1$	\mathcal{O}

Figure 1: Overview of DL constructs.

1.1 Description Logics

Description Logics are a family of logics, closely related to propositional dynamic [Har84] and modal [HC96] logics, that are designed to represent terminological knowledge and that are meant to be practically usable in knowledge representation (KR) systems. For that reason, a strong emphasis is put on keeping the reasoning problems decidable. It is to be noted that all DLs make the Open-World assumption, i.e., the failure to derive a fact from a knowledge base does not imply its opposite.

For now we introduce one of the simplest DL, its syntax and its semantic. We use the DL \mathcal{AL} [SSS91] (\mathcal{AL} stands for “attributive language”) for this purpose because the majority of the other DLs are obtained by augmenting it with other constructs. Each features (or set of features) has a

letter assigned to it (see Figure 1), so a logic is named from the concatenation of the symbols of its features, for example \mathcal{ALU} is \mathcal{AL} augmented with concept union.

The base of a description are the atomic concepts and role names, which are mutually disjoint sets. A description is inductively built from them together with the constructors.

Let us now introduce some conventions for the notation that will be used throughout this document. The letters A and B refer to atomic concepts, C and D refer to concept descriptions. Roles are represented by R . With respect to that notation, the following are also concepts in \mathcal{AL} : \top , \perp , $\neg A$, $C \sqcap D$, $\forall R.C$, $\exists R.\top$.

Here are some examples of what we can express with these constructors to give some intuition before we introduce the formal semantics. Let **Student** and **Worker** be atomic concepts and **hasFriend** be an atomic role. Then the expressions **Student** \sqcap **Worker** and **Student** \sqcap \neg **Worker** are \mathcal{AL} concept expressions which respectively describe students that are also workers, and students that are not workers. Using the role constructs, we can form these concept expressions: **Student** \sqcap \forall **hasFriend**.**Worker** and **Student** \sqcap \exists **hasFriend**. \top , which describes students that only have workers as friends and students that have a friend respectively. Using the bottom concept, one can also describe a student without any friends with the following expression: **Student** \sqcap \forall **hasFriend**. \perp .

The formal definition of \mathcal{AL} 's semantics requires the notion of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consists of a non-empty set $\Delta^{\mathcal{I}}$, the domain of \mathcal{I} , and an interpretation function $\cdot^{\mathcal{I}}$ that maps every atomic concept name A to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and every role name R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. To interpret concept expressions, they are inductively reduced to a simple form using the following definitions:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\perp^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\exists R. \top)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}}\} \\
(\forall R. C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}
\end{aligned}$$

A *general concept inclusion axiom* (GCI) is an expression of the form $C \sqsubseteq D$ for two concepts C and D . A TBox \mathcal{T} is a set of GCIs. An *individual assertion* is of one of the following forms: $a : C$, $(a, b) : R$, or $a \neq b$, for a, b individual names, a (possibly inverse) role R and a concept C . An ABox \mathcal{A} is a set of individual assertions.

An interpretation \mathcal{I} is a model of a TBox \mathcal{T} (written $\mathcal{I} \models \mathcal{T}$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each GCI $C \sqsubseteq D$ in \mathcal{T} . A concept C is called *satisfiable* if there exists an interpretation \mathcal{I} with $C^{\mathcal{I}} \neq \emptyset$. A concept D *subsumes* a concept C (written $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models of \mathcal{T} . Two concepts are *equivalent* (written $C \equiv D$) if they are mutually subsuming each other. The above inference problems can be defined w.r.t. a TBox \mathcal{T} by only considering interpretations that are models of \mathcal{T} . For an interpretation \mathcal{I} , an element $x \in \Delta^{\mathcal{I}}$ is called an *instance* of a concept C if $x \in C^{\mathcal{I}}$. An interpretation \mathcal{I} *satisfies* (is a model of) an ABox \mathcal{A} ($\mathcal{I} \models \mathcal{A}$) if for all individual assertions $\phi \in \mathcal{A}$ we have $\mathcal{I} \models \phi$, where

$$\begin{aligned}
\mathcal{I} \models a : C &\quad \text{if } a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \models a \neq b &\quad \text{if } a^{\mathcal{I}} \neq b^{\mathcal{I}} \\
\mathcal{I} \models (a, b) : R &\quad \text{if } \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}} \\
\mathcal{I} \models (a, b) : \neg R &\quad \text{if } \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \notin R^{\mathcal{I}} \text{ or, equivalently, } \langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}
\end{aligned}$$

An ABox \mathcal{A} is *consistent* with respect to a TBox \mathcal{T} if there is a model \mathcal{I} for \mathcal{T} such that $\mathcal{I} \models \mathcal{A}$.

1.2 Rules

In the scope of rules and DL integration, a particular fragment of the first-order logic seems to be a perfect match, i.e. function-free Horn rules. Indeed, Horn clauses' expressivity can capture some very common form of knowledge that are not representable in DL and this fragment of FOL is tractable. For now, we will introduce regular Horn rules and then only specify how each proposition differs from this common base when we introduce each technique.

Horn rules are in fact Horn clauses rewritten as an implication formula, both forms being equivalent. A Horn clause is a disjunction of literals containing at most one positive literal, the general form being $\neg a_1 \vee \dots \vee \neg a_{n-1} \vee a_n$. The atoms (a_1, \dots, a_n) have different forms depending with which logic the clause is used. This clause rewritten as an implication becomes $a_1 \wedge \dots \wedge a_{n-1} \rightarrow a_n$. The implication form being much more used and natural in our context, it will be used in this document. One must keep in mind that the contrapositive of a Horn rule must also hold, i.e., given a rule $C(x) \rightarrow D(x)$, $\neg D(x) \rightarrow \neg C(x)$ must also hold. The part of the rule that is left of the \rightarrow is the body of the rule, also called antecedent, and the right part is the head (consequent). In the context of description logics, there are two possible forms of atoms: the concept atom ($C(x_1)$) and the role atom ($R(x_1, x_2)$), for C a concept name, R a role name and x_1, x_2 variables. Variables in a Horn clause are thought of as universally quantified. In particular, any variables in a clause body that do not occur in the clause head can be thought of as existentially quantified in the body. The variable names have to be distinct from the the union of roles and concepts names. In \mathcal{AL} , the terms (x_i) can only be variable names but more expressive DLs, like \mathcal{SROIQ} that we introduce in Section 2.1, can also allow other constructs. Given a rule r , we use $head(r) = a_n$ for its head, $body(r) = \{a_1 \dots a_{n-1}\}$ for its body and $Var(r)$ for the variables it uses.

In our context, we only consider rules with exactly one atom in the head and at least one atom in the body. Since Horn rules have by definition at most one positive literal, hence one atom in the head, this only leaves out rules with an empty head and/or empty body. We do not consider empty-headed rules because they are highly similar to queries, which is out of the scope of this work, and the empty-bodied rules are facts, so they should be added directly to the KB.

1.3 DL and Rule Integration

As we have said, the expressivity and/or the modeling potential we could gain by integrating rules with DLs is very desirable. In order to make the motivations clear, here are some examples that even very expressive DLs cannot correctly model.

- Let *hasSon* and *hasChild* be two roles such that $hasSon \sqsubseteq hasChild$ and *Man* is an atomic concept. DLs cannot represent the fact that an *hasChild* with a *Man* for object imply a *hasSon* relationship, but rules can: $hasChild(x, y) \wedge Man(y) \rightarrow hasSon(x, y)$.
- Let *connectedToNode* be a role and *Node* be a concept. We cannot model the notion of a cycle, say a concept representing *Nodes* that is part of a cycle of length 2. Once again a rule can, as follows: $connectedToNode(x, y) \wedge connectedToNode(y, x) \rightarrow PartOfCycle(x)$.

These examples show two types of structures commonly encountered in practice that DLs by themselves cannot deal with and thus illustrate the reasons that drives this research.

In [HPS04], it was shown that determining the consistency of a *SHOIN* (OWL-DL) KB, and evidently any KBs of more expressive logics, w.r.t. a set of function-free Horn rules Γ , i.e., $KB \models \Gamma$, is an undecidable problem. Obviously, it follows that every other reasoning problem is also undecidable. In this section, we will show where the undecidability arises to fully grasp the problem. To achieve that, we will observe some characteristics of the two formalisms and see how their interaction induces undecidability.

First, consider the following concept expression: $Person \sqsubseteq \exists hasParent. Person$. It is trivial to see that this expression can be expanded into an infinite chain of *Person* linked by *hasParent*. This simple axiom illustrates why we have to be careful with the reasoning process in order to ensure its termination. Most DLs can guarantee termination and correctness by only considering certain forms of models, namely models that are either tree or forest-shaped. Models of these shapes have the particularity that if a model is infinite, there is a way to “collapse” it into a finite model. Furthermore, the collapsing of such models is also computationally simple.

Horn rules, on the other hand, ensure termination in a totally different way. Indeed, all variables in these rules are universally quantified. It should be clear that such rules will thus never require the creation of “new” individuals. It follows that the number of individuals is finite and that alone ensures the termination of reasoning.

So where DLs restricts the shape of infinite models, Horn rules allow arbitrary but finite models. That gives some intuition regarding the cause of the undecidability of a logic combining function-free Horn rules and DLs. Indeed, a rule can break the tree-like shape of the models thus prevent the blocking of the model’s generation and jeopardize the termination of reasoning.

1.4 Research Objectives

Considering the problem described above, we will investigate different combinations of rules and description logics in order to discover the combination offering optimal expressivity w.r.t. both formalisms while still being decidable. This type of work being very common in KR, there already are excellent results in that area, but they are all either preventing the rules from changing the terminological knowledge (via DL-safety) or undecidable. For example, let us assume a concept C , an individual a asserted to be a C ($a : C$) and the rule $C(x) \rightarrow D(x)$. In a DL-safe context we can derive that x is also a D , but we cannot derive that $C \sqsubseteq D$ even though it is obviously the case. Note that this example can obviously be encoded directly in DL and avoid this problem, but more complex examples could not, so it does not affect our argument.

This type of inference is highly interesting and no existing combination with an expressive DL allows it. This is the problem we intend to solve with our research. Our research should cover the whole area, from the theoretical definition to the implementation of tools so that users can easily leverage the gained expressivity for their purpose.

On the theoretical level, we will identify the description logic and form of rules that combines with optimal expressivity, define the semantics of the combination, prove the correctness of the formalism and its decidability.

On the practical side, we will develop the tools required to support the creation of a rule-enhanced KB and the reasoning over such a KB in a transparent way for the user. We will also conduct experiments to ensure that reasoning over that logic is sufficiently efficient to be of practical use. That has to be verified because, even though we only consider decidable combinations, the combined complexity might render reasoning intractable in some cases.

1.5 Structure of the thesis

Let us now present the structure of this thesis's content. In Chapter 2 we introduce some preliminary notions which are required to understand the thesis. Namely, we will introduce a very expressive DL, *SROIQ*, which we use in our rewriting, the proposed Web Ontology Language (OWL) 2 and the Semantic Web Rule Language (SWRL), two languages used to write ontologies and rules in the context of the Semantic Web. Both these languages are used by our prototype implementation.

In Chapter 3, we present some notable work that also integrated rules and DLs to some extent. First we present CARIN, an approach based on an hybrid reasoning process. Next we introduce Hybrid Minimum Knowledge Negation-as-Failure (MKNF) knowledge bases which combine DL and logic programming, yielding many interesting features but also an undecidable formalism. We then present a summary of many approaches that reduce DL to Disjunctive Datalog, thus allowing the addition of Datalog rules. Lastly we introduce two other results that are somewhat less significant but interesting nonetheless, the Fire system which applies rules to completions of DL KBs using pattern-matching and an approach that extended DL tableau algorithms to handle rule reasoning. We conclude by analyzing these results to have a complete picture of the current state of the art.

In Chapter 4 we present a class of rules applicable in the context of DL and a technique to rewrite them into DL axioms. We first define these DL rules and provide a graph-based conceptualization for them. Using this conceptualization, we define this rewriting. We then prove the

correctness of the rewriting and show how it works by completing a step by step example.

In Chapter 5 we introduce the implementation we made of the rewriting in two applications. First we present the common framework on which both applications are built. Then we present the command line application, its usage and its output. Next we present the graphical application that enables the editing of ontologies with rules and its features. We conclude by showing the practical usability of our approach using experimental results.

In Chapter 6 we conclude this thesis by summarizing our work, showing its contribution and showing some future research avenues in the field.

Chapter 2

Preliminaries

2.1 *SROIQ*

We now introduce a very expressive description logic called *SROIQ* that was introduced in [HKS06]. This DL incorporates some very useful features that have been requested by the Semantic Web community and for which practical and efficient reasoning are available. Most of these features add expressivity to roles, somehow balancing what we can express about roles w.r.t. what we can say about concepts. These qualities led to the selection of *SROIQ* to be the logical foundation of OWL 2, the upcoming successor of OWL. We will now introduce this DL and define its syntax and semantics. Since *SROIQ* is an extension of the DL \mathcal{AL} we presented in Section 1.1, everything that holds in \mathcal{AL} holds in *SROIQ* so we will only cover constructs not present in \mathcal{AL} .

Let \mathbf{C} be a set of concept names including a subset \mathbf{N} of nominals¹, \mathbf{R} a set of role names including the universal role U , and $\mathbf{I} = a, b, c, \dots$ a set of individual names. The set of roles is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$, where a role R^- is called the inverse role of R .

The notion of an interpretation is extended with nominals, the universal role U and inverse roles. Let C be a nominal, thus also a concept, the interpretation of C , $C^{\mathcal{I}}$, is then a singleton. The

¹A nominal is an concept interpreted as a singleton

universal role U is interpreted as a total relation $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, it clearly follows that $(U^-)^{\mathcal{I}} = (U)^{\mathcal{I}}$. Inverse roles are interpreted in such a way that for each role R , we have

$$(R^-)^{\mathcal{I}} = \{\langle y, x \rangle \mid \langle x, y \rangle \in \mathcal{R}^{\mathcal{I}}\}.$$

To avoid considering roles such as R^{--} , we define a function Inv on roles such that $\text{Inv}(R) = R^-$ if R is a role name, and $\text{Inv}(R) = S$ if $R = S^-$. We will have to deal with strings of roles, so we extend the valuation $\cdot^{\mathcal{I}}$ and the function Inv to such strings: if $w = R_1 \dots R_n$ for R_i roles, then $w^{\mathcal{I}} = R_1^{\mathcal{I}} \circ \dots \circ R_n^{\mathcal{I}}$, where \circ denotes the composition of binary relations, and if $\text{Inv}(w) = \text{Inv}(R_n) \dots \text{Inv}(R_1)$.

A *role box* ($R\text{Box}$) is a set of axioms describing roles and the relations between them. An $R\text{Box}$ \mathcal{R} consists of two components. First, there is a *role hierarchy* \mathcal{R}_h which consists of *role inclusion axioms* (RIAs). These axioms are of the form $w \sqsubseteq R$ where w is a string of roles restricted to certain shapes that we will define shortly. The second component, \mathcal{R}_a , is a set of *role assertions* that state certain characteristics of roles. For instance, the assertion $\text{Irr}(R)$ states that the role R must be interpreted as an irreflexive relation.

To define the possible shapes of the role string w in a RIA $w \sqsubseteq R$, we first have to define a *regular* ordering over roles. A strict partial order \prec on a set M is a binary relation over M that is both irreflexive and transitive. An ordering \prec on a set of roles, and their inverses, is regular if it additionally satisfies that $S \prec R$ implies that $S^- \prec R$ and reversly for all roles R and S .

Definition 1 ((Regular) Role Inclusion Axioms). *An interpretation \mathcal{I} satisfies a role inclusion axiom $S_1 \dots S_n \sqsubseteq R$, if $S_1^{\mathcal{I}} \circ \dots \circ S_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$. An interpretation \mathcal{I} is a model of a role hierarchy \mathcal{R}_h , if it satisfies all RIAs in \mathcal{R}_h , written $\mathcal{I} \models \mathcal{R}_h$. A RIA $w \sqsubseteq R$ is \prec -regular, for \prec a regular ordering over roles and R is a role name, if w is of one of the following forms:*

1. $w = RR$, or
2. $w = R^-$, or
3. $w = S_1 \dots S_n$ and $S_i \prec R$, for all $1 \leq i \leq n$, or

4. $w = RS_1 \dots S_n$ and $S_i \prec R$, for all $1 \leq i \leq n$, or

5. $w = S_1 \dots S_n R$ and $S_i \prec R$, for all $1 \leq i \leq n$.

A role hierarchy \mathcal{R}_h is regular if there exists a regular order \prec over the roles such that each RIA is \prec -regular w.r.t. that same \prec .

The restriction to regularity of the role hierarchy ensures that there are no cyclic dependencies such as $RS \sqsubseteq T$ and $TS \sqsubseteq R$ for instance, since it would imply $R \prec R$ which would violate the irreflexivity of \prec . This restriction is necessary to retain decidability as it was shown in [HS04] that such dependencies would induce undecidability.

Given a role hierarchy \mathcal{R}_h , we define the relation \sqsubseteq to be the transitive-reflexive closure of \sqsubseteq over the GCIs of the form $\{R \sqsubseteq S \text{ and } \text{Inv}(R) \sqsubseteq \text{Inv}(S) \text{ in } \mathcal{R}_h\}$. A role R is a sub-role (resp. super-role) of role S if $R \sqsubseteq S$ (resp. $S \sqsubseteq R$). Two roles R and S are equivalent ($R \equiv S$) if \mathcal{R}_h implies both $R \sqsubseteq S$ and $S \sqsubseteq R$.

We introduce the set *Diag*, which is interpreted, w.r.t. \mathcal{I} , as $\{\langle x, x \rangle \mid x \in \Delta^{\mathcal{I}}\}$ and is required for the definition of the second part of an RBox, the set of role assertions \mathcal{R}_a . Since the interpretation of the universal role U is fixed, it cannot be used in axioms that are elements of \mathcal{R}_a .

Definition 2 (Role Assertions). For roles $R, S \neq U$, we call the assertions $\text{Ref}(R)$, $\text{Irr}(R)$, $\text{Sym}(R)$, $\text{Tra}(R)$, and $\text{Dis}(R, S)$, role assertions, where, for each interpretation \mathcal{I} and all $x, y, z \in \Delta^{\mathcal{I}}$, we have:

$$\begin{aligned} \mathcal{I} \models \text{Sym}(R) & \quad \text{if } \langle x, y \rangle \in R^{\mathcal{I}} \text{ implies } \langle y, x \rangle \in R^{\mathcal{I}}; \\ \mathcal{I} \models \text{Tra}(R) & \quad \text{if } \langle x, y \rangle \in R^{\mathcal{I}} \text{ and } \langle y, z \rangle \in R^{\mathcal{I}} \text{ imply } \langle x, z \rangle \in R^{\mathcal{I}}; \\ \mathcal{I} \models \text{Ref}(R) & \quad \text{if } \text{Diag}^{\mathcal{I}} \subseteq R^{\mathcal{I}}; \\ \mathcal{I} \models \text{Irr}(R) & \quad \text{if } R^{\mathcal{I}} \cap \text{Diag}^{\mathcal{I}} = \emptyset; \\ \mathcal{I} \models \text{Dis}(R, S) & \quad \text{if } R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset. \end{aligned}$$

Roles that are implied by the composition of any roles are called *non-simple*. We define the set of simple roles as follows:

- a role name is simple if it does not occur on the right hand side of a RIA in \mathcal{R}_h , and

- an inverse role R^- is simple if R is, and
- if R occurs on the right hand side of a RIA in \mathcal{R}_h , then R is simple if, for each $w \sqsubseteq R \in \mathcal{R}_h$, $w = S$ for a simple role S .

Definition 3 (RBox). *The set $\mathcal{R} = \mathcal{R}_h \cup \mathcal{R}_a$ is a valid $SR\mathcal{OIQ}$ RBox if \mathcal{R}_h is a regular role hierarchy and \mathcal{R}_a is a finite, simple set of role assertions. We say that an interpretation \mathcal{I} satisfies an RBox \mathcal{R} , written $\mathcal{I} \models \mathcal{R}$, if $\mathcal{I} \models \mathcal{R}_h$ and $\mathcal{I} \models \phi$ for all role assertions $\phi \in \mathcal{R}_a$. If \mathcal{I} satisfies \mathcal{R} , it is called a model of \mathcal{R} .*

We wish to highlight the fact that the restriction to a regular role hierarchy implies the prohibition of equivalent roles, i.e. $R \equiv S$ or $R \equiv S^-$, since it is impossible to find a *strict* ordering over a set containing equivalent elements. This is a purely syntactic restriction since we can just replace all occurrences of one of the roles with the other to eliminate the equivalence and not lose any information. Let us now define the syntax and semantics of $SR\mathcal{OIQ}$ -concepts.

Definition 4 ($SR\mathcal{OIQ}$ Concepts, TBoxes, and ABoxes). *The set of $SR\mathcal{OIQ}$ -concepts is the smallest set such that*

- every concept name (including nominals) and \top, \perp are concepts, and,
- if C, D are concepts, R is a role (possibly inverse), S is a simple role (possibly inverse), and n is a non-negative integer, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\exists S.Self$, $(\geq nS.C)$, and $(\leq nS.C)$ are also concepts.

A general concept inclusion axiom (GCI) is an expression of the form $C \sqsubseteq D$ for two $SR\mathcal{OIQ}$ -concepts C and D . A TBox \mathcal{T} is a finite set of GCIs. An individual assertion is of one of the following forms: $a : C$, $(a, b) : R$, $(a, b) : \neg S$, or $a \neq b$, for $a, b \in \mathbf{I}$ (the set of individual names), a (possibly inverse) role R , a (possibly inverse) simple role S , and a $SR\mathcal{OIQ}$ -concept C . A $SR\mathcal{OIQ}$ -ABox \mathcal{A} is a finite set of individual assertions.

In order to preserve decidability, only simple roles are allowed in number restrictions, ($\geq nS.C$) and ($\leq nS.C$), Self-restrictions, $\exists S.Self$, and disjointness and irreflexivity assertions, $Dis(R, S)$ and $Irr(R)$.

Definition 5 (Semantics). *Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, concepts C, D , roles R, S , and non-negative integers n , the extension of complex concepts is defined inductively by the following equations, where $\#M$ denotes the cardinality of a set M , and concept names, roles, and nominals are interpreted as usual:*

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, & \perp^{\mathcal{I}} &= \emptyset, & (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & \text{(Booleans)} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, & (C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} & \text{(Booleans)} \\
(\exists R.C)^{\mathcal{I}} &= \{x \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} & \text{(exists restriction)} \\
(\exists R.Self)^{\mathcal{I}} &= \{x \mid \langle x, x \rangle \in R^{\mathcal{I}}\} & \text{(\exists R.Self-concepts)} \\
(\forall R.C)^{\mathcal{I}} &= \{x \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\} & \text{(value restriction)} \\
(\geq nR.C)^{\mathcal{I}} &= \{x \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\} & \text{(atleast restriction)} \\
(\leq nR.C)^{\mathcal{I}} &= \{x \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\} & \text{(atmost restriction)}
\end{aligned}$$

2.2 Web Ontology Language (OWL) 2

As we mentioned in the introduction, the Semantic Web is one of the major motivations that drives the research in our field. Description Logics alone were not very useful in that context, a language that could be deployed over the Web was necessary. Hence the XML based languages OWL and OWL 2, which are both relying on DL for their underlying logic. The rewriting we propose requires the expressivity of *SR_QIQ*, which OWL 2 is based on, so we will use this version. We do not really use OWL in the theoretical part of this work, since *SR_QIQ* itself is more usable, but we use it throughout our prototype. In that regard, we will now introduce the OWL 2 ontology language. OWL 2 is an extension of OWL that adds many new constructs, some of which significantly increase expressivity while maintaining backward compatibility with OWL. We will

now overview these novelties, then introduce the semantics of OWL 2 and we finish by exposing the non-structural restrictions that OWL 2 must satisfy.

2.2.1 New Features

The new features can be split into three categories: syntactic improvements, expressive power and datatype support. There are two new syntactic constructs that can simplify writing knowledge. First, `DisjointUnion` allows to say in a single axiom that a description is the union of a set of descriptions that are mutually disjoint. The other new feature is the ability to assert the non-existence of a property using two new predicates `NegativeObjectPropertyAssertion` and `NegativeDataPropertyAssertion`.

The choice of *SROIQ* as the underlying logic increased the expressivity by allowing the addition of these new DL constructs: (i) qualified cardinality restrictions (`ex.:ObjectMinCardinality (2 friendOf hacker)`), (ii) local reflexivity restrictions for simple properties (`ex.:ObjectExist Self (likes)` for narcissists), (iii) reflexive, irreflexive, symmetric, and asymmetric properties for simple properties (`ex.: ReflexiveObjectProperty (knows)`), (iv) disjoint properties for simple properties (`ex.: DisjointObjectProperties (childOf spouseOf)`), and (v) property chain inclusion axioms (`ex.: SubObjectPropertyOf (SubObjectPropertyChain (ownspart) owns)`) if the property inclusions are acyclic.

The support of datatypes is vastly improved by allowing the use of almost all of the XML Schema Datatypes². For instance, we can define new datatypes restrictions that can be used in the ontology, as in `SubClassOf (Adult DataSomeValuesFrom (age DatatypeRestriction (xsd:integerminInclusive "18"^^xsd:integer))`.

2.2.2 Semantics

We now introduce the semantics of OWL 2, which are closely related to those of *SROIQ*. The semantics of all OWL 2 constructs that are present in *SROIQ* are a straightforward mapping to

²see <http://www.w3.org/TR/xmlschema-2/> for more information

the semantics of *SRQIQ*. The constructs of OWL 2 that are not included in *SRQIQ* are all those related to datatype support. The semantics of these datatype predicates are easily mappable to those of the equivalent object property, obviously taking into account that the domain of interpretation is the data domain instead of the interpretation domain. For instance, the correspondance between `ObjectSomeValuesFrom` and `DataSomeValuesFrom` is as follows: both define the possible object of their subject property with the difference that `ObjectSomeValuesFrom` accepts concept expressions and `DataSomeValuesFrom` accepts datatype expressions.

2.2.3 Nonstructural Restrictions on Axioms

OWL 2 introduces a set of nonstructural restrictions on axioms. These restrictions are being enforced to ensure that OWL 2 is contained within *SRQIQ*'s expressivity and are equivalent to *SRQIQ*'s definition of a regular role hierarchy. Given that equivalence, we will not cover these restrictions but mention that they should nonetheless be observed.

2.3 SWRL

The Semantic Web Rule Language (SWRL) [HPSB⁺04] is a W3C proposal for a SW oriented rule language. Being oriented towards the Semantic Web, it was designed to extend OWL with the expressive power of the rules, not the other way around. The rule component of SWRL is largely based on RuleML³, another rule language now mainly used for business rules. We can thus define SWRL as the union of OWL and RuleML which is closely related to Unary/Binary Datalog and function-free Horn rules. SWRL is not that important for the theoretical part of this work, but it is used in our implementation and is the de facto standard, much like OWL. SWRL retains the full expressive power of both OWL and RuleML, but this expressivity comes at a hefty price, the loss of decidability.

³<http://www.ruleml.org>

```

axiom ::= rule
      rule ::= 'Implies('[URIreference]annotationantecedentconsequent)'
antecedent ::= 'Antecedent('atom')'
consequent ::= 'Consequent('atom')'

```

Figure 2: Syntax of SWRL rules.

The undecidability of SWRL can be shown by using it to encode a known undecidable problem. We will use *role-value maps* [SSS91] for that purpose. Let R, S be role chains (i.e. $R_1 \circ \dots \circ R_n$), $R \subseteq S$ and $R = S$ are respectively a containment role-value map and an equality role-value map. The following examples informally show their semantics, the concept $Person \sqcap (hasChild \circ hasFriend \subseteq knows)$ describes the persons knowing all the friends of their children, and $Person \sqcap (marriedTo \circ likesToEat = likesToEat)$ describes persons having the same favorite foods as their spouse. These role-value maps are a well-known undecidable problem and SWRL rules can be used to simulate them, thus showing SWRL's undecidability.

SWRL extends OWL by adding the rules as a type of axiom in a way defined using the abstract syntax in Figure 2. It also extends OWL by including a set of built-in functions that allows one to deal with complex relations and operations over datatypes. The relation functions are mostly equivalent with data ranges from OWL, but the built-in operations provide interesting functions. For example, math built-ins offer modulo, string built-ins offer comparison ignoring case and list built-ins offer evaluation of a list's length. These allow for rather complex rules to be written.

A rule axiom consists of an antecedent (body) and a consequent (head), each of which consists of a (possibly empty) set of atoms. A rule axiom can also be assigned a URI reference, which could serve to identify the rule. Informally, a rule may be read as meaning that if the antecedent holds (is "true"), then the consequent must also hold. An empty antecedent is treated as trivially holding (true), and an empty consequent is treated as trivially not holding (false). Rules with an empty antecedent can thus be used to provide unconditional facts; however such unconditional facts are better stated in OWL itself, i.e., without the use of the rule construct. Non-empty antecedents and consequents hold iff all of their constituent atoms hold, i.e., they are treated as conjunctions of their

atoms. Rules with conjunctive consequents can easily be split up into multiple rules each with an atomic consequent.

Atoms can be of the form $C(x)$, $R(x, y)$, $sameAs(x, y)$ $differentFrom(x, y)$, or $builtIn(r, x, \dots)$ where C is an OWL description or data range, R is an OWL property, r is a built-in relation, x and y are either variables, OWL individuals or OWL data values, as appropriate. Informally, an atom $C(x)$ holds if x is an instance of the class description or data range C , an atom $R(x, y)$ holds if x is related to y by property R , an atom $sameAs(x, y)$ holds if x is interpreted as the same object as y , an atom $differentFrom(x, y)$ holds if x and y are interpreted as different objects, and $builtIn(r, x, \dots)$ holds if the built-in relation r holds on the interpretations of the arguments. Note that $sameAs$ and $differentFrom$ can already be expressed using the combined power of OWL and rules without explicit (in)equality atoms.

Atoms may refer to individuals, data literals, individual variables or data variables. Variables are treated as universally quantified, with their scope limited to a given rule. As usual, only variables that occur in the antecedent of a rule may occur in the consequent (a condition usually referred to as "safety"). This safety condition does not, in fact, restrict the expressive power of the language (because existentials can already be captured using OWL `someValuesFrom` restrictions).

2.3.1 Semantics

The semantics for SWRL is a straightforward extension of the semantics of OWL in which we define bindings, i.e. mappings from elements of OWL interpretations to variables of the rules. We say that a rule is satisfied iff every binding that satisfies the antecedent also satisfies the consequent. The semantic conditions relating to axioms and ontologies are unchanged. In particular, an interpretation satisfies an ontology iff it satisfies every axiom (obviously including rule axioms) and facts in the ontology, and an ontology is consistent iff it is satisfied by at least one interpretation.

Chapter 3

Related Work

The expressivity we can gain by combining Description Logics and rules is very interesting and required for some application, so there have been many efforts in that direction. Among these efforts, many different paradigms were investigated by different groups with varying degrees of success. In this chapter we intend to present the most significative paradigms and the results they obtained, to better understand the challenge we aim to tackle.

3.1 CARIN

CARIN [LR96] is a KR formalism that provides an integration of Datalog with the DL \mathcal{ALCNR} , and any of its sublogics. It extends the \mathcal{AL} -log [DLNS98] language by integrating more expressive DLs (\mathcal{AL} -log is built around the \mathcal{ALC} DL) and by admitting more general DL atoms in the rules. The form of rules is heavily restricted since only ordinary predicates (predicates referring to entities not part of the DL terminology) are allowed to appear in the head of rules. Unfortunately, this formalism is undecidable but refutation-complete, i.e. it terminates if there is an answer for a given query but may not terminate otherwise. A query answering procedure is provided and decidable subsets are identified in [LR98]. These subsets are obtained by restricting to non-recursive rules or by restricting the way variables can be used in role atoms. For both these decidable subsets, a sound and complete reasoning algorithm is provided. These algorithms split the reasoning into

two steps, the DL reasoning and the rule reasoning. Recursive rules are defined as follows. An ordinary predicate p depends on an ordinary predicate q if q appears in the antecedant of a rule which consequent is p . A set of rules is recursive if there is a cycle of dependencies between the ordinary predicates. Let $R(x, y)$ be a role atom and x, y be variables, this restriction over the variables requires that either x or y appears in a non-DL body atom in the same fashion as in DL-safe rules (see 3.4.2 for details).

First in the DL reasoning step, the DL component of a given knowledge base is used to construct a set of completions, each of which is represented by a finite set of DL atoms and determines a canonical model of a program. A completion is an interpretation modeling a knowledge base for which all possible inference is performed. Then the construction of an rule augmented component occurs. This component is composed of all the rules augmented by the assertions contained in each completion of the aforementioned canonical model. For the rule reasoning step, a forward-chaining based reasoning is carried out over the completion included in the canonical model using the added DL-assertions as new facts. An ordinary predicate atom is entailed by the knowledge base if and only if it is entailed by all augmented rule components of the canonical model.

3.2 Description Logic Programs (DLPs)

In this section we introduce an approach to rule and DL integration that differs from most others in the way that it focuses on the intersection of the two formalisms and not on some form of union. More precisely, Description Logic Programs (DLPs) [GHVD03] correspond to the intersection of the DL underlying OWL (*SHOIN*) and the Declarative Logic Programs fragment of FOL, i.e. Horn clauses. Figure 3 (taken from [GHVD03]) shows how this formalism relates to other logics. The main advantage of such a combination is obviously a reduced complexity that ensures that the reasoning services are efficient and scalable. Such services could be provided by highly effective logic programming reasoners.

Although DLPs are an intersection of two formalisms, they still provide a significant expressive

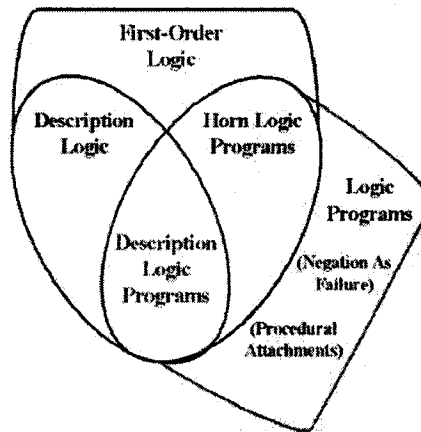


Figure 3: Overview of the relation of DLP with other formalisms.

power. Notably, it includes the expressivity required to state that class D is a Subclass of a class E, that the Domain of a property P is a class C, that the Range of a property P is a class C, that a property P is a Subproperty of a property Q, that an individual b is an Instance of a class C and stating that a pair of individuals (a,b) is an instance of a property P. These constructs are equivalent to the expressivity of RDF(S). Additionally it allows one to express conjunction and disjunction in class descriptions, that a property is transitive, symmetric or the inverse of another.

With this formalism, a bidirectional mapping is provided to and from the two formalisms used in the intersection for knowledge bases and queries. This translation enables two different views of the integration, either rules are added to the ontology or the ontology augments the rules. In the first one, the rules use the vocabulary defined in the ontology, whereas the second one imports the rules' vocabulary into the DL.

There are however restrictions imposed implicitly in the expressivity to ensure that the KB remains in the intersection. The tree model property of DLs, closely related to the correspondance of DLs to FOL with two variables, is what mostly restricts the way rules can be used. This prevents the expression of classes where two different paths of properties lead to a given variable. An example for such a case would be a class "HomeWorker" representing individuals who live and

work at the same location defined as

$$work(x, y) \wedge live(x, z) \wedge loc(y, w) \wedge loc(z, w) \rightarrow HomeWorker(x)$$

Also, Horn logic requires all variables to be universally quantified, thus preventing the assertion of an anonymous individual, i.e. any DL axiom of the form $C \sqsubseteq R.T$. Also, Horn rules cannot contain any negated atoms.

3.3 Hybrid MKNF

Since OWL has been widely used for ontology construction, many considered Logic Programming (LP) as the solution to expressive shortcomings that had been identified, such as exceptions, constraints and, of course, rules. A proposition [MHRS06] has been made for a technique that seamlessly integrates OWL (but also expressive DLs, up to *SR_QIQ*) with LP. It is based on the Minimal Knowledge Negation-as-Failure (MKNF) logic, presented in [Lif03], which can be seen as a modal logic with default models supplying the non-monotonic features. Contrarily to DLPs' intersection, here a union of the two paradigms is proposed. However, this logic is known to be undecidable, a major problem in the DL context.

Having the open-world semantics as a central assumption of DLs, it seemed incompatible with the closed-world semantics of LP. The problem is settled by way of an epistemic operator (the **K** operator) which means, informally, “is known to hold” meaning that it is true in every interpretation. This operator allows us to switch between open and closed world at will, which is interesting since even in a DL context it is sometimes desirable to query the world as-is. A negation-as-failure **not** operator is added as well to allow to specify that “it is possible for α not to hold”. This approach has many benefits that enables one to express n-ary predicates, integrity constraints exceptions in the model and allows for reasoning with the closed-world assumption when desired by using LP constructs over a DL terminology. Rules can also be formulated using this formalism and they are defined using the epistemic operators.

3.4 Datalog-based Approaches

There have been various techniques integrating DL and rules reported that have as a common base the use of Disjunctive Datalog [EGM97] as the formalism used for the reasoning tasks. Such techniques include AL-log [DLNS98], Description Logic Programs [GHVD03] and Horn-SHIQ [HMS05]. Since all these approaches are so closely related, we will focus on the most interesting results using this technique, those of [Ros05]. This work defines a general framework for the integration of DL ontologies and Datalog^{¬v}, meaning a Datalog program with negation as failure and disjunction. This work generalizes the previous results because it proves that any decidable first-order KB (thus any decidable DL KB) augmented with atomic and DL-safe Datalog^{¬v} rules preserves its decidability. In order to create a Datalog program equivalent to the KB and the rules, the KB is first translated into Datalog using basic superposition and then the rules are added to the resulting program (this step is straightforward). This approach is implemented in the KAON2 system. Unfortunately this system is limited to the *SHIQ* logic thus limiting its practical usability since it is less expressive than OWL-DL.

3.4.1 *DL+log*

[Ros06] introduces *DL+log* which we consider apart from the other Datalog-based approaches because it allows a tighter integration of DL and rules, even though currently it is at the expense of the expressivity of the DL, a problem which should be addressed in future work, by not requiring the DL-safeness of the rules. Instead, rules must have what the author calls *weak safeness*. Weak safeness means that variables are not required to appear in a non-DL atom (usually referred to as $\mathcal{O}(x)$) if they do not appear in the head of the rule. Decidability of this approach was demonstrated via the correspondence between reasoning in *DL+log* and conjunctive query containment (checking whether the result of a conjunctive query is a subset of the result of another query for every TBox) in DLs. As mentioned earlier, there is a price to pay for this benefit: currently, decidability of this approach is only proved in *D_{CR}* which lacks prominent features of modern DLs

such as nominals, transitive roles and role inclusion axioms on arbitrary roles. This logic is in fact equivalent to \mathcal{ALCQI} since we can reify the \mathcal{DLR} n-ary relations.

3.4.2 DL-Safety

DL-safety [MSS05] is a restriction over the semantics of the rules that some integration techniques require to retain decidability. It is not to be confused with the classical restriction to safe-rules, which is a restriction over the syntax of the rules, i.e. that variables referenced in the head of the rule are required to be referenced in the body.

As we mentioned, this does not restrict the syntax of the rules because any rule can be transformed into a DL-safe one, but the semantics of the rule is affected. First, we will show how to transform a rule into a DL-safe one and then we will show how and why the semantics are affected.

The method to make a rule DL-safe is rather simple and goes as follows. Let K be a knowledge base containing rule r , to make r DL-safe we add to it an atom of the form $\mathcal{O}(x)$ for every variable x it references. Afterward, we add an assertion of the form $\mathcal{O}(x)$ for every individual x contained in K . We use \mathcal{O} as the non-DL predicate in accordance to a notational convention observed by previous papers on the subject. The addition of the $\mathcal{O}(x)$ atoms and assertions have an obvious effect, only individuals explicitly introduced in the knowledge base can be bound to variables of a rule. That prevents individuals introduced by the application of expansion rules, most notably by existential restrictions, from being bound to variables. This implies that variables can only be bound to a finite number of individuals, hence ensuring termination. Let us now formally define what is a DL-safe rule.

Definition 6 (DL-Safe Rules). *A rule r is DL-safe if each variable occurring in r also occurs in a non-DL atom in the body of r .*

The restriction to DL-safety implies that such rules have different semantics than regular Horn rules. The major effect of such a restriction is that the rules won't have any effect on the TBox, hence missing inferences like rule induced subsumption. Nonetheless, the gained expressivity is still quite interesting.

3.5 Using a FOL Theorem Prover for OWL

Using FOL theorem provers to reason over DLs is a subject that has been investigated numerous times, notably [HS00]. The most used techniques translated the DL into the guarded fragment of FOL such that the prover provided sound and complete reasoning. These techniques have yet to be extended to allow reasoning over the recent DLs that are expressive enough to be interesting in our context. Another approach is introduced in [TRBH04], it proposes to directly translate the DL into FOL using the mappings presented in Figure 4 (taken from [Mot06]). This approach yields a FO theory for which completeness cannot be guaranteed but that is expressive enough to handle very expressive DLs.

A FOL theorem prover can, by definition, reason over any logic that is a subset of FOL and since both OWL 2 and SWRL are such subsets, their union obviously also resides within FOL. This technique can thus rather easily provide an implementation of a SWRL reasoner. This is exactly the approach the reasoner Hoolet¹ implements. The FOL theorem prover used for this is Vampire [RV01], a very competitive implementation, that has not been optimised specifically for the needs of description logics reasoning. Hoolet translates the KB and the rules into a FO theory which is then fed into the Vampire [RV01] theorem prover where it can be used for reasoning tasks such as satisfiability checking and instance checking.

Although such a system has some virtues, it inherently has some major flaws. The non-completeness of the reasoning implied by FO reasoning being the central one. Decidability is one of the defining features of DLs, so, compromising on that aspect is not ideal but such a compromise has to be done for any technique supporting the whole union of DLs and SWRL. Practical experiments showed that the computation times for DL reasoning tasks with Hoolet were unsurprisingly not competitive at all when compared to those obtained by FaCT++ [TH06]. Also, due to that incompleteness, it failed to correctly compute some of the practical tests it was submitted to, especially with tests involving the verification of non-subsumption. This follows from the fact Vampire is based on refutation. Indeed, a subsumption test between concepts C and D tests if

¹available at <http://owl.man.ac.uk/hoolet/>

Mapping Concepts to FOL	
$\pi_y(\top, X) = \top$	
$\pi_y(\perp, X) = \perp$	
$\pi_y(A, X) = A(X)$	
$\pi_y(\neg C, X) = \neg \pi_y(C, X)$	
$\pi_y(C \sqcap D, X) = \pi_y(C, X) \wedge \pi_y(D, X)$	
$\pi_y(C \sqcup D, X) = \pi_y(C, X) \vee \pi_y(D, X)$	
$\pi_y(\forall R.C, X) = \forall y : R(X, y) \rightarrow \pi_x(C, y)$	
$\pi_y(\exists R.C, X) = \exists y : R(X, y) \wedge \pi_x(C, y)$	
$\pi_y(\leq n R.C, X) = \forall y_1, \dots, y_{n+1} : \bigwedge_{i=1}^{n+1} [R(X, y_i) \wedge \pi_x(C, y_i)] \rightarrow \bigvee_{i=1}^{n+1} \bigvee_{j=i+1}^{n+1} y_i \approx y_j$	
$\pi_y(\geq n R.C, X) = \exists y_1, \dots, y_n : \bigwedge_{i=1}^n [R(X, y_i) \wedge \pi_x(C, y_i)] \wedge \bigwedge_{i=1}^n \bigwedge_{j=i+1}^n y_i \not\approx y_j$	
Mapping Axioms to FOL	
$\pi(C \sqsubseteq D) = \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$	
$\pi(C \equiv D) = \forall x : \pi_y(C, x) \leftrightarrow \pi_y(D, x)$	
$\pi(R \sqsubseteq S) = \forall x, y : R(x, y) \rightarrow S(x, y)$	
$\pi(\text{Trans}(R)) = \forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$	
$\pi(C(a)) = \pi_y(C, a)$	
$\pi(R(a, b)) = R(a, b)$	
$\pi(\neg S(a, b)) = \neg S(a, b)$	
$\pi(a \circ b) = a \circ b \text{ for } \circ \in \{\approx, \not\approx\}$	
Mapping KB to FOL	
$\pi(R) = \forall x, y : R(x, y) \leftrightarrow R^-(y, x)$	
$\pi(KB_{\mathcal{R}}) = \bigwedge_{\alpha \in KB_{\mathcal{R}}} \pi(\alpha) \wedge \bigwedge_{R \in N_{R_a}} \pi(R)$	
$\pi(KB_{\mathcal{T}}) = \bigwedge_{\alpha \in KB_{\mathcal{T}}} \pi(\alpha)$	
$\pi(KB_{\mathcal{A}}) = \bigwedge_{\alpha \in KB_{\mathcal{A}}} \pi(\alpha)$	
$\pi(KB) = \pi(KB_{\mathcal{R}}) \wedge \pi(KB_{\mathcal{T}}) \wedge \pi(KB_{\mathcal{A}})$	
Notes:	
(i):	X is a meta-variable and is substituted by the actual term;
(ii):	π_x is obtained from π_y by simultaneously substituting in the definition all $y_{(i)}$ with $x_{(i)}$, π_y with π_x , and vice versa.

Figure 4: Mapping from DL to FOL.

$C \sqcap \neg D$ is satisfiable, if it is not there is a subsumption. For non-subsumptions, this formula is satisfiable thus troublesome for a refutation-based reasoner.

3.6 Other Relevant Work

We now present two other pieces of work on rule and DL integration that obtained interesting results but that are not as significant as the ones previously presented. First we present the Fire system and then we present a technique that merges the rule reasoning directly into the tableaux rules.

3.6.1 Fire

The Fire [Bho05] system couples a DL reasoner (RacerPro) with a rule engine to enable the use of rules in the OWL ontology. The rules are considered as “trigger rules” as opposed to logical implication. The envisioned mode of functioning, we say envisioned because the reasoners not yet implement the required features but they may in a foreseeable future, is as follows. A “cycle” of reasoning starts by the reasoner computing all the possible completions of the KB and determining a canonical model of the KB. After that, the rule engine applies the rules, the possible bindings being found via the RETE pattern matching algorithm, to all these completions and the intersection of all the inference results are applied to the KB. Such cycles are applied as long as they yield new inferences.

3.6.2 Integration with Tableaux Rules

As we have seen in the above section, most of the work for integrating rules and DL do not rely on tableau algorithms for reasoning. Tableaux are a decision and proof procedure that are implemented by most modern DL reasoners such as FaCT++, RacerPro and Pellet. Extending tableaux to handle rules is thus desirable to enable the use of these highly optimized DL reasoners. Such a technique was introduced in [KPS06]. This paper describes an extension to the *SHOTQ* tableaux

algorithm that enables it to deal with DL-safe rules directly in the tableaux. It is achieved by adding a new completion rule to those originally in *SHOIQ*, this completion rule has the lowest priority. This new rule start by generating all the possible bindings (obviously, it only considers asserted individuals) then it verifies that at least one of the rule atom holds for each binding. If so, the expansion of the branch continues. This result in itself is interesting but a system using this new algorithm was developed to have preliminary results regarding practical usability. For small ontologies, the system performs well but the branching introduced by the reasoning over the rules cause the system's performance to drop rapidly as size increases.

3.7 Overall State of the Art

Each research result we just presented offers a knowledge representation paradigm that has sufficient expressivity to partially model both DL axioms and Horn rules. However they use very different ways to do so, i.e. they are not related by how they function but by their common objective, DL and rules integration. Let us now position our research w.r.t. these related works. The approach we investigate is not linked, does not re-use or build upon any of these technique but nonetheless shares their objectives. We intend to complete this objective using a novel and unique approach, so we presented these related works in order to give an overview of what has been done and the results that we aim to improve.

To clearly identify these results we want to improve, we have to synthetize them, taking into account their respective strenghts and weaknesses, into an observation reflecting the current state of affair. Existing rules and DL integration approaches can be, broadly speaking, split in to groups: undecidable formalisms in which rules can modify the terminology, and decidable ones in which rules cannot alter the terminology. For some application domains, such as life sciences, neither of these options is satisfactory. Let us give a concrete example that illustrates the type of inference that DL-safety disables and that we wish to enable. Let the rule $hasChild(x, y) \wedge hasChild(y, z) \wedge Man(z) \rightarrow hasGrandSon(x, z)$ defined w.r.t. the KB shown below.

$$\exists hasGrandSon.\top \sqsubseteq PersonWithGrandSon$$
$$TestSub \equiv \exists hasChild.\exists hasChild.Man$$
$$jack : (\exists hasChild.\exists hasChild.Man)$$
$$joe : (\exists hasChild.\exists hasChild.billy)$$
$$billy : Man$$

It should be obvious that *joe* is an instance of *TestSub*. Suppose that we have two reasoners, one that requires DL-safety and one that does not. Both reasoners can infer that *joe* is an instance of *PersonWithGrandSon* since both *joe* and *billy* are explicitly introduced individuals. The DL-unsafe reasoner can infer that *jack* is an instance of *PersonWithGrandSon* but the DL-safe cannot since the “grand-son” of *jack* is implicitly introduced by the existential restriction. The DL-safe reasoner also fails to infer that $TestSub \sqsubseteq PersonWithGrandSon$ but the DL-unsafe one does. Such limitations severely restricts the use of rules. Hence our objective to bring a third option, one in which rules can have an effect on the terminology while retaining decidability.

Chapter 4

DL Rules and their Rewriting

We now enter the core of the theoretical content of this thesis, the rewriting of logical rules into DL axioms. We will also introduce some accessory concepts required by the rewriting. In this chapter we will first introduce the notion of DL rules, which is a formalization of logical rules considered in the context of description logics. We then define a graph conceptualization and notation of these rules, which provides an elegant and intuitive way to define the procedures composing the rewriting. Next, we expose the rewriting technique itself and the different steps it requires. The semantic correctness of the operations used in the rewriting and of the rewriting itself are shown in the following section. We conclude by showing how the rewriting as a whole works by processing an example step by step.

4.1 DL Rules

Contrary to the field of logic programming, rules used in a DL context do not have widely acknowledged syntax and semantics. Since we will have to represent rules quite often in this thesis, we will now introduce *DL rules*, which is a class of rules satisfying some syntactic restrictions for which we have well defined semantics and also a representation format. This representation format is based on a graph-based paradigm. We chose the name DL rules to highlight the fact that this approach is not bound to DL-safety. Before introducing the representation format, we will present

the restrictions a logical rule has to satisfy to be a DL rule.

We will define the syntactic restrictions required for DL rules by exposing the differences w.r.t. what is probably the most known class of rules, i.e. Horn rules. The general form of a Horn rule is defined as $a_1 \wedge \dots \wedge a_{n-1} \rightarrow a_n$ with the atoms (a_1, \dots, a_n) are either concept atoms, $C(x_i)$, or role atoms, $R(x_i, x_j)$, for C a concept name, R a role name and x_i, x_j variables. To be a DL rule, such a rule must additionally satisfy the following restrictions:

- The variable(s) referenced in the head of the rule have to be referenced in at least one of the body's atoms. This is the standard safety condition, which is unrelated to DL-safety, that ensures a meaningful interpretation since a variable not used in the body but used in the head could be bound to any entity.
- The set of variable names has to be disjoint from the other names used in the ontology, i.e. the union of the sets of names of individuals, concepts and roles.
- We also require rules to be connected. A rule r is connected if, for any pair of variables x, y in r , there exists a sequence x_1, \dots, x_n such that (i) $x_1 = x$, (ii) $x_n = y$ and (iii) for $1 \leq i < n$ there is a role R such that either $R(x_i, x_{i+1})$ or $\text{Inv}(R)(x_{i+1}, x_i)$ is an atom of r .

A rule minimally has to be a valid DL rule for it to be eligible to be rewritten. That implies that rules not satisfying these restrictions are not of any interest in our context. For that reason, we only consider DL rules for the remaining part of this thesis. Knowing that, we consider that the word “rule” implicitly means “DL rule” in this context. Let us now define the semantics of a DL rule w.r.t. an interpretation modelling a DL knowledge base.

Definition 7 (Semantics of DL Rules). *Let \mathcal{I} be an interpretation that is a model of a KB \mathcal{K} , C be a concept of \mathcal{K} , R be a role in \mathcal{K} and π be a mapping $\text{Var}(r) \rightarrow \Delta^{\mathcal{I}}$, i.e. a mapping from the variables of a rule r to individuals in the domain of the interpretation. We write:*

- $\mathcal{I} \models^{\pi} C(x)$ if $\pi(x) \in C^{\mathcal{I}}$, and
- $\mathcal{I} \models^{\pi} R(x, y)$ if $(\pi(x), \pi(y)) \in R^{\mathcal{I}}$.

For a set of atoms $A = \{A_1, \dots, A_n\}$, we write $\mathcal{I} \models^\pi A$ if $\mathcal{I} \models^\pi A_i$ for all A_i in A . Let r be a rule of the form $A \rightarrow H$, where A is a set of atoms and H is an atom. We say that \mathcal{I} satisfies r if, for all π , $\mathcal{I} \models^\pi A$ implies that $\mathcal{I} \models^\pi H$.

4.2 DL Rules as Graph

The abstract syntax of DL rules adequately represents the rules. However, this syntax is not particularly suited to be used to describe transformations on the rules as our rewriting requires. Indeed, when reading the definition of a transformation over this syntax the semantic effect might not be obvious because it is lost in syntactic considerations. That is because the syntax itself carries little semantics. To solve this problem, we introduce a graph conceptualization of rules and a new notation based on that conceptualization which we will use to define the rewriting. We also introduce a simplification technique for rule graphs, called the *skeletonization*.

Thinking of rules as graphs has many advantages, it is intuitive, easy to visualise and it allows a viable notation. For these reasons, we now introduce the notion of a *rule graph*, that is a graph conceptualization of DL rules. A rule graph is a directed labeled graph, which models the body of the rule, augmented with the head of the rule.

A rule graph G is defined by the following expression $G = \langle V, E, \mathcal{L}, H \rangle$ where :

- V is a finite set of vertices, and
- $E \subseteq V \times \mathbf{R} \times V$ is a ternary relation describing the edges, and
- \mathcal{L} is a set of labels, and
- H is the head of the rule.

It should be noted that in the definition of E , the symbol \mathbf{R} is used to represent the set of roles of the KB. Also, a label \mathcal{L}_x of a vertex is a (possibly empty) set of concept expressions that are used in concept atoms over variable x .

4.2.1 Deriving a Graph from a Rule

We now describe the procedure to derive a rule graph from a rule. Let r be a rule defined as $a_1 \wedge \dots \wedge a_n \rightarrow a_h$. We start by defining the rule graph $G_r = \langle V, E, \mathcal{L}, H \rangle$ as an empty rule graph, i.e. $V = \emptyset$ and $H = \emptyset$, and then for each $a_i \in r$, we do the following:

- if a_i is of the form $C(x)$ then
 - if $x \notin V$ then $V = V \cup \{x\}$
 - $\mathcal{L}_x = \mathcal{L}_x \cup \{C\}$
- if a_i is of the form $R(x, x)$ then
 - if $x \notin V$ then $V = V \cup \{x\}$
 - $\mathcal{L}_x = \mathcal{L}_x \cup \{\exists R.Self\}$
- if a_i is of the form $R(x, y)$ then
 - if $x \notin V$ then $V = V \cup \{x\}$
 - if $y \notin V$ then $V = V \cup \{y\}$
 - $E = E \cup \{\langle x, R, y \rangle\}$

The final step is to add the head of the rule, a_h to the graph which simply consists of asserting $H = a_h$.

4.2.2 Notation

The notation we use for rule graphs is very similar to that abstract structure we have just defined, the difference being that we have to further define functions and relations. The notation of a graph G is defined as $G = \langle V, E, L_V, H \rangle$ where

- V is a set of vertices, written $\{x, y, \dots\}$

- E is a set of edges, written $\{R_{x,y}, \dots\}$
- L_V is a set of vertices' labels, written $\{C_x, \dots\}$ where C_x is the label of vertex x
- H is an assertion, written either $x : C$ or $x, y : R$.

4.2.3 Skeletonization

We now introduce the skeletonization which is a simplification technique for rule graphs. This technique aims to simplify the graph by removing edges which can be removed without altering the satisfiability of the rule's body. Let us first define the result of this simplification, the skeleton of the rule graph.

Definition 8. *A graph is a skeleton with respect to a role hierarchy \mathcal{R}_h if there are no edges in E that are implied by other edges in E . We say that an edge $R_{x,y}$ in a graph G is implied w.r.t. a role hierarchy \mathcal{R}_h if, for all interpretations \mathcal{I} , models of \mathcal{R}_h , we have that $\mathcal{I} \models G \setminus \{R_{x,y}\}$ implies $\mathcal{I} \models G$.*

Let us now give some intuition for the definition of an implied edge. As usual, we are only interested in interpretations that model \mathcal{R}_h because if $\mathcal{I} \not\models \mathcal{R}_h$ (hence it is inconsistent) it follows that $\mathcal{I} \not\models G$ for any rule graph G . The idea is that if, for all \mathcal{I} , when \mathcal{I} models $G \setminus \{R_{x,y}\}$, it also models G , then the edge $R_{x,y}$ does not further constrain G , hence it is implied by other edges of G .

We introduce this concept of skeleton because by removing implied edges from G , we maximize the possibilities that G will satisfy the restrictions and be admissible to the rewriting (see next section). We now introduce a new relation, $\overline{\in}$, to simplify the writing of some expressions. The $\overline{\in}$ relation is an extension of the \in relation in such a way that it covers the inverse elements. Let R be a role and x, y be variables. $R_{x,y} \overline{\in} E$ if $R_{x,y} \in E$ or $Inv(R)_{y,x} \in E$. Let G be a rule graph, x, y and z be elements of V_G and R and S be roles. A skeleton of G is obtained by applying the following rules, in that order, to G until none is applicable anymore:

1. Let a RIA of the form $R_1 \circ R_2 \dots R_n \sqsubseteq R$ be in \mathcal{R} , if $R_{x_1, x_2}, \dots, R_{x_n, x_{n+1}} \overline{\in} E$ and $R_{x_1, x_{n+1}} \overline{\in} E$, then remove $R_{x_1, x_{n+1}}$ from E .

2. Let R be a transitive role w.r.t. \mathcal{R} . If $R_{x,y}, R_{y,z}$ and $R_{x,z} \in E$, then remove $R_{x,z}$ from E .
3. Let S be the inverse role of R w.r.t. \mathcal{R} . If $R_{x,y} \in E$ and $S_{y,x} \in E$, then remove $R_{x,y}$.
4. Let R be a super-role of S w.r.t. \mathcal{R} . If $R_{x,y} \in E, S_{x,y} \in E$ and $R \neq S$ then remove $R_{x,y}$ from E .

In order to ensure the removal of the maximal number of edges, we have to apply the rules in a way that respects a certain ordering of roles. We have seen in Section 2.1 that *SR_QIQ* requires the existence of a certain ordering, \prec , over the set of roles. Let T be the set of roles ordered using the inverse of \prec , i.e. $T_{i+1} \prec T_i$, for each T_i , starting with T_1 , we map the role R in each rule with the role T_i and if there exists a mapping for all roles in a rule, this rule is then applied. It is trivial to see that a graph and its skeleton are equivalent since we only remove the edges from the rule graph if they are implied by the remaining ones according to the semantics of equivalent, inverse, transitive roles and RIAs in *SR_QIQ*.

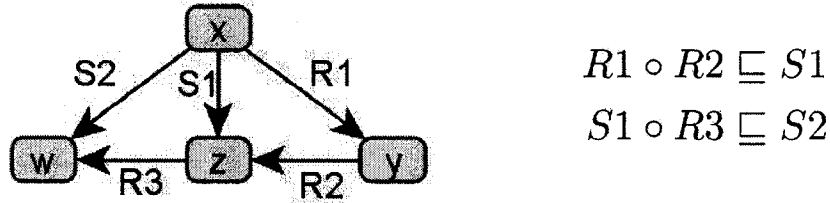


Figure 5: Graph visualization of a rule and its associated RIAs.

We now present an example that will give some intuition about the ordering of roles for skeletonization and also demonstrate its necessity. In Figure 5 we have a graph visualization of the body of a rule and two RIAs as part of the associated role hierarchy. To skeletonize it, we could first remove the S_1 edge (from the R_1 and R_2 edges) but then we could not further simplify it and the graph would still contain a cycle, i.e. it would remain not rewritable. On the other hand, if we start by removing the S_2 edge (which is what the ordering forces us to do), it is still possible to remove the S_1 edge afterward. The resulting graph is a skeleton and acyclic, thus rewritable.

This shows why we skeletonize the rule graphs and why we have to observe a certain order when skeletonizing.

We assume in the definitions of the next section that rule graphs are skeletonized and that the vertices' labels are simplified. The simplification of a label $\mathcal{L} = \{C_1, \dots, C_n\}$, where C_i are concept expressions, w.r.t. a TBox \mathcal{T} is done by applying the following rules until none is applicable anymore:

- if $\mathcal{T} \models \{ \prod_{C_i \in \mathcal{L}} C_i \equiv \top \}$ then $\mathcal{L} = \emptyset$
- if $\mathcal{T} \models \{ C_i \sqsubseteq C_j \}$ then $\mathcal{L} = \mathcal{L} \setminus C_j$

This simplification ensures that labels are expressed in the simplest way possible to avoid unnecessary reasoning. The first rule clears the label of a node if the conjunction of all its components is equivalent to \top , i.e. always true. The second one removes concept expressions that subsume another expression since the subsumer is implied by the subsumee. We now introduce the nucleus of a rule graph, which is used in the rewriting technique.

Definition 9. *The nucleus is a set of nodes that we define differently for concept and role headed rules. For concept-headed rule graphs, the nucleus is only the vertex referenced in the head. As for role-headed rule graphs, it is defined as the path between the two vertices referenced in the head.*

Let us illustrate that notion with two examples. The nucleus of the concept-headed rule $R(x, y) \wedge D(y) \wedge R(y, z) \rightarrow C(x)$ is defined as $\{x\}$ and the nucleus of the role-headed rules $R(x, y) \wedge R(y, z) \wedge C(x) \wedge D(y) \rightarrow T(x, z)$ has the nucleus $\{x, y, z\}$.

4.3 Rewriting Technique

The rewriting technique is composed of two main steps: (i) the folding of the rule graph, in which vertices not required by the head are folded into another, and (ii) the insertion in which the axioms

are generated from the folded graph and inserted in the KB. We describe the details of these operations in the coming sections but we start by defining the criteria a rule must satisfy to be rewritable. After having defined the rewriting for a single rule, we conclude this section by generalizing the rewriting for a rule base.

4.3.1 Admissible Rules

It is an assumption of this research that the complete union of description logics and Horn rules results in an undecidable logic. In our approach, we chose not to semantically restrict the rules so we obviously have to otherwise restrict the combination in order to retain decidability. The restrictions we impose are thus on the form of the rules. The restrictions follow from the encoding of the rules into \mathcal{SROIQ} axioms and since concept and role headed rules are rewritten into different types of axioms, they are subject to different admissibility criteria. We now introduce the class of rules that is admissible to be rewritten, first for concept-headed rules and then for the role-headed rules.

Definition 10. *A rule graph G is concept-headed rule admissible if, when it is considered as an undirected graph, it is a tree, i.e., it does not contain any cycle.*

Definition 11. *Let \prec be an ordering for which the role hierarchy is regular. A rule graph G , with role R in its head and the list of roles l as the roles between the nucleus' vertices, is role-headed rule admissible if:*

1. *It is a tree when considering it as an undirected graph,*
2. *l is of one of following forms:*
 - (a) *$l = RR$ and both vertices of the nucleus have \emptyset as label and all vertices in the rule graph are also in the nucleus, or*
 - (b) *$l = R^-$, or*
 - (c) *$l = S_1, \dots, S_n$ and $S_i \prec R$ for all $1 \leq i \leq n$, or*

- (d) $l = RS_1, \dots, S_n$ and $S_i \prec R$ for all $1 \leq i \leq n$, there are no concept assertions in the first vertex of the nucleus and only one edge comes from this vertex, or
- (e) $l = S_1, \dots, S_n R$ and $S_i \prec R$ for all $1 \leq i \leq n$ and there are no concept assertions in the last vertex of the nucleus and it has no successor in the rule graph, or
- (f) $l = S$ and S is simple.

The restriction for both types of rules to have an acyclic body comes from the fact that *SR_QIQ*, and most other DLs, cannot model cyclic structures with their terminological constructs. This restriction does not imply a loss of generality since cycles can only exist in the ABox, so the application of a cyclic rule over the TBox would not yield any inferences. A DL-safe application of such rules is thus perfectly suited to represent cyclic rules and carry the same semantics. These definitions of admissible rules are complete, meaning that the rewriting of any rule not satisfying these restrictions would yield an invalid *SR_QIQ* KB and that all rules expressible in *SR_QIQ* satisfy these restrictions. The admissibility restrictions for role-headed rules are directly inherited from *SR_QIQ*'s requirement for \prec -regular role hierarchy, which is a certain form of acyclicity in the dependance between roles. This follows from the fact that rules can break this regularity as with the following example. Let R be a sub-role of S , rewriting the rule $S(x, y) \rightarrow R(x, y)$ would break the aforementioned regularity. Indeed, $R \sqsubseteq S$ implies $R \prec S$ and $S(x, y) \rightarrow R(x, y)$ implies $S \prec R$ so it is easy to see that no ordering can satisfy both these constraints.

4.3.2 Folding a Rule Graph

Before we can generate the axioms from a rule graph, we have to reduce it so that is equal to its nucleus. We call this operation the folding of the graph, because vertices are “folded” back into a neighbour. The procedure to fold a graph requires the use of the well known rolling-up technique, so we first define this technique and then the folding procedure.

Vertex Roll-up

The rolling-up technique [Tes01] is a widely known technique in DLs that allows the expression of tree-like structures as concept expressions. Next is the procedure to roll-up a leaf vertex in its ancestor, which we will generalise to the general case. Let x, y be vertices and $R_{x,y}$ be an edge in a rule graph $G = \langle V, E, \mathcal{L}_V, H \rangle$ in which y is a leaf. To roll-up y into x , we execute these two operations:

- if $\mathcal{L}_y \neq \emptyset$ then $\mathcal{L}_x = \mathcal{L}_x \cup \{\exists R. \{ \prod_{C_i \in \mathcal{L}_y} C_i \} \}$
else $\mathcal{L}_x = \mathcal{L}_x \cup \{\exists R. \top\}$
- $V = V \setminus \{y\}$

It is then trivial to show that this can be extended to the rolling-up of any tree-like structure into a given vertex. Indeed, considering that trees with two or more vertices have at least two leaf vertices and that rolling-up a vertex removes it from the graph, we are assured to have leaves to roll-up until the graph has only one vertex left, the desired vertex.

Folding

The goal of the folding of a graph $G = \langle V, E, \mathcal{L}_V, H \rangle$ is to reduce it in such a way that its set of vertices V is equal to its nucleus N , which contains one or many vertices depending on whether the head is a concept or a role. If $V = N$, we do not have to do anything, G is already folded. Otherwise, we select a leaf vertex in V that is not an element of N and roll-it up. This operation is to be repeated until the graph is folded, i.e. $V = N$.

4.3.3 Rule Insertion

Now that the rule graph is reduced to the same vertices as its nucleus, we can use it to generate the axiom(s) required to rewrite it and insert them into the ontology. For both concept or role

headed rules, the generation of the axiom(s) is rather straightforward and obviously so is their insertion. We first define the technique for concept-headed rules and follow with the slightly more complex technique for role-headed rules.

Insertion of a Concept Headed Rule

The nucleus of a concept headed rule graph being a single vertex, and since the rule graph is folded, the rule graphs we have to deal with here are of the form $G = \langle \{x\}, \emptyset, \{\mathcal{L}_x\}, x : C_H \rangle$. In such cases, the rule graph is rewritten as a single axiom, which generation is rather straightforward and highly similar to regular DL concept definition. The axiom is obtained by expanding the following axiom, with respect to the rule graph,

$$\bigcap_{C_i \in \mathcal{L}_x} C_i \sqsubseteq C_H$$

For example, a rule graph $G = \langle \{x\}, \emptyset, \{\mathcal{L}_x = \{C, D\}\}, x : H \rangle$ yields the axiom $C \sqcap D \sqsubseteq H$.

The final step is to insert the generated axiom into the ontology.

Insertion of Role Headed Rule

The generation of the axioms to rewrite these rules is not as simple as the concept headed ones. We usually deal with bigger graphs, of the form $G = \langle \{x_1, \dots, x_n\}, \{R_{x_1, x_2}, \dots, R_{x_{n-1}, x_n}\}, \{\mathcal{L}_{x_1}, \dots, \mathcal{L}_{x_n}\}, x_1, x_n : R_H \rangle$, and we also have to generate and insert more than one axiom.

Before formally defining the technique, we will explain the general idea behind it. Since we have to rewrite the rule as a RIA, which obviously can only contain roles, we first have to rewrite the labels of the nodes. To convert a concept expression to a role expression, we simply add an axiom stating that the label's concept expression implies the existence of a loop edge (using the Self-restriction) labelled with a fresh role name. It follows that all instances of that particular concept expression will have that loop and since we use a fresh role name it also follows that the existence of such loops implies that the individual the role loops over is an instance of the corresponding concept. That part of the technique is not very intuitive, but is highly important

since without this “trick” our rewriting technique could not handle role headed rules, thus making it totally uninteresting. Next we build and insert the RIA. The left hand side is the path from the node that is the subject of the head and the node that is the object. The right hand side is the role of the head. We will show in Section 4.5 that this method, although unintuitive, maintains the semantics of the rule.

The technique is split into three steps defined as :

1. For each $\mathcal{L}_i \in L_V$, we generate an axiom of the form $\prod_{C \in \mathcal{L}_i} C \sqsubseteq \exists newRole_i.Self$ and insert it into the ontology.
2. Let w be a string of roles whose content is built using the following formula: $w = newRole_1 \cup \bigoplus_{i=2:n} R_{x_{i-1}, x_i} \oplus newRole_i$.
3. We just have to insert the axiom, of the form $w \sqsubseteq R_H$, into the ontology.

In the above algorithm, the \oplus symbol stands for the concatenation operator. Let us now summarize this step and then give some intuition on why it works. First, we rewrite the labels of the nodes as loop edges of new roles. The graph is then a only a set of edges in which there is a single path between both nodes used in the head. We build a list of roles by navigating this path and any loops we encounter and then assert that this chain of roles implies the head role. Rewriting concepts as role loops is vital so let us review the general idea. We simply say that being an instance of a given concept implies the existence of a loop of a new role over that instance. Intuitively, every instance has a loop and, since it is a new role, every loop has an instance. The graph is then only a set of edges in which there is a single path between the nodes used in the head so we traverse the graph and build a list of roles that will be asserted to imply the existence of the head role. The upcoming complete example will further illustrate this process.

4.3.4 Generalization to Rule-bases

So far, we defined the rewriting technique w.r.t. to a single rule, obviously in practice we have to deal with sets of rules. The rewriting itself is not affected by the number of rules it has to process,

but we have to review the restrictions for the admissibility of the rules. That follows from the fact that by considering checking one rule at a time, we might miss the effect it can have over each other's regularity and also the effect a combination of rules might have on the regularity of the role hierarchy. For that reason, we will define the criteria for a set of rules to be admissible.

As we have previously mentioned, *SRQIQ* requires the role hierarchy to be \prec -regular. The technique to verify that rewriting a set of rules will not break that regularity is as follows. We start by building a set of inequalities, M , using the role hierarchy and the role headed rules of the set. For all RIAs $R_1, \dots, R_n \sqsubseteq R$ in the role hierarchy \mathcal{R}_h , complete the following operation: $M = M \cup \bigcup_{i=1, \dots, n} \{R_i \prec R\}$. For all role rules, for all edges R_i connecting two vertices of the nucleus, add the inequity $R_i \prec R_H$ to M , where R_H is the role in the head of the rule.

Definition 12. A set of rules S is admissible to be rewritten with respect to a role hierarchy \mathcal{R}_H if:

- all rule graphs in S are acyclic.
- There exists a strict partial regular ordering \prec over the set of inequities M , derived from \mathcal{R}_H and S .

4.4 Walk-through Example

In order to clearly illustrate how the rewriting works, we will now go through every step of the rewriting of a rule. Since the rewriting of concept-headed rules is rather intuitive, and also really close to defining a regular concept, we will use a role-headed rule in this demonstration. The rule we will rewrite is simple yet it is a nice example of the type of rules that could be used in practical applications and induce changes in the terminological knowledge. This rule is defined as follows: $Professor(x) \wedge supervises(x, y) \wedge Student(y) \wedge studiesProgram(y, w) \wedge GraduateProgram(w) \wedge Article(z) \wedge authorOf(y, z) \rightarrow approved(x, z)$. We show the visual graph of this rule in Figure 6.

Before going through the steps of the rewriting itself, we define the rule graph G for our rule.

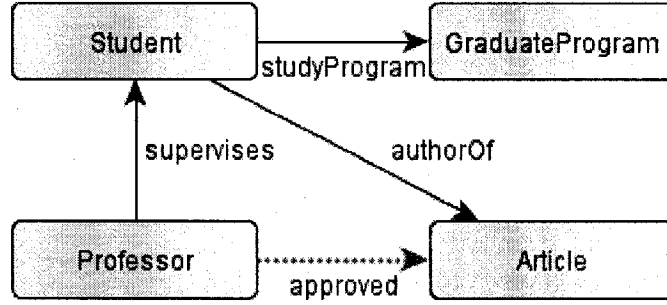


Figure 6: Graphical view of the example's rule.

We do not explicitly go through the step where we derive the graph from the rule because it is rather trivial and it is implicitly shown below.

$$G = \langle \{x, y, w, z\}, \{supervises_{x,y}, studiesProgram_{y,w}, authorOf_{y,z}\}, \\ \{Professor_x, Student_y, GraduateProgram_w, Article_z\}, x, z : approved \rangle$$

The nucleus N of G being defined as $N = \{x, y, z\}$, it follows that G requires to be folded to remove the vertex w . Following the technique, we select vertex w , which is a leaf and not part of the nucleus, to be rolled-up. It is to be rolled-up into its ancestor, the vertex y , and this will augment the label of y with $\exists studiesProgram.GraduateProgram$. The folding stops there because the set of vertices is equal to the nucleus. The rule graph G is now defined as:

$$G = \langle \{x, y, z\}, \{supervises_{x,y}, authorOf_{y,z}\}, \\ \{Professor_x, \{Student, \exists studiesProgram.GraduateProgram\}_y, Article_z\}, \\ x, z : approved \rangle$$

We are now ready to generate the axioms representing the rule and to insert them into the

ontology. There are three steps for the insertion of a role-headed rule graph, for each steps we will recall the operation to be executed and show its output.

1. For each $\mathcal{L}_i \in L_V$, generate $\bigcap_{C \in \mathcal{L}_i} C \sqsubseteq \exists \text{newRole}_i.\text{Self}$ and insert it.
 - Insert $\text{Student} \sqcap \exists \text{studyProgram}.\text{GraduateProgram} \sqsubseteq \exists \text{role}_2.\text{Self}$
 - Insert $\text{Article} \sqsubseteq \exists \text{role}_3.\text{Self}$
 - Insert $\text{Professor} \sqsubseteq \exists \text{role}_1.\text{Self}$
2. Build a string of roles w using the formula: $w = \text{role}_1 \oplus \bigoplus_{i=2:n} R_{x_{i-1}, x_i} \oplus \text{role}_i$.
 - $w = \text{role}_1, \text{supervises}, \text{role}_2, \text{authorOf}, \text{role}_3$
3. Insert an axiom of the form $w \sqsubseteq R_H$ into the ontology.
 - Insert $\text{role}_1 \circ \text{supervises} \circ \text{role}_2 \circ \text{authorOf} \circ \text{role}_3 \sqsubseteq \text{approved}$

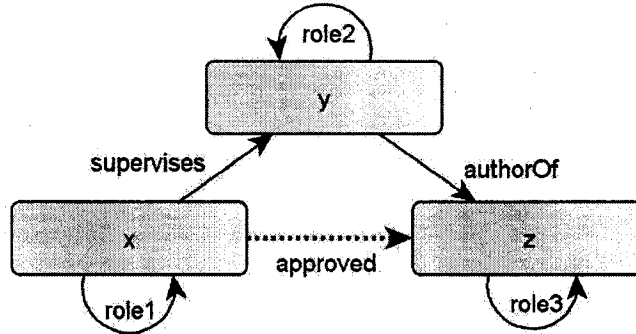


Figure 7: The graph of the rule's body with the labels removed.

Let us now have a closer look at the first step to better illustrate the way it works, which is not intuitive. This step rewrites the labels of the nodes as role restrictions so that the graph is only composed of edges (i.e. roles) which we use to build the RIA. The three axioms we added imply that each instance of the nodes' label has a loop of the corresponding role. The rule graph resulting

$$\begin{aligned}
Student \sqcap \exists studyProgram. GraduateProgram &\sqsubseteq \exists role_1. Self \\
Article &\sqsubseteq \exists role_2. Self \\
Professor &\sqsubseteq \exists role_3. Self \\
role_1 \circ supervises \circ role_2 \circ authorOf \circ role_3 &\sqsubseteq approved
\end{aligned}$$

Figure 8: Axioms generated by the rewriting and added to the ontology.

from that step is shown in Figure 7 and it allows to easily visualize the chain of roles which will imply the head of the role. We summarize the result of the rewriting in Figure 8.

4.5 Correctness of the Rewriting

As we have often pointed out, we consider the semantics of the rules very important in our approach. That is why we will now show the correctness of our rewriting technique, i.e. the preservation of the rules' semantics. We will do so by first considering the different steps on their own and then wrap them up altogether into a theorem. But first let us introduce the notion of an extension of an interpretation which is required to model the KB both before and after a given step since it might requires the addition of axioms into the KB.

Definition 13. \mathcal{I}' is an extension of \mathcal{I} if

1. $\Delta^{\mathcal{I}} = \Delta^{\mathcal{I}'}$
2. if $\mathcal{I} \models \mathcal{K}$ then $\mathcal{I}' \models \mathcal{K}'$, where $\mathcal{K} \subseteq \mathcal{K}'$
3. $C^{\mathcal{I}} = C^{\mathcal{I}'}$ for all concepts C in \mathcal{K}
4. $R^{\mathcal{I}} = R^{\mathcal{I}'}$ for all roles R in \mathcal{K}

This notion of an interpretation's extension is required since the rewriting creates axioms, added to a KB \mathcal{K}' , so we will be working with two KBs and we need to have two interpretations that agree on their common part. For the remaining part of this section when we refer to

π , it is to be a *mapping* from $Var(r) \rightarrow \Delta^{\mathcal{I}}$ where r is a rule. We will abstract the fact we are dealing with rules and consider only the body of the rules. Since we are not modifying the head and folding the graph around the variables referenced in the head, we can do so w.l.o.g. In order to prove the correctness of the rewriting, we will show that the steps composing the folding of a graph are valid and then show that the insertion of a rule maintains its semantics.

Lemma 14 (Rolling-up). *Let \mathcal{I} be an interpretation that models \mathcal{K} , b be the body of a rule and π be a mapping $Var(r) \rightarrow \Delta^{\mathcal{I}}$,*

1. *If $\mathcal{I} \models^{\pi} b$, then there is a rolling-up b_{ru} of b and an interpretation \mathcal{I}' , extension of \mathcal{I} , such that $\mathcal{I}' \models^{\pi} b_{ru}$*
2. *If $\mathcal{I} \models^{\pi} b_{ru}$ for a rolling-up b_{ru} of b , then $\mathcal{I} \models^{\pi} b$.*

Given that this technique is widely known and used, we will not give a proof for it here. The reader may refer to [Tes01] for further information. The correctness of the folding implies that our folding technique is also correct, which we formalize in the following lemma.

Lemma 15. *Let b be the body of a rule. The folding b_f of b is equivalent to b .*

Proof. The operations used in the folding of a rule graph have been shown to yield equivalent graphs. The composition of any of these operations obviously also yields equivalent output. This can be shown as follows. Let f be a function that yields a result equivalent to its input, i.e. $f(x) \equiv x$. It is trivial to show that $f(f(x)) \equiv x$ given that we can replace $f(x)$ by x since they are equivalent. \square

The correctness of folding also implies the correctness of our rewriting over concept-headed rules, considering the close correspondance existing between both technique that we have shown earlier. We thus only have to show that the rewriting of the label of a rule into a self edge is also semantically correct and then build a theorem upon these intermediate results.

Lemma 16 (Label removal). *We use the term “Label Removal” to identify the step where we replace concepts in the label with roles in the role-headed rule insertion algorithm. Let \mathcal{I} be a model of \mathcal{K} , b be the body of a rule and π be a mapping.*

1. *If $\mathcal{I} \models^\pi b$, then there is a label removal b_{lr} of b and an interpretation \mathcal{I}' , extension of \mathcal{I} , such that $\mathcal{I}' \models^\pi b_{lr}$*
2. *If $\mathcal{I} \models^\pi b_{lr}$ for a label removal b_{lr} of b , then $\mathcal{I} \models^\pi b$.*

In this proof, we will show that the rewriting of labels into self edges is consistent by demonstrating that any interpretation that satisfies one of the representations will automatically satisfy the other one. That shows that they are equivalent. For the proof, we will define b as $C(x)$. Note that C could be replaced by any concept expression. Following the label removal we described, \mathcal{K}' would be defined as $\mathcal{K} \cup \{C \sqsubseteq \exists instC.Self\}$ and b_{lr} as $instC(x, x)$. To define \mathcal{I}' , we extend \mathcal{I} with $(instC)^{\mathcal{I}'} = \{(a, a) \in \Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'} \mid a \in C^{\mathcal{I}}\}$.

Proof of (1). Since \mathcal{K}' is only augmented by a RIA with a fresh role name on the right hand side and $\mathcal{I}' \models^\pi \mathcal{K}$ (by definition of an extension), it is trivial to see that $\mathcal{I}' \models^\pi \mathcal{K}'$. To show that $\mathcal{I}' \models^\pi b_{lr}$, we will prove by contradiction, so, we assume that $\mathcal{I}' \not\models^\pi b_{lr}$, i.e., $\mathcal{I}' \not\models^\pi instC(x, x)$. We know that $\mathcal{I}' \models^\pi b$ so $\pi(x) \in C^{\mathcal{I}'}$. The axiom added in \mathcal{K}' gives that $(\pi(x), \pi(x)) \in instC^{\mathcal{I}'}$. The semantics of b_{lr} being $(z, z) \in instC^{\mathcal{I}'}$, we just have to consider $z = \pi(x)$ to have a contradiction. \square

Proof of (2). Since $\mathcal{K} \subseteq \mathcal{K}'$ and $\mathcal{I} \models \mathcal{K}'$, it trivially models \mathcal{K} . We will prove by contradiction for $\mathcal{I} \models^\pi b$, so let's assume that $\mathcal{I} \not\models^\pi b$, i.e. $\mathcal{I} \not\models^\pi C(x)$. Since $\mathcal{I} \models^\pi b_{lr}$, we have $(\pi(x), \pi(x)) \in instC^{\mathcal{I}}$. Since $instC$ is a fresh role name, it can only be introduced by the RIA which means that $\pi(x) \in C^{\mathcal{I}}$, which gives us a contradiction. \square

Let us now give and prove a theorem that combines these results into a coherent proof showing the correctness of our approach.

Theorem 17. *Let r be a rule of the form $b_r \rightarrow h_r$, where b_r is the body and h_r is the head, and a be the axiom rewriting of this rule of the form $a_b \sqsubseteq a_h$. We then have an interpretation \mathcal{I} that models*

\mathcal{K} and satisfies r iff there exists an interpretation \mathcal{I}' , extension of \mathcal{I} , that models \mathcal{K}' and satisfies a . \mathcal{K}' is defined as \mathcal{K} augmented with the axioms generated by the folding of b_r and a .

The proof we use here is not as technical as the previous ones since it is easier to show the correctness of a combination of valid techniques than to prove the correctness directly. Once again we use the bidirectional implication between two constructs to show that they are indeed equivalent and do so by using an extension of the original KB.

Proof. By Lemma 15, we know that b_r is equivalent to a_b . It is trivial to see that the heads are equivalent. For the “if” direction, we have $\mathcal{I} \models r$ and since \mathcal{I}' is an extension of \mathcal{I} , it follows that $\mathcal{I}' \models r$. That means that $\mathcal{I}' \models b_r$, b_r being equivalent to a_b , we also have $\mathcal{I}' \models a_b$. Since $\mathcal{I}' \models \mathcal{K}'$, which contains $a_b \sqsubseteq a_h$, we have that $\mathcal{I}' \models a$.

For the “only if” direction, we have $\mathcal{I}' \models a$, thus $\mathcal{I}' \models a_b$ and by Lemma 15 $\mathcal{I}' \models b_r$. By the definition of an extension and the fact b_r only references roles and concepts in \mathcal{K} , it follows that $\mathcal{I} \models b_r$. The same applies to the head so $\mathcal{I} \models h_r$. That shows that $\mathcal{I} \models b_r$ implies $\mathcal{I} \models h_r$, thus $\mathcal{I} \models r$. \square

This concludes the presentation of the theoretical work included in this thesis. In this chapter particularly, we have presented a class of rules that we could rewrite, we have defined the technique to rewrite and we then proved that the rewritten rules are semantically equivalent to their rewriting, a central point in our approach. The next chapter presents the prototypes we implemented w.r.t. to that rewriting.

Chapter 5

Prototype Implementation

As we previously stated, one of the main advantages of our rule integration approach is that it is fully compatible with existing tools and standards and thus ready to be deployed in real-life scenarios. In order to emphasize that fact and to effectively allow end users to use it, we developed software that fully implements the rewriting technique. It consists of two front-end applications that are built around a common framework that encapsulates all the tasks required to rewrite a rule into DL axioms. The first application is a plug-in for the Protégé¹ [KFNM04] ontology editor that allows to create, modify and remove rules from an ontology. The second one is a command-line application that takes an OWL ontology as input and rewrites all the admissible SWRL rules it contains into DL axioms. Both these applications, and the relevant documentation, are available to download as a single package from http://users.encs.concordia.ca/~f_gasse/.

This chapter will further introduce this software and present some practical results obtained using it. First, we will describe and explain the architecture and features of the common framework as well as the syntax of rules. We will also present some instances of issues that we had to solve and how we solved them. We will then present the two front-ends for this framework, the command-line program and the Protégé plug-in. Considering the user-oriented nature of these components, we will focus on matters of interest for the users, namely, the features, the expected input and output, etc. Finally, we will present the results and conclusions of the tests we conducted to verify

¹<http://protege.stanford.edu>

how reasoners fare with rule-augmented ontologies.

5.1 Architecture of the Common Framework

In this section, we will go over the main tasks that the common framework implements and motivate the design decision that shaped the framework. But first, we shall introduce the syntax we use for rules. This syntax is used as the textual form of rules during processing as well as when editing rules textually in the plug-in.

5.1.1 Rule Syntax

One might wonder why we defined our own syntax when there are options already existing, and why we did not build the syntax around the XML format. We already mentioned that we consider usability as one of the most desirable feature of our approach. From this usability follows the requirement to be able to coexist with other types of rules, so to avoid confusion and the error-prone process of full URI usage, we deemed it better to have a syntax to call our own. The reason why this syntax is not XML based is that the extensibility of XML is not required in this context and nesting an XML rule format into an OWL ontology could lead to unnecessary complications, forced “imports” into host ontologies and a syntax much less readable for humans. These considerations led us to opt for a plain text syntax, which we defined in Figure 9 in a EBNF notation. This syntax is closely related to common human-readable syntax for rules to flatten the learning curve.

In the above grammar, `<className>` and `<propertyName>` refer to classes and properties present in the ontology. We want to point out that if, for any reason, an XML format would become necessary, this syntax is easily rewritable into the SWRL format.

```

<rule> = <atom> [ , { "@" <atom> } ] , ">" , <head>;
<atom> = <propertyAtom> | <classAtom> ;
<head> = <propertyAtom> | <classAtom>;
<propertyAtom> = <propertyName>,
                "(" , <variable> , "," , <variable> , ")" ;
<classAtom> = <className> , "(" , <variable> , ")" ;
<variable> = <letter> [ , { <character> } ] ;
<character> = <letter> | <digit> ;
<letter> = <capitals> | <regular> ;
<capitals> = "A" | "B" | "C" | ... | "X" | "Y" | "Z" ;
<regular> = "a" | "b" | "c" | ... | "x" | "y" | "z" ;
<digit> = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
          "7" | "8" | "9" ;

```

Figure 9: The valid syntax of DL rules in EBNF notation

5.1.2 Persistence

An obviously necessary part of the framework is the input/output package, which has to be able to insert, modify and remove rules from an OWL ontology. In order to minimize the implementation effort, all the rules, denoted by the set R , are removed from the ontology when it is loaded and the (usually modified) set R is inserted back when the ontology is saved. This allows one to handle modifications of a rule using this same code.

Let r be a rule that has to be inserted into an ontology O . r is rewritten into a set of OWL axioms A , each axiom A_i of A has to be inserted into O . When loading the ontology O , since we have to remove the rule from O , we have to remove all the axioms A_i . That implies that we have to be able to identify the axioms we had inserted at the time the ontology was saved. We also have to be able to determine the syntax a rule r had at save time when we load it back so that the user sees the same rules he saved. Since a set of axioms A can be the result of the rewriting of many syntactically different rules, that implies we also have to save the syntax of r into O .

In order to fulfill these requirements, we devised a persistence technique which requires the introduction of two annotation properties in the host ontology. The first one, `dl-rule-axiom`², is

²http://users.encs.concordia.ca/~f_gasse/dl-rule-axiom

used to annotate the axioms related to rules in the ontology, thus identifying them. Since this property is used as a flag, its value is not relevant but our implementation puts the value `addedAxiom` as default. The second one, `dl-rule`³, is used as a non-unique ontology annotation property and its value is the textual form of a rule inserted in that ontology. We will now expose the procedures that save and load ontologies and rules, or more precisely merge an ontology and a set of rules before the saving and separate them after the loading since the actual saving and loading tasks are handled by the OWL API⁴.

Saving

For this task, the input is an ontology and a, possibly empty, set of rules. The task is to merge the set of rules into the ontology, and annotate the relevant axioms so they can be recognized. So for every rule in the set, we check if it is admissible to be rewritten, and:

- If it is, send the rule to the rewriting module and get back a set of axioms. For each axiom, add it to the ontology and annotate it with the property `dl-rule-axiom`. Then annotate the ontology using the property `dl-rule` and set the value as the textual form of the rule.
- If not, convert the rule to a SWRL rule and add the SWRL rule to the ontology.

Loading

The input we have for this task is an ontology that may contain SWRL rules and/or rules rewritten as axioms. The task is to remove the rules of both type from the ontology and add them to a set of rules associated to the ontology. Not surprisingly, this is the mirror of the saving task. First, for all SWRL rules, remove the SWRL axiom from the ontology and add the rule to the set. Second, for all annotations using the property `dl-rule`, parse the value of the annotation into a rule, remove the annotation and add the rule to the set. Finally, for all axioms annotated with the property `dl-rule-axiom`, remove the axiom from the ontology.

³http://users.encs.concordia.ca/~f_gasse/dl-rule

⁴<http://owlapi.sourceforge.net/>

```

<owl:Ontology rdf:about="">
  <cs:dl-rule>
    GradStudent(X1) @ authorOf(X1,X2) &gt; ProlificAuthor(X1)
  </cs:dl-rule>
</owl:Ontology>
[.....]
<rdf:Description>
  <rdf:type rdf:resource="#owl11:Axiom"/>
  <rdf:subject>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="#GradStudent"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#authorOf"/>
          <owl:someValuesFrom rdf:resource="#owl:Thing"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </rdf:subject>
  <rdf:predicate rdf:resource="#rdfs:subClassOf"/>
  <rdf:object rdf:resource="#ProlificAuthor"/>
  <cs:dl-rule-axiom>addedAxiom</cs:dl-rule-axiom>
</rdf:Description>

```

Figure 10: Example of an inserted rule with its annotations

We handle the saving and loading operations this way for two reasons: we can modify rules with no additional code and, more importantly, the end-user is not exposed to the axioms we generate and add to the ontology to replace rules, thus possibly avoiding confusion. The Figure 10 shows the OWL axioms inserted in an ontology to represent the rule `GradStudent(X1) @ authorOf(X1,X2) > ProlificAuthor(X1)`, it also clearly shows how both annotation properties are used.

Note that in this Figure, `cs:` refers to the `http://users.encs.concordia.ca/f_gasse/` namespace.

5.1.3 Other Tasks

Since the Common Framework handles all the tasks that are required by both applications, it obviously implements other tasks than the one we presented so far. Indeed, it also handles, among other things, the admissibility checking, the folding and the rewriting. But since the implementation of these features was nothing else than translating the procedures, we introduced in Chapter 4, into Java, we will not discuss their design.

5.2 Command-line Program

We provided this program to allow rule users who have OWL ontologies containing SWRL rules to enjoy the benefits of our approach without having to modify their ontologies themselves. The program has to be invoked with an OWL file as parameter, which it will search for SWRL rules. For each rule found, it will check if the rule is rewritable into OWL axioms. If it is, the SWRL axiom is removed from the ontology and the OWL axioms and required annotations properties are inserted. Otherwise, the SWRL axiom of that rule is left unchanged.

5.2.1 Invocation

The program provides a set of parameters to modify the way it will execute. The syntax to invoke the program is defined as:

```
DLRuleApp <source> [-t<target>] [-r<report>] [-p] [-i]
```

It is to be noted that as usual, the [] notations indicates optional parameters. Let us now describe the effect and required object of each parameters:

<source> This parameter indicates the file that is to be processed by the program and is the only required parameter. The value should be the physical path of an OWL file. We strongly advise the use of an absolute path enclosed in quotation marks to avoid possible problems with relative paths and spaces in this path. These remarks also apply to the other parameters that are paths, namely <target> and <report>.

-t<target> This parameter allows the specification of a file to store the processed ontology. If specified, the ontology will be written to the file <target>, otherwise the original ontology is overwritten with the processed one.

-r<report> This parameter allows the specification of a file to save the report. If the parameter is present, the report will be saved in the file <report> additionally to being displayed in the console, which is the default.

-p This parameter is used to preview the effect of the execution of the program without actually modifying it. It is to be noted that if this parameter is present, the parameter -t<target> will be overlooked since no modifications are to be carried on.

-i This parameter is used to include in the report the new knowledge about individuals that a rule allowed to infer.

5.2.2 Report

Since the rewriting of a rule modifies its semantics, thus possibly also modifying the architecture of the ontology by implying new subsumption relations for instance, it is a rather important feature to provide the user with some feedback regarding what rules were rewritten. Given that, the program will generate a report giving that very feedback. The report mentions, for every SWRL rules in the ontology, if the rule is rewritable as OWL axioms and, if it is, additionally includes the following informations:

- the OWL axioms added to model this rule, and
- the new subsumption and/or equivalence relations between classes implied by the rule, and
- the new subsumption and/or equivalence relations between properties implied by the rule.

This information will allow the user to ensure that the effects of the rules' rewriting are desired and coherent with the content of the ontology. An example of such a report is presented in Figure 11. In order to keep the reports as readable and concise as possible, we use the functional syntax of OWL, which is arguably the easiest to read and also rather concise, and only show the fragment part of the URI referencing all the entities. One can also add to the reports all the assertions about individuals inferred by rules by using the parameter `-i`. We do not include this in the report by default to avoid generating an endless report, for instance if a rule implies a new subsumption for a class with a 1000 instances. Another parameter, `-p`, cause the application to generate a report to preview the effect it would have while not modifying anything. To avoid confusion, the preview reports are easily recognizable since each line is prefixed with `!PREVIEW!`.

5.3 Protégé plug-in

As we mentioned before, we consider the practical usability of our approach as a very desirable feature. Nowadays, usability implicitly requires the existence of a graphical user interface, so the development of such a front-end was a must to support correctly our approach and favor its

```
#####
RuleID: "diplomaToDrDiploma"
Rule: "hasDiploma(x,y) @ DoctoralDegree(y)
      > hasDoctoralDiploma(x,y) "
Rewritable : NO
#####
RuleID: ""
Rule: "authorOf(x,y)@Article(y)@Conference(z)@acceptedAt(y,z)
      >attends(x,z) "
Rewritable: YES
OWL Axioms: SubClassOf(Article ObjectExistsSelf(role107))
            SubClassOf(Conference ObjectExistsSelf(role108))
            SubObjectPropertyOf(
              SubObjectPropertyChain(authorOf role107
                                      role108 acceptedAt)
              attends)
New Inferences: PublishedAuthor SubClassOf ConferenceAttendee
```

Figure 11: An example of report generated by the command-line application.

adoption. We chose to provide such a front-end in the form of a plug-in for the Protégé ontology editing environment. The plug-in, called “DLRule”, offers all the functionalities required to create and edit rules, save them into an ontology and evaluate their effect on that ontology.

In this section, we will first introduce Protégé and motivate our choice to use this platform. Next, we will describe the plug-in itself, its interface, its main components and some examples of its usage. To conclude this section, we will discuss some design problems we encountered and the solutions we provided as well as a brief overview of how the plug-in compares to other rule editors.

5.3.1 Protégé

Protégé is a feature-rich graphical tool for ontology editing and other knowledge management tasks. It is very widely used in the Semantic Web community by both researchers and end-users, making it a de facto reference in its category. This large user base is a major reason we implemented a plug-in for it since these users are potential testers and adopters for our approach. Other

reasons that motivated our choice are the facts that Protégé is open-source, implemented in Java and provides third-party plug-in developers with an API that facilitates development. The final reason we chose Protégé is that it is developed and supported by academic institutions (Stanford University and the University of Manchester) which favors its long term maintenance.

Since the rewriting of rules is based on features only introduced in OWL 2, we are obviously limited to the use of tools supporting the draft version of the upcoming standard. Such support is only offered in the version 4 of Protégé, a total rewriting of the application which is still in development and only released as beta. Notwithstanding this beta status, the application is surprisingly stable and relatively bug-free so using it is not a source of concerns.

In its basic form, Protégé can edit OWL ontologies graphically, browse the class and property hierarchies, query the ontology, etc. These features can be extended by using the available plug-ins. The reasoning tasks are handled by third-party reasoners. Two reasoners, FaCT++ [TH06] and Pellet [SP06], are bundled with Protégé and they both support OWL2 and are thus compatible with our approach.

5.3.2 Overview of the Interface

The interface of the DLRule plug-in is built within a single panel, which is embedded within a tab in Protégé, along with other plug-ins. It was designed to offer an adequate rule editing environment to both experts and beginners, which we ensured by following the UI design principles presented in [Mar95]. Additionally, we aimed to offer an interface that would be visually and functionally coherent with Protégé. Let us now introduce the general architecture of the plug-in's interface and then further explain the main components and their functionalities. The description of each component will implicitly introduce all the functionalities the plug-in provides.

A general outlook of the plug-in is presented in Figure 12, in which the components are labelled with numbers. We now introduce the name and main function of each component and locate them in the interface using the labeled number.

Rule List (1) This component contains the list of the rules present in the active ontology, whether

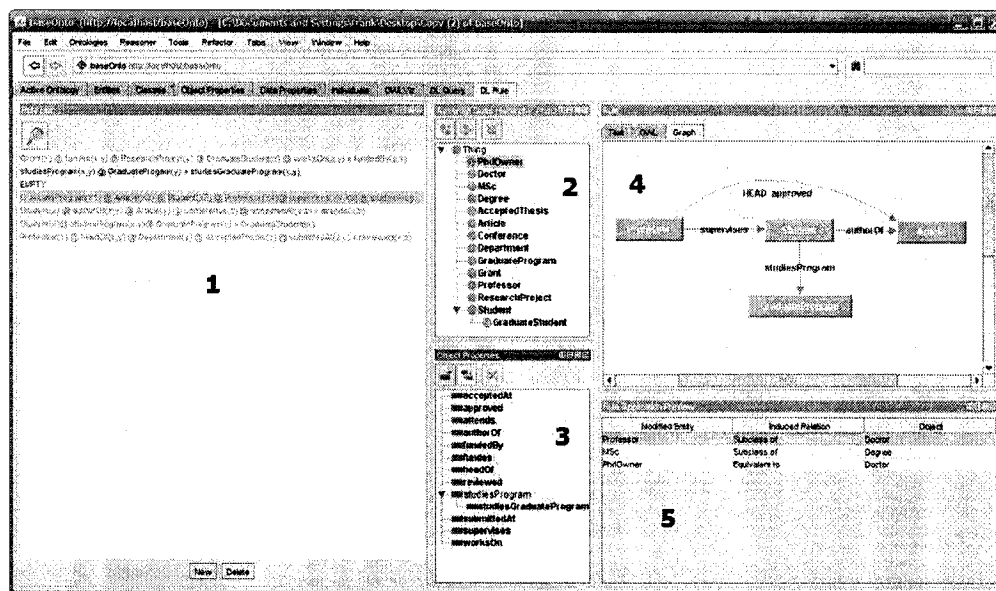


Figure 12: Overview of the DLRule plug-in.

they are written in SWRL or as axioms. The rule selected in that list is called the “active rule”.

Class Hierarchy (2) This component is a standard Protégé widget and displays a tree view of the concept taxonomy of the active ontology. It also allows to create and delete concepts.

Role Hierarchy (3) This component behaves similarly to the previous one, only it is concerned with properties.

Rule Pane (4) The Rule Pane is the actual editing component. It is composed of 3 tabs that offers different visualization and editing modes, either in a textual form, as OWL axioms or as a graph.

Rule Application Previewer (5) This component offers insight on the effect of the application of a rule over the ontology. Given the possible cost to compute the information, this view is computed on demand.

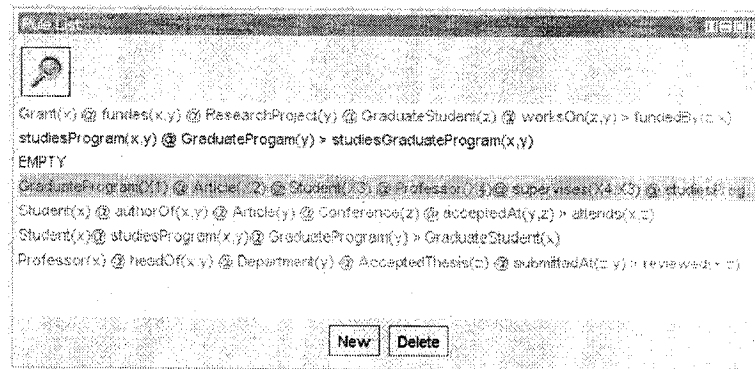


Figure 13: The Rule List component.

With that high-level introduction completed, we now introduce more thoroughly each component, except those provided by Protégé. For each component, we provide a description of all its functionalities, the way it behaves and we also provide more detailed screenshots for each one.

Rule List

The Rule List displays the list of all existing rules in the ontology, whether they are written as SWRL rules or rewritten as axioms. The rule that is selected in that lists is “active”, which means that it will be rendered in the Rule Pane and if the preview action is launched, this rule will be its subject. The list also serves to show the “status” of each rule by changing the font color in which the rule is displayed. The status of a rule is one of three possibilities: it is rewritable as axioms (green font), SWRL rule (orange font) or invalid (red). An invalid rule is one that does not respect the syntax we introduced.

Below the list are two buttons, labelled “New” and “Delete”, that obviously allow the creation of new rules and their deletion. The New button adds an empty rule in the list, which is rendered as “EMPTY”, which can then be selected and modified in the Rule Pane. The Delete button deletes the active rule, if there is one, from the ontology. This components also contains the button, labelled with the magnifier icon, that activates the Previewer with the active rule as subject.

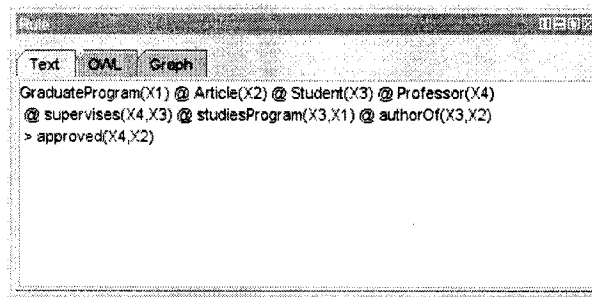


Figure 14: The text tab of the Rule Pane.

Rule Pane

The Rule Pane fulfills two main functions, it allows one to visualize the active rule and also to modify it within different paradigms. It is composed of 3 tabs, named the Text tab, OWL tab and the Graph tab respectively, each supports a different view of the rule.

The *Text* tab (shown in Figure 14) renders the rule in the textual syntax we presented in Figure 9. It allows the rule to be modified and provides some interesting features in that regard, i.e. error highlighting and drag and drop support. The error highlighting feature shows the user the syntactic errors in the rules by underlining in red the problematic parts of the rule. The rule is parsed every second to discover these errors and trigger their highlighting. We also provide a support for drag and drop operations between the class and role hierarchy components and the Text tab. If a concept C is dragged over the text tab, an atom of the form “ $C()$ @ ” is added to the rule at the location the drop occurred. A similar behavior is implemented with the roles, with the difference that the added atom is of the form “ $R(,)$ @ ”. The user then only has to put in the desired variables in the atom.

The *OWL* tab (see Figure 15) is the only of the three that is read-only since it does not display the rule itself but the OWL axioms resulting from the rule’s rewriting. The OWL axioms are rendered with the Manchester OWL Syntax⁵ to offer an OWL based yet easily readable notation. This visualization may not help the editing of the rule per se, but it can be very useful to help users better understand the inner workings of the rewriting.

⁵<http://www.w3.org/2007/OWL/wiki/ManchesterSyntax>

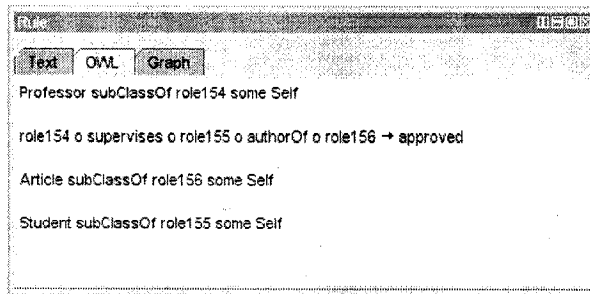


Figure 15: The OWL tab of the Rule Pane.

Finally, we have the *Graph* tab (pictured in Figure 16) that offers a graph-based visual representation of the rule. The graph is derived from the rule in the way as introduced in Section 4.2, so that variables are represented as nodes and the roles between these variables are edges. The head of the rule is represented by appending a string of the form “HEAD <headEntity>”, where “<headEntity>” is the role or concept in the head, to the relevant node or edge depending on the type of the rule. This graph representation offers an intuitive paradigm to validate the rule as well as to modify it.

The technique to modify a rule was made as intuitive as possible but we shall still go over the different use-cases. The creation of a node is simply triggered by a double-click, which will cause the insertion of a node in the graph and the appearance of a class selection window (see the left hand side of Figure 17). In this window, there is a class hierarchy component on the left, a list of concepts on the right with a combo box beneath this list. The hierarchy allows one to select the concept which we want to assert on the node, the list on the right hand side contains concepts asserted on that variable and the combo box allows to specify the concept used in the head if this variable is the target. There are two buttons, labelled with left and right arrows, to add or remove concept assertions.

For the addition of an edge between two nodes, we have to drag the pointer from the subject of the property to the object while keeping the Control key pressed. This triggers the appearance of a Property Selection window to select the desired property. This window behaves in the same

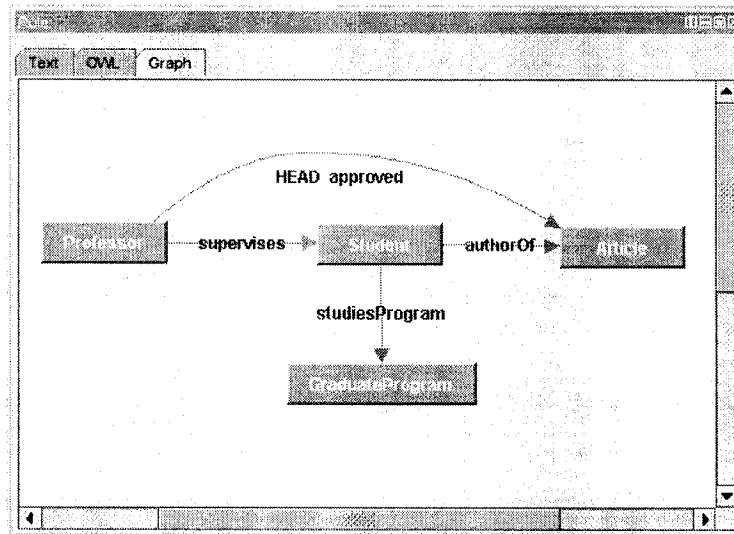


Figure 16: The Graph tab of the Rule Pane.

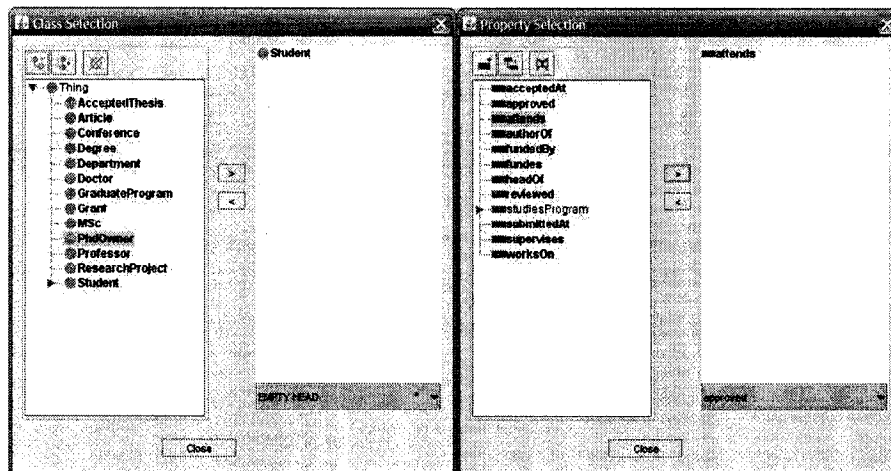
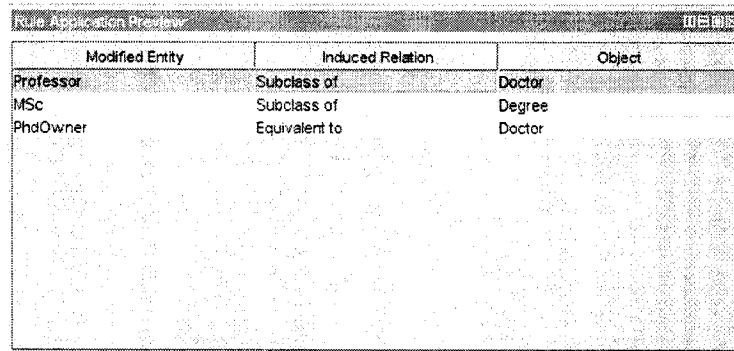


Figure 17: The Class and Property pop-ups from the Graph tab.



Modified Entity	Induced Relation	Object
Professor	Subclass of	Doctor
MSc	Subclass of	Degree
PhdOwner	Equivalent to	Doctor

Figure 18: The Rule Application Previewer component.

way as its class counterpart. To edit an existing entity, node or edge, double-clicking on it causes the relevant selection window to pop-up. Also, pressing the Delete key causes the selected entity to be removed from the graph altogether. It should be noted that when a tab loses the focus, it updates the active rule and the two other tabs with its content. This behavior ensures that all the representations of a rule are consistent.

Rule Application Previewer

The Rule Application Previewer (Figure 18) offers a functionality that is especially interesting in our context since our integration approach allows rules to have an impact on the terminological knowledge of the ontology. The effect of such rules is sometimes not trivial to foresee so the previewer is a great help to ensure a rule has the desired effect. When the previewer is activated, by pressing the magnifier button, the effects of the active rule are computed and each new subsumption and equivalence relations is displayed in a line of the previewer's table. Let \mathcal{K} be a knowledge base and r be a rule, the subsumption $C \sqsubseteq D$ is an effect of r if $\mathcal{K} \not\models C \sqsubseteq D$ and $\{\mathcal{K} \cup r\} \models C \sqsubseteq D$. The effect on intensional knowledge (ABox) is not shown because the amount of such effects could be very large and thus obscure the interesting information in that mass of information.

5.3.3 Opening and Saving Operations

All the input and output operations of our plug-in are embedded into those of Protégé. This means that loading and saving ontologies containing rules are transparent operations for the user. Protégé, in coordination with the plug-in, will complete the required operations with respect to the content of the ontology.

5.3.4 Design Problem

During the plug-in's implementation, we faced a problem that had vast implications on the behavior of the whole application. The rewriting of a rule may require the introduction of new classes and properties which would be better not to appear in the hierarchies to avoid to confuse the user with this information that is irrelevant in its context. Considering that, we opted to isolate Protégé from the ontology containing rules. To do so, when the application loads a rule-augmented ontology, the plug-in removes the rules and all the concepts and properties generated by the rewriting and Protégé then uses this "cleaned" version. The rules and their rewriting are added back into the ontology by the plug-in when the ontology is saved.

This isolation unfortunately cause another problem. Since Protégé works with a version of the ontology stripped of the rules, the taxonomy displayed does not take into account the possible effect of the rules. The effect of this problem is lessened by the fact that the user has access to the Previewer which gives these possible effects. This solution is not perfect, but it offers the best combination of clarity and accuracy.

5.3.5 DLRule vs. Other Rule Editors

We would now like to offer a a brief overview of the comparisons that can be made between this plug-in and other available rule editors (in the ontology context). There are not many such tools, in fact our plug-in is unique in many respects (to the best of our knowledge): it is the only one to support OWL 2 (the *SRITQ* DL), the only one to offer a mechanism to preview the effect

of the rules, the only one bundled as a Protégé 4 plug-in, and the only one to allow a graphical rule specification within Protégé. Let us now compare our plug-in with two competitors, the SWRLTab and OntoStudio.

The Protégé SWRL tab allows to add SWRL rules within an OWL ontology. It only supports a textual form of the rules and is only implemented for Protégé 3.x, thus limiting it to the support of OWL (*SHOIN*(D)). A nice feature of this implementation is the interoperation mechanism with various rule engines it provides.

OntoStudio⁶ provides a rather usable and intuitive graphical rule editor tool. The tool is however not very comparable to our tool because it focuses on RDF and F-Logic ontologies and no reasoning services are provided within the tool. Furthermore, this tool is commercial and only freely usable under certain conditions.

5.4 Practical Reasoning Results

With *SRQIQ* being a decidable logic and the axioms we insert being valid in *SRQIQ*, it is obvious that reasoning over a rule augmented ontology is a decidable problem. What is less obvious is how the currently available reasoners fare with such ontologies in practice. Many factors contribute to that uncertainty. For example, since OWL 2 is still in development, no big ontologies are yet committed to use it so we have little results for large and/or complex ontologies. Also we use newly introduced features which have been implemented recently by the reasoners, hence reasoning over these constructs is probably not optimized as much as other OWL features. Furthermore, our rewriting technique uses extensively some of these new features and in ways they have not been used much.

In order to settle these unknowns, we designed a test suite aimed to verify that reasoning over rule-augmented ontologies was indeed practically usable. Our test is composed of three series of 16 realizations of ontologies where we augment the ontology with additional data between each

⁶<http://www.ontoprise.de/de/en/home/products/ontostudio.html>

iteration. Each series is characterized by the nature of the additional data which is added. For the first series, the added content is the ABox of the original ontology, for the second series, the TBox (including the rules, if applicable) and for the third, both the TBox and ABox are added. This test was carried out with two base ontologies, only one containing rules. The data added between each iteration is 20 times the relevant data the original ontology contained. The ontology we started with is the University Ontology presented in Appendix A, which contains 10 classes, 13 properties, 5 DL rules and 20 individuals (and their assertions). A quick calculation reveals that the biggest ontology processed contained 3200 classes, 4160 properties, 1600 rules and 6400 individuals. We had to stop at 16 executions because the parser is limited to 64000 entities per file and the 17th run reached that threshold.

The reasoner we used for these experiments is FaCT++⁷, version 1.1.11. We used FaCT++ because preliminary experiments showed that it performed and scaled much better than Pellet for reasoning over OWL 2 ontologies. The tests were conducted on a computer operating Windows XP with a Intel Core Duo 2.16GHz CPU and 2 GB of RAM.

Results

The only data we collected during these tests is the time required to realize each ontology since this is the measure reflecting most accurately the computational cost of an operation in this context. The results are shown in the chart in Figure 19. The data underlying the graph is the average computation time of 5 independent runs for iterations of each series.

These data allow us to see the effect of many factors over the computational cost of reasoning, i.e. growing ABox size, TBox size and both, for ontologies containing, or not, rules. That will allow us to isolate the effect of rules alone in an ontology and how they might combine with other factors to affect the cost of reasoning.

⁷<http://code.google.com/p/factplusplus/>

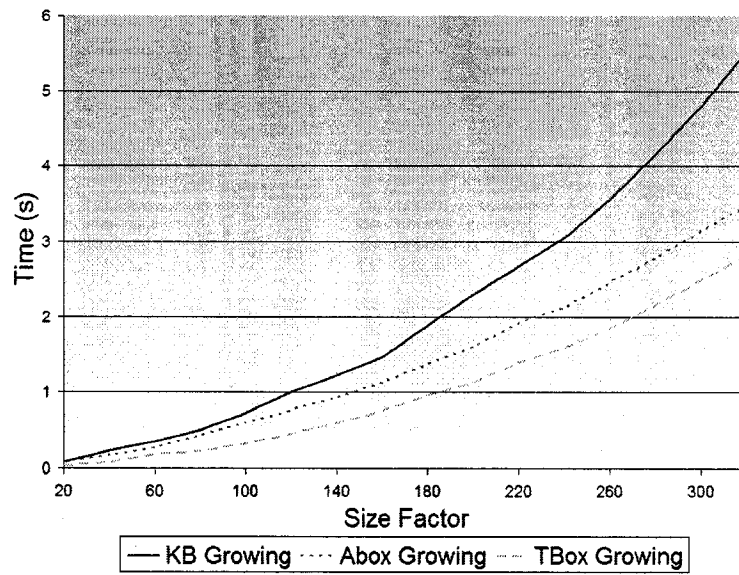


Figure 19: Graph of the realization time of KBs containing rules.

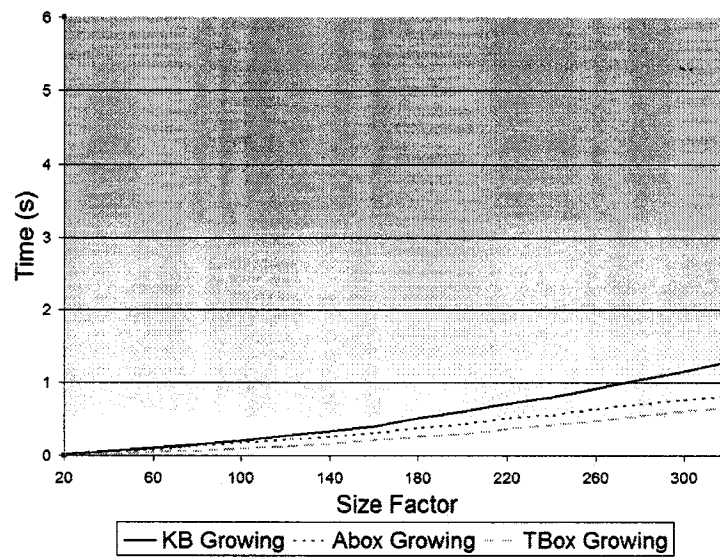


Figure 20: Graph of the realization time of KBs not containing rules.

Analysis

Unfortunately, we have to focus our analysis on the trend the results provides, not on the absolute values, since we lack points of comparison. Indeed, we can hardly compare the results we obtained with results of implementations of other rule integration technique because the semantics are different. The extra expressivity of *SR_{OTQ}* and the DL-unsafety of rules enabling them to have an effect on the taxonomy obviously has a cost. For instance, the computation times of KAON2⁸ over the same test suite are better by an order of magnitude because it uses a sub-logic of *SR_{OTQ}* and DL-safe rules.

This performance difference can be seen as a major issue but we consider it otherwise. The semantics of rules in our approach offer an expressive power that can be of great interest for many types of applications and experience shows that users are usually more concerned with expressivity than performance as long as the latter is still practical, which it is in this particular case. We thus leave to the users to decide if the trade-off between expressivity, of both the rules and the description logic, and performance is of any interest in their particular field of application.

The results we obtained allow us to come to some conclusions. We can see that the addition of rules alone into an ontology does not cause major problems since the growth function of the computation time is, although higher, of the same order as the one for ontology without rules. We can also see that the rules combined with a number of individuals/concepts do not cause an explosion of computation time. These two observations allows us to state that our approach is practical and also scales rather well, as well as DLs scale at least.

⁸<http://kaon2.semanticweb.org/>

Chapter 6

Conclusion

Before presenting the conclusion of this work, we shall point out that there has been parallel work which content overlaps with the content of this thesis. The work presented in [KRH08] also describes rules that can be rewritten into DL axioms but offers a different angle to the work. Indeed, this form of rules is then used to see what rules can be integrated into weaker DLs to maintain their tractability. Also, this work does not cover the practical considerations that are part of this thesis.

Now, the conclusion is structured as such. First, we will summarize the content of the thesis, then we will relate the contribution of this research to the field and finally we present some interesting ideas we uncovered during this research but were left as future work.

6.1 Summary

Let us now go over the content we presented in this thesis and briefly summarize it as we go along. We first presented the Semantic Web and its unsatiable thirst for expressivity, to which we think that combining rules and DL should be part of the solution. We presented these two formalism and their respective expressivity and which, when combined, result in an undecidable formalism. Since this field of research had been navigated before, we presented the most relevant results that were obtained. We then analysed these results and have seen that the current state of affairs let the users choose between the lesser of two evils when in need for the expressivity of both rules

and DLs. Either they used the complete union of the languages and had an undecidable logic, or they restricted the interaction between the rules and the terminology and regained decidability. We defined our research objective with respect to that dilemma and we intended to provide a third way, one in which both the semantics of the rules and the decidability of reasoning were to be preserved.

We then presented the core of this thesis, a rewriting technique that enables us to offer that third way by allowing rules to be rewritten into DL axioms. Since the full union of rules and DLs is undecidable, we devised a new restriction that let the rules modify the terminology while maintaining decidability. We identified and presented such a set of syntactic restrictions. We then introduced the technique used to rewrite rules as axioms. We then showed that rules rewritten in such a fashion were indeed not semantically restricted and decidability follows from their rewriting in a decidable logic.

We completed the thesis by presenting the prototypes implementing our technique. The goal of this part of the project was twofold: first, to show that this approach was usable in real-life scenarios and expressive enough to be interesting in a variety of contexts, and then also to facilitate the adoption of that technique by the potential users. The rule editing tool that we developed allowed us to answer both of these questions. We have shown that reasoning over ontologies augmented with rules was indeed practical and it allowed us to see that users were really interested by the potential of this work and also that they considered that the our tool resulted in a valuable contribution that compared advantageously with other available alternatives.

6.2 Contribution

The goal of any research is obviously to advance the knowledge we have in a certain area. This work contributed to the field on both the theoretical and applied level. On the theoretical level, we identified a combination of rules and DL that evades the restriction to DL-safety and remained decidable. These characteristics allow the rules in our approach to modify the terminology. Our integration paradigm is the first to combine these two features. On the application level, we showed

this combination is: (i) compatible with OWL, (ii) supported by existing reasoners, and (iii) we provided a set of tools to support the community in their use of this combination. The software we implemented also yielded a contribution we did not originally aimed for. In our technique, concept-headed rules are rewritten as concept inclusion axioms, the best way to represent them, such that a user accustomed to using rules can define a whole ontology in the rule interface and the rewritten ontology will be identic to an ontology optimally defined using concept inclusion axioms. This enables the user to choose the paradigm he is more comfortable with. To fully understand the contribution of this work, we will now present the publications that stemmed from it and also relate the contribution to the project's objectives.

6.2.1 Publications

In the world of academic research, publications in peer reviewed publications and conferences are the prime way to confirm the validity of a research and its results. This considered, we have to state that the work presented in this thesis has been acknowledged as valid and interesting by such review committees. Indeed, two articles presenting subsets of the present work have been published.

The first publication, "DLRule: A Rule Editor plug-in for Protégé" [GH08] was presented at the "2008 Workshop on OWL: Experiences and Directions" conference. This workshop is aimed at gathering both users and researchers working with OWL. In accordance with this audience, it presented the basics of the rewriting technique and the features of the Protégé plug-in we developed. Its contribution was to lay the basis of the rewriting technique and to present a new rule editing tool dedicated to OWL users. Users showed keen interest in both the rewriting and the plug-in, proving that it addressed an actual need of the community.

The second article, "Rewriting Rules into *SR_{OIQ}* Axioms" [GSH08] was published at the "2008 International Workshop on Description Logics" which is the most prominent conference in the field of DL. The audience being mostly researchers with extensive knowledge of the field, this paper extensively presented the theory of the rewriting technique and the related proofs which was

a relevant contribution to the field.

6.2.2 Contribution versus Objectives

At the beginning of this research project, we investigated the different approaches to rules and DL integration and identified the shortcomings of these approaches. We then decided on objectives that we intended to complete with our research, overcoming some of the aforementioned issues among others. Let us now restate these objectives and analyze how successfully this research addresses each of them.

Identify the most expressive DL and rule combination

We intended to identify the most expressive combination of DL and rules that did not require the restriction to DL-safety to retain decidability. We have introduced DL rules, a combination of *SR_QIQ* and a restricted form of rules. *SR_QIQ* being the most expressive DL currently available and our rules only restricted to respect *SR_QIQ*'s restrictions, we conjecture that DL rules are the most expressive combination, but we can say for sure that it is a very expressive combination.

Define the semantics and prove the decidability of this combination

The proposed approach rewrites rules as *SR_QIQ* axioms. It follows that the semantics of the rules are the same as *SR_QIQ*'s, also the restrictions over the rules ensure that the rewritten rules are valid *SR_QIQ* axioms and since *SR_QIQ* is a decidable logic, the decidability of the rules is obvious. This objective has been totally accomplished.

Build tools supporting this combination

The softwares we have presented in Chapter 5 provide the required functionalities to insert, modify and remove rules from an OWL ontology in graphical mode within a major ontology editor and also to convert existing SWRL rules into OWL axioms automatically. This software support

is advantageously comparable with other integration technique, which leads us to consider this objective as fulfilled.

Prove the practical usability of this combination

In Section 5.4 we have shown that reasoning over rule augmented ontologies was practical and scalable, so the objective is completed. However, we have to mention that the usability of the software supporting the approach has not been fully evaluated, although the results and feedback we got let us believe it is.

Overall Objectives Completion

The four objectives we had set for this research have been completed with a rather high level of success. Obviously the subjectivity of some of these objectives prevent us to say it is a “total” success, but we confidently say they are achieved.

6.3 Future Work

While conducting this research, we discovered very interesting ideas to further improve rules and DL integration, some of which were presented in this thesis. Unfortunately, some of them were not pursued because they were out of the scope of the presented work. We now introduce the two most promising ideas that we consider as interesting future work.

6.3.1 $SROIQ^+$

Rules of the type $hasChild(x, y) \wedge Man(y) \rightarrow hasSon(x, y)$ cannot be rewritten since naturally $hasSon \sqsubseteq hasChild$, hence $hasSon \prec hasChild$, and the rule implies that $hasChild \prec hasSon$ thus violating the \prec -regularity of roles hierarchy. This type of rules being rather common, it is desirable to solve this problem. A possible solution was provided by Ulrike Sattler, extending $SROIQ$ with a new form of role inclusion axioms, similar to the restrictions found in ALB

[HA00]. In addition to the $SR\mathcal{OIQ}$ RIAs, in $SR\mathcal{OIQ}^+$, an expression of the form $S\dot{r}_C \sqsubseteq R$ or $S\dot{\gamma}_C \sqsubseteq R$ for C a $SR\mathcal{OIQ}$ concept and R a possibly inverse role, and S a possibly inverse *simple* role¹ is also a RIA. Intuitively the construct $R\dot{r}_C \sqsubseteq S$ means that a R relation toward an instance of C implies an S relation, and $R\dot{\gamma}_C \sqsubseteq S$ means that a R relation with an instance of C as subject implies an S relation. The semantics are formally defined as follows: an interpretation \mathcal{I} satisfies

$$\begin{aligned} S\dot{r}_C \sqsubseteq R & \text{ if } \{ \langle x, y \rangle \in S^{\mathcal{I}} \mid y \in C^{\mathcal{I}} \} \subseteq R^{\mathcal{I}} \\ S\dot{\gamma}_C \sqsubseteq R & \text{ if } \{ \langle x, y \rangle \in S^{\mathcal{I}} \mid x \in C^{\mathcal{I}} \} \subseteq R^{\mathcal{I}} \end{aligned}$$

This extension was shown to be decidable and the introduced axioms are treated as usual, i.e. in a non DL-safe way. Since $SR\mathcal{OIQ}^+$ is a super-set of $SR\mathcal{OIQ}$ and decidable, we can seamlessly use it with our rewritten rules thus enabling the aforementioned desirable rule form. We have identified two possible further extensions to $SR\mathcal{OIQ}$ that would provide more expressivity and also possibly retain decidability. The first one is to allow Self-restrictions over complex roles, which would allow the rewriting of cyclic rules. Number restrictions over complex roles are known to be undecidable, but since the Self-restriction is weaker, it might be decidable. The second possible extension is to investigate if we could replace role hierarchies with role conjunction as it was done for $SH\mathcal{IQ}$ in [GHLS07]. It remains to be seen if this would be decidable and possibly which restrictions are required over the role conjunctions to retain it.

Investigating these possible extensions to $SR\mathcal{OIQ}$ and then seeing how this may augment the versatility of our rewriting should be very interesting and may yield some very interesting results.

6.3.2 Cover More Rules with Description Graphs

One may recall that we require a rule to be acyclic in order to be admissible for rewriting. That follows from the fact that even if the RIAs of $SR\mathcal{OIQ}$ can be used to define non-tree-shaped models, which relax the restriction to tree-like models present in previous DLs, we still cannot represent cyclic rules. A recently published article, “Structured Objects in OWL: Representation and Reasoning” [MGHS08], introduces an extension to OWL that allows the modeling of arbitrary

¹Recall that a simple role is one which is not implied by role paths; see [HKS06] for details.

relational structure using *description graphs*. These are labeled directed graphs that consist of a set of vertices labeled with atomic concepts and a set of edges labeled with atomic roles. The semantics of this extension reveals that the description graphs are used to specify a relational structure around an instance of a certain class. The use of SWRL-like rules is also allowed to model similar relational structures around an edge of a given role.

A graph-extended DL knowledge base \mathcal{K} is defined as $\mathcal{K} = \{\mathcal{T}, \mathcal{G}, \mathcal{P}, \mathcal{A}\}$ where \mathcal{T} is a regular TBox, \mathcal{G} is a description graph, \mathcal{P} is a set of SWRL-like rules and \mathcal{A} is an ABox. Atomic roles are divided into two disjoint sets, N_{R_g} and N_{R_t} , where elements of N_{R_g} are called *graph roles* and elements of N_{R_t} are *tree roles*. In order to avoid the undecidability caused by the introduction of arbitrary structures in DLs, some restrictions have to be enforced. First, the structured objects represented by the graphs and the SWRL-like rules are required to have a bounded number of components. This is not a serious restriction since most practical examples of such arbitrary structures are naturally composed of a bounded number of components. Additionally, the TBox \mathcal{T} can only refer to tree roles, \mathcal{P} and \mathcal{G} can only refer to graph roles and the ABox \mathcal{A} can refer to both type of roles. It is to be noted that DL-safety is not required to ensure the formalism's decidability.

It seems a natural fit to use these description graphs, and the associated SWRL-like rules, to model arbitrary rules. This could yet extend the scope of rules usable without DL-safety and yet retain decidability. This is a preliminary idea and the expressivity and usability of such an extension remains to be investigated.

Appendix A

The University Ontology

We included this ontology as an appendix for many reasons, first since all the examples of rules are excerpts from this ontology, it is sometimes useful to be able to see the bigger picture. Also, we used this ontology as the base when generating all the larger ontologies used in our experiments and it is always relevant to see the test data first-hand. We chose to render it using the OWL/XML notation because it is the middle ground between a human-centered notation, the Manchester Syntax, and the very machinesque RDF.

```
<?xml version="1.0"?>
<!DOCTYPE Ontology [
  <!ENTITY cs "http://cs.concordia.ca/" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY owl2 "http://www.w3.org/2006/12/owl2#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
```

Class count	11
Object property count	23
Expressivity	<i>ALERT</i>
SubClasses axioms count	20
GCI count	1
Inverse Object properties axioms count	2
Class assertions axioms count	110
Object property assertion axioms count	139

Figure 21: Statistics of the University Ontology.

```

<!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
<!ENTITY baseOnto "http://users.encs.concordia.ca/~f_gasse/baseOnto#">
<!ENTITY baseOnto2 "http://users.encs.concordia.ca/~f_gasse/baseOnto/">
]>

<Ontology xmlns="http://www.w3.org/2006/12/owl2-xml#"
  xml:base="http://www.w3.org/2006/12/owl2-xml#"
  xmlns:baseOnto="http://users.encs.concordia.ca/~f_gasse/baseOnto#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:baseOnto2="http://users.encs.concordia.ca/~f_gasse/baseOnto/"
  xmlns:owl2="http://www.w3.org/2006/12/owl2#"
  xmlns:cs="http://cs.concordia.ca/"
  URI="http://users.encs.concordia.ca/~f_gasse/baseOnto">

  <Annotation annotationURI="&cs;dl-rule">
    <Constant>
      Professor(x) @ headOf(x,y) @ Department(y) @ AcceptedThesis(z)
      @ submittedAt(z,y) &gt;reviewed(x,z)
    </Constant>
  </Annotation>

  <Annotation annotationURI="&cs;dl-rule">
    <Constant>
      Student(x) @ authorOf(x,y) @ Article(y) @ Conference(z)
      @ acceptedAt(y,z) &gt;attends(x,z)
    </Constant>
  </Annotation>

  <Annotation annotationURI="&cs;dl-rule">
    <Constant>
      Student(x) @ studiesProgram(x,y) @ GraduateProgram(y)
      &gt;GraduateStudent(x)
    </Constant>
  </Annotation>

  <Annotation annotationURI="&cs;dl-rule">
    <Constant>
      Grant(x) @ fundes(x,y) @ ResearchProject(y) @ GraduateStudent(z)
      @ worksOn(z,y) &gt;fundedBy(z,x)
    </Constant>
  </Annotation>

  <Annotation annotationURI="&cs;dl-rule">
    <Constant>
      Professor(x) @ supervises(x,y) @ Student(y) @
      studiesProgram(y,w) @ GraduateProgram(w) @
      Article(z) @ authorOf(y,z) &gt;approved(x,z)
    </Constant>
  </Annotation>

  <SubClassOf>
    <OWLClass URI="&baseOnto;AcceptedThesis"/>
    <OWLClass URI="&owl;Thing"/>
  </SubClassOf>

```



```

</SubClassOf>
<SubClassOf>
  <Annotation annotationURI="&cs;dl-rule-axiom">
    <Constant>addedAxiom</Constant>
  </Annotation>
  <OWLClass URI="&baseOnto;AcceptedThesis"/>
  <ObjectExistsSelf>
    <ObjectProperty URI="&baseOnto2;role111"/>
  </ObjectExistsSelf>
</SubClassOf>
<Declaration>
  <OWLClass URI="&baseOnto;AcceptedThesis"/>
</Declaration>
<SubClassOf>
  <OWLClass URI="&baseOnto;Article"/>
  <OWLClass URI="&owl;Thing"/>
</SubClassOf>
<SubClassOf>
  <Annotation annotationURI="&cs;dl-rule-axiom">
    <Constant>addedAxiom</Constant>
  </Annotation>
  <OWLClass URI="&baseOnto;Article"/>
  <ObjectExistsSelf>
    <ObjectProperty URI="&baseOnto2;role102"/>
  </ObjectExistsSelf>
</SubClassOf>
<SubClassOf>
  <Annotation annotationURI="&cs;dl-rule-axiom">
    <Constant>addedAxiom</Constant>
  </Annotation>
  <OWLClass URI="&baseOnto;Article"/>
  <ObjectExistsSelf>
    <ObjectProperty URI="&baseOnto2;role107"/>
  </ObjectExistsSelf>
</SubClassOf>
<SubClassOf>
  <OWLClass URI="&baseOnto;Conference"/>
  <OWLClass URI="&owl;Thing"/>
</SubClassOf>
<SubClassOf>
  <Annotation annotationURI="&cs;dl-rule-axiom">
    <Constant>addedAxiom</Constant>
  </Annotation>
  <OWLClass URI="&baseOnto;Conference"/>
  <ObjectExistsSelf>
    <ObjectProperty URI="&baseOnto2;role108"/>
  </ObjectExistsSelf>
</SubClassOf>
<SubClassOf>
  <OWLClass URI="&baseOnto;Department"/>
  <OWLClass URI="&owl;Thing"/>
</SubClassOf>
<SubClassOf>
  <Annotation annotationURI="&cs;dl-rule-axiom">

```

```

        <Constant>addedAxiom</Constant>
    </Annotation>
    <OWLClass URI="&baseOnto;Department"/>
    <ObjectExistsSelf>
        <ObjectProperty URI="&baseOnto2;role110"/>
    </ObjectExistsSelf>
</SubClassOf>
<Declaration>
    <OWLClass URI="&baseOnto;Department"/>
</Declaration>
<SubClassOf>
    <OWLClass URI="&baseOnto;GraduateProgram"/>
    <OWLClass URI="&owl;Thing"/>
</SubClassOf>
<SubClassOf>
    <OWLClass URI="&baseOnto;GraduateStudent"/>
    <OWLClass URI="&baseOnto;Student"/>
</SubClassOf>
<SubClassOf>
    <Annotation annotationURI="&cs;dl-rule-axiom">
        <Constant>addedAxiom</Constant>
    </Annotation>
    <OWLClass URI="&baseOnto;GraduateStudent"/>
    <ObjectExistsSelf>
        <ObjectProperty URI="&baseOnto2;role103"/>
    </ObjectExistsSelf>
</SubClassOf>
<SubClassOf>
    <OWLClass URI="&baseOnto;Grant"/>
    <OWLClass URI="&owl;Thing"/>
</SubClassOf>
<SubClassOf>
    <Annotation annotationURI="&cs;dl-rule-axiom">
        <Constant>addedAxiom</Constant>
    </Annotation>
    <OWLClass URI="&baseOnto;Grant"/>
    <ObjectExistsSelf>
        <ObjectProperty URI="&baseOnto2;role105"/>
    </ObjectExistsSelf>
</SubClassOf>
<SubClassOf>
    <OWLClass URI="&baseOnto;Professor"/>
    <OWLClass URI="&owl;Thing"/>
</SubClassOf>
<SubClassOf>
    <Annotation annotationURI="&cs;dl-rule-axiom">
        <Constant>addedAxiom</Constant>
    </Annotation>
    <OWLClass URI="&baseOnto;Professor"/>
    <ObjectExistsSelf>
        <ObjectProperty URI="&baseOnto2;role100"/>
    </ObjectExistsSelf>
</SubClassOf>
<SubClassOf>

```

```

        <Annotation annotationURI="&cs;dl-rule-axiom">
            <Constant>addedAxiom</Constant>
        </Annotation>
        <OWLClass URI="&baseOnto;Professor"/>
        <ObjectExistsSelf>
            <ObjectProperty URI="&baseOnto2;role109"/>
        </ObjectExistsSelf>
    </SubClassOf>
    <Declaration>
        <OWLClass URI="&baseOnto;Professor"/>
    </Declaration>
    <SubClassOf>
        <OWLClass URI="&baseOnto;ResearchProject"/>
        <OWLClass URI="&owl;Thing"/>
    </SubClassOf>
    <SubClassOf>
        <Annotation annotationURI="&cs;dl-rule-axiom">
            <Constant>addedAxiom</Constant>
        </Annotation>
        <OWLClass URI="&baseOnto;ResearchProject"/>
        <ObjectExistsSelf>
            <ObjectProperty URI="&baseOnto2;role104"/>
        </ObjectExistsSelf>
    </SubClassOf>
    <SubClassOf>
        <OWLClass URI="&baseOnto;Student"/>
        <OWLClass URI="&owl;Thing"/>
    </SubClassOf>
    <SubClassOf>
        <Annotation annotationURI="&cs;dl-rule-axiom">
            <Constant>addedAxiom</Constant>
        </Annotation>
        <OWLClass URI="&baseOnto;Student"/>
        <ObjectExistsSelf>
            <ObjectProperty URI="&baseOnto2;role101"/>
        </ObjectExistsSelf>
    </SubClassOf>
    <SubClassOf>
        <Annotation annotationURI="&cs;dl-rule-axiom">
            <Constant>addedAxiom</Constant>
        </Annotation>
        <OWLClass URI="&baseOnto;Student"/>
        <ObjectExistsSelf>
            <ObjectProperty URI="&baseOnto2;role106"/>
        </ObjectExistsSelf>
    </SubClassOf>
    <Declaration>
        <ObjectProperty URI="&baseOnto;acceptedAt"/>
    </Declaration>
    <SubObjectPropertyOf>
        <Annotation annotationURI="&cs;dl-rule-axiom">
            <Constant>addedAxiom</Constant>
        </Annotation>
        <SubObjectPropertyChain>

```

```

        <ObjectProperty URI="&baseOnto2;role100"/>
        <ObjectProperty URI="&baseOnto;supervises"/>
        <ObjectProperty URI="&baseOnto2;role101"/>
        <ObjectProperty URI="&baseOnto;authorOf"/>
        <ObjectProperty URI="&baseOnto2;role102"/>
    </SubObjectPropertyChain>
    <ObjectProperty URI="&baseOnto;approved"/>
</SubObjectPropertyOf>
<Declaration>
    <ObjectProperty URI="&baseOnto;approved"/>
</Declaration>
<SubObjectPropertyOf>
    <Annotation annotationURI="&cs;dl-rule-axiom">
        <Constant>addedAxiom</Constant>
    </Annotation>
    <SubObjectPropertyChain>
        <ObjectProperty URI="&baseOnto2;role106"/>
        <ObjectProperty URI="&baseOnto;authorOf"/>
        <ObjectProperty URI="&baseOnto2;role107"/>
        <ObjectProperty URI="&baseOnto;acceptedAt"/>
        <ObjectProperty URI="&baseOnto2;role108"/>
    </SubObjectPropertyChain>
    <ObjectProperty URI="&baseOnto;attends"/>
</SubObjectPropertyOf>
<Declaration>
    <ObjectProperty URI="&baseOnto;attends"/>
</Declaration>
<Declaration>
    <ObjectProperty URI="&baseOnto;authorOf"/>
</Declaration>
<SubObjectPropertyOf>
    <Annotation annotationURI="&cs;dl-rule-axiom">
        <Constant>addedAxiom</Constant>
    </Annotation>
    <SubObjectPropertyChain>
        <ObjectProperty URI="&baseOnto2;role103"/>
        <ObjectProperty URI="&baseOnto;worksOn"/>
        <ObjectProperty URI="&baseOnto2;role104"/>
        <ObjectProperty URI="&baseOnto;fundesinv"/>
        <ObjectProperty URI="&baseOnto2;role105"/>
    </SubObjectPropertyChain>
    <ObjectProperty URI="&baseOnto;fundedBy"/>
</SubObjectPropertyOf>
<Declaration>
    <ObjectProperty URI="&baseOnto;fundedBy"/>
</Declaration>
<InverseObjectProperties>
    <ObjectProperty URI="&baseOnto;fundes"/>
    <ObjectProperty URI="&baseOnto;fundesinv"/>
</InverseObjectProperties>
<Declaration>
    <ObjectProperty URI="&baseOnto;fundes"/>
</Declaration>
<Declaration>

```

```

    <ObjectProperty URI="&baseOnto;headOf"/>
  </Declaration>
  <SubObjectPropertyOf>
    <Annotation annotationURI="&cs;dl-rule-axiom">
      <Constant>addedAxiom</Constant>
    </Annotation>
    <SubObjectPropertyChain>
      <ObjectProperty URI="&baseOnto2;role109"/>
      <ObjectProperty URI="&baseOnto;headOf"/>
      <ObjectProperty URI="&baseOnto2;role110"/>
      <ObjectProperty URI="&baseOnto;submittedAtinv"/>
      <ObjectProperty URI="&baseOnto2;role111"/>
    </SubObjectPropertyChain>
    <ObjectProperty URI="&baseOnto;reviewed"/>
  </SubObjectPropertyOf>
  <Declaration>
    <ObjectProperty URI="&baseOnto;reviewed"/>
  </Declaration>
  <Declaration>
    <ObjectProperty URI="&baseOnto;studiesProgram"/>
  </Declaration>
  <Declaration>
    <ObjectProperty URI="&baseOnto;submittedAt"/>
  </Declaration>
  <InverseObjectProperties>
    <ObjectProperty URI="&baseOnto;submittedAtinv"/>
    <ObjectProperty URI="&baseOnto;submittedAt"/>
  </InverseObjectProperties>
  <Declaration>
    <ObjectProperty URI="&baseOnto;supervises"/>
  </Declaration>
  <Declaration>
    <ObjectProperty URI="&baseOnto;worksOn"/>
  </Declaration>
  <SubClassOf>
    <Annotation annotationURI="&cs;dl-rule-axiom">
      <Constant>addedAxiom</Constant>
    </Annotation>
    <ObjectIntersectionOf>
      <OWLClass URI="&baseOnto;Student"/>
      <ObjectSomeValuesFrom>
        <ObjectProperty URI="&baseOnto;studiesProgram"/>
        <OWLClass URI="&baseOnto;GraduateProgram"/>
      </ObjectSomeValuesFrom>
    </ObjectIntersectionOf>
    <OWLClass URI="&baseOnto;GraduateStudent"/>
  </SubClassOf>
</Ontology>

```

Bibliography

- [BCM⁺03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [Bho05] Kruthi Bhoopalam. *Fire – A Description Logic Based Rule Engine for OWL Ontologies with SWRL-like Rules*. M. Sc. thesis, Concordia University, September 2005.
- [BvHH⁺04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language reference. W3C Recommendation, 10 February 2004. Available at <http://www.w3.org/TR/owl-ref/>.
- [DLNS98] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. *AL-log: Integrating Datalog and Description Logics*. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.
- [EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Transactions on Database Systems (TODS)*, 22(3):364–418, sep 1997.
- [GH08] Francis Gasse and Volker Haarslev. DLRule: A Rule Editor Plug-in for Protégé. In *Proceedings of the 4th International Workshop: OWL Experiences and Directions (OWLEd 2008), Washington DC, USA, April 1-2, 2008*, 2008.

- [GHLS07] Birte Glimm, Ian Horrocks, Carsten Lutz, and Uli Sattler. Conjunctive Query Answering for the Description Logic *SHIQ*. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.
- [GHVD03] Benjamin N. Grosz, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.
- [GSH08] Francis Gasse, Ulrike Sattler, and Volker Haarslev. Rewriting rules into *SROIQ* axioms. In *Proceedings of the 2008 International Workshop on Description Logics (DL2008), Dresden, Germany, May 13 - 16, 2008*, CEUR Workshop Proceedings. CEUR-WS.org, 2008.
- [HA00] Ullrich Hustadt and Renate A. Schmidt. Issues of Decidability for Description Logics in the Framework of Resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *LNAI*, pages 191–205. Springer Verlag, 2000.
- [Har84] D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Volume II — Extensions of Classical Logic*, pages 497–604. D. Reidel Publishing Company: Dordrecht, The Netherlands, 1984.
- [HC96] G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic*. Routledge, September 1996.
- [HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible *SROIQ*. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 57–67. AAAI Press, 2006.

- [HMS05] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In Leslie Pack Kaelbling and Alessandro Saffioti, editors, *Proc. of the 19th Int. Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 466–471, Edinburgh, UK, July 30–August 5 2005. Morgan Kaufmann Publishers.
- [HPS04] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.
- [HPSB⁺04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004. Available at <http://www.w3.org/Submission/SWRL/>.
- [HS00] Ullrich Hustadt and Renate A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In *Analytic Tableaux and Related Methods*, pages 67–71, 2000.
- [HS04] Ian Horrocks and Ulrike Sattler. Decidability of *SHIQ* with complex role inclusion axioms. *Artificial Intelligence*, 160(1):79–104, 2004.
- [KFN04] Holger Knublauch, Ray W. Ferguson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL plugin: An Open Development Environment for Semantic Web Applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2004.
- [KPS06] Vladimir Kolovski, Bijan Parsia, and Evren Sirin. Extending the *SHOIQ(d)* Tableaux with DL-safe Rules: First Results. In Parsia et al. [PST06].

- [KRH08] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Expressive tractable description logics based on *SRQ* rules. Technical report, Institut AIFB, Universität Karlsruhe (TH), 2008.
- [Lif03] V. Lifschitz. Nonmonotonic databases and epistemic queries. In *Proceedings of IJCAI 91, Sydney, Australia*, pages 381–386, 2003.
- [LR96] Alon Y. Levy and Marie-Christine Rousset. CARIN: A representation language combining horn rules and description logics. In Wolfgang Wahlster, editor, *12th European Conference on Artificial Intelligence, Budapest, Hungary, August 11-16, 1996, Proceedings*, pages 323–327. John Wiley and Sons, Chichester, 1996.
- [LR98] Alon Y. Levy and Marie-Christine Rousset. Combining Horn Rules and Description Logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.
- [Mar95] Aaron Marcus. *Principles of effective visual communication for graphical user interface design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.
- [MGHS08] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulrike Sattler. Representing Structured Objects using Description Graphs. In *Proc. of the 11th Int. Joint Conf. on Principles of Knowledge Representation and Reasoning (KR 2008)*, Sydney, Australia, August 16–19 2008. AAAI Press. To appear.
- [MHR06] Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can OWL and logic programming live together happily ever after? In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*. Springer, 2006.

- [Mot06] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
- [MSS05] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, JUL 2005.
- [PST06] Bijan Parsia, Ulrike Sattler, and David Toman, editors. *Proceedings of the 2006 International Workshop on Description Logics (DL2006), Windermere, Lake District, UK, May 30 - June 1, 2006*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
- [Ros05] Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *Web Semantics Journal*, 3(1):41–60, 2005.
- [Ros06] Riccardo Rosati. *DL+log: Tight Integration of Description Logics and Disjunctive Datalog*. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 68–78. AAAI Press, 2006.
- [RV01] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 376–380, London, UK, 2001. Springer-Verlag.
- [SBLH06] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.
- [SP06] Evren Sirin and Bijan Parsia. Pellet system description. In Parsia et al. [PST06].

- [SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements,. *Artificial Intelligence*, 48:1–26, 1991.
- [Tes01] Sergio Tessaris. *Questions and answers: reasoning and querying in Description Logic*. PhD thesis, University of Manchester, 2001.
- [TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the International Joint Conference on Automated Reasoning(IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.
- [TRBH04] Dmitry Tsarkov, Alexandre Riazanov, Sean Bechhofer, and Ian Horrocks. Using Vampire to reason with OWL. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*, number 3298 in *Lecture Notes in Computer Science*, pages 471–485. Springer, 2004.