FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Inconsistencies in Hybrid Knowledge Bases

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doctor of Technical Sciences

by

Daria Stepanova
Registration Number 1125734

to the Faculty of Informatics
at the Vienna University of Technology

Advisors: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Eiter
Dipl.-Ing. Dr. techn. Michael Fink
O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhard Pichler

The dissertation has been reviewed by:

(O. Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Eiter)

(Asst. Prof. Dr. Domenico Lembo)

Vienna, 05.03.2015

(Daria Stepanova)

Erklärung zur Verfassung der Arbeit

Daria Stepanova
Favoritenstraße, 9/11, 1040, Vienna

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort und Datum) (Unterschrift der Verfasserin)

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor Prof. Thomas Eiter for his invaluable guidance, continuous support, encouragement, patience and most useful feedback during the whole period of my PhD studies. It has been truly inspirational to see how Thomas handles the multitude of demands on his time, and always finds energy for regular meetings, which never failed to produce helpful scientific discussions and advice. I particularly appreciate the painstaking detail, with which Thomas annotated my written work, and made numerous suggestions for improvement of this thesis. I am also very grateful to Dr. Michael Fink for co-supervising this work. Michael has provided plenty of insight, taught me how to write research proposals, and instilled precision in the formulation of ideas. It has been a great privilege for me to work with people of such caliber, who besides being outstanding researchers are also talented teachers. I could not have asked for better supervision.

A special thank goes to Stefan Woltran for coordinating our doctoral program, Eva Nedoma for helping with various administrative issues, Matthias Schlögel and Toni Pisjak for technical support. Moreover, I would like to acknowledge the Fonds zur Förderung der wissenschaftlichen Forschung (`http://www.fwf.ac.at`), for the financial support of the projects Reasoning in Hybrid Knowledge Bases (FWF P20840) and Evaluation of Answer Set Programs with External Source Access (FWF P24090), which I have been involved in.

Many thanks to my colleagues from the KBS group for being an example of how to undertake research work. I would like to especially thank Christoph Redl for his incredible support during the implementation stage, and for always being very responsive and eager to help. Moreover, I am grateful to Thomas Krennwallner for helping me to adapt to the new environment at the very start of my PhD project. I also reserve thanks to Guohui Xiao, Harald Beck, Jörg Pührer, Minh Dao Tran, Magdalena Ortiz, Mantas Simkus and Patrik Schneider for thought provoking discussions. Many thanks to all members of the Doctoral Kollege for the stimulating experiences which span the three and a half years of my PhD journey.

Last but certainly not least, I want to thank my beloved parents Anna and Slav, and especially my dearest brother Roman for the endless love, encouragement and inspiration that they have given me during all years of my life.

Abstract

The development of intelligent systems based on structured representation of knowledge that are capable of exhibiting human-like reasoning is a key goal of artificial intelligence in general and the field of knowledge representation research in particular. Advances in the latter have given rise to various formalisms motivated by different application domains. The most prominent families of such formalisms are ontologies in Description Logics (DLs) and nonmonotonic logic rules. While ontologies are well-suited for terminological modelling especially in the Semantic Web context, rules are widely accepted for common-sense reasoning and solving difficult search problems using declarative means. Many multi-purpose and task-specific applications, however, require a combination of ontologies and rules.

The need for such a combination has led to the development of different approaches within Hybrid Knowledge Bases (KBs). Among several others, loose coupling has emerged as a prominent approach, in which an ontology and rules are treated separately, although, a level of interaction between them is permitted via a well-defined interface. Description Logic (DL)-programs are a representative of loosely coupled hybrid KBs. The bidirectional information exchange between rules and ontology realized in DL-programs makes them powerful systems that can be effectively used for solving advanced reasoning tasks on top of ontologies.

However, this sophisticated information flow can also be a reason for inconsistencies which, as practice shows, occur in DL-programs. An inconsistent system does not yield any useful information, and as such is viewed as broken and in need of repair. Due to a possibly complex interaction between the rules and ontology, the repair problem is far from trivial and represents a significant challenge. Unfortunately, currently available engines for evaluation of DL-programs (e.g. dlvhex, DReW) suffer from the inability to handle inconsistencies with ease. This forms a major obstacle to wider acceptance of such systems.

Therefore, the aim of this thesis is the development of a sophisticated framework for handling inconsistencies in DL-programs. The main results of the effort are briefly summarized below:

- We offer a novel general repair semantics for DL-programs, and facilitate its practical applicability by applying formal methods in computer science to analyze computational complexity.

- We develop algorithms for repairing inconsistent DL-programs over lightweight DLs as well as techniques for their optimization.
- Our repair approach is implemented within the dlvhex system and evaluated on a set of benchmarks. The conducted experiments have revealed the effectiveness of our algorithms both in terms of the performance and quality of computed repairs.

Kurzfassung

Die Entwicklung von intelligenten Systemen auf Basis strukturierter Darstellung von Wissen, welche in der Lage sind, menschenähnlich zu schlussfolgern ist das wichtigste Ziel der Künstlichen Intelligenz im Allgemeinen und des Forschungsgebiets der Wissensrepräsentation und -verarbeitung im Besonderen. Fortschritte in diesem Bereich haben zu underschiedlichen Formalismen geführt, die durch verschiedene Anwendungsgebiete motiviert sind. Die prominentesten Familien von solchen Formalismen sind Ontologien basierend auf Beschreibungslogiken (engl. Description Logics, DLs) und nicht-monotone logische Regeln. Während sich Ontologien sehr gut zur Modellierung von Terminologien insbesonders im Semantic Web Kontext eignen, werden Regeln häufig für Schließen mit Hausverstand (engl. common-sense reasoning) verwendet sowie zur Lösung schwieriger Suchprobleme mit deklarativen Mitteln. Zahlreiche Mehrzweck- und aufgabenspezifische Anwendungen erfordern jedoch eine Kombination aus Ontologien und Regeln.

Der Bedarf für eine solche Kombination hat zur Entwicklung verschiedener Ansätze zu sogenannten hybriden Wissensbasen (engl. hybrid knowledge bases, hybrid KBs) geführt. Neben mehreren anderen Ansätzen ist die lose Kopplung als prominenter Ansatz entstanden, in dem eine Ontologie und Regeln getrennt behandelt werden jedoch eine Interaktionsebene dazwischen über eine gut definierte Schnittstelle erlaubt ist. Beschreibungslogik (DL) Programme sind für lose gekoppelte hybrid KBs repräsentativ. Der in DL-Programmen realisierte bidirektionale Informationsaustausch zwischen den Regeln und der Ontologie macht diese zu leistungsfähige Systemen, die effektiv zur Lösung von anspruchsvollen Schlussfolgerungsproblemen über einer Ontologie verwendet werden können.

Allerdings kann der beidseitige Informationsfluss auch ein Grund für Inkonsistenzen (Widersprüche) sein, die, wie die Praxis zeigt, bei DL-Programmen durchaus auftreten. Eine inkonsistentes System liefert keine nützliche Information; es wird als defekt und reparaturbedürftig angesehen. Aufgrund des komplexen Zusammenspiels zwischen den Regeln und der Ontologie, ist das Reparaturproblem alles andere als trivial und stellt eine bedeutende Herausforderung dar. Unglücklicherweise sind die gegenwärtig verfügbaren Softwaresysteme für die Evaluierung von DL-Programmen (z.B. dlvhex, DReW) unfähig, Unstimmigkeiten mit Leichtigkeit zu behandeln. Dies stellt ein großes Hindernis für eine breitere Akzeptanz von DL-Programmen dar.

Das Ziel dieser Arbeit ist daher die Entwicklung eines ausgefeilten Rahmens für den Umgang mit Inkonsistenzen in DL-Programmen. Die wichtigsten Ergebnisse unserer Bemühungen dazu sind nachstehend kurz zusammengefasst:

- Wir bieten eine neue, allgemeine Reparatur-Semantik für DL-Programme, und wir ermöglichen ihre praktische Anwendbarkeit durch die Anwendung formaler Methoden in der Informatik zur Analyse ihrer Berechnungskomplexität um den Entwurf zu unterstützen.
- Wir entwickeln Algorithmen für die Reparatur von inkonsistenten DL-Programmen mit Ontologien, die in Schmalspur-Beschreibungslogiken (sogenannten „Lightweight DLs") formuliert sind, sowie Verfahren zu deren Optimierung.
- Unsere Reparatur-Ansatz ist im dlvhex System implementiert und wird auf einer Reihe Teställen evaluiert. Die durchgeführten Experimente zeigen, dass unsere Algorithmen sowohl in Bezug auf die Leistung als auch auf die Qualität der berechneten Reparaturen effektiv sind.

Contents

CHAPTER 1

Introduction

This thesis is primarily concerned with the development of methods and algorithms for repairing inconsistent loosely-coupled hybrid Knowledge Bases (KBs). We focus our attention on a prominent example of hybrid KBs, the so-called DL-programs.

Roughly put, in this work we develop a framework for repairing inconsistencies in DL-programs, analyze its complexity as well as provide its implementation and thorough evaluation.

After motivating this work in the next section, we formulate the goals of this thesis in Section 1.2, and conclude by summarizing our main results in Section 1.3 and structure of this document in Section 1.4.

1.1 Motivation and Background

Artificial Intelligence (AI) is a wide and intensively growing field of research, that aims at creating intelligent machines [RN10]. These should be capable of rational thought and purposeful action to effectively interact with the surrounding environment. The tasks the intelligent systems are expected to solve require a considerable amount of complex background knowledge about the world. Knowledge representation (KR) is a subfield of AI which is primarily concerned with the study of formalisms for representing different types of knowledge as well as reasoning algorithms for such formalisms [Lev84].

The development of various logical formalisms for KR has received tremendous attention in the recent years. Here, *Description Logics (DLs)* [BCM$^+$07] is applied as a widely accepted and mature formalism, based on decidable fragments of First-Order Logic (FOL) and closely related to semantic networks [BMPS$^+$91]. DLs are relevant for constructing ontologies, which are conceptualizations of an application domain shared by various task-specific applications. Ontologies describe a given subject domain in terms of concepts, corresponding to sets of domain objects, and roles, depicting relationships among these sets. Ontologies have been remarkably successful in a number of areas, including health care, life science and Semantic Web.

In health care ontologies are extensively used for managing large terminologies e.g. SNOMED [JS08], GALEN [RGG+94], International Classification of Diseases[1]. In Biology and Chemistry they are applied for description of experimental results and other types of data, e.g. Gene Ontology, Sequence Ontology, Protein Ontology, etc. (see the BioPortal[2]). Ontologies play an important role in the Semantic Web [BF00], which is the next generation of the World Wide Web, where information stored on web pages is machine processable and can be accessed by automatic agents. In this context, ontologies are aimed at describing and structuring complex Web resources and enabling automated reasoning over them. Evident success of ontologies in a range of disciplines motivated the World Wide Web Consortium (W3C) to propose the Web Ontology Languages (OWL) as a standard for constructing ontologies.

However, DLs and consequently OWL languages, are insufficiently expressive for the requirements of certain practical problems. For instance, they cannot model closed-world reasoning, nor can they express nonmonotonicity. These features are often essential in various application scenarios. To bridge this gap, knowledge representation approaches which rely on rules in the sense of logic programming (LP) have been developed to the extent that now they are of great importance in different fields. In databases rule languages and their implementations are accepted as efficient and also sufficiently expressive means for querying data repositories. Moreover, rule languages play a vital role in AI in general, where they are effectively used as tools for declarative problem solving. Computationally hard problems can be represented in terms of nonmonotonic logic rules in such a way that the solutions of the former correspond to the models of the latter. This idea is based on the so-called *Answer Set Programming (ASP) paradigm* [GL91a].

While DLs are focused on specifying and reasoning about conceptual knowledge, logic rules serve well for reasoning about individuals; furthermore they target issues associated with nonmonotonic inference as well as non-determinism. Many applications, however, require both the features of DLs and rules. Thus, the natural solution of bringing the representatives of the two together has given rise to *Hybrid Knowledge Bases* [LNS96], which are referred to as a combination of rules and ontologies. There are two main approaches for the depicted combination: *tight-coupling* and *loose-coupling*. The tight-coupling approach is based on the idea of combining the components through a unified encoding, i.e. it aims at using a single formalism to represent the information; the examples include SWRL [HPSB+04], r-hybrid KBs [Ros05] and ELP [KRH08]. The tight-coupling approach works in some cases; however if the pieces of information encoded in the ontology and rules have (completely) different structures, this method might be very costly to implement. A better approach, in such a setting is termed loose-coupling. Although it treats ontology and rules separately, it still permits a level of interaction between them via a well-defined interface. The list of loosely-coupled KBs contains F-Logic KBs [HKE+10], which serve well for reasoning both about classes as well as individuals and DL-programs [EIL+08].

In this work we concentrate on *Description Logic (DL-)programs*, which is a promi-

[1] `http://www.who.int/classification/icd/en`

[2] `http://bioportal.bioontology.org`

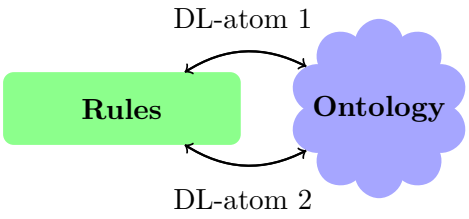


Figure 1.1: Bidirectional information flow in DL-programs

nent realization of the loosely-coupled hybrid KBS. Roughly speaking, a DL-program Π consists of a pair $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ of an ontology $\mathcal{O}$ and a rule set $\mathcal{P}$, where in the bodies of the rules in $\mathcal{P}$ so-called DL-atoms of the form $\mathrm{DL}[\lambda; Q](\vec{t})$ may occur. Informally, $Q(\vec{t})$ is a query to $\mathcal{O}$ (also known as DL-query), and λ is a list of update operations $S\,op\,p$ which specify assertions $S(\vec{c})$ resp. $\neg S(\vec{c})$ for $\mathcal{O}$ depending on the rules predicate p. These assertions are calculated from the valuations of p and temporally added to $\mathcal{O}$ before the query $Q(\vec{t})$ is evaluated. Similarly as in logic programs, the semantics of DL-programs is given in terms of their models (answer sets), which are certain interpretations over the signature of $\mathcal{P}$ [EIL$^+$08]. DL-programs are a powerful framework for knowledge representation. One of the main reasons behind the increasing popularity of DL-programs is the availability of sophisticated solvers (e.g. dlvhex, DReW).

The bidirectional information flow arranged between the ontology and the rules in DL-programs through DL-atoms (see Figure 1.1) makes them highly powerful systems, capable of effectively solving advanced reasoning problems on top of ontologies. However, there is a high possibility of recurring inconsistencies in such hybrid systems. As practice shows, conflicting information might easily arise in DL-programs, even if the ontology and rules are perfectly consistent when considered separately. An inconsistent DL-program is one that does not have any answer sets, i.e. it yields no information. Therefore, it can be viewed as broken and thus unusable.

Even in tightly-coupled hybrid KBs, which are normally built by designated programmers, and whose encoded knowledge can be coordinated to some extent, handling inconsistencies is nontrivial [HLH13]. Resolving inconsistencies in loosely-coupled hybrid KBs becomes an even more challenging task, since in the latter the components of the system are separate and they might be independently designed by different engineers.

Unfortunately, currently available systems for evaluating DL-programs suffer from the inability to solve inconsistencies with ease. This forms a major obstacle for wider acceptance of DL-programs, as it is critically important to analyze the reasons behind the faulty elements in the system as well as provide a user friendly procedure for reasoning based on such a system.

Although a large number of "inconsistency-tolerant" approaches exist (see [FGC$^+$11] for overview), most of them are applicable only to formalisms that are based on a single underlying logic. Conversely, DL-programs represent a hybrid formalism; therefore, the straightforward application of existing approaches is not suitable in the considered setting, and new methods for inconsistency handling are urgently needed. Research on inconsistencies in loosely coupled hybrid KBs was mentioned in [PHE10], but a more in-depth investigation is required, to which this thesis is devoted.

1.2 Goals

In addressing the needs of DL-programs pointed out above, the main goal of this project is to develop a powerful framework for handling inconsistencies in the considered formalism. We strive to formulate theoretical approaches for handling inconsistencies as well as to develop concrete tools which help to materialize the developed techniques. Our research initiative involves both theoretical as well as implementation work.

The key objective of our theoretical work covers the approach for inconsistency handling by repairing DL-programs, and more specifically comprises the following goals:

- ***Repair Semantics for Inconsistent DL-programs***
  The first goal of this thesis is to perform a thorough analysis of reasons for inconsistencies in DL-programs, and develop a methodology for repairing inconsistent loosely coupled hybrid KBs using DL-programs as a prominent example.

- ***Complexity Analysis***
  In real-world environment, we face certain constraints such as limited computation time, or the amount of the available working memory. Thus, the total cost of inconsistency handling process may be unjustifiably high. This demonstrates the need for the accurate evaluation of computational capabilities required for the repair approaches developed in our setting. To ensure the practical applicability, we focus on lightweight ontologies, e.g. *DL-Lite*$_{\mathcal{A}}$ and $\mathcal{EL}$. We aim at analyzing the complexity of repairing inconsistent DL-programs over these lightweight DLs.

- ***Development of Algorithms***
  The third goal of this thesis is to develop comprehensive algorithms for repairing DL-programs as well as to provide suitable optimization techniques.

At the practical level, our research work focuses on bringing together the advanced results, methodologies and algorithms developed within the project into a coherent system architecture. More specifically, we concentrate on the following practical issues:

- ***Implementation***
  As a proof-of-concept, one of our major goals is the implementation of the new algorithms emerging from our research as well as the incorporation of those into the dlvhex system[3] [EIST06]. The practical techniques required a lot of investigation, since no comparable plug-in for handling inconsistency in DL-programs existed before.

- ***Evaluation***
  Finally, our last goal is the evaluation of the developed approaches for inconsistency repair. This includes generation of benchmarks, which represents a significant challenge by itself.

[3] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

1.3 Development of this Work and Main Results

We now briefly summarize our main results and contributions.

In this thesis, we assume that the ontology and the rules of the DL-program are consistent when considered separately and the inconsistencies arise as a result of their combination. The reasons for inconsistencies therefore lie in the *wrong values of some DL-atoms* occurring in the DL-program.

1. We started our research work by identifying DL-atoms which have always the same value regardless of the ontology and interpretation of the DL-program at hand. Knowledge about such DL-atoms helps to decide whether a DL-program repair exists, and it can also be used for optimization purposes. We call such DL-atoms *independent* and developed a sound and complete calculus for their derivation. More specifically, independent DL-atoms fall into two categories: tautologic and contradictory DL-atoms. We show that checking whether a given DL-atom is independent can be done efficiently.

2. Moreover, on the theoretical level we formalize the problem of repairing DL-programs and introduce the notions of *repair* and *repair answer set*. Since the ontology repair is better studied then the rules repair, and as the rules are on top of the ontology (such that their plausibility can be separately assessed) we assume that the reasons for inconsistencies lie in the *ontology*, and in particular in it's ABox (TBox is usually well-developed). The novel notions of repair and repair answer set are, therefore, based on changes of the ontology data part that enable answer sets.

3. We show that repair answer sets do not have higher complexity than ordinary ones (more specifically, weak and FLP answer sets) in case if queries in DL-atoms can be evaluated in polynomial time. To ensure this property, we concentrate on the Description Logic *DL-Lite*$_{\mathcal{A}}$ [CLLR07], which is a prominent DL particularly useful for ontology based data access. Our complexity results are also extended to $\mathcal{EL}$ DL, as for the latter query evaluation is also tractable [BBL05].

4. As clearly not all repairs are equally attractive for a given scenario, in order to discriminate among the repairs, we introduce preference realized by a *selection function* σ, which selects preferred repairs from a set of candidates.

   Special attractive selection functions, and ones we focus on in this work, are called *independent*; they allow one to decide whether a given repair is preferred without looking at other repairs. These functions do not introduce additional complexity for computing preferred repair answer sets, e.g. bounded $\delta^{\pm}$-change, deletion, addition under bounded opposite polarity and others.

5. We show how an algorithm for evaluating DL-programs [EIST05] can be gracefully extended to compute repairs resp. repair answer sets, with possibly integrated selection criteria. In the context of this extension, we introduce a novel interesting

generalized ontology repair problem (ORP). The latter is based on an answer set candidate and DL-atoms of the program. The solution to an ORP is an ABox which ensures simultaneous entailment and non-entailment of sets of queries under possible updates.

6. The naive implementation of the repair answer set computation turns out to be ineffective in practice, since the search space of repair ABoxes is too large in general. Therefore, we propose an alternative improved approach for repair computation which is based on the notion of *support sets*. Intuitively, a support set for a ground DL-atom is a part of its input which together with the ontology TBox is sufficient to derive the DL-query. We faithfully lift support sets to the nonground level, enabling scalability of exploitation.

7. Our optimized algorithm uses *complete support families*, i.e. stocks of support sets such that the value of each DL-atom under every interpretation of the logic program can be decided without ontology access. Fortunately, for *DL-Lite*$_{\mathcal{A}}$ ontologies complete support families are small and easy to compute. Thus the idea is to precompute small support sets for all DL-atoms on a nonground level by exploiting TBox classification, and then for each candidate interpretation obtain the ground instantiations of support sets effectively. These help to prune the search space of the model candidates and also to construct the ABox repair.

8. We generalize the repair approach based on support sets for ontologies in $\mathcal{EL}$. Due to range restrictions and concept conjunctions on the left-hand side of inclusion axioms in $\mathcal{EL}$, a DL-atom accessing an $\mathcal{EL}$ ontology can have arbitrarily large and infinitely many support sets in general. Therefore, we extend repair answer set computation to deal with incomplete (*partial*) support families, such that $\mathcal{EL}$ ontologies can be handled. We formally define both ground and nonground support sets for $\mathcal{EL}$ ontologies and present techniques for their computation.

9. We present a declarative realization of algorithms dealing with both complete and partial support families for determining repair answer set within the dlvhex system[4].

10. We provide a set of benchmarks (inconsistent DL-programs) for evaluating our algorithms. On these benchmark scenarios, we estimate scalability and effectiveness of our approach; the conducted experiments reveal a promising potential of the developed repair semantics for practical settings.

1.4 Thesis Organization

The rest of the work is organized as follows.

[4] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

- In **Chapter 2**, we present some preliminaries on Description Logics and the Ontology Web Language. We, moreover, give an introduction to declarative logic programming, and in particular to the Answer Set Programming paradigm. This chapter also contains a general note on hybrid Knowledge Bases and formally introduces DL-programs. Finally, we recall some notions from the complexity theory and provide relevant complexity results from the fields of Description Logics, ASP and DL-programs.

- **Chapter 3** is on inconsistency management. We discuss the main available approaches for dealing with inconsistencies in databases, ontologies, rules and hybrid formalisms.

- **Chapter 4** deals with the repair semantics and repair answer sets that we introduce. It contains complexity analysis, and repair preferences based selection functions.

- In **Chapter 5**, the algorithms developed in this thesis are presented. This chapter comprises both a naive algorithm description and more sophisticated algorithms designed for DL-programs over *DL-Lite*$_{\mathcal{A}}$ and $\mathcal{EL}$ ontologies. We formally show the correctness of the proposed algorithms.

- In **Chapter 6**, we propose an approach for optimizing the rule part of DL-programs by eliminating independent DL-atoms, i.e. DL-atoms that always have the same value regardless of the underlying ontology. We present a calculus for identifying such independent DL-atoms and a complexity analysis of their computation.

- **Chapter 7** describes the architecture of a prototype implemented as part of the dlvhex system, as well as benchmarks that we constructed for evaluating the developed approaches. We also provide the results of our experiments, their analysis and interpretation.

- Finally, in **Chapter 8** we give a general summary of our work, present concluding remarks and give the future work outline.

1.4.1 Relevant Publications

Most of the presented results in this thesis have been published in refereed articles in the proceedings of international conferences and workshops. We introduced the independent DL-atoms and calculus for their derivation in the proceedings of the *"6th International Conference on Web Reasoning and Rule Systems (RR 2012)"* [EFS12a]. The repair semantics was presented in the proceedings of the *"23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)"* [EFS13a]. The general description of the problem of inconsistency handling in DL-programs was given in the proceedings of the *"6th International Conference on Web Reasoning and Rule Systems (RR 2013)"* [EFS13b] and the

thesis proposal in the proceedings of the *"14th Doctoral Consortium on Knowledge Representation (DC of KR 2014)"* [Ste14]. We provided the notion of support sets as optimization means for evaluating HEX-programs, which are generalizations of DL-programs accessing arbitrary external sources, in the proceedings of the *"28th Conference on Artificial Intelligence (AAAI 2014)"* [EFRS14]. The algorithms for repairing DL-programs over *DL-Lite*$_{\mathcal{A}}$ ontologies based on support sets were published in the proceedings of the *"21st European Conference on Artificial Intelligence (ECAI 2014)"* [EFS14b] and the *"7th International Workshop on Description Logics (DL workshop 2014)"* [EFS14c]. Finally, the algorithms for repairing DL-programs over $\mathcal{EL}$ ontologies were presented in the proceedings of the *"14th European Conference on Logics in Artificial Intelligence (JELIA 2014)"* [EFS14a].

CHAPTER 2

Preliminaries

In this chapter we introduce the background knowledge needed for this thesis. Section 2.1 deals with the introduction and formal details of Description Logics (DLs). We briefly present declarative logic programming approach in Section 2.2, and consider logic programs under answer set semantics (ASP) as its prominent representative in Section 2.3. The ASP and DL knowledge representation formalisms are compared, and existing approaches for their combination are reviewed in Section 2.4. A special focus is put on DL-programs in Section 2.5. We briefly introduce HEX-programs and establish their relation to DL-programs in Section 2.6. Finally, we recall the concepts of the Computational Complexity Theory, and summarize the main existing complexity results for ASP, DLs and DL-programs in Section 2.7.

2.1 Description Logics and OWL

Description Logics (DLs) belong to a family of Knowledge Representation (KR) formalisms, which can be regarded as decidable fragments of First Order Logic (FOL). Before outlining technical details, let us first look at the history and motivation behind the development of this formalism.

In 1970's-80's the subject of knowledge representation gained an increasing momentum, with two main approaches corresponding to logic-based formalisms and non-logic based ones. The former approach is focused on the idea that predicate logic has the right intuitive underpinning and thus can be used as a knowledge representation formalism. Conversely, non-logic based approaches follow a more cognitive way of recording information, and they have been more widely studied in the natural language processing community.

The most prominent formalisms among non-logic ones are *Semantic Networks* [Qui67] and *Frame-based systems* [Min85]. Semantic networks dating back to 1970's represented one of the first attempts of characterizing knowledge and reasoning of a system by means

of network-shaped structures. The nodes of these networks naturally model classes or sets of objects while edges convey information about the relationships among them. The frame-based systems stand for a similar formalism that first appeared in late 1970's. A frame can be viewed as a concept with potential attributes, which in turn correspond to properties of the instances pertaining to the specified concept. The main problem with these formalisms is the lack of formal semantics. This deficiency does not allow the reasoning process to be sufficiently precise and effective for practical applications, which was quickly realized by the research community. To overcome this shortcoming, characterization of the semantics of frames by FOL was promptly proposed by Hayes [Hay80]. The researcher also professed that the semantics of semantic networks can be mapped to FOL as well. Inspired by this result, further studies in Knowledge Representation aimed at overcoming the weakness of earlier formalisms by mapping them to various fragments of FOL. It is at this stage that the logic based and non-logic based approaches converged, giving rise to *Description Logics* (or *terminological systems* and *concept languages* as they were also known as).

DLs were extensively studied in the last decades, and now they are widely accepted as a prominent KR formalism with applications in *ontologies* and *Semantic Web* research. The main three building blocks of DLs are concepts, corresponding to sets of individuals, roles standing for (binary) relationships between individuals, and individual names, which represent single individuals in the domain.[1] For instance, *Child*, *Person*, *Animal* are concepts, intuitively denoting the sets of children, persons and animals respectively. On the other hand, *hasParent*, *hasPet* are roles, which specify the relationships between children and their parents, persons and their pets. Moreover, individual names *john*, *pat* refer to the individuals John and Pat respectively.

Starting from atomic concepts and roles, using various logical constructors one can build complex concept expressions. The most common concept constructors include ones that correspond to the boolean connectives (e.g. $\sqcap$ standing for the set intersection or logical $\wedge$, $\sqcup$ referring to the set union or logical $\vee$), and constructors that allow one to quantify over all domain elements connected through a certain role. For instance, $\exists R.C$, denotes all objects that are connected through the role R to only those objects that are in C.

Example 2.1. The concept expression $\mathit{Child} \sqcap \mathit{Male}$ denotes the set of male children, i.e. boys. The expression $\mathit{Female} \sqcap \exists \mathit{hasParent}.(\exists \mathit{hasSister}.(\mathit{Athlet} \sqcap \mathit{Singer}))$ describes all female individuals whose aunts are athletes and singers. □

A DL ontology normally comprises two reasoning components: a TBox, describing the conceptual knowledge about a domain of interest, and an ABox, specifying the factual data. More specifically, the TBox stores axioms, encoding the *subsumption* (or subset inclusion) and *disjointness* between possibly complex concepts and roles. For example, $\mathit{Male} \sqcup \mathit{Female} \sqsubseteq \mathit{Person}$ is a TBox axiom, stating that all individuals known to be either male or female are persons. The ABox contains facts in the form of ground predicates, e.g. $\mathit{Male}(\mathit{john})$, meaning that John is male. The formal semantics of DLs

[1] In FOL these can be viewed as unary predicate, binary predicate, and constant symbols.

makes it possible to infer new implicit knowledge from the information that is explicitly stated. For instance, given the TBox and ABox axioms from above we can derive the membership of $john$ in the class $Person$. The process of obtaining new inferences from the given information is called *reasoning*. The possibility of reasoning distinguishes DLs from other modern modeling languages like UML.

It is important to mention that a DL ontology does not fully describe a particular situation or state of the world, but it rather captures partial knowledge about the situation, and there may exist different states of the world that satisfy the given knowledge. In other words, not knowing that a statement is explicitly true does not imply the falsity of the statement. For example, if the only information regarding John that can be inferred from an ontology says that John is a citizen of the UK, then we can not conclude that he is not a citizen of Northern Ireland. More formally, in contrast to databases, DLs adhere to the so-called *Open World Assumption*:

Definition 2.2. *Open World Assumption (OWA)* is the assumption, under which the truth-value of a statement is independent of whether or not it is known.

Another specific feature of DL ontologies concerns the possibility of using different individual names when referring to the same object. For illustrating this feature, let us recall the famous children puzzle.

Two fathers ($f1$, $f2$) and two sons ($s1$, $s2$) went to a restaurant and bought three pizzas. When they came back, everyone had a whole pizza. How could this happen?

What confuses the reader in the problem statement is the natural habit of matching different object names (two sons and two fathers) to different objects. The solution to the puzzle reveals that there were in fact a grandfather, his son and his grandson going to the restaurant, meaning that in the problem description the constants $f2$ and $s1$ in fact referred to the same real world object. The *Unique Name Assumption* forbids such confusions in the formal knowledge description.

Definition 2.3. The *Unique Name Assumption (UNA)* is the assumption in a DL ontology that ensures that different names always refer to different entities in the world.

From the theoretical perspective, clearly, the more constructors are allowed in a DL, the more expressive it becomes, and consequently the harder it is to perform reasoning tasks in it. The search for the right balance between expressiveness and availability of effective reasoning has become one of the main scientific ambitions and overarching aims in the DL field. The progress in this direction led to an increasing popularity of ontologies in data-intensive applications, in particular, in the context of intelligent systems and data integration. Driven by this potential applicability of DLs, the idea of *Ontology Based Data Access (OBDA)* [CGL98, CLLR07, CGP12] was introduced. The latter deals with query answering over an incomplete database under the open world semantics, taking into account knowledge provided by an ontology. The *DL-Lite* [CLLR07] family of DLs was specifically designed to serve the OBDA setting. These DLs have very fruitful computational properties, and they are specifically tailored towards effective reasoning

on the one hand, and the ability to represent conceptual modeling formalism on the other hand. The basic core of the family is the *DL-Lite*$_{core}$ DL, in which simple constructs are allowed. Among more expressive extensions there are *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{A}}$, which captures all basic constructs of UML Class Diagram and the Entity Relationship (ER) model. The DL-Lite family also contains many other DLs including *DL-Lite*$_{horn}$, *DL-Lite*$_{krom}$, *DL-Lite*$_{bool}$, and so on, but going further into the details here is out of the scope of this thesis. We refer the interested reader to [ACKZ09] for more information on the DL-Lite family. The $\mathcal{EL}$ family [SPS09] is another well-studied lightweight DL family, which possesses good computational properties.

From the practical perspective a crucial research issue is the actual implementation of inferencing engines. Among the pre-DL reasoners, one can mention Krypton [BGL85], Nikl [KBR86], Loom [MB87], Classic [BBMR89], Back [Pel91], and Kris [BH91]. Unfortunately, many of these first systems were incomplete, meaning that for some ontologies the reasoners did not manage to find all logical inferences. These systems were followed by more effective implementations, which used tableaux algorithms. Among most prominent ones there are fact++[2], pellet[3] and RacerPro[4]. There are also resolution-based algorithms, which are realized in such systems as KAON2[5] [MS06] and HermiT [MSH07, MSH14]. Furthermore, lately the "consequence-based" approaches were implemented in the ELK[6] system [KKS13]. Due to the integration of various advanced optimization techniques, these systems work pretty well on practical instances, despite employing algorithms of high worst-case complexity. This observation allowed for the development of various real-world ontology applications, encoding domain knowledge from disciplines such as geoscience [Goo05, RP05, FMC+09], bioinformatics [The00, SAR+07, SGB00], medicine [SMK+14, SCC97, SGM95] or electrical engineering [UD07].

In our work we focus on two lightweight DLs: *DL-Lite*$_{\mathcal{A}}$ and $\mathcal{EL}$, for which common inference tasks can be solved in polynomial time. The first reason for such a choice obviously stems from their benign computational properties. The second reason is their wide practical applicability: as mentioned *DL-Lite*$_{\mathcal{A}}$ is actively used in the OBDA setting, and $\mathcal{EL}$ proved to be particularly useful for many application domains like biology, medicine, chemistry, policy, etc.

Now we move to the formal syntactic and semantical descriptions of the DLs relevant for this thesis. We consider Description Logic (DL) knowledge bases (KBs) over a signature $\Sigma_o = \langle \mathbf{I}, \mathbf{C}, \mathbf{R} \rangle$ with a set $\mathbf{I}$ of individuals (constants), a set $\mathbf{C}$ of concept names (unary predicates), and a set $\mathbf{R}$ of role names (binary predicates). The sets $\mathbf{I}$, $\mathbf{C}$ and $\mathbf{R}$ are countable.

Definition 2.4. A *DL knowledge base* (or *ontology*) is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ of a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$, which are finite sets of formulas capturing taxonomic resp. factual

[2] https://code.google.com/p/factplusplus
[3] http://clarkparsia.com/pellet/
[4] http://racer.sts.tuhh.de
[5] http://kaon2.semanticweb.org/
[6] https://code.google.com/p/elk-reasoner/

knowledge, whose form depends on the underlying DL.

We denote by $\mathsf{sig}(E)$ the signature of E, where E can be any set of axioms from a given ontology. Throughout the work, we assume that ontologies are under UNA, i.e. different names always denote different individuals. In what follows we describe the formal syntax and semantics of *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ DLs. For overview of other DLs, we refer the reader to [BCM$^+$07].

2.1.1 *DL-Lite*$_\mathcal{A}$: Syntax and Semantics

Syntax. In *DL-Lite*$_\mathcal{A}$, concepts C, and roles R are formed according to the following syntax:

$$C \rightarrow A \mid \exists R \qquad B \rightarrow C \mid \neg C \qquad R \rightarrow U | U^-$$

where $A \in \mathbf{C}$ is an atomic concept, $U \in \mathbf{R}$ an atomic role and U^- denotes the inverse of the atomic role U. *DL-Lite*$_\mathcal{A}$ TBox axioms are then of the form:

$$\begin{array}{lll} C_1 \sqsubseteq C_2, & C_1 \sqsubseteq \neg C_2, & \\ R_1 \sqsubseteq R_2, & R_1 \sqsubseteq \neg R_2, & (\mathit{func}\ R). \end{array}$$

Axioms in the first column are *positive inclusions* (among concepts and roles, respectively), while those in the second column are *disjointness axioms*. Finally, the axiom in the third column (*func* R) is a *functionality axiom*.

An *assertion* is a formula $A(c)$ or $U(c, d)$ where $A \in \mathbf{C}$, $U \in \mathbf{R}$, and $c, d \in \mathbf{I}$ (called *positive*) or its negation, i.e., $\neg A(c)$ resp. $\neg U(c, d)$ (*negative*).[7] The ontology ABox contains a set of assertions over the ontology signature Σ_o.

Example 2.5. An example of a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}$ is given below.

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Child} \sqsubseteq \exists \mathit{hasParent} & (4)\ \mathit{Male}(\mathit{pat}) \\ (2)\ \mathit{Adopted} \sqsubseteq \mathit{Child} & (5)\ \mathit{Male}(\mathit{john}) \\ (3)\ \mathit{Female} \sqsubseteq \neg \mathit{Male} & (6)\ \mathit{hasParent}(\mathit{john}, \mathit{pat}) \end{array} \right\}$$

The axioms (1)-(3) form the ontology TBox, while the assertions (4)-(6) represent the ABox part of $\mathcal{O}$. Intuitively, the TBox states that every child has a parent, every adopted child is a child, and male and female are disjoint classes. □

Semantics. We now turn to the semantics of *DL-Lite*$_\mathcal{A}$ DL. The semantics of DL ontologies $\mathcal{O}$ is based on first-order interpretations [BCM$^+$07], [CLLR07].

Definition 2.6. An interpretation $\mathcal{I}$ is a pair $\langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$, where $\Delta^\mathcal{I}$ is a non-empty interpretation domain, and $\cdot^\mathcal{I}$ is an interpretation function from $\mathcal{I}$ to $\Delta^\mathcal{I}$ which assigns to

Construct	Syntax	Semantics
Individual	a	$a^{\mathcal{I}}$
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Atomic role	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Inverse role	R^-	$\{(o, o') \mid (o, o') \in R^{\mathcal{I}}\}$
Unqualified exist. restriction	$\exists R$	$\{o \mid \forall o'.(o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
Concept negation	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
Role negation	$\neg U$	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \backslash U^{\mathcal{I}}$

Table 2.1: DL *DL-Lite*$_{\mathcal{A}}$ concept and role constructors

each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$. The constructs allowed in *DL-Lite*$_{\mathcal{A}}$ are interpreted as shown in Table 2.1:

The *satisfiability* of an axiom α w.r.t. an interpretation $\mathcal{I}$ is defined as follows:

- $\mathcal{I} \models C(a)$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$;
- $\mathcal{I} \models R(a, b)$, if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$;
- $\mathcal{I} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;
- $\mathcal{I} \models \neg C(a)$, if $a^{\mathcal{I}} \in \Delta \backslash C^{\mathcal{I}}$;
- $\mathcal{I} \models funct(R)$, if the binary relation R is a function such that $(o_1, o_2) \in R^{\mathcal{I}}$ and $(o_1, o_3) \in R^{\mathcal{I}}$ implies that $o_2 = o_3$.

An axiom (resp. TBox, ABox or ontology) is *satisfiable* (or consistent) if it is satisfiable w.r.t. some interpretation $\mathcal{I}$.

We call an ABox $\mathcal{A}$ *consistent with* a TBox $\mathcal{T}$, if $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent, otherwise $\mathcal{A}$ is *inconsistent with* a TBox. An ontology $\mathcal{O}$ is *inconsistent*, if $\mathcal{A}$ is inconsistent with $\mathcal{T}$.

In *DL-Lite*$_{\mathcal{A}}$ ontologies, inconsistency arises by few assertions.

Proposition 2.7 (cf. [CLLR07])**.** *In DL DL-Lite*$_{\mathcal{A}}$*, given an arbitrary TBox* $\mathcal{T}$*, every* $\subseteq$*-minimal ABox* $\mathcal{A}$ *such that* $\mathcal{T} \cup \mathcal{A}$ *is inconsistent fulfills* $|\mathcal{A}| \leq 2$*.*

2.1.2 $\mathcal{EL}$: Syntax and Semantics

The DL $\mathcal{EL}$ is another DL highly relevant for representing lightweight ontologies.

Syntax. In $\mathcal{EL}$ [SPS09] concepts C and roles R obey the following syntax, where $A \in \mathbf{C}$ is an atomic concept and $R \in \mathbf{R}$ is an atomic role:

[7] Negative assertions $\neg F(\vec{t})$ are easily compiled to positive ones using a fresh concept resp. role name $F_\neg$ and $F_\neg(\vec{t})$, $F_\neg \sqsubseteq \neg F$.

$$C \to A \qquad B \to C \mid C \sqcap D \mid \exists R.C \qquad R \to U$$

TBox axioms are of the form $B_1 \sqsubseteq B_2$ (inclusion axiom); ABox assertions are of the form $A(a)$ and $U(a, b)$, where $A \in \mathbf{C}$, $U \in \mathbf{R}$ and $a, b \in \mathbf{I}$.

Example 2.8. As an example let us consider the following ontology from the policy domain [BFS10], whose taxonomy (TBox) $\mathcal{T}$ is given by (1)-(3), while (4)-(9) is a sample data part (ABox) $\mathcal{A}$.

$$\mathcal{O} = \left\{ \begin{array}{lll} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} & & \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAction}.\mathit{Action} \sqcap \exists \mathit{hasSubject}.\mathit{Staff} \sqcap \exists \mathit{hasTarget}.\mathit{Project} & & \\ (3)\ \mathit{BlacklistedStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubject}.\mathit{Blacklisted} & & \\ (4)\ \mathit{StaffRequest}(\mathit{r1}) & (5)\ \mathit{hasSubject}(\mathit{r1}, \mathit{john}) & (6)\ \mathit{Blacklisted}(\mathit{john}) \\ (7)\ \mathit{hasTarget}(\mathit{r1}, \mathit{p1}) & (8)\ \mathit{hasAction}(\mathit{r1}, \mathit{read}) & (9)\ \mathit{Action}(\mathit{read}) \end{array} \right\}.$$

Intuitively, (1) states that every blacklisted staff is staff, while (2) and (3) provide definition of the classes *StaffRequest* and *BlacklistedStaffRequest*. For instance, *StaffRequest* must have an action, a member of staff as a subject, and a project as a target. □

In this work we will need the notion of a normalized TBox, which is its restricted syntactic form, defined in the following way.

Definition 2.9. A TBox is *normalized*, if all of its axioms have one of the following forms:

$$A_1 \sqsubseteq A_2 \qquad A_1 \sqcap A_2 \sqsubseteq A_3 \qquad \exists R.A_1 \sqsubseteq A_2 \qquad A_1 \sqsubseteq \exists R.A_2,$$

where A_1, A_2, A_3 are atomic concepts.

For instance, the axioms (1) and (2) in Example 2.8 are in normal form, while axiom (3) is not.

The following result is instrumental:

Proposition 2.10 (cf. [SPS09])**.** *For any $\mathcal{EL}$ TBox, an equivalent TBox in normal form is constructible in linear time (over an extended signature).*

Example 2.11. The normalized form of the TBox from Example 2.8 looks as follows:

$$\mathcal{T}_{norm} = \left\{ \begin{array}{l} (1*)\ \mathit{StaffRequest} \sqsubseteq \exists \mathit{hasAction}.\mathit{Action} \\ (2*)\ \mathit{StaffRequest} \sqsubseteq \exists \mathit{hasSubject}.\mathit{Staff} \\ (3*)\ \mathit{StaffRequest} \sqsubseteq \exists \mathit{hasTarget}.\mathit{Project} \\ (4*)\ \exists \mathit{hasAction}.\mathit{Action} \sqsubseteq C_{\exists \mathit{hasA}.A} \\ (5*)\ \exists \mathit{hasSubject}.\mathit{Staff} \sqsubseteq C_{\exists \mathit{hasS}.\mathit{St}} \\ (6*)\ \exists \mathit{hasTarget}.\mathit{Project} \sqsubseteq C_{\exists \mathit{hasT}.P} \\ (7*)\ C_{\exists \mathit{hasA}.A} \sqcap C_{\exists \mathit{hasS}.\mathit{St}} \sqsubseteq C_{\exists \mathit{hasA}.A \sqcap \exists \mathit{hasS}.\mathit{St}} \\ (8*)\ C_{\exists \mathit{hasA}.A \sqcap \exists \mathit{hasS}.\mathit{St}} \sqcap C_{\exists \mathit{hasT}.P} \sqsubseteq \mathit{StaffRequest} \end{array} \right\},$$

where C with subscripts are fresh concept names. □

Construct	Syntax	Semantics
Individual	a	$a^{\mathcal{I}}$
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Atomic role	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Qualified exist. restriction	$\exists R.C$	$\{o \mid \exists o'.(o, o' \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}})\}$

Table 2.2: DL $\mathcal{EL}$ concept constructors

Construct	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Atomic role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Full negation	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Unqualified exist. restriction	$\exists R$	$\{o \mid \forall o'.(o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
Bottom	$\bot$	$\emptyset$

Table 2.3: DL $\mathcal{ALC}$ concept constructors

Semantics. The semantics for the DL $\mathcal{EL}$ as for all other DLs is based on first-order interpretations [BCM$^+$07], the interpretation of the constructors allowed in $\mathcal{EL}$ is depicted in Table 2.2.

2.1.3 Beyond Lightweight DLs

Apart from lightweight DLs there are more expressive DLs, The DL known as $\mathcal{ALC}$ is considered the "basic" expressive description logic because it is the minimal one that supports unrestricted use of the basic concept constructors: conjunction, disjunction, negation, and existential and universal restrictions. The term expressive Description Logics usually refers to $\mathcal{ALC}$ and its extensions. Table 2.3.

In various expressive DLs different constructors are allowed, e.g. $\mathcal{ALCFI}$ is $\mathcal{ALC}$ in which additionally functional and inverse roles are allowed. Other expressive DLs that we mention in this thesis include $\mathcal{SHIF}$, $\mathcal{SHOIN}$, $\mathcal{SHOIQ}$.

Apart from letters explained in Table 2.4,

- $\mathcal{S}$ stands for $\mathcal{ALC}$ extended with transitive roles,
- $\mathcal{H}$ stands for role hierarchies,

Construct	$\mathcal{ALC}$	Syntax	Semantics
Disjunction	$\mathcal{U}$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Qualified exist. restriction	$\mathcal{E}$	$\exists R.C$	$\{o \mid \exists o'.(o, o' \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}})\}$
Unqualified number restrictions	$\mathcal{N}$	$\geq k\, R$	$\{o \mid \#\{o' \mid (o, o') \in R^{\mathcal{I}}\} \geq k\}$
		$\leq k\, R$	$\{o \mid \#\{o' \mid (o, o') \in R^{\mathcal{I}}\} \leq k\}$
Qualified number restrictions	$\mathcal{Q}$	$\geq k\, R.C$	$\{o \mid \#\{o' \mid (o, o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \geq k\}$
		$\leq k\, R.C$	$\{o \mid \#\{o' \mid (o, o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq k\}$
Inverse role	$\mathcal{I}$	R^{-}	$\{(o, o') \mid (o, o') \in R^{\mathcal{I}}\}$
Top		$\top$	$\Delta^{\mathcal{I}}$

Table 2.4: Additional concept and role constructors

- $\mathcal{F}$ stands for functionality of roles,
- $\mathcal{O}$ stands for nominals, which means the possibility of using individuals in the TBox.

2.1.4 Ontology Main Reasoning Tasks

The typical reasoning tasks that are associated with DLs include [CLLR07]:

- *Ontology satisfiability* – given an ontology $\mathcal{O}$, verify whether $\mathcal{O}$ admits at least one model;
- *Logical implication of KB assertions*, which consists of the following sub- problems:
  - *Instance checking* – given an ontology $\mathcal{O}$, a concept C and a constant a (resp., a role E and a pair of constants a and b), verify whether $\mathcal{O} \models C(a)$ (resp., $\mathcal{O} \models R(a, b)$);
  - *Subsumption of concepts or roles* – given a TBox $\mathcal{T}$ and two general concepts C_2 and C_2 (resp., two general roles R_1 and R_2), check whether $\mathcal{T} \models C_1 \sqsubseteq C_2$ (resp., $\mathcal{T} \models R_1 \sqsubseteq R_2$);
  - *$\mathcal{T}$-classification* – given a TBox $\mathcal{T}$ compute all subsumption relations followed from $\mathcal{T}$;
  - *Checking functionality* – given a TBox $\mathcal{T}$ and a basic role R, decide whether $\mathcal{T} \models funct(R)$;
- *Query answering* – given an ontology $\mathcal{O}$ and a query q (either a conjunctive query or a union of conjunctive queries) over $\mathcal{O}$, compute the set of answers to q.

2.1.5 OWL: Ontology Web Language

Since the 1990's a number of research efforts have been pursued in terms of applying the idea of knowledge representation from artificial intelligence to the World Wide Web context. This led to the pioneering Semantic Web paradigm. Back then the ultimate goal was to make the semantical side of the Web content more accessible to machines for which the ontological representation of the web pages proved to be more appropriate. The RDF language, and its schematic extension RDF schema (RDFS), were proposed for data interchange in the Web. RDF allows to specify ground binary predicates, and in RDFS definition of subclass relationships, property hierarchies as well as domain and range restrictions on the corresponding properties are possible. However, for encoding ontologies in even lightweight DLs richer constructs are necessary, e.g. class disjointness, range restrictions which are applied only to certain classes, etc. This motivated the development of more powerful ontology languages like Simple HTML Ontology Extensions (SHOE) [HH00], DAML-OIL and DAML-ONT (see [HPSvH03] for earlier ontology languages). Evolution of these first languages, their extension and enhancement has led to the modern Ontology Web Language (OWL), which still partially relies on the RDFS fragments.

OWL can now be regarded as a well-established standard for ontology modeling, and it is widely applied not only in the Semantic Web context, but also in various subject domains including biology, medicine, and many others. The first version of the OWL standard (OWL 1) was delivered in 2004; later, in 2008 its extended and revised successor OWL 2 was released.

The theoretical background of OWL is tightly related to DLs. As we discussed in the previous subsections, DLs have been grouped into families according to their expressive power and computational properties. The grouping criteria is convenient, and therefore it is also applied in OWL, creating further division into three sublanguages: OWL Lite, OWL DL, and OWL Full. OWL Lite is suitable for representing a classification hierarchy with simple constraints, and it is the sublanguage with lower expressive capabilities then the other two. The second expressive sublanguage is OWL DL, which maintains maximal expressiveness while still permitting completeness of computational reasoning and decidability. Finally, OWL Full is the most expressive sublanguage; however, the computational completeness in this language can no longer be guaranteed.

The OWL DL fragment is in turn split into OWL-QL, OWL-RL and OWL-EL, which are lightweight sublanguages with restricted modeling features, but significantly simplified reasoning algorithms. OWL-EL is popular in large biomedical ontologies, OWL-RL is a useful language for reasoning with Web data, and OWL-QL is widely recognized in database applications with an ontological data access layer. Crucially, main computational tasks can be efficiently performed in these languages.

Earlier, it has already been mentioned that in our work we concentrate on ontologies in *DL-Lite*$_{\mathcal{A}}$ and $\mathcal{EL}$, which belong to the OWL 2 QL and OWL 2 EL sublanguages respectively. This is depicted in Figure 2.1. Furthermore, the syntax and semantics of these sublanguages are discussed.

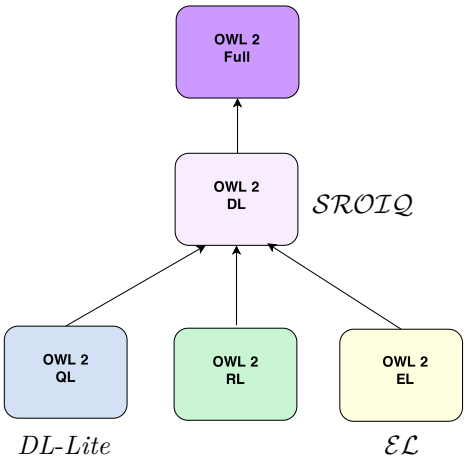


Figure 2.1: OWL 2 sublanguages vs DL fragments

OWL Syntax and Semantics.

The OWL language comes in different syntaxes. The syntactic formats range from the W3C officially required RDF/XML exchange syntax [GS14], to various optional ones including Turtle [Bec04], OWL/ XML [MPPS12b], the Functional-Style Syntax [MPPS12a], the Manchester Syntax [HDG+06]. We focus on the RDF-XML syntax, as it is the most widely used one. The correlation between DL constructs and their syntactic representation in OWL is shown in Table 2.5.

Example 2.12. The class disjointness axiom (2) of Example 2.5 in RDF/XML syntax looks as follows:

```
<owl:Class rdf:about="#Male">
      <owl:disjointWith rdf:resource="#Female"/>
</owl:Class>
```

A more involved equivalence axiom (2) from Example 2.8 is represented as:

```
<owl:Class rdf:about="#StaffRequest">
      <owl:equivalentClass>
          <owl:Class>
              <owl:intersectionOf rdf:parseType="Collection">
                  <owl:Restriction>
                      <owl:onProperty rdf:resource="#hasAction"/>
                      <owl:someValuesFrom rdf:resource="#Action"/>
```

	RDF/XML Syntax	DL Syntax
Axioms	C_1 `rdfs:subClassOf` C_2	$C_1 \sqsubseteq C_2$
	C_1 `owl:disjointWith` C_2	$C_1 \sqsubseteq \neg C_2$
	R_1 `rdfs:subPropertyOf` R_2	$R_1 \sqsubseteq R_2$
	R_1 `owl:propertyDisjointWith` R_2	$R_1 \sqsubseteq \neg R_2$
	C_1 `owl:equivalentclass` C_2	$C_1 \equiv C_2$
	a `rdf:type` C	$C(a)$
	a R b	$R(a, b)$
	`rdf:type owl:NegativePropertyAssertion` `owl:sourceIndividual` a `owl:assertionProperty` R `owl:targetIndividual` b	$\neg R(a, b)$
	R `rdfs:domain` C	$\exists R.\top \sqsubseteq C$
Class expressions	`owl:complementOf` C	$\neg C$
	`owl:intersectionOf`	$C_1 \sqcap C_2$
	`owl:Thing`	$\top$
	`owl:Nothing`	$\bot$
	`owl:restriction owl:onProperty` R `owl:someValuesFrom` C	$\exists R.C$
Property expressions	`owl:inverseOf`	R^-
	R `rdf:type ref:resource="&owl:FunctionalProperty"`	$funct(R)$

Table 2.5: Correspondence between OWL syntactic expressions and DL constructs allowed in *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$

```
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasSubject"/>
                        <owl:someValuesFrom rdf:resource="#Staff"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasTarget"/>
                        <owl:someValuesFrom rdf:resource="#Project"/>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
</owl:Class>
```

□

There are two alternative semantics for OWL ontologies with a correspondence theorem providing a link between them. The first is *Direct Semantics* [HPS12], which assigns meaning directly to ontology structures and is compatible with the DL semantics. The second is the *RDF-based semantics* [CHPS09], which assigns meaning to the RDF graphs. The correspondence theorem defines a precise, close relationship between the Direct and RDF-Based Semantics (see [Krö12] for details). This theorem states that

given an OWL 2 DL ontology, inferences drawn using the Direct Semantics will still be valid if the ontology is mapped into an RDF graph and interpreted using the RDF-Based Semantics.

2.2 Nonmonotonic Logics and Declarative Logic Programming

As discussed in previous sections Description Logics have been accepted as a suitable technical tool for the representation of expert knowledge from various subject domains because of their well-defined semantics and a powerful inference generating mechanism. This tool, however, is not fully adequate for the *commonsense reasoning*, i.e. the ability of simulating the human way of making deductions about everyday situations. The main difficulty stems from the so-called "monotonicity" of the DLs. A logic is *monotonic* if addition of new axioms to a theory in the logic never leads to the loss of any previously derived conclusions from that theory. In other words, whenever a sentence ϕ is a logical consequence of a set of sentences T, ϕ is also entailed from any superset of T. Commonsense reasoning is *nonmonotonic*: we are often forced to withdraw previous conclusions when new knowledge has been obtained. Motivated by this observation, nonmonotonic logics resp. logic-based formalisms have been developed and investigated. The most well-known among them are circumscription [McC80, McC84, Lif85], default logic [Rei80], and nonmonotonic modal logics [MD80, McD82, Moo85]. For an overview of other nonmonotonic formalisms see [Gin87] and references therein.

Another separate direction of research deals with logic programming as a subclass of declarative programming languages, which appeared from the idea of combining logical knowledge representation means and the theory of automated deduction. The first and probably the most well-known logic programming language was Prolog [War77, CKC81, Llo87, Kow88], which at the very start of its development was defined as a small subset of predicate calculus. Further it was extended with nonmonotonic features and the *negation as failure* operator. The initial definition of nonmonotonic features in Prolog was procedural, and then a declarative semantical characterization was given. The latter development established a link between the logic programming and nonmonotonic logics as mentioned above.

Despite being a powerful logic programming language, Prolog still has several aspects which make its applicability for knowledge representation less apparent [EIK09]. Among such aspects [EIK09] mentions the following:

- *Incomplete information handling*, that is the ability to properly complete the missing information with default assumptions, is not supported in Prolog;
- *Termination issue* is striking in Prolog;
- *Ordering of rules in a program and atoms in a rule* is important in Prolog[8], which makes modeling tasks in various subject domains less convenient;

[8]SLD resolution and backtracking are used in Prolog as evaluation techniques.

- *Preference handling*, as the possibility to describe which solutions are preferred to others with respect to some "quality" criterion, is not easily realizable in Prolog.

To overcome the described shortcomings, the Answer Set Programming (ASP) paradigm was proposed in 1991 [GL91a]. Essentially, ASP programs are closely related to Prolog programs. The main difference stems from the semantics, which in the ASP case is purely model-theoretic, and thus the termination of ASP programs is always guaranteed. In what follows we present ASP in more detail.

2.3 Logic Programs under the Answer Set Semantics

Answer Set Programming (ASP) has appeared as a declarative problem solving paradigm oriented towards difficult search problems. ASP has its roots in Logic Programming and Nonmonotonic Reasoning. This formalism is well-suited for modeling and (automatically) solving problems which involve the previously mentioned commonsense reasoning. The basic idea of ASP is to describe problem specifications by means of a nonmonotonic logic program: solutions to instances of such a problem are represented by the intended models (the so-called answer sets, or stable models) of the program at hand. Rules and constraints, which describe the problem and its possible solutions rather than a concrete algorithm, are basic elements of the ASP programs. Such a problem encoding can be then be sent to an answer set (AS) solver as an input. The ASP solver computes some models (answer sets) of the program, from which the solutions of the problem can be extracted. This is done instead of the computation of proofs when dealing with the prolog programs.

The availability of advanced reasoners makes the ASP formalism applicable to solving practical tasks. The standard reasoner comprises two components: a grounder and a solver. The list of the most well-known grounders include: DLV[9], Gringo[10], LParse[11]. Among the solvers one can mention DLV, SModels[12], CModels[13], clasp[14], claspD[15], GnT[16]. Potassco[17] represents a collection of ASP reasoning tools, combining clasp and Gringo into a system architecture. The performance of the ASP solvers is annually appraised at the dedicated ASP competition, with the latest results of 2014 are available at `https://www.mat.unical.it/aspcomp2014/`.

Next we present the formal syntax and semantics of ASP.

Syntax. Let $(\mathbf{P}, \mathcal{C})$ be a function-free first-order vocabulary, consisting of nonempty finite sets $\mathbf{P}$ of predicates and $\mathcal{C}$ of constants. Let, moreover, $\mathbf{V}$ be a set of variables.

[9] `http://www.dlvsystem.com/`
[10] `http://potassco.sourceforge.net/`
[11] `http://www.tcs.hut.fi/Software/smodels/`
[12] `http://www.tcs.hut.fi/Software/smodels/`
[13] `http://www.cs.utexas.edu/~tag/cmodels/`
[14] `http://potassco.sourceforge.net/`
[15] `http://www.cs.uni-potsdam.de/claspD/`
[16] `http://www.tcs.hut.fi/Software/gnt/`
[17] `http://potassco.sourceforge.net/`

We now formally define the building components of logic programs.

- A *term* is either a constant from $\mathcal{C}$ or a variable from $\mathbf{V}$.
- An *atom* is defined as follows $p(t_1, \ldots, t_l)$, where $p \in \mathbf{P}$, each t_i is a term and l is a positive integer, denoting the arity of p.
- A *classical literal* is a positive atom a, and a literal of the form $not\ \ a$ is called a *default-negated* literal.

Definition 2.13. A *disjunctive rule* is a formula of the form

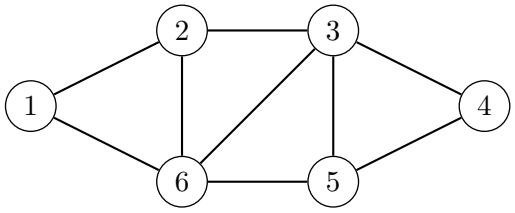
$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, not\ \ b_{k+1}, \ldots, not\ \ b_m, \tag{2.1}$$

where $1 \leq k \leq m$, every a_i $(1 \leq i \leq n)$, is a classical literal, and not denotes default negation.

A rule r of form 2.1 is called a *fact* if $n = 1$ and $m = 0$. We sometimes omit the symbol $\leftarrow$ when referring to facts. A rule without head literals, (i.e. $n = 0$) is a *constraint*. A rule is *positive* if $k = m$. Moreover,

- $H(r) = \{a_1, \ldots, a_k\}$ is called the *head* of r,
- $B(r) = \{b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_n\}$ is called the *body* of r,
  - the set $B^+(r) = \{b_1, \ldots, b_k\}$ is the *positive body* of r, and
  - the set $B^-(r) = \{b_{k+1}, \ldots, b_m\}$ is the *negative body* of r.
- An *extended disjunctive logic program (EDLP)* (also known as DATALOG$^{\vee,\neg}$) is a finite set of rules of the form 2.1.
- A program without disjunction in the heads (i.e. $n = 1$ for all rules of $\mathcal{P}$) is called *extended logic programs* or *normal logic program* (or DATALOG$^{\neg}$).
- An extended logic program without default-negated literals (i.e. all rules r of $\mathcal{P}$ are such that $B^-(r) = \emptyset$) is a *positive logic program* (or DATALOG).
- Furthermore, DATALOG$^{\vee}$ refers to a positive logic program, in which disjunction in heads of rules is allowed.
- If all predicates occurring in a program $\mathcal{P}$ are of arity $l = 0$, then $\mathcal{P}$ is called *propositional*.

Example 2.14. Consider a graph 3-colourability problem, in which we are given a graph and three colours, e.g. red, blue and green, and we aim at finding the assignment of colours to the nodes of a graph in such a way that no adjacent nodes share the same color. The encoding of this problem is as follows.



$$\mathcal{P}^I = \left\{ \begin{array}{l} (1)\ node(1 \ldots 6); \quad (2)\ \ edge(1,2); \quad \ldots \quad (8)\ edge(5,6); \\ (9)\ coloured(V, red) \leftarrow not\ \ coloured(V, blue), not\ \ coloured(V, green), node(V); \\ (10)\ coloured(V, green) \leftarrow not\ \ coloured(V, blue), not\ \ coloured(V, red), node(V); \\ (11)\ coloured(V, blue) \leftarrow not\ \ coloured(V, green), not\ \ coloured(V, red), node(V); \\ (12)\ \bot \leftarrow coloured(V, C), coloured(V, C'), C \neq C'; \\ (13)\ \bot \leftarrow coloured(V, C), coloured(V', C), edge(V, V') \end{array} \right\}$$

The facts of the programs $\mathcal{P}$ describe the nodes and edges of the graph from above. The rules (9)-(11) state that each node has to be colored in at least one of the three colours. The constraint (12) forbids the nodes to be colored in more then one color, while the constraint (13) says that two nodes connected via an edge must have different colours. □

Semantics. The semantics of extended logic program is defined for ground (variable-free) programs. Given a program $\mathcal{P}$ over a signature $\Sigma_{\mathcal{P}} = \langle \mathbf{P}, \mathcal{C} \rangle$, we define the *Herbrand universe* and the *Herbrand base* of $\mathcal{P}$ as follows:.

Definition 2.15. The *Herbrand universe* of a logic program $\mathcal{P}$, denoted by $HU_{\mathcal{P}}$ is the set $\mathcal{C}$. If there are no constants in $\mathcal{P}$, then $HU_{\mathcal{P}} = \{c\}$, where c is an arbitrary constant symbol.

Definition 2.16. The *Herbrand base* of a logic program $\mathcal{P}$, denoted by $HB_{\mathcal{P}}$ is the set of all atoms constructed using predicates from $\mathbf{P}$ and constants from $\mathcal{C}$.

The terms, atoms, rules, programs are *ground*, if they do not contain variables. A *ground instance of a rule* $r \in \mathcal{P}$ is obtained from r by replacing all variables in r with constants from $\mathcal{C}$. A *ground program*, denoted by $ground(\mathcal{P})$, corresponds to a set of all ground instances of all rules in $\mathcal{P}$.

Example 2.17 (cont'd)**.** For the logic program $\mathcal{P}$ from Example 2.14, the grounding $ground(\mathcal{P})$ is obtained by substituting the variables V, V', C, C' with constants from the set $\{1, 2, 3, 4, 5, 6, blue, green, red\}$ in all possible ways. □

The semantics of EDLPs is given first for positive ground programs. Before presenting it formally, we define several important notions first.

- A set $S \subseteq HB_{\mathcal{P}}$ of literals is called *consistent*, if $\{p, \neg p\} \not\subseteq S$ for every atom $p \in HB_{\mathcal{P}}$.

- An *interpretation* I relative to a program $\mathcal{P}$ is a consistent subset of $HB_{\mathcal{P}}$.

A *satisfaction relation* of an interpretation with respect to a certain element is defined as follows:

Definition 2.18. Let $\mathcal{P}$ be a ground logic program. An interpretation $I \subseteq HB(\mathcal{P})$ satisfies

- a literal a, if $a \in I$;
- a default negated literal $not\ \ a$, if $a \notin I$;
- a set of literals, if it satisfies each literal separately;
- a rule r, if $H(r) \cap S \neq \emptyset$ whenever $B^+(r) \subseteq S$ and $B^-(r) \cap S = \emptyset$;
- a logic program $\mathcal{P}$ (I is a model of $\mathcal{P}$), if it satisfies all rules r of $\mathcal{P}$.

Example 2.19. Recall the program from Example 2.14, and let us look at the interpretation $I = \{coloured(1, red), node(1), node(2)\}$ and the following ground rules:

$r_1 : coloured(1, red) \leftarrow not\ \ coloured(1, blue), not\ \ coloured(1, green), node(1);$
$r_2 : coloured(2, green) \leftarrow not\ \ coloured(1, blue), not\ \ coloured(1, red), node(2)$

We have that $I \models node(1)$. Furthermore, $I \models r_1$, and $I \not\models r_2$. □

Definition 2.20. A model I of a logic program $\mathcal{P}$ is called *minimal*, if there is no model I' of $\mathcal{P}$, such that $I' \subset I$.

The semantics of ASP programs is given in terms of *answer sets* (or *stable models*).

Definition 2.21. Given a positive logic program $\mathcal{P}$, an interpretation I is an answer set (stable model) of $\mathcal{P}$ if $I \models \mathcal{P}$ and there does not exist $I' \subset I$, such that $I' \models \mathcal{P}$. $AS(\mathcal{P})$ denotes the set of all answer sets.

The notion of answer sets for logic programs with negation is defined using the *Gelfond-Lifshitz reduct*.

Definition 2.22. Given a logic program $\mathcal{P}$ and an interpretation I of $\mathcal{P}$ the *Gelfond-Lifshitz* (GL-)reduct denoted by $\mathcal{P}^I_{gl}$ is constructed from $ground(\mathcal{P})$ by

- deleting all rules r from $ground(\mathcal{P})$ s.t. $B^-(r) \cap I \neq \emptyset$, and
- deleting the negative body for all of the remaining rules.

Note that for any program and interpretation the GL-reduct is positive. The definition of answer sets for arbitrary logic programs is then as follows:

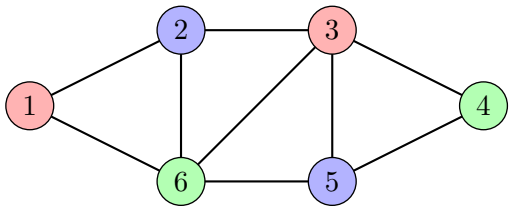
Definition 2.23. An *answer set* of a program $\mathcal{P}$ is an interpretation $I \subseteq HB_{\mathcal{P}}$, such that I is an answer set of the positive program $\mathcal{P}^I_{gl}$.

Example 2.24 (cont'd)**.** Consider an interpretation I of the program $\mathcal{P}$ from Example 2.14, which apart from the graph description contains the facts: $coloured(1, red)$, $coloured(2, blue)$, $coloured(3, red)$, $coloured(4, green)$, $coloured(5, blue)$, $coloured(6, green)$.

The GL-reduct $\mathcal{P}^I_{gl}$ of $\mathcal{P}$ is as follows:

$$\mathcal{P}^I_{gl} = \left\{ \begin{array}{l} (1)\ node(1 \ldots 6); \quad \ldots \quad (8)\ edge(5,6); \\ (9)\ coloured(1, red) \leftarrow node(1); \\ (10)\ coloured(3, red) \leftarrow node(3); \\ (11)\ coloured(4, green) \leftarrow node(4); \\ (12)\ coloured(6, green) \leftarrow node(6); \\ (13)\ coloured(2, blue) \leftarrow node(2); \\ (14)\ coloured(5, blue) \leftarrow node(5) \end{array} \right\}$$

It is easy to see that I is the minimal model of the positive program $\mathcal{P}^I_{gl}$, and thus an answer set of $\mathcal{P}$. It encodes the following valid graph coloring:



□

Recently an alternative elegant characterization of answer sets has been introduced in [FLP11]. It is based on an interesting modification of the GL-reduct of a logic program, called *flp-reduct*: FLP stands for the first letters of the authors' surnames: Faber, Leone, Pfeifer. Intuitively, the *flp*-reduct of a set of rules relative to an interpretation is obtained by removing every rule, the body of which is not satisfied by the interpretation. More formally,

Definition 2.25. Given an ASP program $\mathcal{P}$ and an interpretation I, the set of rules $\mathcal{P}^I_{flp} = \{r \in ground(\mathcal{P}) \mid I \models a, \forall a \in B^+(r);\ I \not\models a, \forall a \in B^-(r)\}$ is the *flp-reduct* of $\mathcal{P}$ relative to I.

For obtaining the *flp*-answer sets the same minimality condition as in traditional answer sets definition is applied.

Definition 2.26. An interpretation I is called an *flp-answer set* of $\mathcal{P}$, if I is a minimal model of the *flp*-reduct $\mathcal{P}^I_{flp}$. AS_{flp} denotes the set of all *flp*-answer sets of $\mathcal{P}$.

Example 2.27 (cont'd)**.** The *flp*-reduct of the program $\mathcal{P}$ from Example 2.14 relative to the interpretation I from Example 2.24 is as follows:

$$\mathcal{P}^I_{flp} = \left\{ \begin{array}{l} (1)\ node(1 \ldots 6); \quad \ldots \quad (8)\ edge(5,6); \\ (9)\ coloured(1, red) \leftarrow not\ \ coloured(1, blue), not\ \ coloured(1, green), node(1); \\ (10)\ coloured(3, red) \leftarrow not\ \ coloured(3, blue), not\ \ coloured(3, green), node(3); \\ (11)\ coloured(4, green) \leftarrow not\ \ coloured(4, blue), not\ \ coloured(4, red), node(4); \\ (12)\ coloured(6, green) \leftarrow not\ \ coloured(6, blue), not\ \ coloured(6, red), node(6); \\ (13)\ coloured(2, blue) \leftarrow not\ \ coloured(2, green), not\ \ coloured(2, red), node(2); \\ (14)\ coloured(5, blue) \leftarrow not\ \ coloured(5, green), not\ \ coloured(5, red), node(5) \end{array} \right\}$$

Since I is a minimal model of $\mathcal{P}^I_{flp}$, we get that I is an *flp*-answer set of $\mathcal{P}$. □

It has been shown in [FLP11] that the stable model semantics and the FLP-semantics coincide for extended disjunctive logic programs. We demonstrate this on the following example

Example 2.28. Consider a simple propositional logic program $\mathcal{P}$.

$$\mathcal{P}^I = \left\{ \begin{array}{ll} (1)\ q \leftarrow not\ p; & (3)\ r \leftarrow p; \\ (2)\ p \leftarrow not\ q; & (4)\ r \leftarrow q \end{array} \right\}$$

The *flp*-reduct $\mathcal{P}^I_{flp}$ relative to $I = \{p, r\}$ is

$$\mathcal{P}^I_{flp} = \left\{ \begin{array}{l} (1)\ p \leftarrow not\ q; \\ (2)\ r \leftarrow p \end{array} \right\}$$

The I is a minimal model of $\mathcal{P}^I_{flp}$, thus it is an *flp*-answer set of $\mathcal{P}$. Consider now the Gelfond Lifshitz reduct $\mathcal{P}^I_{gl}$ of $\mathcal{P}$ relative to I.

$$\mathcal{P}^I_{gl} = \left\{ \begin{array}{l} (1)\ p; \\ (2)\ r \leftarrow p; \\ (3)\ r \leftarrow q \end{array} \right\}$$

Observe that I is a minimal model of $\mathcal{P}^I_{gl}$, hence the stable model of $\mathcal{P}$. □

These two semantics differ when it comes to logic programs with aggregates, which we do not consider in this work.

Reasoning Tasks. The most important reasoning tasks in ASP include the following:

- *Answer set existence:* given a program $\mathcal{P}$, decide whether it has an answer set, i.e. decide whether $AS(\mathcal{P}) \neq \emptyset$;
- *Computation of all answer sets:* given a program $\mathcal{P}$, compute all its answer sets, i.e. find $AS(\mathcal{P})$;

- *Answer set checking:* given a program $\mathcal{P}$ and an interpretation I, decide whether I is an answer set of $\mathcal{P}$, i.e. decide whether $I \in AS(\mathcal{P})$;
- *Brave reasoning:* given a program $\mathcal{P}$ and a ground formula F, decide whether F holds in some answer set of $\mathcal{P}$, denoted by $\mathcal{P} \models_b F$;
- *Cautious reasoning:* given a program $\mathcal{P}$ and a ground formula F, decide whether F holds in all answer sets of $\mathcal{P}$, denoted by $\mathcal{P} \models_c F$.

Example 2.29. Verifying whether a graph encoded in a logic program from Example 2.14 is 3-colourable corresponds to the problem of answer set existence. Checking whether there exists a coloring, in which the node 1 is colored in red corresponds to brave reasoning, and deciding whether the node 1 is red in all possible valid colorings is a cautious reasoning task. □

2.4 Hybrid Knowledge Bases

So far we have considered two well-established formalisms for knowledge representation: DLs and ASP, serving diverse needs. DLs are widely accepted for creating ontologies, which in the Semantic Web context are intended to describe and structure complex web resources. ASP plays an important role in AI and databases. In AI ASP is mainly used for declarative problem solving and commonsense reasoning, and in databases for querying data repositories.

While DLs are monotonic, and they focused on specifying and reasoning in relation to conceptual knowledge, rules target issues associated with nonmonotonic inference. The semantics of DLs uses open domain, which allows one to express the knowledge about anonymous or unnamed individuals. For instance, we can express the information that every person has a parent without explicitly specifying the parent in the ontology. Conversely, the semantics of ASP uses the closed Herbrand domain which is often essential in various application scenarios. However, once the vocabulary is fixed the anonymous individuals can not be modeled in ASP. Although some extensions of ASP, e.g. generalized stable models [FLL07], Quantified Equilibrium Logic [Pea96], Open Answer Set Programming (OASP) [HNV07] convey the open domain assumption, the complexity of reasoning in such extensions increases and sometimes goes even beyond decidability. In DLs the maximal predicate arity is usually restricted to 2, while in ASP the usage of predicates with larger arities is possible.

Many practical applications require the features of both DLs and rules. Thus, the natural solution of combining representatives of Description Logics and rule-based languages has given rise to the notion of *Hybrid Knowledge Bases* [LNS96]. Informally, a hybrid KB is a pair $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O}$ is a DL-based ontology and $\mathcal{P}$ is a set of logical rules. There are three approaches for defining hybrid KBs: *tight coupling*, *embedding* approaches and *loose coupling*. The tight coupling approaches, like SWRL [HPSB$^+$04], r-hybrid KBs [Ros05] and ELP [KRH08], define the interface based on common models. The embedding approaches, like MKNF KBs [MR10], G-hybrid KBs [HPF$^+$06] and Open Answer Set Programming [Hey06], define the interface based on embeddings of

both the ontology and the rules in a single unifying non-monotonic formalism. In the loose coupling approach [dBBC$^+$09] the ontology and the rules act separately but communicate via a well-defined interface, e.g. F-Logic KBs [HKE$^+$10] and Description Logic programs [EIL$^+$08].

In this work, we focus on the loosely coupled hybrid KBs, and in particular consider *Description Logic (DL-) programs*. One of the main reasons behind the increasing popularity of DL-programs is the availability of sophisticated solvers for DL-programs (e.g. dlvhex, DReW). A bidirectional flow of information in the DL-programs between the logic program and the DL ontology is achieved via so-called DL-atoms, which prior to querying ontology can update it with the information derived from the rules.

2.5 DL-programs

We now turn to the formal description of syntax and semantics of DL-programs.

Syntax. A DL-program is a pair $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ over a signature $\Sigma_\Pi = \langle \mathbf{P}, \mathcal{C}, \mathbf{I}, \mathbf{C}, \mathbf{R}, \rangle$ of a finite ontology $\mathcal{O}$ over a signature $\Sigma_\mathcal{O} = \langle \mathbf{I}, \mathbf{C}, \mathbf{R} \rangle$ and a finite set of rules $\mathcal{P}$ over a signature $\Sigma_\mathcal{P} = \langle \mathbf{P}, \mathbf{C} \rangle$. In this work we assume that $\mathcal{C} = \mathbf{I}$, i.e. the set of ontology individuals and constants of the logic program part coincide. The rules r of $\mathcal{P}$ are of the form:

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, not\ \ b_{k+1}, \ldots, not\ \ b_m\,, \tag{2.2}$$

where each a_i, $0 \leq i \leq n$, is an lp-atom and each b_i $1 \leq i \leq m$, is either an *lp-atom* or a *DL-atom*. These atoms are defined as follows:

- an *lp-atom* is a first-order atom $p(\vec{t})$ with predicate p from a set $\mathbf{P}$ of predicate names disjoint with $\mathbf{C}$ and $\mathbf{R}$, and constants from the set $\mathcal{C} = \mathbf{I}$;
- a *DL-atom* $a(\vec{t})$ is of the form $\mathrm{DL}[\lambda; Q](\vec{t})$, where

  $$\lambda = S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m, \quad m \geq 0, \tag{2.3}$$

  is such that, for $1 \leq i \leq m$, $S_i \in \mathbf{C} \cup \mathbf{R}$, $op_i \in \{\uplus, \cup, \cap\}$ is an *update operator*, and $p_i \in \mathbf{P}$ is an *input predicate* of the same arity as S_i. Intuitively, $op_i = \uplus$ (resp., $op_i = \cup$) increases S_i (resp., $\neg S_i$) by the extension of p_i, while $\cap$ constraints S_i to p_i;
- $Q(\vec{t})$ is a *DL-query*, which is of one of the following forms:

  (i) $C(t)$, where C is a concept and t is a term;
  (ii) $R(t_1, t_2)$, where R is a role and t_1, t_2 are terms;
  (iii) $C_1 \sqsubseteq C_2$, where C_1, C_2 are concepts;
  (iv) $R_1 \sqsubseteq R_2$, where R_1, R_2 are roles, or
  (v) $\neg Q'(\vec{t})$ where $Q'(\vec{t})$ is from (i)-(iv).

  We skip $(\vec{t})$ for $\vec{t} = \epsilon$.

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Child} \sqsubseteq \exists \mathit{hasParent} & (4)\ \mathit{Male}(\mathit{pat}) \\ (2)\ \mathit{Adopted} \sqsubseteq \mathit{Child} & (5)\ \mathit{Male}(\mathit{john}) \\ (3)\ \mathit{Female} \sqsubseteq \neg \mathit{Male} & (6)\ \mathit{hasParent}(\mathit{john}, \mathit{pat}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{ischildof}(\mathit{john}, \mathit{alex});\quad (8)\ \mathit{boy}(\mathit{john}); \\ (9)\ \mathit{hasfather}(X, Y) \leftarrow \mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](Y), \mathrm{DL}[;\ \mathit{hasParent}](X, Y) \end{array} \right\}$$

Figure 2.2: DL-program Π over a family ontology

The set of all DL-atoms of a DL-program Π is denoted by DL_Π. The notions of rule's head and body are naturally inherited from normal logic programs, i.e. for a DL-rule r, $H(r) = a_1, \ldots, a_n$ is called the *head* of r, and $B(r) = \{b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m\}$ is called the *body* of r.

Example 2.30. Consider the DL-program Π in Figure 2.2, which captures information about children of a primary school and their parents in simplistic form. It consists of an ontology $\mathcal{O}$ which contains a taxonomy $\mathcal{T}$ of concepts (i.e., classes) in (1)-(3) and factual data (i.e., assertions) $\mathcal{A}$ about some individuals in (4)-(6). The rules $\mathcal{P}$ contain some further facts (7), (8) and proper rules: (9) determines fathers from the ontology, upon feeding information to it.

A DL-atom $\mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](Y)$ contained in the rule (9) first enriches the concept Male in $\mathcal{O}$ by the extension of the predicate boy in $\mathcal{P}$ via $\uplus$, and then queries the concept Male over the modified ontology. □

Semantics. The formal semantics of DL-programs associates Π with a collection of answer sets. Similar as for the case of ordinary logic programs, answer sets are defined in terms of the program's grounding $gr(\Pi) = \langle \mathcal{O}, gr(\mathcal{P}) \rangle$ over $\mathcal{C}$, i.e., $gr(\mathcal{P})$ contains all ground instances of rules r in $\mathcal{P}$ over $\mathcal{C}$. In the remainder, by default we assume that Π is ground.

Definition 2.31. A (Herbrand) *interpretation* of Π is a set $I \subseteq HB_\Pi$ of ground atoms, where HB_Π is the usual Herbrand base w.r.t. $\mathcal{C}$ and $\mathbf{P}$.

The satisfaction relation for an interpretation is defined as follows:

Definition 2.32. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a ground DL-program. An interpretation I satisfies

- an lp-atom a, if $a \in I$;
- a default negated atom $\mathit{not}\ a$ if $a \notin I$;
- a DL-atom a of the form (2.3), if

$$\mathcal{O} \cup \lambda^I(a) \models Q(\mathbf{c}) \tag{2.4}$$

  where $\lambda^I(a) = \bigcup_{i=1}^m A_i(I)$, and

- $A_i(I) = \{S_i(\vec{t}) \mid p_i(\vec{t}) \in I\}$, for $op_i = \uplus$;
- $A_i(I) = \{\neg S_i(\vec{t}) \mid p_i(\vec{t}) \in I\}$, for $op_i = \cup$;
- $A_i(I) = \{\neg S_i(\vec{t}) \mid p_i(\vec{t}) \notin I\}$, for $op_i = \cap$.

- a set S of (DL)-atoms, if it satisfies each atom individually;
- a ground DL-rule $r = a \leftarrow b_1, \ldots, b_k, not\ b_{k+1} \ldots, not\ b_m$, if either $I \not\models B(r)$ or $I \models H(r)$;
- a ground DL-program Π if it satisfies each rule r of Π.

We denote that I satisfies (is a *model* of) an object o (atom, rule, etc.) by $I \models^{\mathcal{O}} o$.

Let us now illustrate the defined notions by the following example.

Example 2.33. In Example 2.30, the interpretation $I = \{ischildof(john, alex), boy(john)$ $hasParent(john, pat)\}$, satisfies $a = \mathrm{DL}[Male \uplus boy; Male](john)$, since it holds that $\mathcal{O} \cup \lambda^I(a) \models Male(john)$. Furthermore, $I \models^{\mathcal{O}} r$, for every rule r of Π and thus $I \models^{\mathcal{O}} \Pi$. □

Various semantics were introduced for DL-programs, see [EIL+08] for overview. As in the ordinary ASP setting, we start the description of the semantics for the most simple class of *positive* DL-programs. Informally, positive DL-programs contain no default negations and involve only monotonic DL-atoms.

Definition 2.34. A ground DL-atom a is *monotonic* relative to $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ if $I \models^{\mathcal{O}} a$ implies $I' \models^{\mathcal{O}} a$, for all $I \subseteq I' \subseteq HB(\Pi)$, otherwise a is nonmonotonic. We denote by $DL^+_\Pi \subseteq DL_\Pi$ the set of all DL-atoms known to be monotonic, and as $DL^?_\Pi = DL_\Pi \backslash DL^+_\Pi$ the set of all other DL-atoms. A DL-program Π is *monotonic*, if all DL-atoms that it involves are monotonic.

Unless stated otherwise, it is assumed that DL^+_Π is the set of all monotonic DL-atoms of Π.

Note that a DL-atom involving the operator $\cap$ may be not monotonic, since an increasing set of ground predicates leads to a reduction of the ABox assertions by which the ontology is updated prior to its querying. On the other hand the DL-atoms that do not contain $\cap$ are guaranteed to be monotonic.

Definition 2.35. A DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is *positive* if (i) $\mathcal{P}$ does not involve default negated atoms, and (ii) every ground DL-atom occurring in Π is monotonic relative to $\mathcal{O}$. It is *normal* if $n = 1$ for every rule of form 2.2.

Like ordinary positive logic programs, every satisfiable positive non-disjunctive DL-program has a single least model reflecting the so-called *least model semantics*. Note that for disjunctive positive logic programs there is a least model in general.

Example 2.36. The non-disjunctive program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ from Figure 2.2 is positive, and its least model is $I = \{ischildof(john, alex), hasParent(john, pat),\ boy(john)\}$. □

For stratified DL-programs, which informally are composed of a layered hierarchy of positive DL-programs, the *iterative least model semantics* can be found in [EIL+08].

For our work the relevant semantics are Strong, Weak and FLP, which we describe in the following.

Strong and weak answer set semantics. The strong semantics for DL-programs is reduced to the least model semantics of positive DL-programs by the following transformation, called the *strong reduct*.

Definition 2.37. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program. The *strong reduct* of Π relative to $\mathcal{O}$ and an interpretation $I \subseteq HB(\Pi)$, denoted $\mathcal{P}^{I,\mathcal{O}}_{strong}$, is the set of all DL-rules obtained from $ground(\Pi)$ by deleting

- every DL-rule r, such that either $I \not\models^{\mathcal{O}} a$ for some $a \in B^+(r) \cap DL^?_\Pi$, or $I \models^{\mathcal{O}} d$ for some $d \in B^-(r)$; and
- from each remaining DL-rule r all literals in $B^-(r) \cup (B^+(r) \cap DL^?_\Pi)$.

Note that the strong reduct of any DL-program is positive by construction, and thus if a given non-disjunctive DL-program is satisfiable, then it must have some minimal model. Formally, strong answer sets are defined as follows:

Definition 2.38. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a non-disjunctive DL-program. A *strong answer set* of Π is an interpretation $I \subseteq HB_\Pi$, such that I is a minimal model of $\langle \mathcal{O}, \mathcal{P}^{I,\mathcal{O}}_{strong} \rangle$. $AS_{strong}(\Pi)$ denotes the set of all strong answer sets of Π.

Example 2.39. Consider a program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ with the ontology $\mathcal{O}$ as in Example 2.30, and the following rules $\mathcal{P}$:

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ ischildof(john, alex); \quad (8)\ boy(john); \quad (9)\ contact(john, alex); \\ (10)\ hasfather(john, pat) \leftarrow \mathrm{DL}[Male \uplus boy;\ Male](john), \mathrm{DL}[;\ hasParent](john, pat); \\ (11)\ contact(john, pat) \leftarrow \mathrm{DL}[;\ hasParent](john, pat), not\ \ omit(john, pat); \\ (12)\ omit(john, pat) \leftarrow \mathrm{DL}[;\ Adopted](john), hasfather(john, pat) \end{array} \right\}$$

The rule (9) is a ground version of the rule (10) from Example 2.30, while the rules (11) and (12) intuitively distinguish adult representatives for children (which can be contacted in case of emergency). Let us look at the interpretation $I = \{hasparent(john, alex), boy(john), hasfather(john, pat), omit(john, pat), contact(john, alex)\}$. Note that all DL-atoms in the program are monotonic. The strong reduct $\mathcal{P}^{I,\mathcal{O}}_{strong}$ contains all facts and rules of $\mathcal{P}$ apart from (11). Since I is the minimal model of the strong reduct $\mathcal{P}^{I,\mathcal{O}}_{strong}$, it is a strong answer set of Π. □

It is known that the strong answer set semantics of a DL-program without DL-atoms coincides with the least model semantics of a DL-program without DL-atoms. Another important property is as follows:

Proposition 2.40 (cf. [EIL$^+$08])**.** *Strong answer sets of a DL-program* Π *are also models of* Π*, and moreover minimal models of* Π *if* $DL_\Pi = DL^+_\Pi$.

For positive (resp. stratified) programs the strong answer sets coincide with the least models if the program is satisfiable (resp. consistent) and do not exist otherwise.

The *weak answer set semantics* is defined in terms of the weak reduct.

Definition 2.41. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program. The *weak reduct* of $\mathcal{P}$ relative to $\mathcal{O}$ and to an interpretation $I \subseteq HB_\Pi$, denoted $\mathcal{P}^{I,\mathcal{O}}_{weak}$ is the ordinary positive program obtained from $ground(\Pi)$ by deleting

- all DL-rules r such that either $I \not\models^{\mathcal{O}} a$ for some DL-atom $a \in B^+(r)$, or $I \models^{\mathcal{O}} l$ for some $l \in B^-(r)$; and
- from every remaining DL-rule r all the DL-atoms in $B^+(r)$ and all the literals in $B^-(r)$.

Notice that $\mathcal{P}^{I,\mathcal{O}}_{weak}$ is an ordinary ground positive program without any DL-atoms and default-negated literals. Therefore we define the weak answer set semantics of general DL-programs by reduction to the least model semantics of ordinary positive programs in the following way.

Definition 2.42. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program. A *weak answer set* of Π is an interpretation $I \subseteq HB_\Pi$ such that I is a minimal model of the ordinary positive program $\mathcal{P}^{I,\mathcal{O}}_{weak}$. $AS_{weak}(\Pi)$ denotes the set of all weak answer sets of Π.

Example 2.43. For the program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ from Example 2.39, the weak reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$ contains only the following facts:

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ ischildof(john, alex); \quad (8)\ boy(john); \quad (9)\ contact(john, alex); \\ (10)\ hasfather(john, pat); \quad (12)\ omit(john, pat) \end{array} \right\}$$

□

The weak answer set semantics of a DL-program Π with no DL-atoms involved coincides with the ordinary answer set semantics of Π.

Another important property states that every weak answer set of a DL-program Π is also a model of KB. However, unlike for the case of strong answer sets, a weak answer set is not necessarily a minimal model. This holds even for DL-programs containing only monotonic DL-atoms. The set of all strong answer sets of a DL-program Π is contained in the set of all its weak answer sets. The converse, however, does not hold in general.

Example 2.44. Consider $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where the ontology $\mathcal{O}$ is empty, i.e. $\mathcal{O} = \emptyset$, and the rule part $\mathcal{P}$ is as follows: $\mathcal{P} = \{man(pat) \leftarrow \mathrm{DL}[Male \uplus man; Male](pat).\}$. We now compute strong and weak answer sets of Π.

Let us look at the interpretation $I_1 = \emptyset$. The strong reduct $\mathcal{P}^{I_1,\mathcal{O}}_{strong} = \mathcal{P}$, since the DL-atom $\mathrm{DL}[Male \uplus man; Male](pat) \notin D^?_\Pi$. Observe that I_1 is a minimal model of $\mathcal{P}^{I_1,\mathcal{O}}_{strong}$, hence we get that $I_1 \in AS_{strong}(\Pi)$. The interpretation I_1 is also a weak answer set of Π, since $\mathcal{P}^{I_1,\mathcal{O}}_{weak} = \emptyset$. Furthermore, Π has $I_2 = \{man(pat)\}$ as another weak answer set. Indeed, $\mathcal{P}^{I_2,\mathcal{O}}_{weak} = \{man(pat).\}$, and clearly I_2 is its minimal model. Note that I_2, however, is not a strong answer set of Π. The strong reduct $\mathcal{P}^{I_2,\mathcal{O}}_{strong} = \mathcal{P}$ of $\mathcal{P}$ relative to I_2 has a model I_1, which is smaller then I_2.

Overall, we have $AS_{strong}(\Pi) = \{I_1\}$, and $AS_{weak}(\Pi) = \{I_1, I_2\}$. □

The strong answer semantics provides stricter conditions on answer sets than the weak one, but the weak semantics remains a reasonable choice if no information about monotonicity of DL-atoms is available.

FLP answer set semantics. The last semantics relevant for our work is the *flp*-semantics, which characterizes the answer sets in terms of the so-called *flp*-reduct.

Definition 2.45. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program. The *flp*-reduct $\mathcal{P}^{I,\mathcal{O}}_{flp}$ of Π relative to $I \subseteq HB_\Pi$ is the rule set

$$\mathcal{P}^{I,\mathcal{O}}_{flp} = \{r^{I,\mathcal{O}}_{flp} \mid r \in gr(\mathcal{P})\}, \tag{2.5}$$

where

- $r^{I,\mathcal{O}}_{flp} = r$, if the body of r is satisfied, i.e., $I \models^{\mathcal{O}} b_i$, for all b_i, $1 \leq i \leq k$ and $I \not\models^{\mathcal{O}} b_j$, for all $k < j \leq m$;
- $r^{I,\mathcal{O}}_{flp}$ is void, if the body of r is not satisfied, i.e., $I \not\models^{\mathcal{O}} b_i$, for all b_i, $1 \leq i \leq k$ and $I \not\models^{\mathcal{O}} b_j$, for all $k < j \leq m$.

Now the *flp*-answer sets are given as follows:

Definition 2.46. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program. An *(flp-)answer set* of $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is any interpretation $I \subseteq HB_\Pi$ that is a $\subseteq$-minimal model of the *flp*-reduct $\mathcal{P}^{I,\mathcal{O}}_{flp}$.

Example 2.47. Consider the DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where we have $\mathcal{O} = \emptyset$ and $\mathcal{P} = \{woman(pat) \leftarrow \mathrm{DL}[Male \sqcap man; \neg Male](pat)\}$. Let us look at the interpretation $I_1 = \{woman(pat)\}$. The *flp*-reduct $\mathcal{P}^{I_1,\mathcal{O}}_{flp}$ of $\mathcal{P}$ relative to I_1 contains all rules of $\mathcal{P}$, i.e. $\mathcal{P}^{I_1,\mathcal{O}}_{flp} = \mathcal{P}$. As I_1 is a minimal model of $\mathcal{P}^{I_1,\mathcal{O}}_{flp}$, we have that $I_1 \in AS_{strong}(\Pi)$. □

The notion of consistency for DL-programs is defined in the usual way:

Definition 2.48. A DL-program Π is *inconsistent*, if it has no answer set.

The *flp*-semantics is attractive as it naturally handles DL-atoms which are not monotonic. The correspondence between the *flp* semantics and strong semantics for DL-programs containing only DL-atoms, known to be monotonic is given in the following proposition.

Proposition 2.49. *[EIST05] Given a DL-program* Π*, I is an flp answer set of* Π *iff I is a strong answer set of* Π*.*

In fact the operator $\Cap$ is rarely used in practice and as shown in [WEY+13], it can be often removed by simple translations.

If the semantics under which the answer sets for a given DL-program are computed is not important, we refer to the answer sets as *x-answer sets*, where x can be substituted by any concrete semantics, e.g. when $x = flp$ we get *flp*-answer sets.

The reasoning tasks for DL-programs are the same as for ASP, and they can be considered under various semantics. For example, decide whether there exists a strong answer set for a DL-program Π is denoted by $AS_{strong}(\Pi) \neq \emptyset$, deciding whether I is a weak answer set of Π is denoted by $I \in AS_{weak}(\Pi)$.

2.6 HEX-programs

Apart from the interaction with the DL ontology through a logic program there are other ways of accessing information from different external sources. An important generalization of DL-programs are *HEX-programs* [EIL+08], which accommodate a universal bidirectional interface for arbitrary sources of external computation. This is achieved by means of the notion of an external atom. Using such external atoms, whose semantics is abstractly modeled by an input-output relationship, one can access different kinds of information and reasoning in a single program. HEX-programs have been successfully used in various kinds of applications. Some examples include multi-agent systems, rule-based policy specification, distributed SPARQL processing, to mention a few.

We assume that for a given HEX-program the vocabulary consists of mutually disjoint sets $\mathcal{C}$ of constants, $\mathbf{V}$ of variables, $\mathbf{P}$ of predicates, $\mathbf{X}$ of external predicates. We recall several notions relevant for HEX-programs.

- A *(signed) literal* is a positive or a negative formula $\mathbf{T}a$ resp. $\mathbf{F}a$, where a is an atom of form $p(\vec{X}) = p(X_1, \ldots, X_\ell)$, with a predicate p and terms $X_1, \ldots, X_\ell \in \mathcal{C} \cup \mathbf{V}$;
- A signed literal or atom is *ground*, if all terms in $\vec{X}$ are constants, and *nonground* otherwise.
- An *assignment* $\mathbf{A}$ over a (finite) set A of ground atoms is a consistent set of ground signed literals $\mathbf{T}a$ and $\mathbf{F}a$, $a \in A$, where $\mathbf{T}a$ expresses that a is true and $\mathbf{F}a$ that a is false.
- An *interpretation* is any assignment $\mathbf{A}$ that is *complete*, i.e., no strictly larger assignment $\mathbf{A}' \supset \mathbf{A}$ over A exists.

Next we recall syntax and semantics of HEX-programs.

Syntax. HEX-programs generalize (disjunctive) extended logic programs under the answer set semantics described earlier with *external atoms*, which enable a bidirectional interaction between a program and external sources of computation. External atoms have a list of input parameters (constants or predicate names) and a list of output parameters.

Definition 2.50. A *ground external atom* $a(\vec{Z})$ is of the form

$$\&g[\vec{Y}](\vec{X}), \tag{2.6}$$

where $\&g \in \mathcal{X}$, $\vec{Y} = Y_1, \ldots, Y_\ell$, and $\vec{X} = X_1, \ldots, X_m$, such that $Y_i, X_j \in \mathbf{P} \cup \mathcal{C} \cup \mathbf{V}$, for $1 \leq i \leq \ell$ and $1 \leq j \leq m$, and $\vec{Z}$ is the restriction of $\vec{Y}$ and $\vec{X}$ to elements from $\mathbf{V}$.

Example 2.51. As an example of an external atom, consider $a(\vec{X}) = \&diff[p, q](\vec{X})$, where p and q are predicates. The atom $a(\vec{X})$ computes the set of all elements X, which are in the extension of p but not in the extension of q. □

HEX-programs are defined as follows:

Definition 2.52. A HEX-*program* consists of rules r of form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, not\ \ b_{k+1}, \ldots, not\ \ b_m\ , \tag{2.7}$$

where each a_i is an (ordinary) atom, each b_j is either an ordinary atom or an external atom, and $n + m > 0$.

Like for ordinary logic programs, we refer to $H(r) = \{a_1, \ldots, a_n\}$ as the *head* of r, and to $B(r) = \{b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_n\}$ as the *body* of r.

Example 2.53. Consider the program Π

$$\begin{array}{lll} d(c) \leftarrow; & q(c) & \leftarrow d(c), \&diff[d, p](c); \\ & p(c) & \leftarrow d(c), \&diff[d, q](c) \end{array}$$

Informally, this program implements a choice from $p(c)$ and $q(c)$. □

A program is *ground*, if it contains no variables. We will also consider non-ground HEX-programs in our examples, for which suitable safety conditions allow to use a *grounding procedure* that transforms the program to a variable-free program with the same answer sets. We thus confine our formal investigations here to ground programs.

Semantics. The semantics of a HEX-program is defined via interpretations $\mathbf{A}$ over the Herbrand base, which is naturally generalized from ordinary logic programs as follows:

Definition 2.54. A *Herdbrand Base* of a HEX-program Π, denoted $HB(\Pi)$ is the set of all atoms constructible from the predicates occurring in Π and the constants from $\mathcal{C}$.

Given a HEX-program Π, satisfaction of (sets of) literals, rules, etc. O w.r.t. an assignment $\mathbf{A}$ over $HB(\Pi)$, denoted $\mathbf{A} \models O$, extends naturally from ordinary [GL91a] to HEX-programs and the satisfaction of a ground external atom $\&g[\vec{y}](\vec{x})$ is a more involved. It is given by the value of a $1+|\vec{y}|+|\vec{x}|$-ary Boolean function $f_{\&g}$. Formally,

Definition 2.55. Let Π be a HEX-program and $\mathbf{A}$ an interpretation. The satisfaction relation is defined as follows:

- for an ordinary atom b, $\mathbf{A} \models b$, if $\mathbf{T}b \in \mathbf{A}$, and $\mathbf{A} \not\models b$, if $\mathbf{F}b \in \mathbf{A}$;
- for a ground external atom $\&g[\vec{p}](\vec{c})$, $\mathbf{A} \models \&g[\vec{p}](\vec{c})$, if $f_{\&g}(\mathbf{A}, \vec{y}, \vec{x}) = 1$, and $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$, if $f_{\&g}(\mathbf{A}, \vec{y}, \vec{x}) = 0$;
- $\mathbf{A}$ satisfies an (ordinary or external) literal *not* b, if $\mathbf{A} \not\models b$;
- $\mathbf{A}$ satisfies a rule of form (2.7), if $\mathbf{A} \models a_i$ for some $1 \leq i \leq k$ or $\mathbf{A} \not\models b_i$ for some $1 \leq i \leq m$ or $\mathbf{A} \models b_i$ for some $m < i \leq n$;
- $\mathbf{A}$ satisfies a ground HEX-program Π ($\mathbf{A}$ is a model of Π), if $\mathbf{A} \models r$ for all rules r of Π.

The answer sets of HEX-programs are defined in terms of the *flp*-reduct.

Definition 2.56. Let Π be a HEX-program and let $\mathbf{A}$ be an assignment. An *flp*-reduct of Π w.r.t. $\mathbf{A}$ is a program $\Pi^{\mathbf{A}}_{flp} = \{r \in \Pi \mid \mathbf{A} \models B(r)\}$.

Definition 2.57. Given a HEX-program Π, an assignment $\mathbf{A}$ is an *flp-answer set* of Π, if $\mathbf{A}$ is a model of $\Pi^{\mathbf{A}}_{flp}$, whose positive part (i.e., $\{\mathbf{T}a \in \mathbf{A}\}$) is subset-minimal. $AS_{flp}(\Pi)$ denotes the set of all *flp*-answer sets of a HEX-program Π.

Example 2.58. Recall the HEX-program from Example 2.53 and consider an assignment $\mathbf{A}_1 = \{\mathbf{T}d(c), \mathbf{F}p(c)\}$. The reduct $\Pi^{\mathbf{A}_1}_{flp}$ of Π relative to $\mathbf{A}_1$ is as follows:

$$\Pi^{\mathbf{A}_1}_{flp} = \{d(c);\ \ p(c) \leftarrow \&diff[d, q](c)\}.$$

Observe, that $\mathbf{A}_1$ is a minimal model of $\Pi^{\mathbf{A}_1}_{flp}$, therefore $\mathbf{A}_1 \in AS_{flp}(\Pi)$.

The assignment $\mathbf{A}_2 = \{d(c), q(c)\}$ is another *flp*-answer set of Π. Indeed, the *flp*-reduct comprises

$$\Pi^{\mathbf{A}_2}_{flp} = \{d(c);\ \ q(c) \leftarrow \&diff[d, p](c)\}.$$

As $\mathbf{A}_2$ is the minimal model of $\Pi^{\mathbf{A}_2}_{flp}$, we get that $\mathbf{A}_2 \in AS_{flp}(\Pi)$. □

Evaluation. The usual way to compute the answer sets of a HEX-program Π is via a transformation to an ordinary ASP program $\hat{\Pi}$ [EFK+14]. Each external atom $a = \&g[\vec{y}](\vec{x})$ in a rule $r \in \Pi$ is replaced by an ordinary *replacement atom* $\hat{a} = e_{\&g[\vec{y}]}(\vec{x})$ (resulting in a rule $\hat{r}$), and a rule $e_{\&g[\vec{y}]}(\vec{x}) \vee ne_{\&g[\vec{y}]}(\vec{x}) \leftarrow$ is added to the program. The

answer sets of the resulting *guessing program* $\hat{\Pi}$ are computed by an ASP solver and projected to non-replacement atoms. However, each answer set $\hat{\mathbf{A}}$ of $\hat{\Pi}$ merely gives rise to a *candidate answer set* of Π, as the guess for $e_{\&g[\vec{p}]}(\vec{c})$ must be checked against the actual value of $\&g[\vec{p}]$. If no discrepancy is found, the model candidate is a *compatible set* of Π. More precisely,

Definition 2.59. Let Π be a HEX-program Π. The *guessing program* $\hat{\Pi}$ is an ASP program obtained from Π as follows:

- each external atom $a = \&g[\vec{p}](\vec{c})$ in a rule r is replaced by an ordinary ground replacement atom $\hat{a} = e_{\&g[\vec{p}]}(\vec{c})$ (resulting in a rule $\hat{r}$), and
- for each external atom $a = \&g[p](c)$ an additional *guessing rule* rule of the form

$$e_{\&g[\vec{p}]}(\vec{c}) \vee ne_{\&g[\vec{p}]}(\vec{c}) \leftarrow \tag{2.8}$$

  is added to $\hat{\Pi}$.

Definition 2.60. A *compatible set* of a HEX-program Π is an assignment $\hat{\mathbf{A}}$ such that

(i) $\hat{\mathbf{A}} \in \mathcal{AS}(\hat{\Pi})$ and

(ii) for all $\&g[\vec{y}](\vec{x})$ in Π, $\mathbf{T}e_{\&g[\vec{y}]}(\vec{x}) \in \hat{\mathbf{A}}$.

As each answer set of Π is a projection of a (unique) compatible set that fulfills an additional minimality check, computing compatible sets is essential for evaluation.

2.6.1 From HEX-programs to DL-programs

As HEX-programs are generalizations of DL-programs, there is a correlation between their syntax and semantics, which we discuss next. Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program, where $\mathcal{O}$ is a consistent ontology fixed as an external source and $\mathcal{P}$ is a set of DL-rules.

DL-atoms are encoded as external atoms of the form $\&DL[c^+, c^-, r^+, r^-, Q](\vec{x})$, binary (resp. ternary) predicates and Q is a string which encodes an ontology query. The query Q is a possibly negated ontology concept or a role name, concept or role subsumption or its negation.

The oracle function of $\&DL$ is defined by

$$\begin{aligned} & f_{\&DL}(\mathbf{A}, c^+, c^-, r^+, r^-, \vec{x}) = 1 \\ & \quad \iff \mathcal{O} \cup U^{\mathbf{A}}(c^+, c^-, r^+, r^-) \models Q(\vec{x}), \end{aligned}$$

where $U^{\mathbf{A}}(c^+, c^-, r^+, r^-)$ is an update to $\mathcal{O}$, specified by the (extension of the) predicates c^+, c^-, r^+, r^-. More specifically, it contains for each $\mathbf{T}c^+(C, a) \in \mathbf{A}$ (resp. $\mathbf{T}c^-(C, a) \in \mathbf{A}$), a concept assertion $C(a)$ (resp. $\neg C(a)$). Updates of roles, generated by the predicates r^+ and r^- are analogous.

Example 2.61. The DL-atom $\mathrm{DL}[\mathit{Male} \uplus \mathit{boy}; \mathit{Male}](X)$ from Example 2.30 is translated to $\&DL[c^+, c^-, r^+, r^-, \mathit{Male}](X)$, s.t. $\mathcal{P}$ is extended by the rule $c^+(\mathit{Male}, X) \leftarrow \mathit{boy}(X)$ in $\mathcal{P}$, and the predicate c^+ does not occur elsewhere in the program $\mathcal{P}$.

The rules $\mathcal{P}$ in the syntax of HEX-programs is as follows:

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{ischildof}(\mathit{john}, \mathit{alex}); \quad (8)\ \mathit{boy}(\mathit{john}); \\ (9)\ \mathit{hasfather}(X, Y) \leftarrow \&DL[c'^+, c'^-, r'^+, r'^-, \mathit{hasParent}](X, Y), \\ \qquad\qquad\qquad\qquad \&DL[c^+, c^-, r^+, r^-, \mathit{Male}](Y); \\ (10)\ c^+(\mathit{Male}, X) \leftarrow \mathit{boy}(X) \end{array} \right\}$$

□

The described transformation has been implemented within the dlplugin[18] and dlliteplugin[19] of the dlvhex system, which is a HEX-program solver.

2.7 Computational Complexity

We now briefly review computational complexity theory and discuss the main relevant complexity results for reasoning problems in DLs, ASP and DL-programs.

We start by formally defining the notion of a problem.

Definition 2.62. A *problem* is a question together with an (in general infinite) set of possible instances $\mathbf{\Pi}$ (i.e. possible inputs) encoded in some meaningful way.

Standard complexity theory deals with *decision problems*.

Definition 2.63. A problem is a *decision problem* if it's question has a yes/no answer, i.e. the answer is "yes", if the problem input falls into the set of yes-instances $\mathbf{Y} \subseteq \mathbf{\Pi}$, and "no" if the problem input falls into the set of no-instances $\mathbf{\Pi} \backslash \mathbf{Y}$.

Example 2.64. The graph 3-colourability problem, which we considered in Example 2.14 is a decision problem, and it can be represented as follows:

GRAPH 3-COLOURABILITY
INSTANCE: A graph $G = (V, E)$.
QUESTION: Is a graph G 3-colourable?

For the **GRAPH 3-COLOURABILITY** problem the set of instances $\mathbf{\Pi}$ is the set of all possible graphs, while the set of "yes"-instances $\mathbf{Y} \subset \mathbf{\Pi}$ is the set of graphs that are 3-colourable, and the set of "no" instances $\mathbf{\Pi} \backslash \mathbf{Y}$ is the set of all other graphs. □

Definition 2.65. *Computational complexity theory* is a field of study that aims at determining the upper and lower bounds on the amount of resources (time, space) needed to solve a computational problem at hand.

[18] `https://github.com/hexhex/dlplugin`
[19] `https://github.com/hexhex/dlliteplugin`

Worst-case complexity analysis measures the complexity in terms of a (complexity) function f, whose argument is the size n of an instance of the problem (i.e., the length of its encoding) and whose result is the amount $f(n)$ of time/space needed in the worst-case to solve a problem instance of size n. Problems that can be solved using a specific set of resources are grouped together to form a *complexity class*. We assume that the reader is familiar with the basic notions from computational complexity theory such as (non-)deterministic Turing machines, problem reductions and completeness (see [Pap94] for an excellent introduction to the field).

All complexity classes can be divided into two groups: time complexity classes and space complexity classes. More formally,

Definition 2.66. A *time complexity class* TIME($f(n)$) (resp. NTIME($f(n)$)) is the set of languages L, s.t. L is decided by a deterministic (resp.) nondeterministic Turing machine in time $O(f(n))$.

Definition 2.67. A *space complexity class* SPACE($f(n)$) (resp. NSPACE($f(n)$)) is the set of languages L s.t. L is decided by a deterministic (resp. nondeterministic) Turing machine within space $O(f(n))$.

As usual we denote by P(resp. NP) a class of decision problems that can be solved on a deterministic (resp. nondeterministic) Turing Machine in polynomial time. AC^0 is the class of problems definable using a family of circuits of constant depth and polynomial size, which can be generated by a deterministic Turing machine in logarithmic space in the size of the input. AC^0 allows one to use polynomially many processors, while keeping the run-time constant. PSpace is the class of problems solvable with polynomial work space. Some major complexity classes are shown in Table 2.7.

The relation among the complexity classes is as follows:

$$\mathsf{AC}^0 \subset \text{LogSpace} \subseteq \text{NLogSpace} \subseteq \mathsf{P}$$
$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \text{PSpace} \subseteq \text{NPSpace}$$
$$\text{NPSpace} \subseteq \text{ExpTime} \subseteq \text{NExpTime} \subseteq \text{2ExpTime} \subseteq \text{2NExpTime}$$

An *Oracle Turing Machines* model computations with calls to subroutines, and they are capable of solving certain decision problems in a single operation. Given a complexity class C and an oracle A, we denote by C^A a class of problems which can be solved by a Turing Machine within the timebound of the class C, where the Turing Machine can invoke an oracle for solving any problem from the class A. Given an instance I of A, such an oracle machine can produce a correct answer for I in one unit of time. Moreover, given a complexity class C, a class co-C denotes the class of problems whose complement language is in C. For example, the problems in NP intuitively ask whether there *exists* a string satisfying certain properties, whereas problems in co-NP ask whether *all* strings satisfy certain properties. A canonical NP-complete problem is deciding satisfiability of a propositional CNF formula, while deciding its validity is a typical co-NP-complete problem. A natural extension is to consider problems which combine existential and

Complexity Class	Definition
LogSpace	$\text{SPACE}(\log_2 n)$
NLogSpace	$\text{NSPACE}(\log_2 n)$
P	$\bigcup_{k \in N} (\text{TIME}(O(n^k)))$
NP	$\bigcup_{k \in N} (\text{NTIME}(O(n^k)))$
PSpace	$\bigcup_{k \in N} (\text{SPACE}(O(n^k)))$
NPSpace	$\bigcup_{k \in N} (\text{NSPACE}(O(n^k)))$
ExpTime	$\bigcup_{k \in N} (\text{TIME}(O(2^{n^k})))$
NExpTime	$\bigcup_{k \in N} (\text{NTIME}(O(2^{n^k})))$
2ExpTime	$\bigcup_{k \in N} (\text{TIME}(O(2^{2^{n^k}})))$
2NExpTime	$\bigcup_{k \in N} (\text{NTIME}(O(2^{2^{n^k}})))$

Table 2.6: Defining complexity classes

universal quantifiers. The complexity classes which emerge from this process make up the *polynomial hierarchy*, which is a hierarchy of complexity classes that generalize the classes of P, NP and co-NP.

Definition 2.68. The *polynomial hierarchy* consists the following sequence of classes:

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$
- $\Delta_{i+1}^P = \mathsf{P}^{\Sigma_i^P}$
- $\Sigma_{i+1}^P = \mathsf{NP}^{\Sigma_i^P}$
- $\Pi_{i+1}^P = \text{co-}\mathsf{NP}^{\Sigma_i^P}$

for all $i \geq 0$. Moreover, the *collective class* is defined as follows: $\text{PH} = \bigcup_{i \geq 0} \Sigma_i^P$.

Note that since $\Sigma_0^P = \mathsf{P}$, we have that $\Sigma_1^P = \mathsf{NP}$, $\Delta_1^P = \mathsf{P}$, and $\Pi_1^P = \text{co-}\mathsf{NP}$. At each level the classes are believed to be distinct, and furthermore each class at each level also includes the classes from the previous levels. Checking whether a ground disjunctive logic program has some answer set is a well-known Σ_2^P-complete problem [EG93].

There are also some problems that are *undecidable*, i.e. decision problems, for which it is known to be impossible to construct a single algorithm that always leads to a correct

	$\mathcal{O}$ satisfiability		$\mathcal{O} \models P(\vec{t})$		$C \sqsubseteq D$	
	in the size of $\mathcal{T}$	in the size of $\mathcal{A}$	in the size of $\mathcal{T}$	in the size of $\mathcal{A}$	in the size of $\mathcal{T}$	in the size of $\mathcal{A}$
DL-Lite$_\mathcal{A}$	NLogSpace	AC^0	NLogSpace	AC^0	NLogSpace	AC^0
$\mathcal{EL}$	P	P	P	P	P	P

Table 2.7: Complexity of Lightweight DLs (completeness results)

Program Class	$AS(\mathcal{P}) \neq \emptyset$	$\mathcal{P} \models_b F$	$\mathcal{P} \models_c F$
Prop. positive lp	trivial	P	P
Prop. normal lp	NP	NP	co-NP
Prop. disjunctive lp	Σ_2^P	Σ_2^P	Π_2^P
DATALOG	trivial	EXPTIME	EXPTIME
DATALOG$^\neg$	NEXPTIME	NEXPTIME	co-NEXPTIME
DATALOG$^{\neg,\vee}$	NEXPTIME$^{\mathrm{NP}}$	NEXPTIME$^{\mathrm{NP}}$	co-NEXPTIME$^{\mathrm{NP}}$

Table 2.8: Complexity of logic programming (completeness results)

Ground $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$		$AS_{strong/flp}(\Pi) \neq \emptyset$	$AS_{weak}(\Pi) \neq \emptyset$
$\mathcal{O}$ in $\mathcal{SHIF}$		NEXPTIME	NEXPTIME
$\mathcal{O}$ in $\mathcal{SHOIN}$		P$^{\mathrm{NEXPTIME}}$	P$^{\mathrm{NEXPTIME}}$
$\mathcal{O}$ in *DL-Lite*$_\mathcal{A}$ or $\mathcal{EL}$	Normal $\mathcal{P}$	Σ_2^P	NP
	Arbitrary $\mathcal{P}$	Σ_2^P	Σ_2^P

Table 2.9: Complexity of DL-programs (completeness results)

yes-or-no answer in finite time. For instance, the problem of deciding satisfiability of a first order formula is well-known to be undecidable.

2.7.1 Complexity of DLs, ASP and DL-programs: Main Results

In this section we provide an overview of the known complexity results for DLs, ASP and DL-programs.

***DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ DL.** The DLs considered in this work belong to the class of lightweight DLs, therefore the reasoning tasks for these DLs can be regarded as computationally "easy". For the *DL-Lite*$_\mathcal{A}$ case the works [CLLR07] and [PLC+08] provide the complexity results, while the complexity of DL $\mathcal{EL}$ was mostly studied in [BBL05]

and [BBL08]. We give the summary of the relevant complexity upper bounds in Table 2.7. For each of the reasoning tasks we present the complexity measured in terms of the size of the TBox and the ABox respectively.

ASP. The complexity of ASP has been extensively studied both in the propositional and the first-order case. We summarize the main known results in Table 2.8, and refer the reader to [DEGV01] for more detailed exposition. The first column of Table 2.8 distinguishes program classes for which the complexity of reasoning tasks is presented. The columns 2-4 specify the reasoning tasks considered, i.e. the second column refers to the checking an answer set existence, the second and third columns are related to the brave and cautious reasoning respectively.

Complexity of various restricted types of logic programs were analyzed in the literature as well, see e.g. [EFFW07].

DL-programs. Extensive overview of the complexity for DL-programs over ontologies in expressive DLs is provided in [EIL$^+$08] and for lightweight DLs some results can be obtained based on [WEY$^+$13]. We provide a complexity summary in Table 2.9.

CHAPTER 3

Inconsistency Management

Inconsistency is known to cause severe problems in logic-based and data intensive systems. An inconsistent logical knowledge base has no model, and the set of its classical consequences is trivialized, i.e. every formula follows from it. This makes query answering from such an inconsistent knowledge base meaningless.

Treating adequately logically contradicting information is a major challenge faced by KR formalisms in various settings. Therefore, inconsistency management has emerged as an important area of research in AI and KR, which has been extensively studied in various fields, e.g. belief revision [AGM85, GR95], knowledge base updates [EEFS05], diagnosis [Rei87], nonmonotonic reasoning [Bre89, SI03] and many others (e.g., [BHS05, Ngu08, MMSA13, Ber11, dCH15]).

There are mainly two approaches for dealing with inconsistency in a logic-based system. These depend upon the practical settings. The first approach seeks to minimize changes to a faulty system (e.g. by deleting information) in the search of satisfiability. The second approach promotes the use of paraconsistent semantics which is capable of producing reasonable inferences from a faulty system. Under these semantics, the contradictory statement does not deduce an arbitrary formula, hence the whole theory is not trivialized. Historically, paraconsistent logics have been developed in the area of philosophical logic [Mid11].

In this chapter we overview existing strategies for handling inconsistencies in the context of databases (Section 3.1), ontologies (Section 3.2), rules (Section 3.3), and hybrid logical formalisms with the focus on DL-programs (Section 3.4). For a good survey of consistency-related issues we refer the reader to [FGC+11].

3.1 Inconsistencies in Databases

Managing inconsistent databases, i.e. databases violating integrity constraints is a common problem that has been intensively studied in the recent years. Even though integrity

constraints successfully capture data semantics, the actual data in the database often fails to satisfy such constraints. This may happen because the data is drawn from a variety of independent sources as in data integration settings [DLLR04, Lem04].

The problem of dealing with inconsistency in databases is tightly related to belief revision and update [AGM85, GR95] in AI, which study the problem of integrating new information with available knowledge. Among the first theoretical approaches to the problem of dealing with incomplete and inconsistent information in databases one can mention the work [IJ84], which mainly focused on issues related to the semantics of incompleteness. The problem of extracting reliable information from inconsistent data was then addressed in [AKWS95], where an extension of relational algebra (namely flexible algebra) was proposed to evaluate queries on data inconsistent w.r.t. key constraints. The first proof-theoretic notion of consistent query answers was introduced in [Bry97], expressing the idea that tuples involved in an integrity violation should not be considered in consistent query answering.

In [ABC99] a different notion of consistent answer based on the notion of repair was introduced. In particular, the authors of [ABC99] show that, for quantifier-free conjunctive queries and binary universal constraints, consistent answers can be evaluated without computing repairs, but by looking only at the specified constraints and rewriting the original query q into a query q' such that the answer of q' on D is equal to the consistent answer of q on D. This technique based on query rewriting was further developed in [CB00] and extended in [FM07,FFP05] to work for a subclass of conjunctive queries with existential quantification in the presence of key constraints. The results provided in [FM07] were further generalized in [Wij09].

Based on the notions of repair and consistent query answer introduced in [ABC99], several works investigated the problem of querying inconsistent data considering more expressive classes of queries and constraints. The notion of consistent answer was extended to the case of aggregate queries in [ABC03], where consistent answers of aggregate queries were investigated in the presence of functional dependencies.

Several works considered logic-based frameworks for exploiting the problem of computing repairs and consistent query answering. Specifically, in [ABC00,ABC03] extended disjunctive logic programs with exceptions were used for the computation of repairs. A further generalization was proposed in [GGZ03], where the authors defined a technique based on the rewriting of constraints into extended disjunctive rules with two different forms of negation (negation as failure and classical negation). This technique was shown to be sound and complete for universally quantified constraints. In [ABC00], a repairing framework based on a non-classical logic (the annotated predicate calculus [KL92]) was proposed, which works for queries that are conjunctions or disjunctions of positive literals in the presence of universal constraints. This strategy was extended in [BB02] to deal with referential integrity constraints. A similar approach was proposed in [BB03], where repairs were specified as the stable models of a disjunctive logic program where the database predicates contain annotations as extra arguments (as opposed to annotated programs that contain annotated atoms).

3.2 Inconsistencies in Description Logics

The problematic ontologies fall into two categories: inconsistent and incoherent. An ontology is *inconsistent*, if it does not have any model. An ontology is *incoherent*, if its TBox contains at least one unsatisfiable concept, i.e. concept that is interpreted as the empty set in all models of the TBox.

For fixing inconsistent ontologies a number of techniques were proposed. In particular, [LLR^{++}11, Bie12] studied consistent query answering over DL-Lite ontologies based on the repair technique (see [Ber11]), using minimal deletion repairs. A work [RRGM12] is based on ABox cleaning in *DL-Lite*$_{\mathcal{A}}$ ontologies. There an inconsistent ontology (with consistent TBox) is repaired by identifying and eliminating minimal conflict sets, causing, i.e. explaining, inconsistency, thus resulting in maximal deletion repairs. A recent work [NMSN14] proposes an approach for efficient inconsistency detection in distributed *DL-Lite*$_{\mathcal{A}}$ knowledge bases. It relies on the generation of so-called clash queries in the distributed setting. The clash queries essentially amount to conflict sets considered by Lembo *et al* [LLR^{+}11].

A vast amount of research has been devoted to studying inconsistencies in ontologies in general without distinguishing the data part from the TBox, e.g. [PSK05, JHQ^{+}09, HvHtT05, HvHH^{+}05, WWT10]. For example, the work [JHQ^{+}09] proposes a technique for repairing inconsistent ontologies by generating minimal inconsistent subsets for resolving inconsistencies. A similar approach reported in [HvHH^{+}05] provides methodologies for extracting minimal inconsistent and maximal consistent subontologies. The authors of [HRDA11] deal with the identification of contradiction derivations under the integrity constraint rules defined in a logic program. For resolving detected inconsistencies, they generate all possible Minimal Inconsistent Resolve Candidates (MIRCs). These are defined as a set of asserted ontology triples whose removal results in a consistent subontology.

As for incoherent ontologies, a recent attempt described in [SGWB10] uses default logics for relaxing the axioms that cause incoherence, and shows how probabilistic description logics can be used to resolve conflicts. Other earlier approaches for debugging terminologies and resolving incoherence were proposed in [KPSG06, Sch05].

The most prominent techniques for paraconsistent reasoning in ontologies are based on the use of additional truth values standing for undefined (i.e. neither true nor false) and over-defined (or contradictory, i.e. both true and false). Such multi-valued semantics are defined for Description Logics in [MH09]. In [GCS10] the authors suggest to apply querying only on a consistent fragment of the inconsistent ontology. Identifying consistent fragments is based on a selection function, which can be defined by some syntactic or semantic relevance [HvHtT05]. One of the recent attempts for dealing with inconsistencies in ontologies was presented in [GCS10], where ontologies are expressed as defeasible logic programs (DeLP). For a given query, a dialectical analysis is performed on the DeLP program, where all arguments in favor and against the query results are taken into account [GCS10].

3.3 Inconsistencies in Logic Programs

Inconsistent logic programs are programs, lacking answer sets. A subset of inconsistent programs is called *incoherent*; these are lacking answer set for a particular reason of cyclic dependence of some atom on its default negation. The efforts towards detecting and solving inconsistencies in logic rules are mostly described in the papers that focus on debugging of logic programs (e.g. [GPST08, PSEK09, Syr06], etc).

The approach by Syränen [Syr06] is designed to find reasons for the absence of answer sets. It addresses the issue of debugging incoherent logic programs, which is adapted from the field of model-based diagnosis [Rei87]. A generalization of the same problem is given in the work [GPST08] which provides explanations why interpretations are not answer sets of a program under consideration. The latter method relies on a meta-programming technique, i.e., a program over a metalanguage manipulates another program over an object language. An important contribution of that work is a classification scheme of errors that is based on a rather intuitive characterization of answer sets [Lee05]. The consistency-restoring rules of Balduccini and Gelfond [BG03] are another related approach in this regard. The authors of this work define a method that allows a reasoning system to find the best explanation for conflicting observations. Furthermore,some previous work on the properties of odd and even cycles in a program and their effect on the existence and number of answer sets is given in such works as [YY94, LZ04].

Among the best known approaches in the area of paracoherent reasoning for logic programs are 3-valued stable models [Prz91], L-stable models [ELS97], revised stable models (Pereira *et. al*), regular models (You *et. al*), p-stable models (Osorio *et. al*) and semi-stable models (Sakama *et. al*). For more discussion of 3-valued stable and regular models as well as many other semantics coinciding with them, see [ELS97]. The paracoherent semantics proposed in [EFM10] is designed for paracoherent programs. The authors propose a semantic characterization of semi-stable models in terms of bi-models and of semi-equilibrium models was given. One of the weak points of this work is the absence of support of modularity properties of the programs. Therefore, lifting the semantics for programs with modules is an interesting and challenging issue. Such extension should exist, since the Equilibrium Logic is rather flexible.

Another issue with regard to [EFM10] is to lift the paracoherent semantics to the paraconsistent one, where the strong negation will also be allowed. The semantics can be further enhanced for treating programs with strong negation, nested programs, etc.

A recent work [DBV12] surveys and explains the application of the approximation fixpoint theory to the semantics of logic programming and answer set programming and generalizations of these.

3.4 Inconsistencies in Hybrid Logical Formalisms

In general, combining different pieces of knowledge is more vulnerable for contradictions than individual representations. Therefore, inconsistency handling in hybrid theories

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Child} \sqsubseteq \exists \mathit{hasParent} & (4)\ \mathit{Male}(\mathit{pat}) \\ (2)\ \mathit{Adopted} \sqsubseteq \mathit{Child} & (5)\ \mathit{Male}(\mathit{john}) \\ (3)\ \mathit{Female} \sqsubseteq \neg \mathit{Male} & (6)\ \mathit{hasParent}(\mathit{john}, \mathit{pat}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{ischildof}(\mathit{john}, \mathit{alex}); \quad (8)\ \ \mathit{boy}(\mathit{john}); \\ (9)\ \mathit{hasfather}(\mathit{john}, \mathit{pat}) \leftarrow \mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](\mathit{pat}), \mathrm{DL}[;\ \mathit{hasParent}](\mathit{john}, \mathit{pat}); \\ (10)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[;\ \mathit{Adopted}](\mathit{john}), \mathit{hasfather}(\mathit{john}, \mathit{pat}), \mathit{ischildof}(\mathit{john}, \mathit{alex}), \\ \qquad\qquad \mathit{alex} \neq \mathit{pat}, \mathit{not}\ \mathrm{DL}[\mathit{Child} \uplus \mathit{boy};\ \neg \mathit{Male}](\mathit{alex}) \end{array} \right\}$$

Figure 3.1: Inconsistent DL-program Π over a family ontology

poses a far non-trivial challenge. These logical theories might turn out to be inconsistent even if the pieces of information are perfectly consistent when considered separately.

Most of the approaches for handling inconsistencies in hybrid formalisms are based on paraconsistent reasoning. For example, the works [KAH08,HLH13] present a paraconsistent semantics for tightly-coupled MKNF hybrid KBs, which is based on the four-valued logic proposed by Belnap [Bel77]. A translation of this semantics to the stable model semantics via a linear transformation operator is also described by the authors. The work [Fin12a] presents a paraconsistent semi-equilibrium model semantics for abstract hybrid logical theories, given by a classical first order theory and a rules part as a set of declarative logic program rules.

The authors of [EFSW10,EFW14] focus on explaining inconsistency in multi-context systems (MCS), where decentralized and heterogeneous system parts interact via non-monotonic bridge rules. The techniques described in [EFSW10] characterize inconsistency in terms of bridge rules that are involved: either by pointing out rules which need to be altered for restoring consistency, or by finding combinations of rules which cause inconsistency. Among other related works in the context of MCS there are [BE07,BEFS10].

3.4.1 Analyzing Inconsistencies in DL-programs

Before delving into details about the existing research related to inconsistent DL-programs, let us first consider an inconsistent version of Example 2.30, depicted in Figure 3.1.

Example 3.1. In Chapter 2 we have already seen the ontology $\mathcal{O}$ and the rules (7)-(9) of Π from Figure 3.1. Consider now the additional constraint (10). Intuitively, it states that a child can not have two male parents unless it is adopted. Observe, that *pat* is a father of *john*, and *john* is not known to be adopted. Furthermore, *alex* is a parent of *john* who is not provably $\neg \mathit{Male}$. Therefore, the constraint (10) is applied for any model candidate, and Π does not have any answer sets. □

There are multiple strategies that one could follow for resolving inconsistencies in Π from the above example. Indeed, one could assume that the logic rules $\mathcal{P}$ of Π are not appropriate or that the ontology $\mathcal{O}$ contains some faulty (incomplete) information

bits. Under the former view suitable changes applied to $\mathcal{P}$ would resolve the problem. For instance, the constraint (10) can be dropped or changed to a rule by introducing *affiliated*(*john*) in its head. Under the latter view, i.e. if the reasons for conflicts are in $\mathcal{O}$, resolving them could be done by replacing *pat* with *alex* in axioms (4) and (6) or by adding the assertion *Adopted*(*john*) to $\mathcal{O}$. In addition to possible problems in rules or the ontology, the interfaces between them can be not fully correct. Provided that Π contains an additional fact *woman*(*alex*), for resolving contradictions in Π it would be plausible to extend the input signature λ of the DL-atom $a = \mathrm{DL}[\mathit{Child} \uplus \mathit{boy}; \neg \mathit{Male}](\mathit{alex})$ by adding $\mathit{Female} \uplus \mathit{woman}$ to it. This would lead to a being true in the model candidates, and thus the constraint (10) would not be applied any longer.

Summarizing these observations, we can naturally distinguish the following possible sources of inconsistency in a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$:

(1) the rules $\mathcal{P}$;
(2) the ontology $\mathcal{O}$;
(3) the interface $\mathrm{DL}[\lambda; Q](\vec{t})$ between $\mathcal{P}$ and $\mathcal{O}$;
(4) combinations of (1)-(3).

Handling inconsistencies in DL-programs is a relatively new issue, since the formalism of DL-programs itself is quite young. There were very few works that targeted this problem before the start of this thesis, and all of them focused on inconsistency tolerance. For example, in [PHE10], rules with DL-atoms that amount to queries over inconsistent ontologies are suppressed. The semantics for general hybrid theories presented in [Fin12a] pays particular attention to inconsistency due to the lack of stability in models (termed paracoherence), and it is in line with paraconsistent semantics for Description Logics [MHL08].

The problem of repairing DL-programs, i.e. changing formulas to obtain consistency (which is a natural and ubiquitous approach in knowledge representation) has not been attacked earlier. As we have seen due to multiple sources of inconsistencies (1)-(4), different possibilities to restore inconsistency exist. We now shed some light on the techniques that can be applied to restoring consistency of DL-programs while focusing on changing rules, ontology and interfaces respectively.

Repairing rules. Repairing rules in a DL-program subsumes repair of ordinary nonmonotonic programs, and thus poses a challenge as such, especially if repair goes beyond merely dropping rules.

Inconsistent DL-programs can be seen as programs with bugs, and for fixing these bugs suitable debugging techniques are required. In fact, the development of debugging techniques for DL-programs has started in [OPT12]. The proposed debugging approach is built on standard answer set debugging techniques [Syr06, GPST08, PÏ14]; the debugging is done in a user-interactive way, and it proceeds by stepping through the rules of the DL-program, and distinguishing at each step a set of rules that are active together with an intermediate interpretation. Faulty rules are identified, once a conflict is reached in the stepping process.

Debugging techniques can be applied to *modules* of DL-programs [DEFK09]. A given DL-program can be split into independent components such that each component is evaluated separately, and then the results are conveniently combined. Having modules of a DL-program clearly identified, one can focus on debugging/repairing the rules of each of the modules individually.

Repair of interfaces. Repairing interfaces might be addressed in different ways. One possibility is to modify the update specification λ in a DL-atom $\mathrm{DL}[\lambda; Q](\vec{t})$ and/or the query $Q(t)$ to effect a different information flow between the rules and the ontology. However, the search space for changes here is large and needs to be appropriately limited. Furthermore, user intervention will most likely be indispensable.

Another possibility is to change query evaluation: rather than simply expanding the ontology $\mathcal{O}$ with assertions assembled from the update specification λ, one incorporates them into $\mathcal{O}$ in a way such that consistency is preserved, using a revision or update operator [AGM85, GR95]; it remains to identify suitable such operators. Alternatively, the DL-queries can be evaluated using a suitable paraconsistent semantics.

Repairs of interfaces can be done in groups. One can build a partition of DL-atoms according to their mutual dependence, e.g. dependence w.r.t. value, update, DL-query. Repairing interfaces can be done in such partitions, for instance if an update $P \uplus q$ is removed from a DL-atom $\mathrm{DL}[P \uplus, q; D](\vec{t})$, then it might be worthwhile to consider the removal of the same update from the interface $\mathrm{DL}[P \uplus q, Q \uplus m; D](\vec{t})$. Further syntactical similarities among DL-atoms can be exploited.

Orthogonal to the actual repair techniques is a problem of selecting a concrete set of interfaces that should be changed from all possibilities. That is, assume we identified a set A of faulty DL-atoms, such that the logical value of at least one of them must be relaxed or changed for the overall system to become consistent. From the purely technical point of view, any atom a from the set A could be modified to do the job. However, obviously we are interested in making the best possible choice for such a. That is, we do not want to modify the interfaces which are a priori irrefutable.

The question that arises in this setting is how to find such irrefutable atoms and how to compare two interfaces with respect to their correctness. In many cases pure syntactic analysis will not bring satisfactory results, and thus additional semantic information about DL-atoms needs to be taken into account whenever possible. Ideally such domain dependent information should be provided by the designer of the program in such a way that it can be exploited during the inconsistency handling process. However, it is not clear

- what kind of information the program designer should provide about the DL-atoms;
- how this information could be represented in a convenient way;
- what framework can one offer to the user for encoding the additional information about the DL-atoms (e.g. forms of tagging DL-atoms).

These are open questions, to which answers are far nontrivial, but they go beyond the scope of this thesis. We leave the problem of modifying interfaces for future work.

Repair of ontology. Finally repair of ontologies is an important aspect and the main focus of this thesis. Naturally, the possible changes in the ontology repair concern the modifications of the TBox, ABox or both of them. Normally, the modifications of the TBox are less obvious, as the TBoxes are often carefully constructed, and consulting domain experts is often vital for making reasonable changes. If, however, the TBox still has to be repaired, application of techniques for dealing with incoherent ontologies need to be adapted accordingly.

In this work we concentrate on modifying the data part of the ontology (i.e. its ABox) for restoring consistency. The approaches that we propose are discussed in the next chapters.

The ideas and techniques that we mentioned above can be combined in the general setting of handling inconsistencies. For example, if it was decided to make suitable changes to the rules, the one can still choose to consider paraconsistent evaluation of DL-queries in the same scenario. The general semantics of DL-programs can be extended to semi-equilibrium model semantics [EFM10], and yet some changes to the ontology could be still allowed. A policy language that could make suggestions in the interactive mode about the suitable techniques for drawing plausible inferences from a DL-program could effectively guide the inconsistency handling process.

CHAPTER 4

DL-program Repair Semantics

As discussed earlier the powerful formalism of DL-programs permits a bidirectional information flow between the rule part and the ontology, which makes it attractive for various application scenarios. This information flow, however, can have unforeseen effects and cause inconsistency such that no answer set (i.e., model), of a DL-program exists. Absence of answer sets often makes the DL-program unusable, which calls for dealing with this problem, e.g. by computing a repair. In this chapter we tackle the problem of repair computation from the theoretical perspective by introducing the repair semantics and analyzing its complexity. To illustrate the notion of inconsistency in DL-programs, we turn to an extended version of Example 3.1 from the family domain.

Example 4.1. Consider the DL-program Π from Figure 4.1, which is an extended version of Example 3.1 with the additional rules (11) and (12). Intuitively, single out contact persons for children, which by default are the parents; for adopted children, fathers from the ontology are omitted, if some other contact exists.

The program Π does not have any answer sets, and therefore it is inconsistent. The inconsistency arises in this program as *john*, who is not provably adopted, has *pat* as father by the ontology, and by the local information possibly also *alex*; this causes the constraint (10) to be violated. □

The previous works that considered the problem of inconsistencies in hybrid formalisms, e.g. [PHE10, Fin12b] focused on inconsistency tolerance. Their approach was to tolerate inconsistency by suppressing or weakening information that leads to inconsistency in model building. However, the problem to *repair* the program, i.e., change formulas in it to obtain consistency (which is a natural and ubiquitous approach in data and knowledge representation), has to the best of our knowledge not been considered.

We study this issue, which due to the interaction between the rules and the ontology is nontrivial and at least as challenging as for these parts. Ontology repair has been

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Child} \sqsubseteq \exists \mathit{hasParent} & (4)\ \mathit{Male}(\mathit{pat}) \\ (2)\ \mathit{Adopted} \sqsubseteq \mathit{Child} & (5)\ \mathit{Male}(\mathit{john}) \\ (3)\ \mathit{Female} \sqsubseteq \neg \mathit{Male} & (6)\ \mathit{hasParent}(\mathit{john}, \mathit{pat}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{ischildof}(\mathit{john}, \mathit{alex}); \quad (8)\ \mathit{boy}(\mathit{john}); \\ (9)\ \mathit{hasfather}(X, Y) \leftarrow \mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](Y), \\ \qquad\qquad\qquad\qquad \mathrm{DL}[;\ \mathit{hasParent}](X, Y); \\ (10)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[;\ \mathit{Adopted}](X), Y_1 \neq Y_2, \\ \qquad\qquad \mathit{hasfather}(X, Y_1), \mathit{ischildof}(X, Y_2), \\ \qquad\qquad \mathit{not}\ \mathrm{DL}[\mathit{Child} \uplus \mathit{boy};\ \neg \mathit{Male}](Y_2) \\ (11)\ \mathit{contact}(X, Y) \leftarrow \mathrm{DL}[;\ \mathit{hasParent}](X, Y), \\ \qquad\qquad\qquad\qquad \mathit{not}\ \mathit{omit}(X, Y); \\ (12)\ \mathit{omit}(X, Y) \leftarrow \mathrm{DL}[;\ \mathit{Adopted}](X), Z \neq Y, \\ \qquad\qquad\qquad\qquad \mathit{hasfather}(X, Y), \mathit{contact}(X, Z) \end{array} \right\}$$

Figure 4.1: Extended version of a DL-program Π over a family ontology

studied in many works, e.g., in [LLR^{++}11, Bie12] for consistent query answering; repairing nonmonotonic logic programs instead is less developed (cf. [SI03]). In the light of this and as the rules are on top of the ontology (such that their plausibility can be separately assessed), we take the view that the latter, and here in particular its data part might not be fully correct.

In the above example, suitable changes of the ontology facts make the DL-program consistent. For example, deleting $\mathit{hasParent}(\mathit{john}, \mathit{pat})$ from $\mathcal{A}$ leads to the answer set $I_1 = \{\mathit{ischildof}(\mathit{john}, \mathit{alex}),\ \mathit{boy}(\mathit{john})\}$, alternatively the addition of $\mathit{Adopted}(\mathit{john})$ leads to $I_2 = \{\mathit{ischildof}(\mathit{john}, \mathit{alex}),\ \mathit{boy}(\mathit{john}),\ \mathit{hasfather}(\mathit{john}, \mathit{pat}), \mathit{contact}(\mathit{john}, \mathit{pat})\}$; yet other possibilities exist.

However, not all repairs might be acceptable; the question is how to find suitable repairs which additionally do not increase the complexity of DL-programs.

In this chapter we tackle this challenge with the following contributions:

(1) We first formalize repairing DL-programs and introduce the notions of repair and repair answer set in Section 4.1. They are based on changes of the assertions in the ontology that enable answer sets. As it turns out, repair answer sets do not have higher complexity than ordinary answer sets (more precisely, weak and FLP answer sets) if queries in DL-atoms are evaluable in polynomial time; to ensure this, we concentrate on the lightweight Description Logic *DL-Lite*$_\mathcal{A}$ [CLLR07]. We also consider a bit complexity for repair answer set computation of DL-programs over Description Logic $\mathcal{EL}$, in which like in *DL-Lite*$_\mathcal{A}$ the query answering is feasible in polynomial time [BBL05].

(2) We model repair preference by functions σ introduced in Section 4.2. These functions select preferred repairs from a set of candidates. As preference is a known source of complexity, we focus on selections σ with a benign independence property (which some practicable σ selections enjoy). Importantly, the locally selected ABoxes also yield, modulo a conditional check on the rules part, the σ-selected repairs of the program.

(3) As a subtask of repair computation, we introduce a *generalized ontology repair problem* (ORP) in Section 4.3, which arises from a candidate answer set and the DL-atoms of the program. It consists of two sets D_1, D_2 of entailment and non-entailment queries to the ontology, with possible temporary assertions, and asks for an ABox satisfying these sets.

(4) Furthermore, we analyze the complexity of the ORP problem. Unsurprisingly, it is intractable (NP-complete) for *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ in general, but we show that NP-hardness holds also in plain ontology settings, due to the temporary assertions. However, in Section 4.4 we also identify several tractable cases of σ-selections for *DL-Lite*$_\mathcal{A}$ that are useful in practical applications.

(5) To ensure usefulness of repairs we propose in Section 4.5 various domain-dependent types of restrictions that can be applied on top of σ-selections when computing repairs. Incorporating domain-specific knowledge into the repair process contributes to the practicability of our approach.

While we focus here on lightweight DLs, i.e. *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$, our repair framework and basic results can be extended to DL-programs with ontologies in other Description Logics as well. Further ideas regarding an extension of our approach and possible directions of future work are summarized in Section 4.6.

4.1 Repair Answer Sets

We now turn straight to repairing an inconsistent DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$. In our setting, we assume that the rule part $\mathcal{P}$, which is on top of the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, is reliable and that the cause for inconsistency is in the latter. Thus when searching for a repair, modifications should only be applied to $\mathcal{O}$. In principle, the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ of the ontology could be subject to change; however, as usually the TBox is well-developed and a suitable TBox change is less clear in general (the more by an external user), we confine to change only the ABox.

Hence given a possibly inconsistent DL-program, our goal is to find an ABox $\mathcal{A}'$ such that replacing the ABox $\mathcal{A}$ by $\mathcal{A}'$ makes the DL-program consistent. The answer sets of such a "repaired" DL-program are then referred to as *repair answer sets* of the program.

Formally, they are defined as follows.

Definition 4.2. Given a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, an ABox $\mathcal{A}'$ is an *x-repair* of Π, where $x \in \{\mathit{flp}, \mathit{weak}\}$, if

(i) $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ is consistent, and
(ii) $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$ has some x-answer set.

By $rep_x(\Pi)$ we denote the set of all x-repairs of Π.

An interpretation I is an *x-repair answer set* of Π, if $I \in AS_x(\Pi')$, where $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$, $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$, and $\mathcal{A}' \in rep_x(\Pi)$. By $RAS_x(\Pi)$ we denote the set of all x-repair answer sets of Π.

$AS_x(\Pi) \mid RAS_x(\Pi) \neq \emptyset$?	normal Π	disjunctive Π
$x = weak$	$\mathsf{NP} \mid \mathsf{NP}$	$\Sigma_2^P \mid \Sigma_2^P$
$x = \mathit{flp}$	$\Sigma_2^P \mid \Sigma_2^P$	$\Sigma_2^P \mid \Sigma_2^P$

Table 4.1: Complexity of deciding weak and *flp* answer set existence for ground DL-programs over *DL-Lite*$_\mathcal{A}$ ontologies (completeness results)

Furthermore, by $rep_x^I(\Pi) = \{\mathcal{A}' \in rep_x(\Pi) \mid I \in AS_x(\Pi'), \Pi' = \langle \mathcal{O}', \mathcal{P}\rangle, \mathcal{O}' = \langle \mathcal{T}, \mathcal{A}'\rangle\}$ we denote the set of all ABoxes $\mathcal{A}'$ under which I becomes an x-answer set of Π.

Example 4.3. Reconsider Π in Example 4.1. The interpretation $I_1 = \{boy(john), ischildof(john, alex)\}$ is an *flp*-repair answer set with *flp*-repair $\mathcal{A}'_1 = \{Male(john), Male(pat)\}$. Another *flp*-repair for I_1 is $\mathcal{A}'_2 = \{hasParent(john, pat), Female(pat), Male(john)\}$. The interpretation I_1 is also a *weak*-repair answer set with the *weak*-repairs $\mathcal{A}'_1$ and $\mathcal{A}'_2$.

4.1.1 Complexity of RAS existence for DL-programs over *DL-Lite*$_\mathcal{A}$ DL

We now look at the problem of deciding whether a given DL-program $\Pi = \langle \mathcal{O}, \mathcal{P}\rangle$ has an x-(repair) answer set for $x \in \{flp, weak\}$. Table 4.1 compactly summarizes our complexity results for this problem for $\mathcal{O}$ in *DL-Lite*$_\mathcal{A}$.

Before formally addressing the complexity of repair answer sets, we first state the following proposition:

Proposition 4.4. *Given any $I \subseteq HB_\Pi$, $\mathcal{O}$ in DL-Lite$_\mathcal{A}$, and a DL-atom $a = \mathrm{DL}[\lambda; Q](\vec{t})$, deciding $I \models^\mathcal{O} a$ is feasible in polynomial time.*

Proof. Deciding whether $I \models^\mathcal{O} a$ is equivalent to checking $\mathcal{O} \cup \lambda^I(a) \models Q(\vec{t})$. As instance checking is known to be polynomial [CLLR07] in *DL-Lite*$_\mathcal{A}$, the result immediately follows. □

Now we are ready to formally prove the complexity results for checking the repair answer set existence that we obtained.

Theorem 4.5. *Given a ground DL-program $\Pi = \langle \mathcal{O}, \mathcal{P}\rangle$, deciding whether $RAS_x(\Pi) \neq \emptyset$ is* NP*-complete for normal Π and $x = weak$ if the ontology $\mathcal{O}$ is in DL-Lite$_\mathcal{A}$ DL.*

Proof. (Membership) Let $\Pi = \langle \mathcal{O}, \mathcal{P}\rangle$ be a DL-program, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$. We guess an interpretation I, the value of the DL-atoms and the repair ABox $\mathcal{A}'$.

We then check whether I is a repair answer set. For that we first (i) evaluate all DL-atoms over $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}'\rangle$ and compare their values with the guessed values. If this

test succeeds, then we (ii) check whether the interpretation candidate I is minimal of the reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$. The check (i) is feasible in polynomial time, which follows from the Proposition 4.4. As for the check (ii), observe that the reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$ is a normal positive logic program without any DL-atoms. Hence, it must have a single model. We can check whether I is a model of $\mathcal{P}^{I,\mathcal{O}'}_{weak}$ in polynomial time. The above algorithm solves the target problem, which proves its membership in NP.

(Hardness) To prove hardness we provide a reduction from the NP-complete 3SAT problem to deciding weak repair answer set existence for a DL-program over a *DL-Lite*$_{\mathcal{A}}$ ontology as follows.

Let $\phi = \bigwedge_{1 \leq j \leq m} C_j$ be an instance of the 3SAT problem, where each C_j is a disjunction of three atoms over the variables $x_1, \ldots, x_n$. From this we construct a DL-program $\Pi = \langle \mathcal{T}, \mathcal{A}, \mathcal{P} \rangle$.

- Construction of the TBox $\mathcal{T}$:

  We introduce concept names X_i and $\bar{X}_i$, for each variable x_i occurring in ϕ. Moreover, we introduce a concept name C_j for each clause C_j in ϕ. Then the TBox $\mathcal{T}$ is constructed as follows:

  - $X_i \sqsubseteq C_j$ iff x_i is a disjunct in C_j;
  - $\bar{X}_i \sqsubseteq C_j$ iff $\neg x_i$ is a disjunct in C_j;
  - $X_i \sqsubseteq \neg \bar{X}_i$ and $\bar{X}_i \sqsubseteq \neg X_i$ for all pairs X_i, $\bar{X}_i$;

- The ABox is $\mathcal{A} = \{D(b)\}$, where D and b are a fresh concept and a fresh constant respectively.
- Construction of $\mathcal{P}$:

  We introduce fresh ground atoms $p_i(b)$ (resp. $\bar{p}_i(b)$) for each x_i occurring positively (resp. negatively) in ϕ. The rules of $\mathcal{P}$ are as follows:

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ \bot \ \leftarrow\ \mathrm{DL}[;\, D](b); \\ (2)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[;\, C_1](b); \\ \quad \ldots \\ (3)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[;\, C_m](b); \\ (4)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[\lambda_1;\, \neg C_1](b); \\ \quad \ldots \\ (5)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[\lambda_m;\, \neg C_m](b); \\ (6)\ p_i(b) \mid p_i \text{ occurs in } \lambda_j, 1 \leq j \leq m; \\ (7)\ \bar{p}_i(b) \mid \bar{p}_i \text{ occurs in } \lambda_j, 1 \leq j \leq m \end{array} \right\},$$

  where for each x_i, we have $X_i \uplus p_i$ (resp. $\bar{X}_i \uplus \bar{p}_i$) occurs in λ_j if $\neg x_i$ (resp. x_i) is a disjunct in C_j of ϕ. In addition, $\mathcal{P}$ contains the facts $p_i(b)$ (resp. $\bar{p}_i(b)$) iff x_i (resp. $\bar{x}_i$) occurs in some λ_j.

We claim that ϕ is satisfiable iff $RAS_{weak}(\Pi) \neq \emptyset$, i.e. there exists an interpretation I and an ABox $\mathcal{A}'$ such that I is a weak answer set of $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$.

($\Rightarrow$) Suppose that ϕ is satisfiable and $A(\phi)$ is a satisfying assignment. From this we construct a repair answer set I of Π and a repair ABox $\mathcal{A}'$ as follows. The interpretation I contains the set of all ground atoms that occur as facts in $\mathcal{P}$. The ABox $\mathcal{A}'$ is such that $X_i(b)$ (resp. $\bar{X}_i(b)$) is in $\mathcal{A}'$ if x_i is true (resp. false) under the assignment $A(\phi)$.

Example 4.6. Let $\phi = x_1 \vee \neg x_2 \vee x_3$ be an instance of a 3SAT problem. From this we construct a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ as follows:

The TBox is $\mathcal{T} = \{X_1 \sqsubseteq C_1;\ \bar{X}_2 \sqsubseteq C_1;\ X_3 \sqsubseteq C_1\}$; the ABox is $\mathcal{A} = \{D(b)\}$; the program is

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ \bot \leftarrow \mathrm{DL}[;D](b); \\ (2)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[;C_1](b); \\ (3)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[\bar{X}_1 \uplus \bar{p}_1,\ X_2 \uplus p_2, \bar{X}_3 \uplus \bar{p}_3;\ \neg C_1](b); \\ (4)\ \bar{p}_1(b);\quad (5)\ p_2(b);\quad (6)\ \bar{p}_3(b) \end{array} \right\}.$$

□

Now we show that I is a weak repair answer set of Π with the repair $\mathcal{A}'$. For that we prove that I is a weak answer set of $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. Observe that the body of the rule (1) is not satisfied, as $\mathcal{A}'$ does not contain the fact $D(b)$. Furthermore, the DL-atoms $\mathrm{DL}[;C_1](b), \ldots, \mathrm{DL}[;C_m](b)$ evaluate to true under $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$, since $\mathcal{O}' \models C_j(b)$ for all $1 \leq j \leq m$ by construction. Moreover, each $d_j = \mathrm{DL}[\lambda_j;\ \neg C_j](b)$ evaluates to true under I, because the ontology $\mathcal{O}' \cup \lambda^I(d_j)$ is unsatisfiable (by construction $X_i(b) \in \mathcal{A}'$ or $\bar{X}_i(b) \in \mathcal{A}'$ for some $X_i \sqsubseteq C_j$ resp. $\bar{X}_i \sqsubseteq C_j$), and thus each $\neg C_j(b)$ is trivially entailed. Therefore, none of the constraints of $\mathcal{P}$ is present in the program reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$. The reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$ contains only facts of the program, from which we get that I is a weak repair answer set of Π.

($\Leftarrow$) Let I be a weak repair answer set of Π and let $\mathcal{A}'$ be its respective repair. Then all DL-atoms of Π apart from $\mathrm{DL}[;D](b)$ are true. This means that for all C_j it holds that $\mathcal{O}' \models C_j(b)$. The ontology $\mathcal{O}' = \mathcal{T}, \mathcal{A}'$ is satisfiable, therefore $X_i(b)$ and $\bar{X}_i(b)$ can not simultaneously be in $\mathcal{A}'$. Therefore, either

(i) $C_j(b) \in \mathcal{A}'$ or

(ii) $X(b) \in \mathcal{A}'$, such that $X \sqsubseteq C_j$ is in $\mathcal{T}$.

If (i) was true, then the bodies of the constraints (4) would be satisfied, which contradicts I being a repair answer set. Thus, it holds that some $X(b) \in \mathcal{A}'$ such that $X \sqsubseteq C_j \in \mathcal{T}$. Therefore, from the repair ABox $\mathcal{A}'$ a satisfying assignment $A(\phi)$ can be constructed as follows: $A(\phi)$ such that $A(x_i) = \mathit{true}$ (resp. $A(a_i) = \mathit{false}$) if $X_i(b) \in \mathcal{A}'$ (resp. $\bar{X}_i(b) \in \mathcal{A}'$). The assignment $A(\phi)$ witnesses satisfiability of ϕ. □

Theorem 4.7. *Given a ground DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, deciding whether $RAS_x(\Pi) \neq \emptyset$ is Σ^P_2-complete for normal Π and $x = \mathit{flp}$, if $\mathcal{O}$ is in DL-Lite$_\mathcal{A}$ DL.*

Proof. (Membership) We can guess a repair $\mathcal{A}'$ together with an interpretation I and then check whether I is an *flp*-repair answer set of $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$, where $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$. Constructing the reduct $\mathcal{P}^{I,\mathcal{O}'}_{flp}$ is polynomial, as we only need to pick those rules of Π whose body is satisfied by I, and all DL-atoms can be evaluated in polynomial time. With the reduct $\mathcal{P}^{I,\mathcal{O}'}_{flp}$ at hand we then need to check whether (i) $I \models \mathcal{P}^{I,\mathcal{O}'}_{flp}$, and (ii) I is a minimal model of $\mathcal{P}^{I,\mathcal{O}'}_{flp}$. The check (i) can be done in polynomial time. For (ii) we have that the interpretation I is not a minimal model of $\mathcal{P}^{I,\mathcal{O}'}_{flp}$ iff there exists an interpretation $I' \subset I$ such that $I' \models \mathcal{P}^{I,\mathcal{O}'}_{flp}$. A guess for I' is verifiable in polynomial time, thus deciding whether I is not an answer set of $\mathcal{P}^{I,\mathcal{O}'}_{flp}$ is in NP. From this we get that deciding whether I is an answer set of $\mathcal{P}^{I,\mathcal{O}'}_{flp}$ is in co-NP. Hence for the check (ii) we need to make a call to a co-NP oracle. Since having an oracle for co-NP is equivalent to having an oracle for NP, we get that the overall problem can be solved in $\mathsf{NP}^{\mathsf{NP}} = \Sigma^P_2$.

(Hardness) We prove the Σ^p_2-hardness result by a reduction from the problem of deciding validity of a QBF formula

$$\phi = \exists x_1 \dots x_n \forall y_1 \dots y_m E, \quad n, m \geq 1, \tag{4.1}$$

where $E = \chi_1 \vee \dots \vee \chi_r$ is a DNF formula, and each $\chi_k = l_{k_1} \wedge l_{k_2} \wedge l_{k_3}$ is a conjunction of literals over atoms $x_1, \dots, x_n, y_1, \dots, y_m$.

For each atom x_i we introduce a fresh concept X_i, and for each atom y_j we introduce a fresh concept Y_j and a fresh logic program predicate y_j. Furthermore, we introduce an additional fresh predicate w. Given ϕ, we construct $\Pi = \langle \emptyset, \mathcal{A}, \mathcal{P} \rangle$ with $\mathcal{A} = \{X_1(b), \dots, X_n(b)\}$ and $\mathcal{P}$ as follows:

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[;\, X_i](b), \mathit{not}\ \mathrm{DL}[;\, \neg X_i](b); \\ (2)\ \bot \leftarrow \mathrm{DL}[;\, Y_j](b); \\ (3)\ \bot \leftarrow \mathrm{DL}[;\, \neg Y_j](b); \\ (4)\ w(b) \leftarrow \mathit{not}\ w(b); \\ (5)\ y_j(b) \leftarrow w(b); \\ (6)\ w(b) \leftarrow f(l_{k_1}), f(l_{k_2}), f(l_{k_3}) \end{array} \right\},$$

where $f(x_i) = \mathrm{DL}[X_i \uplus w;\, X_i](b)$,
$f(\neg x_i) = \mathrm{DL}[X_i \uplus w;\, \neg X_i](b)$,
$f(y_j) = \mathrm{DL}[Y_j \uplus y_j, Y_j \uplus w;\, Y_j](b)$,
$f(\neg y_j) = \mathrm{DL}[Y_j \cap y_j, Y_j \uplus w;\, \neg Y_j](b)$.

Intuitively, the rules of the form (1) of $\mathcal{P}$ ensure that for each x_i at least one of $X_i(b)$ and $\neg X_i(b)$ is present in the repair ABox $\mathcal{A}'$, while the rules (2) and (3) forbid that $Y_j(b)$ resp. $\neg Y_j(b)$ is in $\mathcal{A}'$. The rule (4) forces each consistent *flp*-repair answer set of Π to contain $w(b)$. The rule (5) ensures that the ground atoms of the form $y_j(b)$ are also contained in each repair answer set. Finally, the rules of the form (6) are present in $\mathcal{P}$ for

each clause χ_k of ϕ. For each literal l_{k_h} in χ_k these rules have a DL-atom $f(l_{k_h})$ in the body, which poses to the ontology under some updates an instance query corresponding to the literal l_{k_h}.

We now formally show that ϕ is valid iff $RAS_{\mathit{flp}}(\Pi) \neq \emptyset$.

($\Rightarrow$) Let ϕ be valid and let $A(\phi)$ be a satisfying assignment, i.e. for all extensions of A to variables $y_1, \ldots, y_m$ it holds that $A(\phi)$ is true. From this we construct a repair ABox $\mathcal{A}'$ as follows. If $A(x_i) = \mathit{true}$, then $X_i(b) \in \mathcal{A}'$, otherwise $\neg X_i(b) \in \mathcal{A}'$. By construction the repair $\mathcal{A}'$ represents a maximal consistent subset of $\{X_i(b),\ \neg X_i(b) \mid 1 \leq i \leq n\}$. Therefore, the constraints (1)-(3) are not violated under $\mathcal{A}'$.

We now show that for any interpretation I the body of at least one rule of the form (6) of Π' must be satisfied by I. Let us consider various possibilities for an interpretation I of Π'.

- $I \cap \{y_1(b), \ldots, y_m(b)\} = \emptyset$. Let us look at an extension A' of A, under which all variables y_j of ϕ are false. Since $A'(\phi) = \mathit{true}$, there must exist a clause χ_k, such that $A'(\chi_k) = \mathit{true}$. Consider the rule r_k of the form (6) that corresponds to χ_k. The clause χ_k is a conjunction of literals, thus all of its conjuncts over y_j must be negative. We have that each $\neg y_j$ occurring in χ_k corresponds to a DL-atom of the form $f(\neg y_j) = \mathrm{DL}[Y_j \sqcap y_j, Y_j \uplus w;\ \neg Y_j](b)$. As $y_j(b) \notin I$, it holds that $\lambda^I(f(y_j)) = \{\neg Y_j(b)\}$, leading to $I \models^{\mathcal{O}'} f(\neg y_j)$. All DL-atoms of the forms $f(x_i)$ and $f(\neg x_i)$ are satisfied by the construction of $\mathcal{A}'$.

- $I \cap \{y_1(b), \ldots y_m(b)\} \neq \emptyset$. Let us look at an extension A' of A such that

$$A'(y_j) = \begin{cases} \mathit{true}, & \text{if } y_j(b) \in I \\ \mathit{false}, & \text{if } y_j(b) \notin I. \end{cases}$$

  Since $A(\phi)$ is a satisfying assignment of ϕ, there must exist a clause χ_k in ϕ such that $A'(\chi_k) = \mathit{true}$. Let us look at the rule r_k of the form (6) corresponding to χ_k. For all literals l_{k_h} we have that $I \models f(l_{k_h})$. Indeed, if l_{k_h} is a literal over x_i, then the corresponding DL-atom is true by construction of $\mathcal{A}'$. If $l_{k_h} = y_j$ then as $A'(y_j) = \mathit{true}$ we have that $y_j(b) \in I$ and thus $\lambda^I(f(y_j)) = \{Y_j(b)\}$. Similarly, if $l_{k_h} = \neg y_j$, then $\lambda^I(f(\neg y_j)) = \{\neg Y_j(b)\}$. Therefore, $I \models f(l_{k_h})$ for all l_{k_h} occurring in χ_k.

So we have that for any I the body of at least one rule r_k of the form (6) must be satisfied, and hence the rule r_k must be present in the reduct $\mathcal{P}^{I,\mathcal{O}'}_{\mathit{flp}}$. Moreover, if Π' has some *flp*-answer set I, then it must contain $w(b)$ (this follows from $w(b) \leftarrow \mathit{not}\ w(b)$), and thus the rule of the form (4) is not in $\mathcal{P}^{I,\mathcal{O}'}_{\mathit{flp}}$. Finally, according to the rules (5) the answer set I should also contain all $y_j(b)$ for $1 \leq j \leq m$.

Since there are no other atoms which could potentially be in the answer set, we now show that $I = \{w(b), y_1(b), \ldots, y_m(b)\}$ is a minimal model of $\mathcal{P}^{I,\mathcal{O}}_{\mathit{flp}}$. First, obviously I satisfies all rules of the reduct; we only need to show its minimality. Towards a contradiction, assume that there is an interpretation $I' \subset I$, such that $I' \models \mathcal{P}^{I,\mathcal{O}'}_{\mathit{flp}}$.

There are two possibilities: either $w(b) \in I'$ or $w(b) \notin I'$. The former can not be true, as then there is some $y_j(b)$, such that $y_j(b) \notin I'$, and hence for some rule r of the form (5) we have that r is not satisfied by I'. If the latter holds, then we know that there are no rules of the form (6), whose body is satisfied by I'. Consider an extension A'' of the assignment A to the atoms y_j, such that $A''(y_j) = true$, if $y_j(b) \in I'$, and $A''(y_j) = false$ otherwise. We know that $A''(\phi) = true$, i.e. there is a disjunct χ_k in ϕ, such that $A''(\chi_k) = true$. Let us look at the rule r_k corresponding to the conjunct χ_k. All DL-atoms $f(x_i)$ are satisfied by I', due to the construction of the ABox $\mathcal{A}'$. The DL-atoms of the forms $f(y_j)$ are satisfied by I', because $y_j(b) \in I'$, and thus $\lambda^I(f(y_j)) \models Y_j(b)$. Similarly, as $y_j(b) \notin I'$, we have that $\lambda^I(f(\neg y_j)) \models \neg Y_j(b)$, leading to the satisfaction of the DL-atoms of the form $f(\neg y_j)$ by I'. Hence I' must satisfy $B(r_k)$; but since $w(b) \notin I'$, we have that $I' \not\models r_k$, leading to a contradiction. Therefore, I is indeed an *flp*-repair answer set of Π.

($\Leftarrow$) Let $I \in RAS_{flp}(\Pi)$ be some *flp*-repair answer set of Π with a repair ABox $\mathcal{A}'$, i.e. $I \in AS_{flp}(\langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle)$. Since I is a repair answer set, the repair ABox $\mathcal{A}'$ must contain a nonempty consistent subset of $\{X_i(b), \neg X_i(b)\}$, $1 \leq i \leq n$ because of constraints of the form (1). We construct an assignment A of ϕ from $\mathcal{A}'$ as follows:

$$A(x_i) = \begin{cases} true, & \text{if } X_i(b) \in \mathcal{A}' \\ false, & \text{if } \neg X_i(b) \in \mathcal{A}'. \end{cases}$$

We now show that A is a satisfying assignment of ϕ, i.e. for any extension A' of A to the values of y_j, we have that $A'(\phi) = true$. Towards a contradiction, assume that this is not the case, i.e. there exists an extension A' of A to the values of y_j, such that $A'(\phi) = false$, that is $A'(\chi_k) = false$ for all clauses χ_k of ϕ.

Let us now look at the interpretation I' of Π', such that $y_j(b) \in I'$, if $A'(y_j) = true$ and $y_j(b) \notin I'$, if $A'(y_j) = false$. We know that $I \supset I'$ is a minimal model of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$. Therefore, it must hold that $I' \not\models r$ for some rule r of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$, i.e. $I' \models B(r)$, but $I' \not\models H(r)$. Observe that the reduct $\mathcal{P}_{flp}^{I,\mathcal{O}'}$ contains only the rules (5) and (6). Since $w(b) \notin I'$, the rule r that I' does not satisfy can not be of the form (5), hence it must be of the form (6). Let us look at the corresponding clause χ_k in ϕ. We know that $A'(\chi_k) = false$, i.e. there is a conjunct l_{k_h} in χ_k, such that $A'(l_{k_h}) = false$. We distinguish the following cases:

- l_{k_h} is a literal over x_i. We know that $\lambda^{I'}(f(l_{k_h})) = \emptyset$, because $w(b) \notin I$. Thus it must be true that $\mathcal{A}' \models f(l_{k_h})$. Since $\mathcal{A}'$ is a repair, by Definition 4.17 it must be consistent. Thus the query of $f(l_{k_h})$ must be explicitly present in $\mathcal{A}'$, i.e. $X_i(b) \in \mathcal{A}'$, if $l_{k_h} = x_i$; $\neg X_i(b) \in \mathcal{A}'$, if $l_{k_h} = \neg x_i$. However, then by construction of A' we have that $A(l_{k_h}) = true$, which leads to a contradiction.

- l_{k_h} is a literal over y_j. There are two possibilities: either $l_{k_h} = y_j$ or $l_{k_h} = \neg y_j$.

  - First assume that $l_{k_h} = y_j$. We know that the corresponding DL-atom $f(y_j) = \mathrm{DL}[Y_j \uplus y_j, Y_j \uplus w; Y_j](b)$ is true under I'. Since the repair ABox $\mathcal{A}'$ is

consistent and does not contain any concepts of the form $Y_j(b)$, it must hold that $\lambda^{I'}(f(y_j)) \models Y_j(b)$. Observe that $w(b) \notin I'$, thus it must be true that $y_j(b) \in I'$; however, then $A'(l_{k_h}) = true$, contradicting our assumption.

– Now suppose that $l_{k_h} = \neg y_j$. Again we have that $I' \models f(\neg y_j)$, where $f(\neg y_j) = \mathrm{DL}[Y_j \cap y_j, Y_j \uplus w; \neg Y_j](b)$. Since $w(b) \notin I'$, it must hold that $y_j(b) \notin I'$. Therefore, $A'(y_j) = false$, i.e. $A'(l_{k_h}) = true$ leading to a contradiction.

We have shown that A' is a satisfying assignment for ϕ for each extension of A to variables y_j, from which the validity of ϕ follows.

□

Theorem 4.8. *Given a ground DL-program* Π*, deciding whether* $RAS_x(\Pi) \neq \emptyset$ *is* Σ_2^P*-complete for arbitrary* Π *and* $x = weak$*.*

Proof. (Membership) The overall algorithm of deciding the existence of a weak repair answer set proceeds as follows: We guess an interpretation I, values of DL-atoms and an ABox $\mathcal{A}'$ and then check whether:

(1) the real values of DL-atoms over I and $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ coincide with the guessed values;
(2) I is a minimal model of $\mathcal{P}^{I,\mathcal{O}'}_{weak}$.

The condition (1) can be checked in polynomial time. Regarding the condition (2) the reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$ can be clearly constructed in polynomial time for $\mathcal{O}$ in *DL-Lite*$_\mathcal{A}$. $\mathcal{P}^{I,\mathcal{O}}_{weak}$ is a propositional program. Deciding whether I is its minimal model can be verified with a call to an NP oracle, from which the membership in Σ_2^P follows.

(Hardness) To prove Σ_2^P hardness, we provide a reduction from the problem of validity of a QBF formula $\phi = \exists x_1 \dots x_n \forall y_1 \dots y_m E$, $n, m \geq 1$. We may assume that $E = C'_1 \vee \dots \vee C'_r$ and each $C'_i = L_{i,1} \wedge L_{i,2} \wedge L_{i,3}$ is a conjunction of literals over atoms $x_1, \dots, x_n, y_1, \dots, y_m$. Deciding whether ϕ is valid is Σ_2^P-hard.

We construct the DL-program $\Pi = \langle \emptyset, \{C(b)\}, \mathcal{P} \rangle$, which is depicted in Figure 4.2. Intuitively, $x_i(b)$ and $\bar{x}_i(b)$ correspond to the literals x_i and $\neg x_i$ of ϕ. Similarly, $y_j(b)$ and $\neg y_j(b)$ refer to y_j and $\neg y_j$ of ϕ. If I is a *weak*-repair answer set then $w(b)$ must be in I (follows from (3)). Furthermore, all $y_j(b)$ and $\bar{y}_j(b)$ must be in I because of the rules (4) and (5). Due to the rule (1) I must also contain at least one of $x_i(b)$ and $\bar{x}_i(b)$; by the minimality of I it can contain only one such atom. Every rule r_k of the form (7) corresponds to some clause χ_k of ϕ, and every DL-atom a_{k_h} in r_k to a literal l_{k_h} in χ_k. The interpretation I must satisfy $B(r_k)$ at least for one rule r_k of the form (7), which follows from the minimality of I. Satisfaction of $B(r_k)$ for such rule r_k guarantees the validity of ϕ, since a DL-atom a_{k_h} can be satisfied by I only due to its update $\lambda^I(a_{k_h})$, which depends on the guesses of atoms over x_i, if l_{k_h} is of the form x_i or $\neg x_i$, and always entails the respective DL-query otherwise.

We now formally show that ϕ is valid iff some *weak*-repair answer set of Π exists.

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ x_i(b) \vee \bar{x}_i(b); \\ (2)\ y_j(b) \vee \bar{y}_j(b); \\ (3)\ w(b)\ \leftarrow\ not\ w(b); \\ (4)\ y_j(b)\ \leftarrow\ w(b); \\ (5)\ \bar{y_j}(b)\ \leftarrow\ w(b); \\ (6)\ w(b)\ \leftarrow y_j(b), \bar{y_j}(b); \\ (7)\ w(b)\ \leftarrow\ f(l_{k_1}), f(l_{k_2}), f(l_{k_3}); \\ (8)\ \bot\ \leftarrow\ \mathrm{DL}[;\ C](b); \\ (9)\ \bot\ \leftarrow\ \mathrm{DL}[;\ X_i](b); \\ (10)\ \bot\ \leftarrow\ \mathrm{DL}[;\ \neg X_i](b); \\ (11)\ \bot\ \leftarrow\ \mathrm{DL}[;\ Y_j](b); \\ (12)\ \bot\ \leftarrow\ \mathrm{DL}[;\ \neg Y_j](b) \end{array} \right\},$$

$$\begin{array}{rl} \text{where } f(x_i) & = \mathrm{DL}[X_i\ \uplus\ x_i;\ X_i](b), \\ f(\neg x_i) & = \mathrm{DL}[X_i \uplus \bar{x}_i;\ \neg X_i](b), \\ f(y_j) & = \mathrm{DL}[Y_j\ \uplus\ y_j, Y_j \uplus w;\ Y_j](b), \\ f(\neg y_j) & = \mathrm{DL}[Y_j \cap y_j, Y_j \uplus w;\ \neg Y_j](b). \end{array}$$

Figure 4.2: DL-program Π from the proof of Theorem 4.8

($\Rightarrow$) Let ϕ be valid. Then there exists a satisfying assignment $A(\phi)$, such that every extension A' of A to $y_1, \ldots, y_m$ satisfies ϕ. Let I be the following interpretation:

$$I = \{x_i(b) : A(x_i) = true\} \cup \{\bar{x}_j(b) : A(x_j) = false\} \cup \{y_1(b), \ldots, y_m(b), w(b)\}.$$

We prove that I is a *weak*-repair answer set of Π, i.e. some repair ABox $\mathcal{A}'$ exists, such that $I \in AS_{weak}(\langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle)$. Let us consider $\mathcal{A}' = \emptyset$. We have that the bodies of the constraints (8)-(12) are not satisfied, therefore they are not present in $\mathcal{P}^{I,\mathcal{O}'}_{weak}$. Furthermore, at least one rule of the form (7) is satisfied by I. Indeed, let us look at some extension A' of the assignment A to the variables y_j. We have that $A'(\phi) = true$. Therefore, for some clause χ_k of ϕ, $A'(\chi_k) = true$. Consider the corresponding rule r_k of the form (7) in $\mathcal{P}$. By construction of I all DL-atoms of the form $f(x_i)$ or $f(\neg x_i)$ are satisfied by I. The rest of the DL-atoms are also satisfied by I due to $w(b) \in I$ and $y_j(b) \in I$. This means that $\mathcal{P}^{I,\mathcal{O}'}_{weak}$ contains the rules (1), (2), (4)-(6) and a fact $w(b)$.

Clearly, $I \models \mathcal{P}^{I,\mathcal{O}'}_{weak}$; it is left to show that I is a minimal model of $\mathcal{P}^{I,\mathcal{O}'}_{weak}$. Assume to the contrary, that some $I' \subset I$ exists, such that $I \models \mathcal{P}^{I,\mathcal{O}'}_{weak}$. Observe that I' must coincide with I on $x_1, \ldots, x_n, \bar{x}_1, \ldots \bar{x}_n$ and must contain $w(b)$, as the latter is a fact of $\mathcal{P}$. In addition, at least one of y_j, $\bar{y}_j$ is in I' due to the rules of the form (2). If $I \backslash I'$ contains atoms over y_j or $\bar{y}_j$, then some rule of the form (4) or (5) is not satisfied by I'. Thus I' coincides with I, and we obtain that I is indeed minimal.

($\Leftarrow$) Suppose that I is a *weak*-repair answer set of Π, i.e. it is a weak answer set of $\Pi' = \langle \emptyset, \mathcal{A}', \mathcal{P} \rangle$ for some $\mathcal{A}'$. Let the assignment $A(\phi)$ to the atoms $x_1, \ldots, x_n$ be defined by

$$A(x_i) = \begin{cases} true, & \text{if } x_i(b) \in I \\ false, & \text{if } \bar{x}_i(b) \in I \end{cases}$$

The assignment $A(\phi)$ is well defined, as I can by minimality contain exactly one atom out of $x_i(b)$ and $\bar{x}_i(b)$ for every $1 \leq i \leq n$. The interpretation I must contain $w(b)$ because of the rule (3), and therefore all $y_j(b)$ and $\bar{y}_j(b)$ because of the rules (4) and (5). As I is a repair answer set, it holds that for each $I' \subset I$, that contains exactly one of $y_j(b)$, $\bar{y}_j(b)$, and coincides with I' on $x_i, \bar{x}_i$, there must be some k, such that $I' \models^{\mathcal{O}'} f(l_{k_h})$, $1 \leq h \leq 3$. Note that $I' \models^{\mathcal{O}'} f(l_{k_i})$ holds if either of the following is true: (i) the DL-query of the DL-atom $f(l_{k_i})$ is explicitly present in $\mathcal{A}'$; (ii) the update $\lambda^{I'}(f(l_{k_h})) \cup \mathcal{O}'$ is inconsistent or (iii) the DL-query of $f(l_{k_i})$ occurs in the update. The case (i) could not hold due to the constraints (9)-(12). The case (ii) is not possible either, since the TBox is empty and inconsistency can be obtained only if the negation of the DL-query is explicitly present in the ABox, which is again forbidden by (9)-(12). This means that all $f(k_{k_h})$ are true under I because of the right choice of atoms $x_i(b)$, present in the interpretation I. Thus ϕ evaluates to true. □

4.1.2 Extending Complexity Results for DL-programs over $\mathcal{EL}$ DL

In fact, for DL-programs over $\mathcal{EL}$ ontologies the complexity results from above should be inherited too. Intuitively, the main reason for that is the tractability of query answering in $\mathcal{EL}$. The result of Theorem 4.5 extends to $\mathcal{EL}$, and the hardness proof can be easily adjusted (one can replace $X_i \sqsubseteq \neg \bar{X}_i$ by $X_i \sqcap \bar{X}_i \sqsubseteq A$ and $\neg C_j$ by A in the construction exploited in the proof of Theorem 4.5). We now establish the NP-hardness result for repair answer set existence for normal DL-programs over $\mathcal{EL}$ ontologies under the additional restriction that all of the DL-atoms occurring in the program have empty updates.

Theorem 4.9. *Given a ground DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, deciding the repair answer set existence of Π, (i.e. $RAS_x(\Pi) \neq \emptyset$) is NP-complete for normal Π and $x = weak$, if the ontology $\mathcal{O}$ is in $\mathcal{EL}$ DL; hardness holds even if all DL-atoms of Π have empty input signature, i.e. they are of the form* DL[; Q]($\vec{t}$).

Proof. (Membership) Since instance query entailment checking is in $\mathcal{EL}$ feasible in polynomial time, for evaluation of DL-atoms we obtain tractability results similarly as in Proposition 4.4. The algorithm for repair answer set computation first guesses an interpretation candidate I and values of the DL-atoms and then checks whether the real values of the DL-atoms coincide with the guessed values, and whether the interpretation I is a minimal model of the weak reduct of the repaired program. As both of checks are feasible in polynomial time, membership in NP is obtained.

(Hardness) To prove hardness we provide a reduction from the NP-complete 3SAT problem to deciding weak repair answer set existence for a DL-program over the $\mathcal{EL}$ ontology as follows.

Let $\phi = \bigwedge_{1 \leq j \leq m} C_j$ be an instance of the 3SAT problem, where each C_j is a disjunction of three literals over the variables $x_1, \ldots, x_n$. From this we construct a DL-program $\Pi = \langle \mathcal{T}, \mathcal{A}, \mathcal{P} \rangle$.

- Construction of the TBox $\mathcal{T}$:

  We introduce a concept name X_i (resp. $\bar{X}_i$) for each variable x_i occurring in ϕ. Moreover, for each $1 \leq i \leq n$ we introduce a fresh concept name B_i. Then the TBox $\mathcal{T}$ is constructed as follows:

  - $X_i \sqcap \bar{X}_i \sqsubseteq B_i$, for all $1 \leq i \leq n$;

- The ABox is $\mathcal{A} = \emptyset$;
- Construction of $\mathcal{P}$:

  We introduce fresh ground atoms $p_j(b)$ for each clause C_j of ϕ. The rules of $\mathcal{P}$ include the following:

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ \bot \leftarrow \mathit{not}\ \ \mathrm{DL}[;\, X_1](b), \mathit{not}\ \ \mathrm{DL}[;\, \bar{X}_1](b); \\ \quad \ldots \\ (2)\ \bot \leftarrow \mathit{not}\ \ \mathrm{DL}[;\, X_n](b), \mathit{not}\ \ \mathrm{DL}[;\, \bar{X}_n](b); \\ (3)\ \bot \leftarrow \mathrm{DL}[;\, B_1](b); \\ \quad \ldots \\ (4)\ \bot \leftarrow \mathrm{DL}[;\, B_n](b); \\ (5)\ p_1(b) \leftarrow \mathrm{DL}[;\, X_i](b)\ \mid \text{if } x_i \text{ is a disjunct in } C_1; \\ (6)\ p_1(b) \leftarrow \mathrm{DL}[;\, \bar{X}_i](b)\ \mid \text{if } \neg x_i \text{ is a disjunct in } C_1; \\ \quad \ldots \\ (7)\ p_m(b) \leftarrow \mathrm{DL}[;\, X_i](b)\ \mid \text{if } x_i \text{ is a disjunct in } C_m; \\ (8)\ p_m(b) \leftarrow \mathrm{DL}[;\, \bar{X}_i](b)\ \mid \text{if } \neg x_i \text{ is a disjunct in } C_m; \\ (9)\ \bot \leftarrow \mathit{not}\ p_1(b); \\ \quad \ldots \\ (10)\ \bot \leftarrow \mathit{not}\ p_m(b) \end{array} \right\},$$

We claim that ϕ is satisfiable iff $RAS_{weak}(\Pi) \neq \emptyset$, i.e. there exists I and $\mathcal{A}'$ such that I is a weak answer set of $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$.

($\Rightarrow$) Suppose that ϕ is satisfiable and $A(\phi)$ is its satisfying assignment. From this we construct a repair answer set I of Π and a repair ABox $\mathcal{A}'$ as follows. An interpretation I contains the set of atoms $p_1(b), \ldots, p_m(b)$. An ABox $\mathcal{A}'$ is such that $X_i(b)$ (resp. $\bar{X}_i(b)$) is in $\mathcal{A}'$ iff x_i is true (resp. false) under the assignment $A(\phi)$.

Now we show that I is a weak repair answer set of Π with the repair $\mathcal{A}'$. Observe that either $\mathrm{DL}[;\, X_i](b)$ or $\mathrm{DL}[;\, \bar{X}_i](b)$ evaluates to true for all i under $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ as

$A(\phi)$ is a full assignment, i.e. each x_i is set to either true or false in A. Furthermore, $A(\phi)$ represents a consistent set of literals, therefore not both $X_i(b)$ and $\bar{X}_i(b)$ can be present in $\mathcal{A}'$, meaning that the bodies of the constraints (3) and (4) are not satisfied. For each $p_j(b)$ occurring in the set of rules (5)-(8), there is at least one rule with a DL-atom whose value is true under $\mathcal{O}'$, as every clause of ϕ is satisfied by A. As the reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$ contains only facts, occurring in I, the result immediately follows.

($\Leftarrow$) Let I be a weak repair answer set of Π and $\mathcal{A}'$ be a respective repair. Note that the only possible repair answer set of Π is the one that contains $p_j(b)$ for all $1 \leq j \leq m$, since otherwise some of the constraints (9)-(10) are violated. By the constraints (1)-(2), the repair must contain either $X_i(b)$ or $\bar{X}_i(b)$; the constraints (3)-(4) forbid both $X_i(b)$ and $\bar{X_i}(b)$ to be in $\mathcal{A}$. Finally, as the bodies of the rules (5)-(8) must be satisfied, the repair encodes a satisfying assignment of ϕ. □

Other complexity results presented for *DL-Lite*$_\mathcal{A}$ earlier in this section also carry over for the $\mathcal{EL}$ DL.

4.2 Selection Functions

Clearly, not all repairs are equally useful or interesting for a certain scenario. For instance, repairs that have no common assertions with the original ABox might be unwanted; repairs that introduce assertions that are not in the initial ABox; repairs that would cause non-minimal change etc. Formally, we model preferred repairs using a *selection* function:

Definition 4.10. A *selection function* is a function $\sigma : 2^{\mathcal{AB}} \times \mathcal{AB} \rightarrow 2^{\mathcal{AB}}$, where $\mathcal{AB}$ is the set of all ABoxes, that given a set S of ABoxes and an ABox $\mathcal{A}$, returns a set $\sigma(S, \mathcal{A}) \subseteq S$ of *preferred* (or *selected*) ABoxes.

This notion captures a variety of selection principles, including minimal repairs according to some preference relation, or some global selection property. We then define:

Definition 4.11. Given $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, and a selection σ, we call $rep_{(\sigma,x)}(\Pi) = \sigma(rep_x(\Pi), \mathcal{A})$ the (σ, x)-*repairs* of Π. An interpretation $I \subseteq HB_\Pi$ is a (σ, x)-*repair answer set* of Π, if $rep^I_{(\sigma,x)}(\Pi) \neq \emptyset$, where $rep^I_{(\sigma,x)}(\Pi) = rep_{(\sigma,x)}(\Pi) \cap rep^I_x(\Pi)$; by $RAS_{(\sigma,x)}(\Pi)$ we denote the set of all such repair answer sets.

Example 4.12. Consider a DL-program $\Pi = \langle \emptyset, \mathcal{A}, \mathcal{P} \rangle$, where $\mathcal{A} = \{Child(john)\}$ and $\mathcal{P}$ is as follows:

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ male(john); \\ (2)\ pupil(john) \leftarrow \mathrm{DL}[;\ studiesAt](john, sch80); \\ (3)\ boy(john) \leftarrow \mathrm{DL}[Child \uplus boy;\ Child](john), male(john); \\ (4)\ \bot \leftarrow boy(john), not\ pupil(john) \end{array} \right\}$$

The interpretation $I = \{male(john), pupil(john), boy(john)\}$ is a (σ,*weak*)-repair answer set of Π with a possible (σ,*weak*)-repair $\mathcal{A}' = \{studiesAt(john, sch80)\}$, i.e.

$I \in RAS_{\sigma,weak}(\Pi)$ and $\mathcal{A}' \in rep^I_{\sigma,weak}(\Pi)$, where σ chooses repairs $\mathcal{A}'$, such that the set difference between $\mathcal{A}$ and $\mathcal{A}'$ contains at most 2 assertions. Indeed, we have that $\mathcal{P}^{I,\mathcal{O}'}_{weak} = \{male(john);\ pupil(john);\ boy(john)\}$, and clearly I is its minimal model.

Moreover, $I \in RAS_{\sigma,\mathit{flp}}(\Pi)$, and $\mathcal{A}'' \in rep^I_{\sigma,\mathit{flp}}(\Pi)$ is its (σ,*flp*)-repair, where $\mathcal{A}'' = \{Child(john), studiesAt(john, sch80)\}$. To verify this, observe that the reduct $\mathcal{P}^{I,\mathcal{O}''}_{\mathit{flp}}$ contains the fact (1) and the rule (4), and I is a minimal model of $\langle \mathcal{A}'', \mathcal{P}^{I,\mathcal{O}''}_{\mathit{flp}} \rangle$, where $\mathcal{O}'' = \langle \emptyset, \mathcal{A}'' \rangle$.

Note that while $\mathcal{A}'' \in rep^I_{\sigma,weak}(\Pi)$, we have that $\mathcal{A}' \notin rep^I_{\sigma,\mathit{flp}}(\Pi)$. More specifically, I is not a minimal model of $\langle \mathcal{P}^{I,\mathcal{O}'}_{\mathit{flp}}, \mathcal{O}' \rangle$, where $\mathcal{P}^{I,\mathcal{O}'}_{\mathit{flp}} = \mathcal{P}^{I,\mathcal{O}''}_{\mathit{flp}}$ and $\mathcal{O}' = \langle \emptyset, \mathcal{A}' \rangle$, since there is a smaller model $I' = I \setminus \{boy(john)\}$, which satisfies all rules of $\mathcal{P}^{I,\mathcal{O}'}_{\mathit{flp}}$.

The repair $\mathcal{A}'_1 = \{Male(john), Male(pat)\}$ from Example 4.3 is in $rep^{I_1}_{\sigma_1,x}(\Pi)$ for $I_1 = \{boy(john), ischildof(john, alex)\}$, where $x \in \{weak, \mathit{flp}\}$ and σ_1 selects deletion repairs, i.e. subsets of $\mathcal{A}$. The ABox $\mathcal{A}_2 = \{hasParent(john, pat), Male(john), Female(pat)\}$ is also in $rep^{I_1}_{\sigma_2,\mathit{flp}}(\Pi)$, where $x \in \{weak, \mathit{flp}\}$, and σ_2 selects repairs $\mathcal{A}'$, which differ from $\mathcal{A}$ only on assertions over gender predicates *Male*, *Female*, and $|\mathcal{A}| = |\mathcal{A}'|$. Consequently, $I_1 \in RAS_{\sigma_1,x}(\Pi)$ and $I_2 \in RAS_{\sigma_2,x}(\Pi)$ for $x \in \{weak, \mathit{flp}\}$. □

In general, even polynomially computable selections σ may incur intractability, like e.g. selecting ABoxes $\mathcal{A}'$ with set-minimal change to $\mathcal{A}$, or with smallest Dalal (Hamming) distance. Naturally, we aim at selections that are useful in practice and have benign computational properties, which are pragmatic specifically for our problem.

4.2.1 Independence Property

Definition 4.13. A selection $\sigma : 2^{\mathcal{AB}} \times \mathcal{AB} \to 2^{\mathcal{AB}}$ is *independent*, if $\sigma(S, \mathcal{A}) = \sigma(S', \mathcal{A}) \cup \sigma(S \setminus S', \mathcal{A})$ whenever $S' \subseteq S$.

Example 4.14. All selection functions considered in Example 4.12 are independent. The selection function σ, which seeks for repairs $\mathcal{A}'$ that contain minimal number of changes in assertions over *Adopted* predicate w.r.t. $\mathcal{A}$ is not independent, since to find the preferred σ-repair one needs to compute all repair candidates first, and then choose the best one among them. □

Independence allows us to decide whether a given repair $\mathcal{A}' \in S$ is selected by σ without looking at other repairs, and composition works here easily. This makes the introduced property valuable, since independent selection functions of different kind can be conveniently combined without a major increase in the complexity. Formally,

Proposition 4.15. *If σ_1 and σ_2 are independent selection functions then their composition $\sigma_1 \circ \sigma_2$ is also independent.*

Proof. We show that whenever $S' \subseteq S$ it holds that $\sigma_1(\sigma_2(S, \mathcal{A}), \mathcal{A}) = \sigma_1(\sigma_2(S', \mathcal{A}), \mathcal{A}) \cup \sigma_1(\sigma_2(S \setminus S', \mathcal{A}), \mathcal{A})$. By independence of σ_2 we have $\sigma_2(S, \mathcal{A}) = \sigma_2(S', \mathcal{A}) \cup \sigma_2(S \setminus S', \mathcal{A})$. Hence, $\sigma_2(S', \mathcal{A}) \subseteq \sigma_2(S, \mathcal{A})$, and thus by independence of σ_1 we get $\sigma_1(\sigma_2(S, A), A) =$

$\sigma_1(\sigma_2(S', \mathcal{A}), \mathcal{A}) \cup \sigma_1(\sigma_2(S, \mathcal{A}) \setminus \sigma_2(S', \mathcal{A}), \mathcal{A})$. As $\sigma_2(S, \mathcal{A}) \setminus \sigma_2(S', \mathcal{A}) = \sigma_2(S \setminus S', \mathcal{A})$, the result follows. □

Clearly, set-minimal change and smallest Dalal distance are not independent, as to decide whether $\mathcal{A}' \in \sigma(S, \mathcal{A})$ one has to compare $\mathcal{A}'$ with all other ABoxes from S. On the other hand, selecting all ABoxes such that $\mathcal{A}' \subseteq \mathcal{A}$, is obviously independent. The latter, and several other independent selections that are useful in practice, will be considered in the next section.

Independence leads to the following beneficial property.

Proposition 4.16. *For every* Π *and selection* σ*, if* σ *is independent, then* $rep^I_{(\sigma,x)}(\Pi) \subseteq rep_{(\sigma,x)}(\Pi)$*, for every* $I \subseteq HB_\Pi$*.*

Proof. By definition $rep_{(\sigma,x)}(\Pi) = \sigma(rep_x(\Pi), \mathcal{A})$ and $rep^I_{(\sigma,x)}(\Pi) = \sigma(rep^I_x(\Pi), \mathcal{A})$. Now as $rep^I_x(\Pi) \subseteq rep_x(\Pi)$ and σ is independent, we obtain $\sigma(rep_x(\Pi), \mathcal{A}) = \sigma(rep^I_x(\Pi), \mathcal{A}) \cup \sigma(rep_x(\Pi) \setminus rep^I_x(\Pi), \mathcal{A})$, from which the result follows. □

Proposition 4.16 implies that if we can turn an interpretation I into an answer set of Π by a σ-selected repair from the repairs which achieve this for I, then I is a σ-repair answer set of Π; that is, *local selection* is enough for a *global* σ-repair answer set. This will be exploited in Section 4.4.

4.3 Ontology Repair Problem

In this section we introduce the Ontology Repair Problem (ORP), which is an important subtask of repair answer set computation. We present here the general complexity results for ORP.

Intuitively, an ORP is a problem of identifying an ABox under which a simultaneous entailment and non-entailment of sets of queries under possible updates is guaranteed. Let us now provide a formal definition for this repair problem.

Definition 4.17. An *ontology repair problem (ORP)* is a triple $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$ where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an ontology and $D_i = \{\langle U^i_j, Q^i_j \rangle \mid 1 \leq j \leq m_i\}$, $i = 1, 2$, are sets of pairs where each U^i_j is an ABox and each Q^i_j is a DL-query. A *repair* (*solution*) for $\mathcal{ORP}$ is any ABox $\mathcal{A}'$ such that

(i) the ontology $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ is consistent;

(ii) $\langle \mathcal{T}, \mathcal{A}' \rangle \cup U^1_j \models Q^1_j$ holds for $1 \leq j \leq m_1$;

(iii) $\langle \mathcal{T}, \mathcal{A}' \rangle \cup U^2_j \not\models Q^2_j$ holds for $1 \leq j \leq m_2$.

For the illustration of the ORP problem, we refer to the ontology from Example 4.1.

Example 4.18. Consider an $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$, where $\mathcal{O}$ is as in Example 4.1, and the sets D_1 and D_2 are as follows:

- $D_1 = \{\langle U_1^1, Q_1^1\rangle, \langle U_2^1, Q_2^1\rangle, \langle U_3^1, Q_3^1\rangle\}$, where
  - $U_1^1 = \{Male(john)\}$, $Q_1^1 = Male(pat)$;
  - $U_2^1 = \emptyset$, $Q_2^1 = hasParent(john, pat)$;
  - $U_3^1 = \{Child(john)\}$, $Q_2^1 = Male(alex)$;
- $D_2 = \{\langle U_1^2, Q_1^2\rangle\}$, where
  - $U_1^2 = \emptyset$, $Q_1^2 = Adopted(john)$.

One of the possible solutions to the described ORP is the ABox $\mathcal{A}' = \{Male(alex), hasParent(john, pat), Male(pat)\}$. Indeed, it easy to verify that

- $\langle \mathcal{T}, \mathcal{A}'\rangle \cup \{Male(john)\} \models Male(pat)$, $\langle \mathcal{T}, \mathcal{A}'\rangle \models hasParent(john, pat)$, $\langle \mathcal{T}, \mathcal{A}'\rangle \cup Child(john) \models Male(alex)$;
- $\langle \mathcal{T}, \mathcal{A}'\rangle \not\models Adopted(john)$. □

We now analyze the complexity of the ORP problem in the general setting.

4.3.1 Complexity Results

Non-surprisingly, the Ontology Repair Problem is intractable in general; however, this holds already for very simple ontologies, which we show in the next proposition.

Proposition 4.19. *Deciding whether an Ontology Repair Problem $\mathcal{ORP} = \langle \mathcal{T}, \mathcal{A}, D_1, D_2\rangle$ has some repair $\mathcal{A}'$ is* NP*-complete, and* NP*-hard even if $\mathcal{T}$ contains only positive concept inclusions and $\mathcal{A} = \emptyset$.*

Proof. A guess for $\mathcal{A}'$ is verifiable in polynomial time, as deciding all $\langle \mathcal{T}, \mathcal{A}' \cup U_i\rangle \models Q_i$ is polynomial in *DL-Lite*$_\mathcal{A}$ (see Table 2.7). NP-hardness is shown by a reduction from SAT instances $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ over atoms $x_1, \ldots, x_n$. We construct the ORP problem $\mathcal{ORP} = \langle\langle \mathcal{T}, \emptyset\rangle, D_1, D_2\rangle$, using concepts $X_i, \bar{X}_i$ for the x_i, C_j for the χ_j, and a fresh concept A as follows:

- $\mathcal{T} = \{X_j \sqsubseteq C_i, \bar{X}_{j'} \sqsubseteq C_i \mid 1 \leq i \leq m, x_j \in \chi_i, \neg x_{j'} \in \chi_i\}$
- $D_1 = \{\langle \emptyset, C_i(b)\rangle, \langle U_i, \neg C_i(b)\rangle, \langle V_j, A(b)\rangle \mid 1 \leq i \leq m,\ 1 \leq j \leq n\}$,

  where $U_i = \{\bar{X}_k(b), X_{k'}(b) \mid x_k \in \chi_i, \neg x_{k'} \in \chi_i\}$,

  $V_j = \{\neg X_j(b), \neg \bar{X}_j(b)\}$,
- $D_2 = \{\langle \emptyset, \neg C_i(b)\rangle \mid 1 \leq i \leq m\} \cup \{\langle \emptyset, A(b)\rangle\}$.

Intuitively, by D_2 a repair $\mathcal{A}'$ must not contain $\neg C_i(b)$ nor $A(b)$, and must be consistent. By D_1 the repair must entail $C_i(b)$. Therefore, for each i, the ABox $\mathcal{A}'$ must contain some X_k (resp. $\bar{X}_k$), such that $X_k \sqsubseteq C_i$ (resp. $\bar{X}_k \sqsubseteq C_i$). Moreover, adding either U_i or V_j to $\mathcal{A}'$ causes inconsistency. The former implies that $\mathcal{A}'$ contains some $\neg\bar{X}_k(b)$ (resp. $\neg X_k(b)$) such that $x_k \in \chi_k$ ($\neg x_k \in \chi_k$), and the latter implies that at least one of $\neg X_k(b)$ and $\neg\bar{X}_k(b)$ must be in $\mathcal{A}'$ for all $1 \leq k \leq n$. Since in addition the ABox is consistent as argued above, it can not contain both $\neg\bar{X}_k$ and $\neg X_k$ thus $\mathcal{A}'$ represents a consistent choice of literals that satisfies ϕ.

We formally show that ϕ is satisfiable iff $\mathcal{ORP}$ has a repair.

($\Rightarrow$) Let A be a satisfying assignment for ϕ. We construct a repair ABox $\mathcal{A}'$ for $\mathcal{ORP}$ as follows: if a variable x_k is set to true in the satisfying assignment of ϕ, then we add $X_k(b)$ and $\neg\bar{X}_k(b)$ to the ABox $\mathcal{A}'$, otherwise, i.e. if x_k is set to false in A, we add $\bar{X}_k(b)$ and $\neg X_k(b)$ to $\mathcal{A}'$. We now verify whether the constructed ABox is indeed a repair for $\mathcal{ORP}$ by checking whether it satisfies the conditions (i) to (iii) of Definition 4.17.

(i) $\mathcal{T} \cup \mathcal{A}'$ is consistent, since A is a consistent set of literals not both $X_k(b)$ and $\neg X_k(b)$ (resp. $\bar{X}_k(b), \neg\bar{X}_k(b)$) can be present in $\mathcal{A}'$).

(ii) We check whether for all $\langle U_i^1, Q_i^1\rangle \in D_1$ it holds that $\mathcal{T} \cup \mathcal{A}' \cup U_i^1 \models Q_i^1$. Let us first consider $\langle \emptyset, C_i\rangle$, $1 \leq i \leq m$. Observe that A is a satisfying assignment of ϕ, therefore each clause of ϕ is satisfied under A. Thus, for each clause either there exists a variable x_j occurring as a disjunct in the clause C_i positively and being set to *true* in the satisfying assignment A or occurring negatively as a disjunct in C_i and being set to false in A. By construction of $\mathcal{A}'$, we have that $\mathcal{T} \cup \mathcal{A}' \models C_i(b)$ for all $1 \leq i \leq m$ due to the inclusion $X_j \sqsubseteq C_i$ (resp. $\bar{X}_j \sqsubseteq C_i$). Similarly, we have that for all U_i, $\mathcal{A}' \cup U_i$ is inconsistent and, therefore trivially entails $\neg C_i(b)$. Finally, since the assignment A is full, each x_i has a truth value. Hence, due to the form of updates V_j, we have that $\mathcal{A}' \cup V_j$ is inconsistent for all j, and thus the queries $A(b)$ are also entailed.

(iii) It is left to show that for all $\langle U_i^2, Q_i^2\rangle \in D_2$ we have that $\mathcal{T} \cup \mathcal{A}' \cup U_i^2 \not\models Q_i^2$. The latter holds since the ontology $\langle \mathcal{T}, \mathcal{A}'\rangle$ is consistent, and there is no way to derive either $\neg C(b)$ or $A(b)$ by means of the TBox axioms and the facts in $\mathcal{A}'$.

The above shows that the ABox $\mathcal{A}'$ is indeed a solution to the $\mathcal{ORP}$.

($\Leftarrow$) Now assume that there exists an ABox $\mathcal{A}'$ that is a solution to $\mathcal{ORP}$. We show that then the formula ϕ is satisfiable. First since $\mathcal{T} \cup \mathcal{A}' \cup V_j \models A(b)$ and $\mathcal{T} \cup \mathcal{A}' \not\models A(b)$, it must be the case that $\mathcal{T} \cup \mathcal{A}' \cup V_j$ is inconsistent. Moreover, due to $\mathcal{T} \cup \mathcal{A}' \not\models \neg C_i(b)$, we know that the inconsistency must occur due to the facts $X_j(b)$, $\bar{X}_j(b)$. Therefore, for each j either $\neg X_j(b)$ or $\neg\bar{X}_j(b)$ must be in $\mathcal{A}$. Observe now, that due to $\langle \emptyset, C_i(b)\rangle, \langle U_i, \neg C_i(b)\rangle \in D_1$, the ABox $\mathcal{A}'$ must contain such $X_j(b)$ (resp. $\bar{X}_j(b)$), that x_j (resp. $\neg x_j$) is a disjunct in the clause χ_i. Moreover, due to $\langle U_i, \neg C_i(b)\rangle$ for some k such that $x_k \in \chi_i$ (or $\neg x_k \in \chi_i$), it must hold that $\neg\bar{X}_k(b)$ (resp. $\neg X_k(b)$) is in $\mathcal{A}'$. The above argument shows that the ABox $\mathcal{A}'$ encodes a satisfying assignment A for ϕ: if $X_i(b) \in \mathcal{A}'$, then x_i is true in A; if $\bar{X}_i(b) \in \mathcal{A}'$, then x_i is false in A. □

In fact, even if both TBox and ABox are empty, then the problem still stays intractable, which is formally proved in the following proposition.

Theorem 4.20. *Deciding if an Ontology Repair Problem $\mathcal{ORP}=\langle\mathcal{O}, D_1, D_2\rangle$ has some repair is* NP*-hard even if $\mathcal{T}=\mathcal{A}=\emptyset$.*

Proof. NP-hardness of $\mathcal{ORP}$ holds by a reduction from SAT. Given $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ on atoms $x_1, \ldots, x_n$, we construct $\mathcal{ORP} = \langle\langle\emptyset, \emptyset\rangle, D_1, D_2\rangle$, with concepts $X_j, \bar{X}_j$ for the x_j and a fresh concept A, such that

- $D_1 = \{\langle U_i, A(b)\rangle, \langle V_j, A(b)\rangle \mid 1 \leq i \leq m\}$, where
  - $U_i = \{\bar{X}_j(b),\ X_{j'}(b) \mid x_j \in \chi_i, \neg x_{j'} \in \chi_i\}$,
  - $V_j = \{\neg X_j(b), \neg \bar{X}_j(b)\}$,
- $D_2 = \{\langle\emptyset, A(b)\rangle\}$.

Intuitively, by D_2 a repair $\mathcal{A}'$ must not contain $A(b)$, and by D_1 adding either (i) U_i or (ii) V_j to $\mathcal{A}'$ causes inconsistency. By (i) $\mathcal{A}'$ must contain at least one $\neg\bar{X}_j(b)$ (resp. $\neg X_j(b)$) such that $x_j \in \chi_i$ ($\neg x_j \in \chi_i$), and by (ii) at least one of $X_j(b), \bar{X}_j(b)$ must be in $\mathcal{A}$. Furthermore, D_2 forbids both $X_j(b), \neg X_j(b)$ (resp. $\bar{X}_j(b), \neg\bar{X}_j(b)$) to be in $\mathcal{A}'$. Thus $\mathcal{A}'$ encodes a consistent choice of literals that satisfies ϕ. □

We note that ORP has two sources of NP-hardness, viz. the data part (as in the proof above), and the taxonomy, which under σ-repairs may derive further assertions. Furthermore, each ORP can be encountered in some DL-program setting; we show this on an example.

Example 4.21. Consider the Ontology Repair Problem $\mathcal{ORP} = \langle\mathcal{O}, D_1, D_2\rangle$, where $D_1 = \{\delta_1\}$, $D_2 = \{\delta_2\}$, such that $\delta_1 = \langle\{C(c), \neg D(c)\}, \neg E(c)\rangle$, and $\delta_2 = \langle\{D(d), \neg S(d)\}, C(d)\rangle$. We introduce predicates $p_C^{\delta_1}, p_D^{\delta_1}$ for δ_1 and $p_D^{\delta_2}, p_S^{\delta_2}$ for δ_2 and construct $\Pi = \langle\mathcal{O}, \mathcal{P}_I \cup \mathcal{P}_{DL}\rangle$, where

$$\mathcal{P}_I = \{p_C^{\delta_1}(c);\ p_D^{\delta_1}(c);\ p_D^{\delta_2}(d);\ p_S^{\delta_2}(d)\},$$

$$\mathcal{P}_{DL} = \left\{ \begin{array}{ll} \bot \leftarrow not\ \underbrace{\mathrm{DL}[C \uplus p_C^{\delta_1}, D \uplus p_D^{\delta_1}; \neg E](c)}_{a_1}; & (1) \\ \bot \leftarrow \underbrace{\mathrm{DL}[D \uplus p_D^{\delta_2},\ S \uplus p_S^{\delta_2}; C](d)}_{a_2}\} & (2) \end{array} \right\}.$$

Then, the program Π has a single repair answer set candidate, in which the atom a_1 has to evaluate to true and a_2 to false respectively. This gives rise to $\mathcal{ORP}$; the rule (1) effects the pair δ_1 in D_1 and the rule (2) the pair δ_2 in D_2. □

Generalizing the above example, for each $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$ one can construct a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, such that solutions of $\mathcal{ORP}$ correspond to repairs of Π. The construction of $\mathcal{P}$ proceeds as follows. A DL-atom a_i^j is created for every pair $\langle U_i^j, Q_i^j \rangle \in D_j$, such that the DL-query of a_i^j is Q_i^j, and the input signature λ_i^j encodes the update U_i^j: for every $C(\vec{t}) \in U_i^j$ (resp. $\neg C(\vec{t})$) the signature λ_i^j contains a triple $C \uplus p_C$ (resp. $C \uplus p_C^{i,j}$). Furthermore, for each such triple the fact $p_C^{i,j}(\vec{t})$ is added to $\mathcal{P}$.

The rules of $\mathcal{P}$ ensure that all DL-atoms a_i^j are true for $j = 1$ and false for $j = 2$. That is, the logic program part $\mathcal{P}$ of Π contains

- a constraint $\bot \leftarrow \mathit{not}\ a_{i_1}^1$, for every $a_{i_1}^1$, and
- a constraint $\bot \leftarrow a_{i_2}^2$, for every $a_{i_2}^2$.

As there are no predicates in $\mathcal{P}$ apart from those occurring in facts, the only possible repair answer set I of Π contains all facts of $\mathcal{P}$. Therefore, the update $\lambda^I(a_i^j)$ of every a_i^j corresponds exactly to U_i^j, and the constraints of $\mathcal{P}$ guarantee the simultaneous entailment and non-entailment of sets of queries under possible temporary updates encoded by the given $\mathcal{ORP}$.

4.4 Tractable ORP Cases

The ontology considered in the proof of Theorem 4.20 is empty and still we obtain intractable results. In what follows we aim at finding tractable cases for the ORP problem given that $\mathcal{O}$ is in *DL-Lite*$_\mathcal{A}$ DL.

If there are few ground DL-atoms in the program then the ORP becomes tractable. However, in real settings the ground program is normally obtained from a nonground one, and it is often large with many DL-atoms. Therefore, the pairs D_1 and D_2 are hard to control in practice, and to gain tractability for ORP, we consider restrictions on repairs and the ontology. We present four tractable cases of σ-repairs with independent selection function σ, which are arguably useful in practice. In what follows, let $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$.

4.4.1 Bounded $\delta^\pm$-change

A natural restriction that one could exploit is to bound the distance from the original ABox, i.e.

$$\sigma_{\delta^\pm,k}(S, \mathcal{A}) = \{\mathcal{A}' \mid |\mathcal{A}' \triangle \mathcal{A}| \leq k\}, \text{[1]} \tag{4.2}$$

.

The formal complexity results for this setting are stated in the following proposition.

Proposition 4.22. *Deciding if an Ontology Repair Problem $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$ has a $\delta^\pm, k$-change repair is polynomial.*

[1] $\mathcal{A}' \triangle \mathcal{A} = (\mathcal{A}' \cap \mathcal{A}) \cup (\mathcal{A} \cap \mathcal{A}')$

Proof. As the number m of possible ABox assertions is polynomial in the size of $\mathcal{T}$ and $\mathcal{A}$, traversing all $O(\binom{m}{k})$ possible $\mathcal{A}'$ and checking the repair condition can be done in polynomial time. □

We illustrate this type of a repair by the following example.

Example 4.23. For the DL-program from Example 4.1 one of $\delta^{\pm}, k$-change repairs for $k = 2$ is $\mathcal{A}' = \mathcal{A}\backslash\{Male(pat)\} \cup \{Female(pat)\}$. Another possible repair is $\mathcal{A}' = \mathcal{A}\backslash\{Male(pat)\} \cup \{Male(mat)\}$ provided that mat is a constant from the ontology signature. □

The $\delta^{\pm}$-change repairs are arguably useful in practice. The repairs that restore consistency by getting rid of such deficiencies as typos and syntactical inaccuracies fall into this repair category. For instance, in Example 4.23 the fact $Male(pat)$ was in the ontology instead of $Male(mat)$, as the letters p and m were confused at the data engineering process. In such scenarios one can search for repairs by applying selective changes to certain ontology assertions. These selective changes include modifications of the predicate or constants occurring in the assertion, i.e. $P(\vec{t})$ could be changed to $P(\vec{t'})$ or $P'(\vec{t})$.

To ensure tractability, the number of constants or predicates with which the initial facts can be modified is bounded by k. Under this restriction, an ABox $\mathcal{A}$ with at most m assertions allowed for modification has $((k+1)^2-1)^m$ repair candidates. Since both k and m are bounded, deciding whether the $\delta^{\pm}$-solution for ORP exists is polynomial. The alternatives (i.e. constants and predicates) used for fixing initial facts can be created by partitioning the elements of the ontology signature into subsets based on their syntactical similarity. For example, the constants mat and pat differ just in a single letter, therefore they will most likely be determined to the same partition. With such partitions at hand for a certain fact there are just few possibilities for changing it. For creating the mentioned partitions one could explore well-known metrics for measuring distances between strings like Hamming or Levenshtein distance [Lev66].

Switching positions of constants in role assertions is another special setting with obvious practical applications.

Example 4.24. For instance, $\mathcal{A}' = \mathcal{A}\backslash\{hasParent(john, pat)\} \cup \{hasParent(pat, john)\}$ would be a plausible repair for the DL-program Π. □

4.4.2 Deletion repair

Another important restriction is to allow only to delete assertions from the original ABox i.e., use

$$\sigma_{del}(S, \mathcal{A}) = \{\mathcal{A}' \mid \mathcal{A}' \subseteq \mathcal{A}\}. \tag{4.3}$$

Example 4.25. In Example 4.1, each $\mathcal{A}' \subset \mathcal{A}$ except $\{Male(pat), hasParent(john, pat)\}$ is a deletion repair. □

To achieve tractability, we exclude non-containment ($\not\sqsubseteq$) DL-queries, i.e., of the form $\neg Q$ where Q is an inclusion or disjointness axiom, from $\mathcal{P}$; let us call such ORPs $\not\sqsubseteq$-*free*. We now make a reasonable and necessary assumption that the original ontology is consistent and for the deletion repair obtain the following results.

Theorem 4.26. *Deciding whether a $\not\sqsubseteq$-free Ontology Repair Problem $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$ with consistent $\mathcal{O}$ has a σ_{del}-repair is polynomial.*

Proof. The proof exploits the following property of $\not\sqsubseteq$ DL-queries.

Lemma 4.27. *If $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent, then $\langle \mathcal{T}, \mathcal{A} \rangle \cup U_j^i \models Q_j^i$ iff $\langle \mathcal{T}, \mathcal{A}_0 \rangle \cup U_j^i \models Q_j^i$ for some $\mathcal{A}_0 \subseteq \mathcal{A}$ with $|\mathcal{A}_0| \leq 1$.*

That is, at most one assertion α from $\mathcal{A}$ is sufficient to derive the query. This follows from a respective result for empty U_j^i and instance queries Q_j^i (see Proposition 2.7).

Now if $\mathcal{T} \cup U_j^i \models Q_j^i$, we can drop $\langle U_j^i, Q_j^i \rangle$ from $\mathcal{ORP}$ if i=1, and stop if $i = 2$ as no repair exists. Otherwise, we let the set $Supp_j^i$ of Q_j^i contain all assertions α such that $\mathcal{T} \cup \{\alpha\} \cup U_j^i \models Q_j^i$. Then, any repair $\mathcal{A}'$ must fulfill $\mathcal{A}' \cap Supp_j^1 \neq \emptyset$ for each j (i.e., be a *hitting set*), and must be disjoint with each $Supp_{j'}^2$. Let then $\mathcal{S}_j := (Supp_j^1 \cap \mathcal{A}) \setminus \bigcup_{j'} Supp_{j'}^2$. A σ_{del}-repair $\mathcal{A}'$ exists iff each $\mathcal{S}_j$ is nonempty; the hitting sets of the $\mathcal{S}_j$ are all the σ_{del}-repairs. The construction of the $\mathcal{S}_j$ and the check can be done in polynomial time, thus the overall problem is tractable. Furthermore, the (possibly exponentially many) σ_{del}-repairs can be output in total polynomial time.

□

If non-containment queries are allowed in DL-atoms, then the problem of computing deletion repairs for ORP stays NP-complete.

Theorem 4.28. *Deciding whether an Ontology Repair Problem $\mathcal{ORP} = \langle D_1, D_2, \mathcal{O} \rangle$ with a consistent $\mathcal{O}$ has a σ_{del} repair is* NP*-complete.* NP*-hardness holds even if either*

(i) for all $U_i^k \in D_j$ it holds that $U_i^k = \emptyset$, where $k \in \{1, 2\}$ or
(ii) $\mathcal{T} = \emptyset$, i.e. the TBox of the original ontology is empty.

Proof. To prove membership in NP we provide an algorithm for deciding whether a solution for an $\mathcal{ORP}$ exists. Our algorithm proceeds as follows. We guess a repair ABox $\mathcal{A}' \subseteq \mathcal{A}$ out of 2^n repair candidates, where $n = |\mathcal{A}|$ is the number of assertions in the original ABox $\mathcal{A}$ of $\mathcal{O}$. We then check whether the guess that we made is correct, i.e. $\mathcal{A}'$ satisfies the conditions (i)-(iii) of Definition 4.17. This test is polynomial, since it involves a bounded number of polynomial query evaluations over *DL-Lite*$_\mathcal{A}$ ontology (see Table 2.7). From this the membership in NP is obtained.

We prove NP-hardness for the cases (i) and (ii) separately.

(i) NP-hardness for (i) is shown by a reduction from SAT instances $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ over atoms $x_1, \ldots, x_n$. We construct the ORP problem $\mathcal{ORP} = \langle \langle \mathcal{T}, \emptyset \rangle, D_1, D_2 \rangle$, where all U_i^k are empty for $k \in \{1, 2\}$. In our construction we use concepts $X_j, \bar{X}_j, X'_j$ for the x_j, C_i for the χ_i as follows:

- $\mathcal{T} = \{X_j \sqsubseteq C_i, \mid x_j \in \chi_i, 1 \leq i \leq m\} \cup$
  $\{\bar{X}_{j'} \sqsubseteq C_i, \mid \neg x_{j'} \in \chi_i, 1 \leq i \leq m\} \cup$
  $\{\bar{X}_k \sqsubseteq \neg X'_k \mid 1 \leq k \leq n\}$
- $\mathcal{A} = \{X_j(b), \bar{X}_j(b) \mid 1 \leq j \leq n\}$
- $D_1 = \{\langle \emptyset, C_i(b) \rangle \mid 1 \leq i \leq m\}$
- $D_2 = \{\langle \emptyset, \neg(X_j \sqsubseteq X'_j) \rangle \mid 1 \leq j \leq n\}$

Intuitively, the queries in D_1 ensure that at least one $X_j(b)$ (resp. $\bar{X}_j(b)$) is present in the ontology ABox, such that x_j (resp. $\neg x_j$) is a disjunct in χ_i. The queries in D_2 forbid both $X_j(b)$ and $\bar{X}_j(b)$ to be in the ABox, which is expressed by the non-containment query $\neg(X_j \sqsubseteq X'_j)$ and TBox axioms of the form $\bar{X}_j \sqsubseteq \neg X'_j$. Therefore, the solution to $\mathcal{ORP}$ encodes a satisfying assignment of ϕ.

We now formally prove that ϕ is satisfied iff σ_{del} solution for $\mathcal{ORP}$ exists.

($\Rightarrow$) Let ϕ be satisfiable, and let A be a satisfying assignment of ϕ. From this we construct a solution $\mathcal{A}'$ to $\mathcal{ORP}$ as follows: if $A(x_j) = true$, then $X_j(b) \in \mathcal{A}'$, otherwise, $\bar{X}_j(b) \in \mathcal{A}'$. The ontology $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ is clearly consistent. Assume towards a contradiction that $\mathcal{A}'$ is not a solution to $\mathcal{ORP}$. That is, either (1) D_1 contains a tuple $\langle \emptyset, Q_i^1 \rangle$, such that $\mathcal{O}' \not\models Q_i^1$ or (2) some $\langle \emptyset, Q_j^2 \rangle$ exists in D_2, such that $\mathcal{O}' \models Q_j^2$. If (1) holds then $C_i(b)$ is not entailed for some i from $\mathcal{O}'$. That means that there is a conjunct $\chi_i \in \phi$, such that for none of its disjuncts x_j (resp. $\neg x_j$) we have the corresponding assertion $X_j(b)$ (resp. $\bar{X}_j(b)$) in $\mathcal{A}'$. Hence by construction none of the literals in χ_i is true under A, meaning that $A(\chi_i) = false$ and thus $A(\phi) = false$, i.e. contradiction. If (2) holds then for some j we have that $X_j(b)$ and $\neg X'_j(b)$ are entailed by $\mathcal{O}'$. Since by construction of $\mathcal{A}'$ it holds that $\mathcal{A}' \subseteq \mathcal{A}$, the only scenario when both $X_j(b)$ and $\neg X'_j(b)$ are entailed is when both $X_j(b), \bar{X_j}(b) \in \mathcal{A}'$. This can not happen, since $\mathcal{A}'$ is built from a satisfying assignment A of ϕ, and thus it represents a consistent set of values for x_j. Hence we obtain a contradiction.

($\Leftarrow$) Let $\mathcal{A}'$ be a σ_{del} solution to $\mathcal{ORP}$. From this we construct a satisfying assignment A for ϕ as follows:

$$A(x_j) = \begin{cases} true, & \text{if } X_j(b) \in \mathcal{A}' \text{ or } X_j(b), \bar{X}_j(b) \notin \mathcal{A}' \\ false, & \text{if } \bar{X}_j(b) \in \mathcal{A}'. \end{cases}$$

We show that $A(\phi) = true$. Observe that for every C_i there must exist X_j (resp. $\bar{X}_j$), such that $X_j \sqsubseteq C_i$ (resp. $\bar{X}_j \sqsubseteq C_i$) due to the tuples $\langle \emptyset, C_i(b) \rangle$ in D_1 and the fact that $\mathcal{A}' \subseteq \mathcal{A}$. Thus by construction of A in each clause χ_i some disjunct is true. It is left to show that A is well defined, i.e. it is not the case that (i) either $A(x_j) = true$ or $A(x_j) = true$ is defined for every j, and (ii) it is not the case that $A(x_j) = true$ and $A(x_j) = false$ for some j. In other words we need to show that

$A(x_j) \neq A(\neg x_j)$. Towards a contradiction suppose that this is not the case. Then for some i it holds that $A(x_j)$ and $A(\neg x_j)$ have the same value. Then $X_j(b)$ and $\bar{X}_j(b)$ are entailed from $\mathcal{O}$ for some j, and therefore $X'_j(b)$ is also entailed from $\mathcal{O}$ due to $\bar{X}_j \sqsubseteq X'_j \in \mathcal{T}$. However, this means that $\mathcal{O}' \models \neg(X_j \sqsubseteq X'_j)$, which is forbidden by the respective tuple $\langle , \neg(X_j \sqsubseteq X'_j)\rangle$ in D_2. The latter means that $\mathcal{A}'$ is not a solution to $\mathcal{ORP}$, leading to a contradiction. Thus A is a satisfying assignment of ϕ.

(ii) NP-hardness for (ii) is shown by a reduction from monotone not-all-equal SAT (NAE-SAT) instances $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ over atoms $x_1, \ldots, x_n$ [GJ79]. In monotone NAE-SAT, all occurrences of literals in clauses are positive, but a formula is "satisfied" only if there is an assignment under which both a literal assigned to true and a literal assigned to false occur in each clause. We construct the ORP problem $\mathcal{ORP} = \langle\langle\emptyset, \emptyset\rangle, D_1, D_2\rangle$, using concepts $X_j, \bar{X}_j$ for the x_j, C_i for the χ_i as follows:

- $\mathcal{A} = \{X_j(b), \bar{X}_j(b) \mid 1 \leq j \leq n\}$
- $D_1 = \{\langle\{\neg X_j(b) \mid x_j \in \chi_i\}, C_i(b)\rangle,$
  $\langle\{\neg \bar{X}_{j'}(b) \mid x_{j'} \in \chi_i\}, C_i(b)\rangle \mid 1 \leq i \leq m\}$
- $D_2 = \{\langle\emptyset, \neg(X_j \sqsubseteq \neg\bar{X}_j)\rangle, \langle\emptyset, C_i(b)\rangle \mid 1 \leq j \leq n, 1 \leq i \leq m\}$

Intuitively, queries Q_i^1 can be satisfied only if the repair ABox $\mathcal{A}'$ is inconsistent with the respective updates U_i^1, since $\mathcal{T} = \emptyset$ and explicit presence of $C_i(b)$ in $\mathcal{A}'$ is forbidden by tuples $\langle\emptyset, C_i(b)\rangle \in D_2$. Therefore, for every χ_i there must be some $X_j(b) \in \mathcal{A}'$, such that x_j is a conjunct in χ_i, which is ensured by $\langle\{X_j \mid x_j \in \chi_i\}, C_i\rangle \in D_1$. However, also some $\bar{X}_{j'}(b)$ must be in $\mathcal{A}'$, such that $x_{j'} \in \chi_i$, which is ensured by $\langle\{X_{j'} \mid x_{j'} \in \chi_i\}, C_i\rangle \in D_1$. By $\langle\emptyset, \neg(X_j \sqsubseteq \bar{X}_j)\rangle$, the indices j and j' must be different, thus the repair ABox encodes a consistent choice of values for variables in ϕ, corresponding to the satisfying assignment of ϕ.

We now formally show that ϕ is a positive instance of monotone NAE-SAT iff the $\mathcal{ORP}$ has some solution.

($\Rightarrow$) Let ϕ be a positive instance of monotone NAE-SAT, and let A be the witnessing assignment. From this we construct the solution $\mathcal{A}'$ to $\mathcal{ORP}$ as follows. $X_i(b) \in \mathcal{A}'$, if $A(x_j) = \mathit{true}$, and $\bar{X}_j(b) \in \mathcal{A}'$, if $A(x_j) = \mathit{false}$. Since for every clause χ_i some $x_j \in \chi_i$ must be set to true, we have that some $X_j(b) \in \mathcal{A}'$, and hence the query of $\langle\{\neg X_j(b) \mid x_j \in \chi_i\}, C_i(b)\rangle \in D_1$ is satisfied by inconsistency. Similarly, queries of tuples $\langle\{\neg \bar{X}_j(b) \mid x_j \in \chi_i\}, C_i(b)\rangle \in D_1$ are satisfied, as at least one x_j in χ_i is set to false, and by construction the respective $\bar{X}_j(b)$ is in $\mathcal{A}'$. The queries in D_2 are satisfied, since A represents a consistent choice of values for x_j, and thus both $X_j(b)$ and $\bar{X}_j(b)$ can not be present in $\mathcal{A}'$.

($\Leftarrow$) Let $\mathcal{A}'$ be a solution to the $\mathcal{ORP}$. From this we construct the assignment of ϕ as follows.

$$A(x_j) = \begin{cases} true, & \text{if } X_j(b) \in \mathcal{A}', \\ false, & \text{if } \bar{X}_j(b) \in \mathcal{A}'. \end{cases}$$

Since C_i can not be in $\mathcal{A}'$ by $\langle \emptyset, C_i \rangle \in D_2$ and $\mathcal{T} = \emptyset$, we have that all queries in D_1 are entailed by inconsistency introduced by the updates, and hence in every clause at least one of x_j must be true and at least one $x_{j'}$ must be false. Furthermore, the assignment A represents a consistent set of values for x_j by construction, since for all j not both $X_j(b)$ and $\bar{X}_j(b)$ can be in $\mathcal{A}'$ due to $\langle \emptyset, \neg(X_j \sqsubseteq \bar{X}_j) \rangle \in D_2$.

□

4.4.3 Deletion δ^+

This selection combines deletion and small change in a prioritized way. First one deletes assertions from $\mathcal{A}$ (assumed to be consistent) according to some polynomial method μ (using domain knowledge etc.) until some $\mathcal{A}_0 = \mu(\mathcal{O}) \subseteq \mathcal{A}$ results that satisfies Definition 4.17 (iii). If $\mathcal{A}_0$ is a repair, it is the result; otherwise, one looks for a close repair with bounded δ^+ change. That is

$$\sigma_{del,\delta^+}(S, \mathcal{A}) = \begin{cases} \{\mu(\mathcal{O})\}, & \text{if } \mu(\mathcal{O}) \in S \\ \sigma_{del,\delta^+}(S, \mu(\mathcal{O})), & \text{if } \mu(\mathcal{O}) \notin S. \end{cases} \tag{4.4}$$

Example 4.29. If $\mu(\mathcal{O})$ drops unreliable information about the gender of certain persons in Example 4.1 (e.g. *pat*), $\mathcal{A}_0$=$\{Male(john),\ hasParent(john, pat)\}$ is a deletion repair. If the constraint

$$\bot \leftarrow \text{DL}[; hasParent](X, Y), not\ \text{DL}[; Male](Y), not\ \text{DL}[; Female](Y)$$

(the gender of parents must be known) were in $\mathcal{P}$, one would have to add $Female(pat)$ to $\mathcal{A}_0$ to obtain a deletion-δ^+ repair. □

Then one can try all possible combinations of k assertions that can be added to the ABox $\mathcal{A}'$ such that along with condition (iii), also (ii) and (i) of the repair definition hold. Observe that $\mu(\mathcal{O})$ is selected by an independent selection function σ_{del}, which chooses subsets of $\mathcal{A}$. Furthermore, $\sigma_{\delta+}$ is applied to $\mu(\mathcal{O})$, the selection $\sigma_{\delta+}$ chooses an ABox $\mathcal{A}' \supseteq \mu(\mathcal{O})$, such that $\mathcal{A}' \setminus \mu(\mathcal{O})$ contains not more then k assertions. The selection $\sigma_{\delta+}$ is independent by Proposition 4.15, as it is a composition of σ_{del} and $\sigma_{\delta+}$ both of which are independent. Since both σ_{del} and $\sigma_{\delta+}$ are realizable in polynomial time, the tractability of the overall problem is obtained.

4.4.4 Addition under bounded opposite polarity

Repairs by unbounded additions become tractable, if few of them are positive resp. negative, i.e., the number of assertions with opposite polarity is bounded (which by Theorem 4.20 is necessary). That is, if $\mathcal{A}^+$ (resp., $\mathcal{A}^-$) is the positive (negative) part of an ABox $\mathcal{A}$, then

$$\sigma_{bop}(S, \mathcal{A}) = \{\mathcal{A}' \supseteq \mathcal{A} \mid |\mathcal{A}'^{+} \setminus \mathcal{A}| \leq k \text{ or } |\mathcal{A}'^{-} \setminus \mathcal{A}| \leq k\}. \tag{4.5}$$

The following result is instrumental.

Theorem 4.30. *For a $\not\sqsubseteq$-free Ontology Repair Problem $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and $\mathcal{T}$ has no disjointness axioms* [2] *deciding whether some σ_{bop}-repair exists is polynomial.*

Proof. We prove the statement for the case when few negative assertions are added to the ABox, i.e. $\mathcal{A}'^{-} \setminus \mathcal{A} \leq k$. The case when few positive assertion are added to the ABox, i.e. $\mathcal{A}'^{+} \setminus \mathcal{A} \leq k$ is completely symmetric, and our proof can be easily adapted to treat is it as well.

We provide an extension of the method for deletion repairs. Assuming that $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent (otherwise no σ_{bop}-repair exists), we proceed as follows:

1. Like for deletion repairs, we compute the sets $Supp^i_j$. We simplify $\mathcal{ORP}$, resp. quit if no repair can exist, checking also whether $Supp^i_j \cap \mathcal{A} \neq \emptyset$ (as then Q^i_j is entailed). More specifically, whenever $U^i_j \cup \mathcal{A} \cup \mathcal{T} \models Q^i_j$ or $Supp^i_j \cap \mathcal{A} \neq \emptyset$
   - we drop $\langle U^i_j, Q^i_j \rangle$ from D_i, if $i = 1$, and
   - we quit, if $i = 2$.
2. We then let $\mathcal{S}_j = Supp^1_j \setminus (\mathcal{A} \cup \bigcup_{j'} Supp^2_{j'})$. Similar as in the proof of Theorem 4.26, the σ_{bop}-repairs are then of the form $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}$ where $\mathcal{H}$ is a hitting set of the $\mathcal{S}_j$, but we must ensure that $\langle \mathcal{T}, \mathcal{A}' \rangle$ is consistent as $\mathcal{H}$ consists of new assertions.
3. We choose a set $\mathcal{H}^- \subseteq \bigcup_j \mathcal{S}_j$ of at most k negative assertions, which is a partial hitting set, and check that $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \rangle$ is consistent. If yes, we remove $\mathcal{S}_j$ if it intersects with $\mathcal{H}^-$ and remove otherwise from $\mathcal{S}_j$ each positive assertion α such that $\neg\alpha$ is entailed by $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \rangle$, and all negative assertions.
4. Then, for every hitting set $\mathcal{H}^+$ of $\mathcal{S}'_j$, the ABox $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}^- \cup \mathcal{H}^+$ is a σ_{bop}-repair. On the other hand, some σ_{bop}-repair with few negative additions exists only if some choice for $\mathcal{H}^-$ succeeds.

The crucial point for the correctness of this method is that, if $\mathcal{T}$ has no disjointness axioms, by adding to $\mathcal{A} \cup \mathcal{H}^-$ positive assertions $\mathcal{H}^+$ we can not infer new negative

[2] Disregarding axioms $F_\neg \sqsubseteq \neg F$ to compile negative assertions.

assertions, unless inconsistency emerges; this is exploited in Step 3, which limits the candidate space for positive hitting sets *a priori*.

We now show the correctness of the proposed algorithm formally. Suppose that given the Ontology Repair Problem $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$ as an input to the algorithm from above, the ABox $\mathcal{A}'$ was produced as the output after execution of the Steps 1-3. We prove that the ABox $\mathcal{A}'$ is indeed a σ_{bop}-repair for $\mathcal{ORP}$, i.e. we prove that the conditions that a σ_{bop}-repair needs to satisfy are indeed satisfied by $\mathcal{A}'$.

(i) $\mathcal{T} \cup \mathcal{A}' \cup U_j^1 \models Q_j^1$ for all $\langle U_j^1, Q_j^1 \rangle \in D_1$.

Towards a contradiction, suppose that there is some $\langle U_{j_i}^1, Q_{j_i}^1 \rangle \in D_1$, such that $\mathcal{A}' \cup \mathcal{T} \cup U_{j_i}^1 \not\models Q_{j_i}^1$. We know that by construction, it either holds that (1) $U_{j_i}^1 \cup \mathcal{T} \models Q_{j_i}^1$; (2) $\mathcal{A} \cap Supp_{j_i}^1$, i.e. $\mathcal{A} \cup \mathcal{T} \models Q_{j_i}^1$; (3) $\mathcal{H}^- \subseteq \mathcal{A}'$ hits $\mathcal{S}_{j_i}$ or (4) $\mathcal{H}^+ \subseteq \mathcal{A}'$ hits $\mathcal{S}_{j_i}$. For (1) and (2) we immediately get a contradiction. For (3) it holds that $\mathcal{H}^- \cap Supp_{j_i}^1 \neq \emptyset$. Therefore, there is $\alpha \in \mathcal{A}'$, such that $\{\alpha\} \cup U_j^1 \cup \mathcal{T} \models Q_{j_i}^1$.

(ii) $\mathcal{T} \cup \mathcal{A}' \cup U_{j'}^2 \not\models Q_j'^2$ for all $\langle U_{j'}^2, Q_{j'}^2 \rangle \in D_2$.

To the contrary, assume that there exists some j_i', such that $\mathcal{T} \cup \mathcal{A}' \cup U_{j_i'}^2 \models Q_{j_i'}^2$. There are several possibilities: (1) $U_{j_i'}^2 \cup \mathcal{T} \models Q_{j_i'}^2$; (2) there is $\alpha \in \mathcal{A}$, such that $\{\alpha\} \cup U_{j_i'}^2 \cup \mathcal{T} \models Q_{j_i'}^2$; (3) there is $\alpha \in \mathcal{H}^-$, such that $\{\alpha\} \cup \mathcal{T} \cup U_{j_i'}^2 \models Q_{j_i'}^2$; (4) there is $\alpha \in \mathcal{H}^+$, such that $\{\alpha\} \cup \mathcal{T} \cup U_{j_i'}^2 \models Q_{j_i'}^2$. Observe that if (1) or (2) were the case, then the algorithm would terminate at Step 1, and no repair $\mathcal{A}'$ would be in the output. For the case (3) we have that $\mathcal{H}^- \cap Supp_{j_i'}^2 \neq \emptyset$. However, according to the Step 3 of our algorithm, it holds that $\mathcal{H}^- \subseteq \bigcup_j (Supp_j^1 \setminus (\mathcal{A} \cup \bigcup_{j'} Supp_{j'}^2))$, meaning that $\mathcal{H}^- \cap Supp_{j_i'}^2 = \emptyset$, which leads to a contradiction.

(iii) $\mathcal{A}' \supseteq \mathcal{A} \mid |\mathcal{A}'^- \setminus \mathcal{A}| \leq k$, i.e. there are at most k negative assertions in the ABox $\mathcal{A}'$.

Finally, we show that the number of negative assertions in $\mathcal{A}' \setminus \mathcal{A}$ is indeed bounded by k. Towards a contradiction, suppose that there are more then k negative assertions in $\mathcal{A}' \setminus \mathcal{A} = \mathcal{H}^+ \cup \mathcal{H}^-$. According to the Step 3 of our algorithm, it holds that $\mathcal{H}^-$ contains at most k negative assertions. Therefore, the rest of the negative assertions must be in $\mathcal{H}^+$. The set $\mathcal{H}^+$ is constructed at Step 4 as a hitting set of sets $\mathcal{S}_j$, which due to the Step 3 contain only positive assertions. Therefore, there are no negative assertions in the set $\mathcal{H}^+$, moreover $\mathcal{T} \cup \mathcal{H}^+ \cup \mathcal{H}^-$ infers only at most k negative assertions, since $\mathcal{T}$ contains only positive inclusions and $\mathcal{A} \cup \mathcal{H}^+ \cup \mathcal{H}^- \cup \mathcal{T}$ is guaranteed to be consistent at Step 3.

The above shows that the output $\mathcal{A}'$ is indeed a σ_{bop}-repair for the $\mathcal{ORP}$ with at most k negative assertions. The case when few positive assertions are allowed for addition is completely symmetric.

Finally, we show that if a given $\mathcal{ORP}$ has σ_{bop} repairs, then after executing the Steps 1-3 some σ_{bop} repair is guaranteed to be found, i.e. $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}^+ \cup \mathcal{H}^-$, such that

$|\mathcal{H}^-| \leq k$. Assume towards a contradiction that this is not the case. We distinguish the cases based on states of the algorithm at which the computation could have terminated.

- Suppose that the computation terminated at (1). Then there is some $\langle U_j^2, Q_j^2 \rangle \in D_2$, such that either (i) $U_j^2 \cup \mathcal{T} \models Q_j^2$ or (ii) $\mathcal{A} \cap Supp_j^2 \neq \emptyset$. If (i) holds then by monotonicity we have that for any $\mathcal{A}'$ the condition (iii) of Definition 4.17 is not satisfied, i.e. $\mathcal{ORP}$ does not have any solutions, which contradicts our assumption. If (ii) is the case, then there is some $\alpha \in \mathcal{A}$, such that $\alpha \cup \mathcal{T} \models Q_j^2$. Again due to monotonicity, for any ABox $\mathcal{A}' \supseteq \mathcal{A}$ it is true that $\mathcal{A}' \models Q_j^2$. Thus all repairs $\mathcal{A}'$ for $\mathcal{ORP}$ are such that $\mathcal{A}' \not\supseteq \mathcal{A}$. Therefore, no σ_{bop} repair exists for $\mathcal{ORP}$, contradicting our assumption.

- Assume that we have reached (2), and constructed the sets $\mathcal{S}_j$. Suppose that the computation stopped at (2), i.e. no hitting set $\mathcal{H}$ of $\mathcal{S}_j$ was found. This means that some j_1 exists, such that $\mathcal{S}_{j_1} = \emptyset$. Therefore, by construction of $\mathcal{S}_{j_1}$ it holds that $Supp_{j_1} \setminus (\mathcal{A} \cup \bigcup_{j'} Supp_{j'}^2) = \emptyset$. Since all $\langle U_j^1, Q_j^1 \rangle$, such that $Supp_j^1 \cap \mathcal{A} \neq \emptyset$ were removed from D_1 at (1), we have that for all $\alpha \in Supp_{j_1}^1$, it holds that $\alpha \in Supp_k^2$ for some k. Hence, for all ABoxes $\mathcal{A}' = \mathcal{A} \cup \alpha$ some $\langle U_k^2, Q_k^2 \rangle \in D_2$ exists, such that $\langle \mathcal{T}, \mathcal{A}' \rangle \models Q_k^2$, meaning that $\mathcal{ORP}$ does not have any solutions, which leads to a contradiction.

- Suppose that the state (3) has been reached, i.e. some repair candidate $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}$ was identified at (2), where $\mathcal{H}$ is a hitting set of $\mathcal{S}_j$. At (3) we picked some set $\mathcal{H}^-$ and updated every $\mathcal{S}_j$ by removing appropriate assertions from $\mathcal{S}_j$. Computation could not have stopped at (3), therefore, we are guaranteed to reach (4). Assume that the algorithm terminated at (4). Then it must be the case that no hitting set $\mathcal{H}^+$ of updated $\mathcal{S}_j$ has been found at (4); that is for all choices of $\mathcal{H}^-$ at (3) some j_1 exists, such that $\mathcal{S}_{j_1} = \emptyset$ at (4). Consider some particular $\mathcal{H}^- \subseteq \bigcup_j \mathcal{S}_j$ of at most k assertions computed at (3), such that $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \rangle$ is consistent. We have that $\mathcal{S}_{j_1} \cap \mathcal{H}^- = \emptyset$ at (3), since otherwise $\mathcal{S}_{j_1}$ would have been removed and would have not been considered in the computation of a hitting set $\mathcal{H}^+$ at (4). We have that for all positive $\alpha \in \mathcal{S}_{j_1}$, the ontology $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \cup \{\alpha\} \rangle$ is inconsistent. As $\langle U_{j_1}^1, Q_{j_1}^1 \rangle$ was not dropped at (1), we have that $\langle \mathcal{T}, U_{j_1}^1 \cup \mathcal{A} \rangle \not\models Q_{j_1}^1$. Therefore, it follows by Lemma 4.27 that no σ_{bop}-repair exists, such that $\mathcal{A}'^- \setminus \mathcal{A} \leq k$, leading to a contradiction.

We have shown that if $\mathcal{ORP}$ has solutions with at most k negative assertions, then some such solution will be found by our algorithm. The argument can be accordingly adjusted to prove the statement for few positive assertions are allowed for addition. □

4.4.5 Applicability of independent selections

Like for relational databases, our tractable cases fit real applications, e.g. in case of deletion repairs (observing that non-subsumption queries are insignificant for practical DL-programs) and scenarios akin to key-constraint violations in databases. Restoring

consistency by removing conflicting pieces of data is a common approach in data management.

Composability of independent selections adds to their applicability. Moreover, they may be combined with DB-style factorization and localization techniques (see [Ber11] and references therein) and with local search to compute closest repairs.

Bounding the number of changes, especially additions, is also compliant with practice, where too many potential repairs suggest human intervention (cf. [Ber11]). Finally, one may increase the bound in iterative deepening (assuming that not many changes are needed).

4.4.6 ORP for DL-programs over $\mathcal{EL}$ DL

Unfortunately most of the tractable results presented earlier in this section will not be inherited to the $\mathcal{EL}$ DL. The only polynomial case is the $\delta^{\pm}$-change repairs. For the rest of the repairs the algorithms proving tractability for DL *DL-Lite*$_{\mathcal{A}}$ that we provided, do not work for the DL $\mathcal{EL}$. All of the algorithms are based on small portions of the data that are sufficient for the query entailment (support sets as we call them). The attractive property of *DL-Lite*$_{\mathcal{A}}$ states that these support sets are of bounded size (at most of size 2) and there are polynomially many of them. Therefore, they can be easily constructed. The $\mathcal{EL}$ DL does not have this property because of the range restrictions and concept conjunctions allowed on the left-hand side of inclusion axioms. There might be arbitrarily large and infinitely many support sets for instance queries in $\mathcal{EL}$. In Chapter 5 we introduce the support sets formally and present optimized algorithms that allow to still obtain reasonably good results in practice for repair computation of DL-programs over $\mathcal{EL}$ DL.

4.5 Domain-based Restrictions on Repairs

In previous sections we have proposed several technical means for treating inconsistencies in DL-programs. We have presented some repair forms that are practically usable and computationally effective, but until now no domain knowledge has been incorporated into the DL-program repair process. It is natural, however, to believe that the end users of DL-programs will wish to contribute to the repair by sharing their subject expertise.

Qualitative and domain-dependent aspects of repairs are of crucial importance for their practicability. These qualitative aspects formulated in terms of additional local restrictions put on repairs help to effectively filter out the irrelevant repair candidates. For example, availability of meta information about the trustfulness of certain ontology pieces allows to instantly adjust the repair process accordingly.

Example 4.31. Being aware of the *unreliability* of ontology facts about the individual *john* in Example 4.1 motivates one to consider the repair $\mathcal{A}' = \mathcal{A} \backslash \{hasParent(john, pat)\}$ for the DL-program Π in the first instance.

Knowing additionally that the whole set of *Adopted* children is very likely to be incomplete adds $\mathcal{A}'' = \mathcal{A} \cup \{Adopted(john)\}$ to the set of the available repair possibilities. □

In contrast to the solution that involves dropping certain data pieces, there might be information stored in the ontology that is *safe* and should necessarily be kept unchanged. In such cases one might naturally aim to leave the original set of safe facts in the ontology.

Example 4.32. For instance, in contrast to Example 4.31 we might want to avoid dropping the data about those individuals who belong to the concept *Child* but not known to be *Adopted*; then the repair $\mathcal{A}'$ is no longer among the preferred options. □

The guidelines on the operations that are allowed to be applied to the ontology could clearly influence the repair process further.

Example 4.33. If in Example 4.31 additions to the ontology are strongly prohibited, then the repair $\mathcal{A}''$ is automatically dropped from the set of leading candidates. □

In some scenarios one could be aware of various dependencies among the data parts stored in the ontology. Consequently, deletion of a certain fact might force the deletion of another one. Additions in such settings might be likewise guided.

Example 4.34. Consider a variant of Example 4.1, in which each *Adopted* child stored in the ontology is desired to have a certain identification number (*ID*) assigned to it through the predicate *hasID*. This additional constraint could clearly be expressed by the TBox axiom $Adopted \sqsubseteq \exists hasID$. However, this restriction might not be a formal requirement, but rather a wish of the user, for whom it is more convenient to track adopted children by their IDs. Thus the TBox axiom might not be present in the ontology explicitly. In such a setting the repair $\mathcal{A}''$ from Example 4.31 in which information about *john*'s adoption is added, is not among the best repair candidates any longer, as together with this new information, the additional knowledge about the *ID* of *john* should be available.

Similarly, if not only adopted children, but all the persons in the database are required to have an ID, and they are indeed specified in the original version of the ontology, the repair $\mathcal{A}' = \mathcal{A} \setminus \{Male(pat)\} \cup \{Male(mat)\}$ from Example 4.23 forces one to delete the *ID* of *pat* and add the ID of *mat*; in case the latter is not known, such repair becomes meaningless. □

Integration of domain restrictions into repair computation process. The wide spectrum of potential restrictions that could be applied to the repair candidates motivates one to consider various possible ways of integrating additional domain knowledge into the repair computation process. Three global modes of repairing inconsistent DL-programs seem reasonable in this context:

(1). The first mode suggests the computation of repair candidates with some σ-function, followed by a post-filtering of the candidates taking into account the domain knowledge. If some of the protected ontology elements are no longer present in the repair

candidate, and the their reintroduction violates the repair conditions, then one proceeds with the analysis of a next repair candidate. Otherwise, the desired repair is computed and the computation process terminates.

(2). The second mode assumes that the domain knowledge is encoded in the selection function and consequently all identified repairs *a priori* satisfy the requirements posed by the domain knowledge.

Example 4.35. Suppose we want to compute the $\delta^{\pm}$ repairs with the desired property expressed in Example 4.34, i.e. in the repairs for all *Adopted* children their *ID* should be known. Then our problem converges to the problem of computing $\delta^{\pm}$ repairs of the original DL-program extended by the following rules that conveniently encode the additional requirement:

(1) $assigned(X) \leftarrow \mathrm{DL}[; Adopted](X), \mathrm{DL}[; ID](Y), \mathrm{DL}[; hasID](X, Y)$;
(2) $\bot \leftarrow \mathrm{DL}[; Adopted](X), not\ assigned(X)$.

The repair of the extended program will correspond to the repairs of the original program post-filtered by the respective domain-specific condition. □

(3). The third mode is the combination of the first two, where some domain conditions are incorporated into the repair search process, such that the computed repairs are satisfied, but still post-filtering conditions can be applied.

(4). The mode (2) can be extended to support prioritized repair computation. That is, first one aims at finding the best repairs that fully satisfy the domain specific requirements, and then if such search does not bring any results, the requirements are weakened accordingly or even dropped altogether. This mode is suitable for the settings in which such requirements are not strict but rather reflect certain preference criteria.

Example 4.36. Recall the setting from Example 4.35. We first aim at repairs, such that *ID*s of all adopted children are known, i.e. we search for repair answer sets of the original DL-program Π extended by the rule (1) and the constraint (2) from Example 4.35 encoding this repair property. Once some repair answer set is found, the computation terminates and the result is output. If no repair answer set was identified, then one might want to relax the repair condition by allowing some adopted children to be without *ID*s but bounding their number. For that the constraint (2) might be changed to a rule (2') with $not\_assigned(X)$ in the head. We then aim at finding the best repair answer set I of the original DL-program extended by the rules (1) and (2'). Here the best repair answer set will be an interpretation I which contains a bounded number of facts over the predicate *not_assigned*. Once such I is found, any repair $\mathcal{A}' \in rep^{I}_{\sigma,x}(\Pi)$ is guaranteed to be preferred, where σ is a $\delta^{\pm}$-change selection function. □

Note that clearly in some settings selecting preferred repairs with respect to certain domain specific criteria (e.g. maximal number of *adopted* children whose *ID* is known) might require non-independent σ-selections. E.g. in Example 4.36 one would need to compute all repair answer sets and compare them with respect to the number of atoms over the predicate *not_assigned* in order to identify the preferred repair.

All of the discussed domain-specific repair preferences can be combined and ordered in various ways. The techniques for their computation heavily depend on the application scenario, and in different concrete settings could be adapted and extended.

4.6 Conclusion, Related Work and Outlook

Our main results presented in this chapter are summarized as follows. The general framework for repairing inconsistent DL-programs has been introduced. The overall complexity of DL-program repair has proved to be Σ_2^p-complete in the general case. As a subproblem we have identified an interesting ontology repair problem ORP which non-surprisingly turned out to be NP-complete. We, furthermore, established useful classes of repairs in practice that are tractable for the well-known Description Logics *DL-Lite*$_\mathcal{A}$.

Managing inconsistent DL-programs has focused so far on inconsistency tolerance, rather than on repair as we considered. Pührer *et al* in [PHE10] avoid unintuitive answer sets caused by inconsistency in DL-atoms, and dynamically deactivate rules to discard spoiled information; they pointed ontology repair out as an issue, but left it open. Fink in [Fin12b] presented a paraconsistent semantics, based on the logic of here-and-there.

Repairing ontologies was considered in many works, often to deal with inconsistency. In particular, Lembo *et al* [LLR^{++}11] and Bienvenu [Bie12] studied consistent query answering over DL-Lite ontologies based on the repair technique (see [Ber11]), using minimal deletion repairs (which amount to non-independent σ-selections). Calvanese *et al* [COSS12a] studied query answers to *DL-Lite*$_\mathcal{A}$ ontologies that miss expected tuples, and defined abductive explanations corresponding to repairs. They analyzed the complexity of explanation existence for various preferences that amount to non-independent σ-selections. While their problem can be viewed as a special ORP for atomic queries, we deal–also in comparison to the aforementioned works–with a more general problem, where mixed entailment and non-entailment of queries must be satisfied, and moreover under ABox updates. Repairing inconsistent non-monotonic logic programs is less developed; Sakama *et al* [SI03] used extended abduction to delete minimal sets of rules (but also adding rules can remove inconsistency).

Our sketched ideas on domain-dependent restrictions on repairs are related to the inconsistency policy for databases discussed e.g. in [SCM12, MPP$^+$14]. The authors presented the preference-based techniques for repairing databases. To the best of our knowledge this topic has not been considered before in the context of DL-programs.

There are several issues for future work. One is extending the inconsistency framework that we presented to more expressive DLs like Horn-$\mathcal{SHIQ}$, $\mathcal{SHIQ}$, or $\mathcal{SROIQ}$. Related to this is also to consider DL-programs with richer queries, e.g., with (unions of) conjunctive queries (cf. [EIKS08]), or more generally to consider logic programs that access multiple and perhaps also heterogeneous ontologies.

Orthogonal to this are further σ-selections for repairs, both independent and non-independent ones. The latter may cause intractability in very simple settings, as they open an exponential search space (e.g., subset-minimal or least, minimal Dalal distance repair). However, neighborhood search on top of σ (bounding e.g. distance from the original ABox) allows to compute locally optimal σ-repairs without loss of tractability.

Following approaches from database research, one can try to axiomatize repairs of a DL-program by means of a logical theory, such that the models of the theory correspond to the possible repairs of the DL-program. In other words, instead of explicitly computing the possible repairs, we may attempt to characterize their formal properties, and subsequently represent them in compact logical terms, all at once. This research direction is connected to the problem of identifying necessary and sufficient conditions for the repair existence. These conditions give new insights into the repair process, and provide with the new comprehensive understanding of repairs. For analysis of repair structures in DL-programs conflict hypergraphs [ABC01, CM05] from the database theory can be explored and adapted for DL-programs.

Another direction of research is the extension of the introduced repair semantics for inconsistency tolerant cautious and brave query answering over DL-programs, and complexity analysis of such an extension. Similar problems were studied in the context of Description Logics, e.g. [LLR$^+$11, BR13, BBG14], etc. The brave inconsistency tolerant reasoning, i.e. checking whether $I \models^{\mathcal{O}'} p(\vec{t})$ for some repair answer set I with a repair $\mathcal{A}'$ could be roughly compared to the reasoning in DLs under the AR semantics [LLR$^+$11]. The cautious reasoning in DL-programs, i.e. checking whether $I \models^{\mathcal{O}'} p(\vec{t})$ for all $I \in RAS(\Pi)$ is remotely related to the IAR semantics of DLs [LLR$^+$11]. However, an extensive study of the topic is left for future work.

CHAPTER 5

Algorithms for Repair Answer Set Computation

In this chapter we provide algorithms for repair answer set computation. In Section 5.1 we show how an algorithm that is used for evaluating DL-programs can be gracefully extended to naively compute repairs resp. repair answer sets, with an optional selection σ featuring independence. The support sets for DL-atoms as optimization means are introduced in Section 5.2. We discuss the forms of support sets for DL-atoms accessing *DL-Lite*$_\mathcal{A}$ ontologies in Section 5.3. Section 5.4 presents an algorithm for repair answer set computation based on complete support families, which is effective for DL-programs over *DL-Lite*$_\mathcal{A}$ ontologies. The construction of support sets for DL-atoms accessing $\mathcal{EL}$ ontologies is discussed in Section 5.5. Due to range restrictions and concept conjunctions on the left-hand side of inclusion axioms in $\mathcal{EL}$, a DL-atom accessing an $\mathcal{EL}$ ontology can have arbitrarily large and infinitely many support sets in general. Therefore, in Section 5.6 we consider approaches for computing incomplete (partial) support families. Moreover, we analyze the TBox structure ensuring bound on the size of support sets and their number in Section 5.7. The extension of the algorithm for repair computation to partial support families is described in Section 5.8. Finally, discussion of other optimization means for repair answer set algorithms is given in Section 5.9. The related work and outline are presented in Section 5.10.

5.1 Naive Repair Answer Set Computation Algorithm

We first present the algorithm for evaluating DL-programs from [EFK$^+$12] (given there for HEX-programs). We then describe the core procedure of its extension that checks whether a given interpretation of a DL-program is a (σ, x)-repair answer set, where x stands for the semantics used.

Algorithm 5.1: *AnsSet*: Compute $AS_x(\Pi)$

Input: A DL-program Π, $x \in \{weak, flp\}$
Output: $AS_x(\Pi)$
for $\hat{I} \in AS(\hat{\Pi})$ **do**
(a) **if** $CMP(\hat{I}, \Pi) \wedge xFND(\hat{I}, \Pi)$ **then**
output $\hat{I}|_\Pi$
end
end

5.1.1 Evaluation of DL-programs

DL-programs can be seen as a special case of HEX-programs, whose evaluation of which was briefly discussed in Chapter 2. Like for HEX-programs, the evaluation of DL-programs builds on a rewriting $\hat{\Pi}$ of Π, where DL-atoms a are replaced by ordinary atoms (replacement atoms) e_a, together with a guess on their truth by additional 'choice' rules. Given an interpretation $\hat{I}$ over this extended language, we use $\hat{I}|_\Pi$ to denote its restriction to the original language of Π. A crucial notion is that of compatibility:

Definition 5.1. [EFK$^+$12] A *compatible set* of a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is an interpretation $\hat{I}$, such that

(i) $\hat{I}$ is an answer set of $\hat{\Pi}$, and
(ii) $e_a \in \hat{I}$ iff $\hat{I}|_\Pi \models^{\mathcal{O}} a$, for all $a = \mathrm{DL}[\lambda; Q](c)$ of Π.

Conversely, given an interpretation I of Π, we denote by I_c the interpretation of $\hat{\Pi}$, such that I_c coincides with I on nonreplacement atoms, and each replacement atom e_a is in I_c (i.e., true) iff $I \models^{\mathcal{O}} a$ for the respective DL-atoms a. Clearly, from each interpretation $\hat{I}$ of $\hat{\Pi}$ we can uniquely construct an interpretation $\hat{I}|_\Pi$ of Π.

With these concepts in place, we are ready to describe the basic algorithm *AnsSet* (cf. Algorithm 5.1) for evaluating a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ adapted from [EFK$^+$12].

First, $\hat{\Pi}$ is evaluated by an ordinary ASP solver. For every answer set $\hat{I}$, in (a) the function *CMP* checks for compatibility of every answer set $\hat{I}$ against the values of the DL-atoms, i.e. verifies the guess. The test *xFND* checks for foundedness, i.e., whether $\hat{I}|_\Pi$ is a $\subseteq$-minimal model of the reduct $\mathcal{P}_x^{\hat{I}|_\Pi, \mathcal{O}}$. In case of $x = weak$ it just returns true, otherwise ($x = flp$) it checks for disjointness with unfounded sets as defined in [EFK$^+$12].

An important link between the answer sets of Π and $\hat{\Pi}$ is:

Proposition 5.2. *Let $\Pi = \langle \mathcal{P}, \mathcal{O} \rangle$ be a DL-program. If $I \in AS_x(\Pi)$ then $I_c \in AS_x(\hat{\Pi})$.*

Proof. Suppose towards a contradiction that $I \in AS_x(\Pi)$, but $I_c \notin AS_x(\hat{\Pi})$. Then it holds that I_c is not a $\subseteq$-minimal model of $\hat{\Pi}_x^{I_c}$. There are two possibilities: either (i) there exists a rule r in $\hat{\Pi}_x^{I_c}$, such that $I_c \not\models r$ or (ii) there exists $I' \subset I_c$, which is a model of $\hat{\Pi}_x^{I_c}$. Before considering these cases, let us compare the reducts $\mathcal{P}_x^{I, \mathcal{O}}$ and $\hat{\Pi}_x^{I_c}$. Clearly, all normal rules r without replacement atoms that are in $\hat{\Pi}_x^{I_c}$ are also in $\mathcal{P}_x^{I, \mathcal{O}}$.

Now the rules r' that contain replacement atoms in $\hat{\Pi}_x^{I_c}$ are also present in $\mathcal{P}_x^{I,\mathcal{O}}$ as rules r'', they coincide with r' on normal atoms, do not contain any DL-atoms if $x = weak$ and contain all DL-atoms if $x = flp$.

First assume that the case (i) holds. We distinguish the following possibilities:

- r does not contain any replacement atoms. As $I \in AS_x(\Pi)$, I must satisfy r, therefore r is also satisfied by I_c.
- r contains some replacement atoms. Since r is in $\hat{\Pi}_x^{I_c}$, its body must be satisfied by I_c. That is all atoms in the positive part of the body of r are in I_c and those in its negative part are not in I_c. The body of the same rule in $\mathcal{P}_x^{I,\mathcal{O}}$ is satisfied by I too, which means that the head of r is in I and hence also in I_c. The latter means that r is satisfied by I_c.

Now suppose that (ii) holds, i.e. there exists some $I' \subset I_c$, such that I' satisfies all rules in $\hat{\Pi}_x^{I_c}$. First suppose that $S = I_c \backslash I'$. Observe that S must contain normal atoms. Otherwise, there is a replacement atom e_a, which is in I_c but not in I' and therefore $I \models a$, but $I \backslash S \not\models a$. The latter can not happen, as $I \backslash S = I$. Therefore, S must contain normal atoms. We now look at $I'' = I \backslash S$. We know that $I'' \subset I$ is not a model of $\mathcal{P}_x^{I,\mathcal{O}}$ (otherwise I would not be an answer set of Π). This means that there is a rule r in $\mathcal{P}_x^{I,\mathcal{O}}$, such that $I'' \models b(r)$, but $h(r) \notin I''$. Observe that the same rule must be present in the reduct $\hat{\Pi}_x^{I_c}$ with the only possible difference that it contains replacement atoms. I' must satisfy this rule by our assumption. This, however, means that the head of this rule is in I' and thus in I''. Therefore, $I'' \models r$, meaning that I'' is a model of $\mathcal{P}_x^{I,\mathcal{O}}$, and hence $I \notin AS(\Pi)$, which leads to a contradiction. □

While *AnsSet* is clearly sound, from the above result its completeness follows, i.e. restricting the search to $AS_x(\hat{\Pi})$ does not yield any loss of answer sets.

5.1.2 Naive Repair Algorithm

In this subsection, we first aim at a procedure for computing (σ, x)-repairs of a DL-program given an independent selection function σ. Then, we describe how its main subroutine can be used for an extension of *AnsSet* that computes answer sets if some exist, and (σ, x)-repair answer sets otherwise.

A first key observation is that Proposition 5.2 can be generalized to repair answer sets, more precisely:

Proposition 5.3. *If $I \in RAS_x(\Pi)$ then $I_c \in AS_x(\hat{\Pi})$.*

Proof. By definition of $RAS_x(\Pi)$, we get that $I \in AS(\Pi')$, where $\Pi' = \langle \mathcal{O}', P \rangle$, $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ and $\mathcal{A}' \in rep_x(\Pi)$. Since by Proposition 5.2 $I_c \in AS_x(\hat{\Pi}')$ and $\hat{\Pi} = \hat{\Pi}'$, the result immediately follows. □

Thus, our approach is to traverse $AS(\hat{\Pi})$ and check for each answer set $\hat{I}$ whether it is a (σ, x)-repair answer set of Π. The latter proceeds in two steps. In the first step we

Algorithm 5.2: *RepAns*: Compute (σ, x)-repairs $rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$ of Π

Input: $\Pi{=}\langle\mathcal{O}, \mathcal{P}\rangle$, $\mathcal{O}{=}\langle\mathcal{T}, \mathcal{A}\rangle$, $\hat{I}{\in}AS(\hat{\Pi})$, σ, $x{\in}\{weak, \mathit{flp}\}$
Output: $rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$
(a) **for** $\mathcal{A}' \in ORP(\hat{I}, \Pi, \sigma)$ **do**
(b) | **if** $CMP(\hat{I}, \langle\mathcal{T}, \mathcal{A}', \mathcal{P}\rangle) \wedge xFND(\hat{I}, \langle\mathcal{T}, \mathcal{A}', \mathcal{P}\rangle)$ **then**
| | output $\mathcal{A}'$
| **end**
end

Algorithm 5.3: *RepAnsSet*: Compute a set $RAS_{(\sigma,x)}(\Pi)$ of (σ, x)-repair AS of Π

Input: $\Pi{=}\langle\mathcal{O}, P\rangle$, $\mathcal{O}{=}\langle\mathcal{T}, \mathcal{A}\rangle$, σ, $x{\in}\{weak, \mathit{flp}\}$
Output: $I \in RAS_{(\sigma,x)}(\Pi)$
for $\hat{I} \in AS(\hat{\Pi})$ **do**
| **if** $RepAns(\Pi, \mathcal{O}, \hat{I}, \sigma, x) \neq \emptyset$ **then**
| | output $\hat{I}|_\Pi$
| **end**
end

search for potential σ-repairs of the ontology such that the condition (ii) of Definition 5.1 is satisfied for $\hat{I}$. In the second step we check the minimality condition of $\hat{I}$ with respect to the given semantics x and the (σ, x)-repair that was found. Hence, our extension is driven by the following idea: whenever $\hat{I}$ turned out to be not compatible, we aim at computing a (σ, x)-repair which will turn it into a compatible set. Otherwise, we follow the further step of *AnsSet* and check the minimality of I w.r.t. semantics x.

Our approach is given in Algorithm 5.2, which computes the set $rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$ of all ABoxes under which $\hat{I}$ becomes a (σ, x)-repair answer set. A procedure *RepAns* calls the subroutine $ORP(\hat{I}, \Pi, \sigma)$ to compute σ-repairs of the corresponding ORP. Further on, *RepAns* re-uses functions *CMP* and *xFND* to check whether $\hat{I}$ is an answer set of $\hat{\Pi}'$ and that it is founded w.r.t. $\Pi' = \langle\mathcal{O}', \mathcal{P}\rangle$, where $\mathcal{O}' = \langle\mathcal{T}, \mathcal{A}'\rangle$. It thus computes the set of all ABoxes under which $\hat{I}$ becomes a (σ, x)-repair answer set.

We demonstrate how the algorithm *RepAns* works on the following example

Example 5.4. Let $\Pi = \langle\mathcal{O}, \mathcal{P}\rangle$ be a DL-program, where

$$\mathcal{O} = \left\{ \begin{array}{ll} A \sqsubseteq \neg C & D(c) \\ A \sqsubseteq D & \neg E(c) \\ A(c) & C(c) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} p(c);\quad r(c);\quad q(c) \leftarrow \mathrm{DL}[C \uplus r;\, D](c); \\ \bot \leftarrow\ \mathrm{DL}[D \uplus p, E \uplus r;\, \neg C](c) \end{array} \right\}.$$

Let $a_1 = \mathrm{DL}[C \uplus r;\, D](c)$ and $a_2 = \mathrm{DL}[D \uplus p, E \uplus r;\, \neg C](c)$, and consider the interpretation $\hat{I} = \{p(c), r(c), q(c),\, e_{a_1}\}$, i.e., a_1 is guessed true and a_2 false. The corresponding ORP is given by $\mathcal{ORP} = \langle \mathcal{O}, D_1, D_2 \rangle$, where $D_1 = \{\langle \{\neg C(c)\}; D(c) \rangle\}$ and $D_2 = \{\langle \{D(c), \neg E(c)\}; \neg C(c) \rangle\}$. Now if the selection function σ selects the repairs that are subsets of the original ABox, i.e. we are interested in deletion repairs, then as a possible solution to $\mathcal{ORP}$ we get $\mathcal{A}' = \{D(c), C(c)\}$. Since $\mathcal{A}'$ is a solution to the $\mathcal{ORP}$, the compatibility check $CMP(\hat{I}, \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle)$ succeeds. Furthermore, the interpretation $\hat{I}|_\Pi$ is a *weak*-repair answer set. For checking whether $\hat{I}|_\Pi$ is an FLP-repair answer sets, we construct the reduct $\mathcal{P}^{I,\mathcal{O}'}_{flp} = \{p(c);\quad q(c);\quad q(c) \leftarrow \mathrm{DL}[C \uplus r;\, D](c)\}$. One can see that $\hat{I}|_\Pi$ is a minimal model of the reduct, hence the check $xFND(\hat{I}, \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle)$ succeeds and the repair $\mathcal{A}'$ is output to the user. □

Let then *RepAnsSet* (Algorithm 5.3) be the algorithm that iteratively calls *RepAns* for every $\hat{I} \in AS(\hat{\Pi})$, and that outputs any $\hat{I}$ where the result of *RepAns* is nonempty, i.e. some repair $\mathcal{A}'$ was computed. We then get:

Theorem 5.5. *RepAns and RepAnsSet are sound and complete for* $rep_{(\sigma,x)}(\Pi)$ *and* $RAS_{(\sigma,x)}(\Pi)$*, respectively, for independent selection* σ*.*

Proof. Soundness. Let $\mathcal{A}'$ be an output of *RepAns*. Towards a contradiction, suppose $\mathcal{A}' \notin rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$. Then $\hat{I}|_\Pi \notin AS(\Pi')$, where $\Pi' = \langle \mathcal{T}, \mathcal{A}', P \rangle$ and $\mathcal{A}'$ is σ-selected. Clearly, $\mathcal{A}'$ is σ-selected, since otherwise $\mathcal{A}' \notin ORP(\hat{I}, \Pi, \sigma)$ and $\mathcal{A}'$ is not in the output. As it holds that $\hat{I} \in AS(\hat{\Pi})$, it must hold that either $\hat{I}$ is not a compatible set of Π' or it is not x-founded. If either of these cases is true, then the corresponding procedure *CMP* or *xFND* returns false and $\mathcal{A}'$ is not in the output, which leads to contradiction.

Completeness. Let $rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$ be the set of all σ-selected repairs for Π that turn $\hat{I}|_\Pi$ into an x-repair answer set. Towards a contradiction, assume that there exists some $\mathcal{A}' \in rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$ which is not an output of the algorithm *RepAns*. Then either (1) $\mathcal{A}' \notin ORP(\hat{I}, \Pi, \sigma)$; (2) $CMP(\hat{I}, \langle \mathcal{T}, \mathcal{A}', P \rangle) = false$ or (3) $xFND(\hat{I}, \langle \mathcal{T}, \mathcal{A}', P \rangle) = false$. If (1) holds, then $\mathcal{A}'$ is not a solution of the ORP problem. Thus either $\langle \mathcal{T}, \mathcal{A}' \rangle$ is unsatisfiable (contradiction to $\mathcal{A}' \in rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$ by the definition of repair) or the actual values of the DL-atoms do not coincide with the replacement atoms in $\hat{\Pi}$ (contradiction due to the failure of the compatibility check). Finally, if either (2) or (3) holds then we obtain a contradiction, since $\mathcal{A}' \in rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$ implies $\hat{I}|_\Pi$ should be compatible and x-founded.

From the argument above and Proposition 5.3 soundness and completeness of the algorithm *RepAnsSet* is established. □

Similarly, *RepAns* could be intertwined with *AnsSet* for an extension that computes answer sets if some exist, and repair answer sets otherwise: while iterating over $\hat{I} \in AS(\hat{\Pi})$ and checking for compatibility and foundedness, also *RepAns* is called for every $\hat{I}$, as long as no answer set is found.

A natural question is whether computing repair answer sets via compatible sets $\hat{I}$ of Π makes repair answer set checking for $\hat{I}|_\Pi$ easier than for arbitrary interpretations I. Unfortunately, this is not the case, thus we obtain a strengthening of the results of Theorems 4.5 and 4.7.

Theorem 5.6. *For ground $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ and $I \subseteq HB_\Pi$, deciding whether $I \in RAS_x(\Pi)$ is* NP*-complete for x=weak and Σ_2^p-complete for x=flp; hardness holds even if $I = \hat{I}|_\Pi$ for a compatible set $\hat{I}$ of Π.*

Proof. In the proofs of Theorems 4.5 and 4.7, both repair candidates I that we considered are such that $I = \hat{I}|_\Pi$ for some compatible set $\hat{I}$ of Π. From this observation the result immediately follows. □

In our algorithm we aim at finding repairs on the fly, i.e. we look at the first answer set, check for its compatibility and in case if it is not compatible try to immediately find a repair before checking the compatibility of the next answer set of $\hat{\Pi}$. We proceed in this way until we find some compatible answer set or until we go through all answer sets of $\hat{\Pi}$. The results of the calls to *RepAns* might be stored, and they could be partially reused during further computation. E.g. if no repair was identified for $\hat{I}$ then no repair will be found for $\hat{I}'$, if all values of DL-atoms and their respective updates in $\hat{I}$ and $\hat{I}'$ coincide. Eventually, all (σ, x)-repair answer sets or just a single best repair option could be output.

A possible alternative approach would be to go through all answer sets of $\hat{\Pi}$, and only if none of them turned out to be compatible, aim at computing repairs for some of them. This approach, however, is less effective in general, since answer set candidates might need to be traversed more then once.

The realization of algorithms for repair answer set computation that we presented in this section is very natural and flexible, as it can be applied to DL-programs over ontologies in various DLs. However, the proposed algorithms turn out to be too naive and do not scale for practical applications due to large number of ABoxes to be checked in general. This calls for the development of suitable optimization techniques and more effective algorithms, to which we devote the rest of this chapter.

Clearly, the possible optimizations often depend on the underlying DL, in which the ontology accessed by the rules of the DL-program, is encoded. Motivated by the complexity results shown in Chapter 4 and the observation that computing repairs for DL-programs over lightweight DLs does not yield any complexity increase compared to the standard answer set computation, we naturally focus on *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ DLs.

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAction}.\mathit{Action} \sqcap \exists \mathit{hasSubject}.\mathit{Staff} \sqcap \exists \mathit{hasTarget}.\mathit{Project} \\ (3)\ \mathit{BlacklistedStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubject}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubject}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \\ (7)\ \mathit{hasTarget}(r1, p1) \quad (8)\ \mathit{hasAction}(r1, read) \quad (9)\ \mathit{Action}(read) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (10)\ \mathit{projfile}(p1); \quad (11)\ \mathit{hasowner}(p1, john); \\ (12)\ \mathit{chief}(Y) \leftarrow \mathit{hasowner}(X, Y), \mathit{projfile}(X); \\ (13)\ \mathit{grant}(X) \leftarrow \mathrm{DL}[\mathit{Project} \uplus \mathit{projfile}; \mathit{StaffRequest}](X), \mathit{not}\ \mathit{deny}(X); \\ (14)\ \mathit{deny}(X) \leftarrow \mathrm{DL}[\mathit{Staff} \uplus \mathit{chief}; \mathit{BlacklistedStaffRequest}](X); \\ (15)\ \bot \leftarrow \mathit{hasowner}(Y, Z), \mathit{not}\ \mathit{grant}(X), \\ \qquad\qquad \mathrm{DL}[; \mathit{hasTarget}](X, Y), \mathrm{DL}[; \mathit{hasSubject}](X, Z). \end{array} \right\}$$

Figure 5.1: DL-program Π over a policy ontology

5.2 General Notion of Support Sets for DL-atoms

In this section we propose the notion of *support sets* for DL-atoms, which serve as optimization means for repair computation. Intuitively, a support set for a DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ is a portion of its input that, together with ABox assertions, is sufficient to conclude that the query $Q(\vec{t})$ will evaluate to true, i.e, that given a subset $I' \subseteq I$ of an interpretation I and a set $\mathcal{A}' \subseteq \mathcal{A}$ of ABox assertions from the ontology, we can conclude that $I \models^{\mathcal{O}} Q(\vec{t})$. The evaluation of d w.r.t. I reduces then to test whether some support set $S = I \cup \mathcal{A}$ exists; to this end, a sufficient collection of such sets S can be precomputed and stored.

For repair as well as standard answer set computation the support sets can be effectively used, as they allow to reduce the number of ontology accesses in answer set checking, and also to prune the candidate space of answer sets effectively. In particular, if a rich enough collection of support sets is available, the ontology access can be *entirely* eliminated. This approach is highly attractive and promises performance gains if a suitable representation of such a *complete set of support sets* is efficiently computable. As we later see the DL-programs over *DL-Lite*$_{\mathcal{A}}$ DL fortunately possess this property. Exploiting partial (i.e. incomplete) support families, however, also provides computational gains; we present algorithms based on partial support families for DL-programs over $\mathcal{EL}$ DL as well.

5.2.1 Ground Support Sets

Before introducing formal details of support sets, let us first present a DL-program over an $\mathcal{EL}$ ontology, which will be referred to as our running example.

Example 5.7. Consider the DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ in Figure 5.1 formalizing an access policy over an ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ [BFS10], which we have already seen in Example 2.8. Besides facts (10), (11) and a simple rule (12), the rule part $\mathcal{P}$ of Π contains defaults (13), (14) expressing that staff members are granted access to project files unless they are blacklisted, and a constraint (15), which forbids that owners of project information lack access to it. Both parts, $\mathcal{P}$ and $\mathcal{O}$, interact via DL-atoms, such as $\mathrm{DL}[\mathit{Project} \uplus \mathit{projfile}; \mathit{StaffRequest}](X)$. The latter specifies that additional assertions $\mathit{Project}(c)$ are considered for each individual c, such that $\mathit{projfile}(c)$ is true in an interpretation of $\mathcal{P}$, before all instances X of *StaffRequest* are retrieved from $\mathcal{O}$. Inconsistency arises as *john*, the chief of project *p1* and owner of its files, has no access to them. □

We define support sets on the ground level as follows.

Definition 5.8 (Ground Support Sets)**.** Let $d(\vec{c}) = \mathrm{DL}[\lambda; Q](\vec{c})$ be a ground DL-atom of a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$. Then a *ground support set* for d is a subset of the Herbrand Base $S = \{p_i(\vec{t}) \in HB_\Pi \mid P_i \circ p_i \in \lambda, \circ \in \{\uplus, \uplus\}\}$, such that for all interpretations $I, I' \supseteq S$, it holds that $I \models^{\mathcal{O}} d$ iff $I' \models^{\mathcal{O}} d$. Moreover, S is positive (resp. negative), if for every interpretation $I \supseteq S$ it holds that $I \models^{\mathcal{O}} d$ (resp. $I \not\models^{\mathcal{O}} d$).

By $\mathcal{S}_{gr}(d)$ we denote the set of all ground support sets for a ground DL-atom d; for any $\mathcal{S} \subseteq \mathcal{S}_{gr}(d)$, we denote by $\mathcal{S}^+$ (resp. $\mathcal{S}^-$) the set of all positive (resp. negative) support sets $S \in \mathcal{S}$ for d.

Example 5.9. Recall Π and $d(\mathit{r1}) = \mathrm{DL}[\mathit{Project} \uplus \mathit{projfile}; \mathit{StaffRequest}](\mathit{r1})$ from Example 5.7. A positive ground support set for $d(\mathit{r1})$ is $S = \{\mathit{projfile}(\mathit{p1})\}$. Indeed, for all interpretations $I \supseteq \{\mathit{projfile}(\mathit{p1})\}$, it holds that $\mathcal{A} \cup \mathcal{T} \cup \lambda^I(d) \models \mathit{StaffRequest}(\mathit{r1})$. □

Example 5.10. Consider Π and a grounding $d(\mathit{john}) = \mathrm{DL}[\mathit{Male} \uplus \mathit{boy}; \mathit{Male}](\mathit{john})$ of the DL-atom from the rule (9) of Example 4.1. A positive ground support set for $d(\mathit{john})$ is $S = \{\mathit{boy}(\mathit{john})\}$, since for all $I \supseteq \mathit{boy}(\mathit{john})$, we have that $\mathcal{A} \cup \mathcal{T} \cup \lambda^I(d) \models \mathit{Male}(\mathit{john})$. □

Support sets are grouped into so-called *support families*.

Definition 5.11. A *support family* is a set of support sets.

An important notion is that of *completeness* for a support family. Informally, a complete support family for a ground DL-atom is a support family that is sufficient for determining the value of the DL-atom under all possible interpretations.

Definition 5.12. A support family $\mathcal{S} \subseteq \mathcal{S}_{gr}(d)$ for a ground DL-atom d is *complete*, if for each interpretation I some $S \in \mathcal{S}$ exists, such that $I \supseteq S$.

Given a complete support family $\mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^-$ for d, we can decide on the value of d w.r.t. an interpretation I by simply checking if some $S \in \mathcal{S}$ "covers" I, i.e., $S \subseteq I$ holds. In fact, to decide this we need just one of $\mathcal{S}^+$ and $\mathcal{S}^-$; for space reasons, storing the

smaller set is naturally preferable. A support family is *+-complete* (resp. *−-complete*) for d, if it equals $\mathcal{S}^+$ (resp. $\mathcal{S}^-$) for some complete support family $\mathcal{S} \subseteq \mathcal{S}_{gr}(d)$.

With a complete support family at hand, ontology accesses during DL-program evaluation and repair can be fully avoided, which promises significant performance improvements. This beneficial property is now formally stated.

Proposition 5.13. *Let d be a ground DL-atom, let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, and let I be an interpretation. Then, $I \models^{\mathcal{O}} d$ iff some $S \in \mathcal{S}^+$ exists, such that $S \subseteq I$, where $\mathcal{S}^+$ is a +-complete support family for d.*

Proof. ($\Rightarrow$) Towards a contradiction, suppose that $I \models^{\mathcal{O}} d$, but no $S \in \mathcal{S}^+$ exists, such that $S \subseteq I$. As $\mathcal{S}^+$ is a +-complete support family for d by our assumption, it must be the case that for all $S \subseteq I$, we have that $S \in \mathcal{S}^-$, where $\mathcal{S}^- = \mathcal{S} \setminus \mathcal{S}^+$ is some −-complete support family for d. Since every support set in −-complete family is negative, it must hold that $I \not\models^{\mathcal{O}} d$, leading to a contradiction.

($\Leftarrow$) Suppose to the contrary that some $S \subseteq I$ exists, such that $S \in \mathcal{S}^+$, but $I \not\models^{\mathcal{O}} d$. Since $\mathcal{S}^+$ is +-complete, it must hold that all support sets in $\mathcal{S}^+$ are positive, hence S is positive. Therefore, by definition, for all $I' \supseteq S$ it is true that $I' \models^{\mathcal{O}} d$, and in particular, $I \models^{\mathcal{O}} d$, which leads to a contradiction. □

To simplify presentation, in this work we exploit only positive support sets, i.e. portions of the ontology input, that ensure that the DL-atom will be true, thus +-completeness is of a particular interest for us here. In what follows when we refer to completeness, we mean +-completeness.

Example 5.14. Let us consider $d(john) = \mathrm{DL}[Male \uplus boy; Male](john)$ and the modified version of ontology from Example 4.1.

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ Child \sqsubseteq \exists hasParent & (4)\ Female(alex) \\ (2)\ Adopted \sqsubseteq Child & (5)\ \neg Male(pat) \\ (3)\ Female \sqsubseteq \neg Male & (6)\ hasParent(john, pat) \end{array} \right\}$$

We have that the family $\mathcal{S} = \{S_1, S_2, S_3\}$, where $S_1 = \{boy(john)\}$, $S_2 = \{boy(alex)\}$, $S_3 = \{boy(pat)\}$ is complete for d. Indeed, $d(john)$ evaluates to true only for interpretations I, such that $I \supseteq S_i$ for some $1 \leq i \leq 3$. □

5.2.2 Nonground Support Sets

Intuitively, support sets reflect the relevant part of an external source (ontology in our case). Thus different ground support sets can be similar with respect to their structure. With this motivation in mind we lift support sets to the nonground level. Given a set of nonground atoms, in general, one can not decide on the value of the DL-atom without any knowledge about the ontology that it accesses. Thus nonground support sets are useful only if they work on a conditional basis and take ontology information into account. In our framework this is served by so-called *conditional guards* (γ), which allow to elegantly specify the support sets on a nonground level as follows:

Definition 5.15 (Nonground Support Sets)**.** Let Π be a DL-program and let $d(\vec{X}) = \mathrm{DL}[\lambda; Q](\vec{X})$ be a DL-atom of Π. A positive *nonground support set* S for $d(\vec{X})$ is a pair $\langle N, \gamma \rangle$, where

- $N \subseteq \{p_i(\vec{Y}) \mid P_i \circ p_i \in \lambda, \circ \in \{\uplus, \Cup\}\}$ is a set of nonground atoms over the input signature λ of d;
- $\gamma : \mathcal{C}^{|\vec{X}|} \times \mathit{grnd}_{\mathcal{C}}(N) \to \{0, 1\}$ is a Boolean function (called the guard), such that for all $\vec{c} \in \mathcal{C}^{|\vec{X}|}$ and $N_{gr} \in \mathit{grnd}_{\mathcal{C}}(N)$ it holds that $\gamma(\vec{c}, N_{gr}) = 1$ only if N_{gr} is a ground support set for $d(\vec{c})$.

By $\mathcal{S}_{ngr}(d)$ we denote the set of all nonground support sets for d.

In this definition, $\mathit{grnd}_{\mathcal{C}}(N)$ is the support family, i.e. set of support sets constructed from N by replacing all variables with constants from $\mathcal{C}$ in all possible ways. Intuitively, the guard γ is an abstract function that checks a condition under which the ground atoms over predicates in N form a ground support set. Definition 5.15 is generic, and it allows to exploit support sets for answer set programs with arbitrary external sources, as it is shown in [EFRS14].

Example 5.16. The DL-atom $d(X) = \mathrm{DL}[\mathit{Project} \uplus \mathit{projfile}; \mathit{StaffRequest}](X)$ has $S_1 = \langle \emptyset, \gamma' \rangle$ as a support set, where $\gamma' : \mathcal{C} \times \emptyset \to \{0, 1\}$ is such that $\gamma'(c, \emptyset) = 1$ only if $\mathit{StaffRequest}(c) \in \mathcal{A}$.

Another support set is $S_2 = \langle \mathit{projfile}(Y), \gamma \rangle$, where $\gamma : \mathcal{C} \times \mathit{grnd}_{\mathcal{C}}(\mathit{projfile}(Y)) \to \{0, 1\}$ is such that $\gamma(c, \mathit{projfile}(c')) = 1$ only if the ABox $\mathcal{A}$ contains the assertions $\mathit{hasAction}(c, c_1)$, $\mathit{Action}(c_1)$, $\mathit{hasSubject}(c, c_2)$, $\mathit{Staff}(c_2)$, and $\mathit{hasTarget}(c, c')$, where c_1, c_2 are arbitrary constants from $\mathcal{C}$. □

Example 5.17. The DL-atom $d(X) = \mathrm{DL}[\mathit{Male} \uplus \mathit{boy}; \mathit{Male}](X)$ from Example 5.10 has a nonground support set $S_1 = \langle \mathit{boy}(Y), \gamma \rangle$, where $\gamma : \mathcal{C} \times \mathit{grnd}_{\mathcal{C}}(\mathit{boy}(Y)) \to \{0, 1\}$ is such that $\gamma(c, \mathit{boy}(c')) = 1$ only if the ABox $\mathcal{A}$ contains the assertion $\neg \mathit{Male}(c')$ or $\mathit{Female}(c')$. The set $S_2 = \langle \mathit{boy}(X), \top \rangle$ is also a nonground support set for $d(X)$, where the guard $\top$ returns 1 for each grounding of $\mathit{boy}(X)$; hence $\{\mathit{boy}(c)\}$ is a ground support set of $d(c)$, for all $c \in \mathcal{C}$. □

The notion of completeness is now generalized for nonground support sets.

Definition 5.18. A family $\mathcal{S} \subseteq \mathcal{S}_{ngr}(d)$ of nonground support sets is said to be *complete* for a (non-ground) DL-atom $d(\vec{X})$, if for every $\vec{c} \in \mathcal{C}^{|\vec{X}|}$ and ground support set $S \in \mathcal{S}_{gr}(d(\vec{c}))$ of $d(\vec{c})$, some $S' = \langle N, \gamma \rangle$ exists in $\mathcal{S}$, such that $S \in \mathit{grnd}_{\mathcal{C}}(N)$ and $\gamma(\vec{c}, S) = 1$.

Example 5.19. Consider a nonground support family $\mathcal{S} = \{S_1, S_2, S_3\}$ for $d(X) = \mathrm{DL}[\mathit{Male} \uplus \mathit{boy}; \mathit{Male}](X)$, with S_1 and S_2 as in Example 5.17, and $S_3 = \langle \emptyset, \gamma' \rangle$, where $\gamma' : \mathcal{C} \times \{\emptyset\} \to \{0, 1\}$ is such that $\gamma'(c, \emptyset) = 1$ only if the ABox $\mathcal{A}$ contains the assertion $\mathit{Male}(c)$. It is not difficult to check that $\mathcal{S} \subseteq \mathcal{S}_{ngr}(d)$ is a complete nonground support family for $d(X)$. □

Nonground support sets are compact representations of ground ones. Given a complete nonground support family for a DL-atom $d(\vec{X})$ and an interpretation I, one can decide on the value of some grounding $d(\vec{c})$ of $d(\vec{X})$ under I by applying syntactic matching. We now state this formally.

Proposition 5.20. *Let $\mathbf{S} \subseteq \mathcal{S}_{ngr}(d)$ be a complete nonground support family for a DL-atom $d(\vec{X})$. Then for every $\vec{c} \subseteq \mathcal{C}^{|\vec{X}|}$, and every interpretation I we have that $I \models^{\mathcal{O}} d(\vec{c})$ iff there exists some $S \subseteq I$ and some $S' \in \mathbf{S}$, such that $S \in grnd_{\mathcal{C}}(S')$, and $\gamma(\vec{c}, S) = 1$.*

Proof. ($\Rightarrow$) Towards a contradiction, suppose that there exists some $\vec{c} \in \mathcal{C}^{|\vec{X}|}$ and some interpretation I, such that $I \models^{\mathcal{O}} d(\vec{c})$, but for all $S \subseteq I$ and all $S_i \in \mathbf{S}$ it holds that either $S \notin grnd_{\mathcal{C}}(S_i)$ or $\gamma(\vec{c}, S) = 0$. Consider a set of all ground support sets $\mathcal{S}_{gr}(d(\vec{c}))$ for $d(\vec{c})$. Since $I \models^{\mathcal{O}} d(\vec{c})$, it must hold that some $S' \subseteq I$ exists, such that $S' \in \mathcal{S}_{gr}(d(\vec{c}))$. However, then by Definition 5.18, it holds that for $\vec{c}$ and N_{gr} some nonground support set $S_k = \langle N, \gamma \rangle$ in $\mathbf{S}$ exists, such that $S' \in grnd_{\mathcal{C}}(S_k)$ and $\gamma(\vec{c}, S') = 1$, which leads to a contradiction.

($\Leftarrow$) Consider some $\vec{c}$ and some interpretation I. To the contrary, assume that for some $N_{gr} \subseteq I$ and $S_i = \langle N, \gamma \rangle$ in $\mathbf{S}$, we have that $N_{gr} \in grnd_{\mathcal{C}}(S_i)$ and $\gamma(\vec{c}, N_{gr}) = 1$, but $I \not\models^{\mathcal{O}} d(\vec{c})$. Since S_i is a nonground support set and $\gamma(\vec{c}, N_{gr}) = 1$, by Definition 5.15 we have that N_{gr} is a ground support set for $d(\vec{c})$; but then since $N_{gr} \subseteq I$ by our assumption, it must hold that $I \models^{\mathcal{O}} d(\vec{c})$, i.e. contradiction. □

The abstract definition of nonground support sets leaves room for flexible realization of the conditional guard γ. A natural one is by (unions of) conjunctive queries (UCQs) over the ontology ABox viewed as a database. In Example 5.16, the guard γ of S_1 takes as input a constant $c \in \mathcal{C}$ and a ground instance of form $projfile(c')$, and returns 1 if the Boolean CQ $q(c) \leftarrow \exists X, X'\ \phi(X, X')$ evaluates to true, where $\phi(X, X') = hasAction(c, X) \wedge Action(X) \wedge hasSubject(c, X') \wedge Staff(X') \wedge hasTarget(c, c')$.
The UCQ $q(c) \leftarrow \exists X, X'\ \ \phi(X, X') \vee \psi(X, X')$, where $\psi(X, X') = hasAction(c,X) \wedge Action(X) \wedge hasSubject(c, X') \wedge Blacklisted(X') \wedge hasTarget(c, c')$, is more general; even more general guards are possible (e.g. nonrecursive datalog programs).

Further we provide methods for constructing support sets for DL-programs over *DL-Lite*$_{\mathcal{A}}$ and $\mathcal{EL}$ ontologies that allow us to just work with ontology predicates. To simplify matters, we first show how the lp-input of DL-atoms can be shifted to the ontology.

5.2.3 Shifting Lemma

Let us define input assertions for a DL-atom as follows.

Definition 5.21. Given a DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ and $P \circ p \in \lambda$, $\circ \in \{\uplus, \sqcup\}$, we call $P_p(c)$ an *input assertion for* d, where P_p is a fresh ontology predicate and $c \in \mathcal{C}$. By $\mathcal{A}_d$ we denote the set of all such assertions.

For a TBox $\mathcal{T}$ and a DL-atom d, we let $\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\} \cup \{P_p \sqsubseteq \neg P \mid P \uplus p \in \lambda\}$, and for an interpretation I, we let $\mathcal{O}_d^I = \mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\}$. We then have:

Proposition 5.22. *For every $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ and interpretation I, it holds that $I \models^{\mathcal{O}} d$ iff $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon; Q](\vec{t})$ iff $\mathcal{O}_d^I \models Q(\vec{t})$.*

Proof. We prove each "if" direction of the statement separately.

- We first show that if $I \models^{\mathcal{O}} d$ then $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon; Q](\vec{t})$. Let $I \models^{\mathcal{O}} d$. That means that $\mathcal{O} \cup \lambda^I(d) \models Q(\vec{t})$. By definition, we have that $\lambda^I(d) = \{P(\vec{t}) \,|\, p(\vec{t}) \in I \text{ and } P \uplus p \in \lambda\} \cup \{\neg P(\vec{t}) \,|\, p(\vec{t}) \in I \text{ and } P \uplus p \in \lambda\}$. Therefore, $\mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\} \cup \{P_p \sqsubseteq \neg P \mid P \uplus p \in \lambda\} \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\} \models \lambda^I(d)$, i.e. $\mathcal{O}_d^I \models \lambda^I(d)$, and hence $\mathcal{O}_d^I \models Q(\vec{t})$. Therefore, we get that $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon; Q](\vec{t})$.

- We now prove the opposite direction, i.e. if $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon; Q](\vec{t})$ then $I \models^{\mathcal{O}} d$, where $d = \mathrm{DL}[\lambda; Q](\vec{t})$. Let $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon; Q](\vec{t})$. Then we have that $\mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\} \models Q(\vec{t})$. By construction of $\mathcal{O}_d^I$, λ must be as follows: $P \uplus p \in \lambda$ iff $P_p \sqsubseteq P \in \mathcal{T}_d$; $P \uplus p \in \lambda$ iff $P_p \sqsubseteq \neg P \in \mathcal{T}_d$. Therefore, for all $P' \in \mathsf{sig}(\mathcal{A} \cap (\mathcal{A}_d \backslash \mathcal{A}))$, we have that if $\mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \mid p(\vec{t}) \in I\} \models P'(\vec{t'})$ then $\mathcal{T} \cup \mathcal{A} \cup \lambda^I(d) \models P'(\vec{t'})$. As $Q \notin \mathsf{sig}(\mathcal{A}_d \backslash \mathcal{A})$, we obtain that $\mathcal{O} \cup \lambda^I(d) \models Q(\vec{t})$, and hence $I \models^{\mathcal{O}} d$.

- The last implications, i.e. $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon; Q](\vec{t})$ iff $I \models^{\mathcal{O}_d^I} Q(\vec{t})$ are immediate from the definition of a DL-atom's satisfaction by an interpretation.

□

Unlike $\mathcal{O} \cup \lambda^I(d)$, in $\mathcal{O}_d^I$ there is a clear distinction between native assertions and input assertions for d w.r.t. I (via facts P_p and axioms $P_p \sqsubseteq P$), mirroring its lp-input. The shifting lemma allows us to work only with ontology predicates when constructing support sets for DL-atoms, as we see next.

5.3 Support Sets for DL-atoms over *DL-Lite*$_{\mathcal{A}}$ Ontologies

We now discuss exact forms of support sets and their computation for ontologies in *DL-Lite*$_{\mathcal{A}}$. In view of the property that in *DL-Lite*$_{\mathcal{A}}$ a single assertion is sufficient to derive a query from a consistent ontology [CLLR07], we obtain that ground support sets can be at most of size 2. Therefore, given a DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ over a *DL-Lite*$_{\mathcal{A}}$ ontology, each of its support sets has one of three predetermined forms, which we now formalize.

Proposition 5.23. *Let $d = \mathrm{DL}[\lambda; Q](\vec{t})$ be a ground DL-atom over a consistent DL-Lite$_{\mathcal{A}}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, and let $\mathcal{S} \subseteq \mathcal{S}_{gr}(d)$ be a complete ground support family for d. Then for each $S \in \mathcal{S}$ one of the following must hold:*

(1) $S = \emptyset$, and some $P(\vec{c}) \in \mathcal{A}$ exists, s.t. $P(\vec{c}) \cup \mathcal{T} \models Q(\vec{t})$;

(2) $S \supseteq \{p(\vec{c})\}$*, such that* $P_p(\vec{c}) \cup \mathcal{T}_d$ *is consistent and* $P_p(\vec{c}) \cup \mathcal{T}_d \models Q(\vec{t})$*;*

(3) $S \supseteq \{p(\vec{c})\}$*, and some* $P'(\vec{d}) \in \mathcal{A}$ *exists s.t.* $P_p(\vec{c}) \cup P'(\vec{d}) \cup \mathcal{T}_d$ *is inconsistent;*

(4) $S \supseteq \{p(\vec{c}), p'(\vec{d})\}$*, such that* $P_p(\vec{c}) \cup P'_{p'}(\vec{d}) \cup \mathcal{T}_d$ *is inconsistent.*

Proof. As at most one assertion α is needed to derive an instance query from a consistent $\mathcal{O}$, we get that if $\alpha \in \mathcal{A}$, then the support set encoding this knowledge is empty. The support set is of form (1) if α is in the update. At most two ABox assertions are needed to make a *DL-Lite*$_\mathcal{A}$ ontology inconsistent. Given that $\mathcal{O}$ is originally consistent, we get support sets of forms (2) and (3). □

The shifting lemma allows one to rewrite an lp-input of a DL-atom to ontology assertions. Therefore, every support set for a DL-atom d containing ground atoms over the logic program signature can be rewritten to a set of ontology assertions from $\mathcal{A}_d$. Viewing such rewriting as a normalization step one can express support set information conveniently and succinctly using only ontology predicates. Exploiting Proposition 5.23 we define such *normalized* support sets for DL-atoms accessing a *DL-Lite*$_\mathcal{A}$ ontology as follows:

Definition 5.24 (Normalized Ground *DL-Lite*$_\mathcal{A}$ Support Sets)**.** Given a ground DL-atom $d = DL[\lambda; Q](\vec{t})$, a set S of assertions from $\mathcal{A} \cup \mathcal{A}_d$ is a *normalized ground DL-Lite*$_\mathcal{A}$ *support set* for d w.r.t. a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, if either

- $S = \{P(\vec{c})\}$ and $\mathcal{T}_d \cup S \models Q(\vec{t})$, or
- $S = \{P(\vec{c}), P'(\vec{c'})\}$ such that $\mathcal{T}_d \cup S$ is inconsistent.

By $Supp_\mathcal{O}^{DL\text{-}Lite_\mathcal{A}}(d)$[1] we denote a family of all normalized ground support sets for d.

Let us illustrate the normalization of support sets by an example.

Example 5.25. Consider the DL-atom $d(john) = \mathrm{DL}[Male \uplus, boy; Male](john)$ and its support sets from Example 5.14. The support set $S = \{boy(john)\}$ corresponds to the normalized support set $\{Male_{boy}(john)\}$. Similarly the support sets S_2 and S_3 correspond to the normalized support sets $\{Male_{boy}(pat)\}$, $\{Male_{boy}(alex)\}$ respectively. □

Apart from the maximal number of assertions that participate in support sets for DL-atoms accessing *DL-Lite*$_\mathcal{A}$ ontologies, there is also a limit on the number of constants that can occur in such support sets. In fact, in Definition 5.24 $\vec{c} \cup \vec{c'}$ can involve at most 3 constants, which we now formally show.

Proposition 5.26. *Let* $d = DL[\lambda; Q](\vec{t})$ *be a ground DL-atom, and let* S *be its ground normalized support set w.r.t. a DL-Lite*$_\mathcal{A}$ *ontology* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$*. Then* S *involves at most 3 constants.*

[1] We omit the superscript *DL-Lite*$_\mathcal{A}$ in $Supp_\mathcal{O}^{DL\text{-}Lite_\mathcal{A}}(d)$ when it is clear from the context that $\mathcal{O}$ is a *DL-Lite*$_\mathcal{A}$ ontology.

Proof. By Definition 5.24 there are two possibilities: either (i) S is unary or (ii) it is binary. In case of (i) we have that S is either a concept or a role assertion, and therefore, it involves at most two constants. For the case (ii) it holds that $S \cup \mathcal{T}_d$ is inconsistent. Hence S represents a binary conflict set. It has been shown in [LLR+11] that S can be a binary conflict set only due to one of the following reasons:

- $\mathcal{T} \models C \sqsubseteq \neg D$, and $S = \{C(a), D(a)\}$, in which case S involves only 1 constant;
- $\mathcal{T} \models R \sqsubseteq \neg R'$, and $S = \{R(a,b), R'(a,b)\}$, thus S involves at most 2 constants;
- $\mathcal{T} \models C \sqsubseteq \neg\exists R$ or $C \sqsubseteq \neg\exists R^-$, and $S = \{C(a), R(a,b)\}$ resp. $S = \{C(a), R(b,a)\}$, i.e. S involves at most 2 constants;
- $\mathcal{T} \models \exists R \sqsubseteq \neg C$ or $\exists R^- \sqsubseteq \neg C$, and $S = \{R(a,b), C(a)\}$ resp. $S = \{R(b,a), C(a)\}$, in which case S involves 2 constants maximum;
- $\mathcal{T} \models \exists R \sqsubseteq \neg\exists R'$, and $S = \{R(a,b), R'(a,c)\}$, thus there are 3 constants occurring in S (similarly for the cases $\mathcal{T} \models \exists R^- \sqsubseteq \neg\exists R'^-$, $\mathcal{T} \models \exists R \sqsubseteq \neg\exists R'^-$, $\mathcal{T} \models \exists R^- \sqsubseteq \neg\exists R'^-$);
- $funct(R) \in \mathcal{T}$, and $S = \{R(a,b), R(a,c)\}$ with $b \neq c$, in which case again at most 3 constants appear in S (the case when $funct(R^-) \in \mathcal{T}$ is analogous).

As we have considered all possibilities for binary conflict sets, the statement is proved. □

Normalized support sets are linked to interpretations by the following notion.

Definition 5.27. A normalized support set S of a DL-atom d is *coherent* with an interpretation I, if for each $P_p(\vec{c}) \in S$ it holds that $p(\vec{c}) \in I$.

The evaluation of d w.r.t. I reduces to the search for coherent normalized support sets.

Proposition 5.28. *Let d be a ground DL-atom, let $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ be a DL-Lite$_\mathcal{A}$ ontology, and let I be an interpretation. Then, $I \models^\mathcal{O} d$ iff some $S \in Supp_\mathcal{O}(d)$ exists such that S is coherent with I.*

Proof. ($\Rightarrow$) Suppose $d = DL[\lambda; Q](\vec{t})$ evaluates w.r.t. $\mathcal{O}$ and I to true, i.e., $\lambda^I(d) \cup \mathcal{O} \models Q(\vec{t})$. Towards a contradiction, assume no $S \in Supp_\mathcal{O}(d)$ is coherent with I. There are two cases:

(1) $\lambda^I(d) \cup \mathcal{O}$ is consistent. Proposition 2.7 implies that an assertion $\alpha \in \lambda^I(d) \cup \mathcal{A}$ must exist such that $\mathcal{T} \cup \{\alpha\} \models Q(t)$. If $\alpha \in \mathcal{A}$ then $Supp_\mathcal{O}(d)$ contains $\{\alpha\}$ by (i) of Definition 5.8, which trivially is coherent with I and thus contradicts the assumption. If $\alpha \in \lambda^I(d)$, then α is an input assertion for d. For $\alpha_d \in \mathcal{A}_d$, we then obtain that $\{\alpha_d\} \in Supp_\mathcal{O}(d)$ according to (i) of Definition 5.8, again a contradiction due to coherence with I.

(2) $\lambda^I(d) \cup \mathcal{O}$ is inconsistent. From Proposition 2.7 and consistency of $\mathcal{O}$, it follows that some $\delta \in \lambda^I(d)$ exists such that either (a) $\mathcal{T} \cup \{\delta\}$ is inconsistent, or (b) some $\gamma \in \mathcal{A} \cup \lambda^I(d)$ exists such that $\mathcal{T} \cup \{\delta, \gamma\}$ is inconsistent. In case a), we obtain $\{\delta_d\} \in Supp_{\mathcal{O}}(d)$, for the corresponding input assertion $\delta_d \in \mathcal{A}_d$. by (i) of Definition 5.8; this is a contradiction, as $\{\delta_d\}$ is coherent with I. In case b), we similarly conclude that either $\{\delta_d, \gamma\} \in Supp_{\mathcal{O}}(d)$ or $\{\delta_d, \gamma_d\} \in Supp_{\mathcal{O}}(d)$, depending on whether $\gamma \in \lambda^I(d)$, according to (ii) of Definition 5.8. Again this is a support set coherent with I, contradiction.

($\Leftarrow$) Suppose some $S \in Supp_{\mathcal{O}}(d)$ is coherent with I. Assume towards a contradiction that $I \not\models^{\mathcal{O}} d$. Again we consider two cases:

(1) $\mathcal{T}_d \cup S$ is consistent. Then, $\mathcal{T}_d \cup S \models Q(\vec{t})$ by item (i) of Definition 5.8. Since S is coherent with I, we conclude that $\mathcal{O}_d^I \models Q(\vec{t})$ which implies $I \models^{\mathcal{O}} d$ by Proposition 5.22. Contradiction.

(2) $\mathcal{T}_d \cup S$ is inconsistent. Then, due to coherence with I, so is $\mathcal{O}_d^I$, and trivially $\mathcal{O}_d^I \models Q(\vec{t})$; again we arrive at a contradiction by concluding that $I \models^{\mathcal{O}} d$ from Proposition 5.22. □

As a simple consequence, we get:

Corollary 5.29. *Given a ground DL-atom d and a DL-Lite$_{\mathcal{A}}$ ontology $\mathcal{O}$, there exists an interpretation I such that $I \models^{\mathcal{O}} d$ iff $Supp_{\mathcal{O}}(d) \neq \emptyset$.*

We now present the normalized nonground support sets for *DL-Lite*$_{\mathcal{A}}$ ontologies.

Definition 5.30 (Normalized Nonground *DL-Lite*$_{\mathcal{A}}$ Support Sets)**.** Let $\mathcal{T}$ be a *DL-Lite*$_{\mathcal{A}}$ TBox, and let $d(\vec{X}) = \mathrm{DL}[\lambda; Q](\vec{X})$ be a DL-atom. Suppose that $V = \{X, Y, Z\}$ is a set of distinct variables, such that $\vec{X} \subseteq V$, and $\mathcal{C} = \{a, b, c\}$ is a set of constants. A *normalized nonground DL-Lite*$_{\mathcal{A}}$ *support set* for d w.r.t. $\mathcal{T}$ is a set $S = \{P(\vec{Y})\}$ resp. $S = \{P(\vec{Y}), P'(\vec{Y'})\}$ such that

(i) $\vec{Y}, \vec{Y'} \subseteq V$ and

(ii) for each substitution $\theta : V \to \mathcal{C}$, the instance $S\theta = \{P(\vec{Y}\theta)\}$ (resp. $S\theta = \{P(\vec{Y}\theta), P'(\vec{Y'}\theta)\}$) is a support set for $d(\vec{X}\theta)$ w.r.t. $\mathcal{O}_{\mathcal{C}} = \mathcal{T} \cup \mathcal{A}_{\mathcal{C}}$, where $\mathcal{A}_{\mathcal{C}}$ is the set of all possible assertions over $\mathcal{C}$.

By $\mathbf{Supp}_{\mathcal{O}}^{DL\text{-}Lite_{\mathcal{A}}}(d)$[2] we denote the family of all normalized nonground *DL-Lite*$_{\mathcal{A}}$ support sets for d.

Here $\mathcal{A}_{\mathcal{C}}$ takes care of any possible ABox, by considering the maximal ABox (as $\mathcal{O} \subseteq \mathcal{O}'$ implies $Supp_{\mathcal{O}}(d) \subseteq Supp_{\mathcal{O}'}(d)$); three variables suffice as at most three different constants are involved.

Example 5.31. The support set S_1 for a DL-atom $d(X) = \mathrm{DL}[Male \uplus boy; Male](X)$ from Example 5.19 corresponds to normalized support sets $\{Male_{boy}(Y), \neg Male(Y)\}$ and $\{Male_{boy}(Y), Female(Y)\}$, while support sets S_2 and S_3 correspond to $\{Male_{boy}(X)\}$ and $\{Male(X)\}$ respectively. □

[2]Similarly as for normalized ground support families the superscript *DL-Lite*$_{\mathcal{A}}$ may be omitted.

Normalized nonground support sets S for *DL-Lite*$_\mathcal{A}$ that we presented are sound, i.e. each instance $S\theta$ that matches with $\mathcal{A} \cup \mathcal{A}_d$ is a support set of the ground DL-atom $d\theta$ w.r.t. $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$. They are also complete, i.e., every normalized support set S of a ground DL-atom d w.r.t. $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ results as such an instance, and thus can be determined by syntactic matching.

Clearly, normalized support sets as defined above may be subsumed by other support sets; e.g., $S = \{A(X), R(X,Y)\}$ is subsumed by $\{A(X)\}$. To keep the support set information succinct, removal of such S is desired.

Definition 5.32. A family $\mathbf{S} \subseteq \mathbf{Supp}_\mathcal{O}(d)$ of normalized nonground support sets for a (non-ground) DL-atom $d(\vec{X})$ w.r.t. a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}$ is *complete*, if for every θ: $\vec{X} \to \mathcal{C}$ and $S \in \mathit{Supp}_\mathcal{O}(d(\vec{X}\theta))$, some $S' \in \mathbf{S}$ exists such that $S = S'\theta'$, for some extension $\theta' : V \to \mathcal{C}$ of θ to V, where $V = \{X, Y, Z\}$ is a set of distinct variables, such that $\vec{X} \subseteq V$.

Example 5.33. Reconsider the nonground support family $\mathcal{S} = \{S_1, S_2, S_3\}$ from Example 5.19 for the DL-atom DL[*Male* $\uplus$ *boy*; *Male*](X). The support set S_1 corresponds to the normalized support sets $\{\mathit{Male}_{boy}(Y), \neg \mathit{Male}(Y)\}$ and $\{\mathit{Male}_{boy}(Y), \mathit{Female}(Y)\}$, S_2 to $\{\mathit{Male}_{boy}(X)\}$, and S_3 to $\{\mathit{Male}(X)\}$.

5.3.1 Determining Normalized Nonground Support Sets

Our technique for computing the normalized[3] nonground support sets for DL-atoms over *DL-Lite*$_\mathcal{A}$ ontologies is based on TBox classification, which is one of the main ontology reasoning tasks. This reasoning service computes complete information about the TBox constraints specified at the conceptual level.

More formally, given a TBox $\mathcal{T}$ over a signature Σ_o, the TBox classification $\mathit{Clf}(\mathcal{T})$ determines all subsumption relations $P \sqsubseteq (\neg)P'$ between concepts and roles P, P' in Σ_o that are entailed by $\mathcal{T}$. This can be exploited for our goal to compute nonground support sets, more precisely a complete family $\mathbf{S}$ of such sets.

TBox classification is well studied in Description Logics [BCM$^+$07]. For example, [KKS13] discusses it for $\mathcal{EL}$ w.r.t. concept hierarchy, and [LSS13a] studies it for the OWL 2 QL profile. Respective algorithms are thus suitable and also easily adapted for the computation of (a complete family of) nonground support sets for a DL-atom $d(\vec{X})$ w.r.t. $\mathcal{O}$. In principle, one can exploit Proposition 5.22 and resort to $\mathcal{T}_d$, i.e., compute the classification $\mathit{Clf}(\mathcal{T}_d)$, and determine nonground support sets of $d(\vec{X})$ proceeding similar as for computing minimal conflict sets [RRGM12]. To determine inconsistent support sets, perfect rewriting [CLLR07] can be done over $\mathit{Pos}(\mathcal{T})$, i.e., the TBox obtained from $\mathcal{T}$ by substituting all negated concepts (roles) $\neg C$ ($\neg R$, $\neg\exists R$, $\neg\exists R^-$) with positive replacements $\overline{C}$ ($\overline{R}$, $\overline{\exists R}$, $\overline{\exists R^-}$).

In practice (and as in our implementation), it can nonetheless be worthwhile to compute $\mathit{Clf}(\mathcal{T})$ first, as it is reusable for all DL-atoms. The additional axioms in $\mathcal{T}_d$,

[3] In further exposition of our results in this section we deal only with normalized support sets, and the word "normalized" is often omitted.

Algorithm 5.4: *SupRAnsSet*: all deletion repair answer sets

Input: $\Pi{=}\langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$
Output: $flpRAS(\Pi)$
(a) compute a complete set $\mathbf{S}$ of nongr. supp. sets for the DL-atoms in Π
(b) **for** $\hat{I} \in AS(\hat{\Pi})$ **do**
  $D_p \leftarrow \{d \mid e_d \in \hat{I}\}$; $D_n \in \{d \mid ne_d \in \hat{I}\}$; $\mathbf{S}^{\hat{I}}_{gr} \leftarrow Gr(\mathbf{S}, \hat{I}, \mathcal{A})$;
(c) **if** $\mathbf{S}^{\hat{I}}_{gr}(d) \neq \emptyset$ *for* $d \in D_p$ *and every* $S \in \mathbf{S}^{\hat{I}}_{gr}(d)$ *for* $d \in D_n$ *fulfills* $S \cap \mathcal{A} \neq \emptyset$ **then**
(d) **for** *all* $d \in D_p$ **do**
(e) **if** *some* $S \in \mathbf{S}^{\hat{I}}_{gr}(d)$ *exists s.t.* $S \cap \mathcal{A} = \emptyset$ **then** pick next d
else remove each S from $\mathbf{S}^{\hat{I}}_{gr}(d)$ s.t. $S \cap \mathcal{A} \cap \bigcup_{d' \in D_n} \mathbf{S}^{\hat{I}}_{gr}(d') \neq \emptyset$
(f) **if** $\mathbf{S}^{\hat{I}}_{gr}(d) = \emptyset$ **then** pick next $\hat{I}$
end
(g) $\mathcal{A}' \leftarrow \mathcal{A} \setminus \bigcup_{d' \in D_n} \mathbf{S}^{\hat{I}}_{gr}(d')$;
(h) **if** $flpFND(\hat{I}, \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle)$ **then** output $\hat{I}|_\Pi$
end
end

i.e., those of form $P_p \sqsubseteq (\neg)P$ (according to the update operators), are handled when determining the nonground support sets for a particular DL-atom from $Clf(\mathcal{T})$.

Example 5.34. Consider the DL-atom $d = \mathrm{DL}[Male \uplus boy; Male](X)$ from Example 5.17. For computing a complete family $\mathbf{S}$ of nonground support sets for d w.r.t. $\mathcal{O}$, we may refer to $\mathcal{T}_d = \mathcal{T} \cup \{Male_{boy} \sqsubseteq Male\}$ and its classification $Clf(\mathcal{T}_d)$. Hence, $S_1 = \{Male(X)\}$ and $S_2 = \{Male_{boy}(X)\}$ are the only unary nonground support sets of d. Further nonground support sets are obtained by computing minimal conflict sets, yielding $\{P(\vec{Y}), \neg P(\vec{Y})\}$ for each $P \in \mathbf{C} \cup \mathbf{R}$, as well as $\{Male_{boy}(Y), \neg Male(Y)\}$, $\{Male(Y), Female(Y)\}$, and $\{Male_{boy}(Y), Female(Y)\}$. However, since we are interested in completeness w.r.t. $\mathcal{O}$ and $\mathcal{O}$ is consistent, pairs not involving input assertions can be dropped (as they will not have a match in $\mathcal{A}$). We call the remaining two sets S_3 and S_4 respectively. Hence, $\mathbf{S} = \{S_1, S_2, S_3, S_4\}$ is a complete support family for d w.r.t. $\mathcal{O}$. □

5.4 Repair Computation Based on Complete Support Families

Complete support families can be fruitfully exploited for the DL-atom evaluation only if the support sets are small and easily computable. Luckily for *DL-Lite*$_\mathcal{A}$ as we have shown the support sets are of size at most 2, and their computation can be done efficiently.

Using support sets, we can completely eliminate the ontology access for the evaluation of DL-atoms. In a naive approach, one precomputes all support sets for all ground DL-atoms with respect to relevant ABoxes, and then uses them during the repair answer set

computation. This does not scale in practice, since support sets may be computed that are incoherent with all candidate repair answer sets.

An alternative is to fully interleave the support set computation with the search for repair answer sets. Here we construct coherent ground support sets for each DL-atom and interpretation on the fly. As the input to a DL-atom may change in different interpretations, its support sets must be recomputed, however, since reuse may not be possible; effective optimizations are not immediate.

A better solution, and the one that we exploit in this work, is to precompute support sets on a nonground level, that is, schematic support sets, prior to repair computation. Furthermore, in that we may leave the concrete ABox open; the support sets for a DL-atom instance are then easily obtained by syntactic matching.

We are now ready to describe our optimized algorithm *SupRAnsSet* (see Algorithm 5.4), which avoids multiple interface calls and merely needs to access the ontology once. Given a (ground) DL-program Π for input, *SupRAnsSet* proceeds as follows.

We start (a) by computing a complete family $\mathbf{S}$ of nonground support sets for each DL-atom. Afterwards the replacement program $\hat{\Pi}$ is created and its answer sets are computed one by one. Once an answer set $\hat{I}$ of $\hat{\Pi}$ is found (b), we first determine the sets of DL-atoms D_p (resp. D_n) that are guessed true (resp. false) in $\hat{I}$. Next, for all ground DL-atoms in $D_p \cup D_n$, the function $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$ instantiates $\mathbf{S}$ to relevant ground support sets, i.e., that are coherent with $\hat{I}$ and match with $\mathcal{A} \cup \mathcal{A}_d$. We then check in (c) for atoms in D_p (resp. D_n) without support (resp. input only support). If either is the case, we skip to (b), the next model candidate, since no repair exists for the current one. Otherwise, in a loop (d) over atoms in D_p—except for those supported input only (e)—we remove support sets S that are conflicting w.r.t. D_n. Intuitively, this is the case if S hinges on an assertion $\alpha \in \mathcal{A}$ that also supports some atom $d' \in D_n$ (hence α needs to be deleted; note that due to consistency of $\mathcal{A}$, even inconsistent support of d' leaves no choice). If this operation leaves the atom from D_p under consideration without support (check at (f)), then no repair exists and the next model candidate is considered. Otherwise (exiting the loop at (g)), a potential deletion repair $\mathcal{A}'$ is obtained from $\mathcal{A}$ by removing assertions that occur in any support set for some atom $d' \in D_n$. An eventual check (h) for foundedness (minimality) w.r.t. $\mathcal{A}'$ determines whether a deletion repair answer set has been found.

Example 5.35. Suppose $\{e_a, ne_b\} \subseteq \hat{I}$ for the DL-atom $a = \mathrm{DL}[; \mathit{hasParent}](\mathit{john}, \mathit{pat})$ and the DL-atom $b = \mathrm{DL}[\mathit{Male} \uplus \mathit{boy}; \mathit{Male}](\mathit{pat})$ from Example 4.1. Then, we get $\mathbf{S}^{\hat{I}}_{gr}(a) = \{\{\mathit{hasParent}(\mathit{john}, \mathit{pat})\}\}$ we reach the else part of Step (e) where nothing is removed from $\mathbf{S}^{\hat{I}}_{gr}(a)$, since $\mathbf{S}^{\hat{I}}_{gr}(b) = \{\{\mathit{Male}(\mathit{pat})\}\}$ and $\mathbf{S}^{\hat{I}}_{gr}(a) \cap \mathbf{S}^{\hat{I}}_{gr}(b) = \emptyset$. Hence, at Step (g) we must drop $\mathit{Male}(\mathit{pat})$ from $\mathcal{A}$ to make $\hat{I}$ a deletion repair answer set. □

As can be shown, algorithm *SupRAnsSet* correctly computes the deletion repair answer sets of the input DL-program. For the completeness part, i.e., that all deletion repair answer sets are indeed produced, the following proposition is crucial.

Proposition 5.36. *Given a DL-program* Π*, let* $\hat{I}$ *be an answer set of* $\hat{\Pi}$ *such that* $I = \hat{I}|_\Pi$ *is an answer set of* $\Pi = \langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$*. If* $\hat{I}$ *is a compatible set for* $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle$ *where* $\mathcal{A}' \supseteq \mathcal{A}$*, then* I *is an answer set for* $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle$*.*

Proof. Assume that I is an answer set of $\Pi = \langle \mathcal{O}, P \rangle$, where $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ and that $\hat{I}$ is a compatible set for $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$ where $\mathcal{O}' = \mathcal{T} \cup \mathcal{A}'$ and $\mathcal{A}' \supset \mathcal{A}$. Towards contradiction, suppose I is not an answer set of Π. Hence, $I = \hat{I}|_\Pi$ is not a minimal model of $\Pi'^{I,\mathcal{O}}_{\mathit{flp}} = \langle \mathcal{T} \cup \mathcal{A}', P^{I,\mathcal{O}}_{\mathit{flp}} \rangle$. That is, some $I' \subset I$ exists such that $I' \models^{\mathcal{O}'} P^{I,\mathcal{O}}_{\mathit{flp}}$. We then obtain that also $I' \models^{\mathcal{O}} P^{I,\mathcal{O}}_{\mathit{flp}}$; this contradicts that I is an answer set of Π. Indeed, suppose that $I' \not\models^{\mathcal{O}} P^{I,\mathcal{O}}_{\mathit{flp}}$. Then some rule $r \in P^{I,\mathcal{O}}_{\mathit{flp}}$ of form (2.2) is violated wrt. I' and $\mathcal{O}$, i.e., (i) $I' \models^{\mathcal{O}} b_i$ for each $1 \leq i \leq k$, (ii) $I' \not\models^{\mathcal{O}} b_j$ for each $k < j \leq m$, and (iii) $I' \not\models^{\mathcal{O}} a_h$ for each $1 \leq h \leq n$. By monotonicity of $I \models^{\mathcal{O}} a$ w.r.t. I and $\mathcal{O}$, we conclude $I' \models^{\mathcal{O}'} b_i$, $I' \not\models^{\mathcal{O}'} b_j$ (as $\hat{I}$ is a compatible set for both $\hat{\Pi}$ and $\hat{\Pi}'$), and $I \not\models^{\mathcal{O}} b_j$, and $I' \not\models^{\mathcal{O}'} a_h$. But then $I' \not\models^{\mathcal{O}'} P^{I,\mathcal{O}}_{\mathit{flp}}$, which is a contradiction. Hence, I' does not exist and I is an answer set of Π'. □

Armed with this result, we establish the correctness result.

Theorem 5.37. *SupRAnsSet is sound and complete w.r.t. deletion repair answer sets.*

Proof. Soundness. Suppose *SupRAnsSet* outputs $I = \hat{I}|_\Pi$. We can get to (h) only if $\hat{I}$ is an answer set of $\hat{\Pi}$; furthermore, by setting $\mathbf{S}^{\hat{I}}_{gr}$ to $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$ in (b) and by the further modifications, it is ensured at (h) that each DL-atom $a \in D_p$ has some coherent support set that matches with $\mathcal{A}'$ (i.e., $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \neq \emptyset$), while no DL-atom $a' \in D_n$ has such a support set. Thus from Proposition 5.28, it follows that $\hat{I}$ is a compatible set for $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', P \rangle$; hence $I \models \Pi'$. Furthermore, as *flpFND*$(\hat{I}, \mathcal{T} \cup \mathcal{A}', P)$ succeeds, I is a minimal model of $\Pi'^{I,\mathcal{O}}_{\mathit{flp}}$. Hence I is an answer set of Π', and thus a deletion repair answer set of Π.

Completeness. Suppose I is a deletion repair answer set. That is, for some $\mathcal{A}' \subseteq \mathcal{A}$, we have that I is an answer set of $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', P \rangle$. This implies Proposition 5.2 that $\hat{I}$ is an answer set of $\hat{\Pi}$ and thus will be considered in (b), with D_p and D_n reflecting the (correct) guess for $I \models^{\mathcal{O}'} a$ for each DL-atom a, where $\mathcal{O}' = \mathcal{T} \cup \mathcal{A}'$. From Proposition 5.28 and completeness of $\mathbf{S}$, we obtain that each $a \in D_p$ has $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \neq \emptyset$ and each $a \in D_n$ has $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) = \emptyset$. The initial $\mathbf{S}^{\hat{I}}_{gr}$ is such that $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \subseteq \mathbf{S}^{\hat{I}}_{gr} = Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a)$ holds for each DL-atom a; in further steps, the algorithm removes all support sets $S \in Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a)$ for $a \in D_p$ from $\mathbf{S}^{\hat{I}}_{gr}(a)$ such that such that $S \cap S' \cap \mathcal{A} \neq \emptyset$ for some support set $S' \in Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a')$ and $a' \in D_n$, and removes all assertions in $S' \cap \mathcal{A}$ from $\mathcal{A}$. Importantly no removed S is in $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a)$, since by the assertion that $\mathcal{T} \cup \mathcal{A}$ is consistent, $|S' \cap \mathcal{A}| = 1$ must hold. Thus step (g) will be reached, and the variable $\mathcal{A}'$ is assigned an ABox $\mathcal{A}''$ such that $\mathcal{A}' \subseteq \mathcal{A}'' \subseteq \mathcal{A}$. Since $\hat{I}$ is a compatible set for $\Pi'' = \langle \mathcal{T} \cup \mathcal{A}'', \mathcal{P} \rangle$ and I is an answer set of Π', by Proposition 5.36 I is also an answer set of Π'', and thus I is a minimal model of $\Pi''^{I,\mathcal{O}}_{\mathit{flp}} = \langle \mathcal{T} \cup \mathcal{A}'', \mathcal{P}^{I,\mathcal{O}}_{\mathit{flp}} \rangle$. Hence, the test *flpFND*$(\hat{I}, \mathcal{T} \cup \mathcal{A}', \mathcal{P})$ in step (h) (where $\mathcal{A}'$ has value $\mathcal{A}''$) succeeds, and $\hat{I}_\Pi$, i.e, I is output. □

5.5 Support Sets for DL-atoms over $\mathcal{EL}$ Ontologies

The DL-atoms accessing $\mathcal{EL}$ ontologies can have arbitrarily large support sets of no particular structure. Moreover, there can be infinitely many such support sets in general. While for acyclic TBoxes (which is a property often met in practice) the latter is excluded, complete support families can anyway be very large and constructing them as well as managing might be impractical. Despite this observation, we still make use of support families, but omit the requirement for their completeness.

We now provide a method for support set construction that like in the *DL-Lite*$_\mathcal{A}$ case allows us to just work with ontology predicates when constructing nonground support sets. As negation is not available nor expressible in $\mathcal{EL}$ ($\bot$ is unavailable), here we restrict our attention to DL-atoms DL$[\lambda; Q](\vec{c})$ with positive updates, i.e. $\circ \in \{\uplus\}$ for all $P \circ p \in \lambda$.

For construction of support sets for DL-atoms over $\mathcal{EL}$ ontologies, it is natural to exploit (conjunctive) query answering methods in $\mathcal{EL}$ (e.g., [Ros07, LTW08, KLT$^+$10, SMH12]). Most of them are based on rewriting the query and the TBox into a datalog program over the ABox; to construct guard functions that use a datalog rewriting of the TBox seems thus suggestive.

Suppose we are given a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an $\mathcal{EL}$ ontology and a DL-atom $d(\vec{X}) = \text{DL}[\lambda; Q](\vec{X})$. Our method for constructing nonground support sets for $d(\vec{X})$ consists of the following three steps.

Step 1. DL-query Rewriting over the TBox. The first step exploits the rewriting of the DL-query Q of $d(\vec{X})$ over the TBox $\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\}$ into a set of datalog rules, see e.g. Figure 5.2. At the preprocessing stage, the normalization technique is first applied to the TBox $\mathcal{T}_d$. This technique restricts the syntactic form of TBoxes by decomposing complex axioms into syntactically simpler ones. For this purpose, a minimal required set of fresh concept symbols is introduced. Given a TBox $\mathcal{T}_d$, its normalized form $\mathcal{T}_{dnorm}$ is computed in linear time[4] [BBL05]. We then rewrite the part of the TBox, relevant for the query at hand, into a datalog program $Prog_{Q,\mathcal{T}_{dnorm}}$ using the translation given in Table 5.1, which is a variant of [PUMH10, ZPR09]. When rewriting axioms of the form $A_1 \sqsubseteq \exists R.A_2$ (fourth axiom in Table 5.1) we introduce fresh constants (o_{A_2}) to represent "unknown" objects. A similar rewriting is exploited in the REQUIEM system (where function symbols are used instead of fresh constants). As a result we obtain:

Lemma 5.38. *For any data part, i.e., ABox $\mathcal{A}$ over sig($\mathcal{T}_d$), and any ground assertion $Q(\vec{c})$, it holds that $Prog_{Q,\mathcal{T}_{dnorm}} \cup \mathcal{A} \models Q(\vec{c})$ iff $\mathcal{T}_{dnorm} \cup \mathcal{A} \models Q(\vec{c})$ iff $\mathcal{T}_d \cup \mathcal{A} \models Q(\vec{c})$.*

Step 2. Query Unfolding. The second step proceeds with the standard unfolding of the rules of $Prog_{Q,\mathcal{T}_{dnorm}}$ w.r.t. the target DL-query Q. We start with the rule that

[4]Linear complexity results are obtained under the standard assumption in DLs that each of the atomic concepts is of constant size, i.e., the length of a binary string representing an atomic concept does not depend on the particular knowledge base.

Axiom	Datalog rule
$A_1 \sqsubseteq A_2$	$A_2(X) \leftarrow A_1(X)$
$A_1 \sqcap A_2 \sqsubseteq A_3$	$A_3(X) \leftarrow A_1(X), A_2(X)$
$\exists R.A_2 \sqsubseteq A_1$	$A_2(X) \leftarrow R(X,Y), A_3(Y)$
$A_1 \sqsubseteq \exists R.A_2$	$R(X, o_{A_2}) \leftarrow A_1(X)$
	$A_2(o_{A_2}) \leftarrow A_1(X)$

Table 5.1: $\mathcal{EL}$ TBox rewriting

$$Prog_{Q,\mathcal{T}_{dnorm}} = \left\{ \begin{array}{l} (1')\ \mathit{Staff}(X) \leftarrow \mathit{Blacklisted}(X); \\ (2')\ C_{\exists \mathit{hasA}.A}(X) \leftarrow \mathit{hasAction}(X, Y), \mathit{Action}(Y); \\ (3')\ C_{\exists \mathit{hasS}.\mathit{St}}(X) \leftarrow \mathit{hasSubject}(X, Y), \mathit{Staff}(Y); \\ (4')\ C_{\exists \mathit{hasT}.P}(X) \leftarrow \mathit{hasTarget}(X, Y), \mathit{Project}(Y); \\ (5')\ C_{\exists \mathit{hasA}.A \sqcap \exists \mathit{hasS}.\mathit{St}}(X) \leftarrow C_{\exists \mathit{hasA}.A}(X), C_{\exists \mathit{hasS}.\mathit{St}}(X); \\ (6')\ \mathit{StaffRequest}(X) \leftarrow C_{\exists \mathit{hasA}.A \sqcap \exists \mathit{hasS}.\mathit{St}}(X), C_{\exists \mathit{hasT}.P}(X); \\ (7')\ \mathit{Project}(X) \leftarrow \mathit{Project}_{\mathit{projfile}}(X). \end{array} \right\}$$

Figure 5.2: DL-query rewriting for DL[*Project* $\uplus$ *projfile*; *StaffRequest*](X) over $\mathcal{T}_{dnorm}$

has Q in the head and expand its body using other rules of the program $Prog_{Q,\mathcal{T}_{dnorm}}$. By applying this procedure exhaustively, we get a number of rules which correspond to the rewritings of the query Q over $\mathcal{T}_{dnorm}$. Note that it is not always possible to obtain all of the rewritings effectively, since in general there might be infinitely many of them (exponentially many for acyclic $\mathcal{T}$). We discuss possible restrictions in the next section.

Step 3. Support Set Extraction. The last step is devoted to the extraction of nonground support sets from the rewritings computed in Step 2. We select those that contain only predicates from $\mathcal{T}_d$ and obtain a set of rules r of the form

$$Q(\vec{X}) \leftarrow P_1(\vec{Y_1}), \dots P_k(\vec{Y_k}), P_{k+1 p_{k+1}}(\vec{Y}_{k+1}), \dots, P_{n p_n}(\vec{Y_n}), \tag{5.1}$$

where each P_i is a native ontology predicate if $1 \leq i \leq k$, and a predicate mirroring lp-input of d otherwise. From such rules r, we construct pairs $S = \langle N, \gamma \rangle$, where

- $N = \{p_i(\vec{Y_i}) \mid P_{i p_i}(\vec{Y_i}) \in B(r), k+1 \leq i \leq n\}$;
- $\gamma : \mathcal{C}^{|\vec{X}|} \times grnd_{\mathcal{C}}(N) \to \{0, 1\}$ is such that $\gamma(\vec{c}, N_{gr}) = 1$ only if $Q(\vec{c})$ follows from $r \cup \mathcal{A} \cup \{P_{i p_i}(\vec{t}) \in \mathcal{A}_d \mid p_i(\vec{t}) \in N_{gr}\}$.

Then the following holds.

Proposition 5.39. *Let $d(\vec{X}) = \mathrm{DL}[\lambda; Q](\vec{X})$ be a DL-atom of a program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, is an $\mathcal{EL}$ ontology. Every set S constructed following Step 1-3 is a nonground support set for $d(\vec{X})$.*

Proof. Towards a contradiction assume that S is not a nonground support set for $d(\vec{X})$. This means that either (1) N is not a set of nonground predicates from λ or (2) the function γ of Definition 5.15 is not correct.

The predicates of the form P_P in the TBox $\mathcal{T}_d$ are obtained from λ of $d(\vec{X})$ by construction and clearly so are the predicates $P_{j_{p_j}}$ of each rule r. Thus the predicates in N are indeed nonground predicates from the input signature of $d(\vec{X})$, therefore (1) can not hold.

Assume that (2) is true, that is the function γ is not correct, i.e. some $\vec{c} \in \mathcal{C}^{|X|}$, and $N_{gr} \in grnd_{\mathcal{C}}(N)$ exist, such that $\gamma(\vec{c}, N_{gr}) = 1$, but N_{gr} is not a positive ground support set for $d(\vec{c})$. The latter means that some interpretation $I' \supseteq N_{gr}$ exists such that $I' \not\models^{\mathcal{O}} d(\vec{c})$. Therefore, by Proposition 5.22 we have that $\mathcal{O}_d^{I'} \not\models Q(\vec{c})$, i.e. $\mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I'\} \not\models Q(\vec{c})$. Observe that by Step 3 of our algorithm for every rule r, that contains only predicates from $\mathcal{T}_d$ and that is obtained from standard rules unfolding of $Prog_{Q,\mathcal{T}_{dnorm}}$ w.r.t. Q, it holds that $r \cup \mathcal{A} \cup \{P_{ip_i}(\vec{t}) \in \mathcal{A}_d \mid p_i(\vec{t}) \in N_{gr}\} \models Q(\vec{c})$. However, then we also have that $Prog_{Q,\mathcal{T}_{dnorm}} \cup \mathcal{A} \cup \{P_{ip_i}(\vec{t}) \in \mathcal{A}_d \mid p_i(\vec{t}) \in N_{gr}\} \models Q(\vec{c})$, and thus by Lemma 5.38 and monotonicity of $\mathcal{O}$, it holds that $\mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I'\} \models Q(\vec{c})$, leading to a contradiction. Therefore, γ is correct, and $S = \langle N, \gamma \rangle$ is indeed a nonground support set for $d(\vec{X})$.

□

We now illustrate the computation of nonground support sets for DL-atoms over $\mathcal{EL}$ ontologies on the following example.

Example 5.40. Consider a DL-atom DL[*Project* ⊎ *projfile*; *StaffRequest*](X) accessing an $\mathcal{EL}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ from Figure 5.1. The datalog rewriting for d computed at Step 1 is given in Figure 5.2. In Step 2 we obtain the following query unfoldings for *StaffRequest*:

(1) $StaffRequest(X) \leftarrow StaffRequest(X)$;
(2) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'), Staff(Y'), hasTarget(X,Y''), Project_{projfile}(Y'')$;
(3) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'), Staff(Y'), hasTarget(X,Y''), Project(Y'')$;
(4) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'), Blacklisted(Y'), hasTarget(X,Y''), Project(Y'')$;
(5) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'), Blacklisted(Y'), hasTarget(X,Y''), Project_{projfile}(Y'')$.

Finally, at Step 3 we extract support sets from the unfoldings (1)-(5). For example, support sets S_1 and S_2 as in Example 5.16 are obtained from the unfoldings (1) and (2) respectively. From the rule (3) we get the support set $S_3 = \{\emptyset, \gamma\}$, where $\gamma : \mathcal{C} \times \{\emptyset\} \rightarrow \{0,1\}$ is such that $\gamma(c, \{\emptyset\}) = 1$ only if the ABox $\mathcal{A}$ contains the following assertions: $hasAction(c, c1), Action(c1), hasSubject(c, c2), Staff(c2), hasTarget(c, c3), Project(c3)$,

where *c1*, *c2*, *c3* are arbitrary constants from $\mathcal{C}$. From the rules (4) and (5) the support sets are similarly determined. □

Like in the *DL-Lite*$_\mathcal{A}$ case when working with support sets for DL-atoms over $\mathcal{EL}$ DL, we can restrict ourselves to the ontology predicates and operate only on them. More specifically, rules of the form (5.1) fully reflect nonground support sets as given in Definition 5.15, and ground instantiations of such rules over constants from $\mathcal{C}$ implicitly correspond to ground support sets. We, therefore, present $\mathcal{EL}$ support sets similar as it is done in Definitions 5.24 and 5.30 for *DL-Lite*$_\mathcal{A}$.

Definition 5.41 (Normalized Ground $\mathcal{EL}$ Support Sets)**.** Given a ground DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$, a set S of assertions from $\mathcal{A} \cup \mathcal{A}_d$ is a *normalized ground $\mathcal{EL}$ support set* for d w.r.t. an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ in $\mathcal{EL}$, if $S \cup \mathcal{T}_d \models Q(\vec{t})$.

By $Supp^{\mathcal{EL}}_{\mathcal{O}}(d)$ we denote the family of all normalized ground $\mathcal{EL}$ support sets.

Example 5.42. In Example 5.9, the support set $S = \{projfile(p1)\}$ for the DL-atom $\mathrm{DL}[Project \uplus projfile; StaffRequest](r1)$ corresponds to the normalized support set $S' = \{hasAction(r1, read), hasTarget(r1, p1), Project_{projfile}(p1), hasSubject(r1, john), Staff(john), Action(read)\}$.

Definition 5.43 (Normalized Nonground $\mathcal{EL}$ Support Sets)**.** Suppose that $d(\vec{X}) = \mathrm{DL}[\lambda; Q](\vec{X})$ is a DL-atom and $\mathcal{T}$ is an $\mathcal{EL}$ TBox. Any unfolding of the DL-query $Q(\vec{X})$ in $Prog_{Q,\mathcal{T}_d norm}$, which contains only atoms over predicates from $\mathsf{sig}(\mathcal{T}_d)$ is a *normalized nonground $\mathcal{EL}$ support set* for d w.r.t. $\mathcal{T}$.

We denote by $\mathbf{Supp}^{\mathcal{EL}}_{\mathcal{O}}(d)$ a set of all normalized nonground $\mathcal{EL}$ support sets for d.

Like in the *DL-Lite*$_\mathcal{A}$ case we often omit the superscript $\mathcal{EL}$ in $Supp^{\mathcal{EL}}_{\mathcal{O}}(d)$ (resp. in $\mathbf{Supp}^{\mathcal{EL}}_{\mathcal{O}}(d)$), if it is clear from the context that $\mathcal{O}$ is in $\mathcal{EL}$.

Example 5.44. The nonground support sets S_1 and S_2 from Example 5.16 for the DL-atom $\mathrm{DL}[Project \uplus projfile; StaffRequest](X)$ correspond to the normalized support sets $S'_1 = \{StaffRequest(X)\}$ and $S'_2 = \{hasAction(X, Y_1), Action(Y_1), hasSubject(X, Y_2), Staff(Y_2), hasTarget(X, Y_3), Project(Y_3)\}$ respectively.

Furthermore, the support set S_3 from Example 5.40 maps to the following normalized support set: $S'_3 = \{hasAction(X, Y), Action(Y), hasSubject(X, Y'), Project(Y''), Staff(Y'), hasTarget(X, Y'')\}$. □

From every normalized support set one can easily reconstruct a support set in the sense of generic Definition 5.15. When referring to support sets we often mean normalized support sets, i.e. DL-query unfoldings.

According to novel results [HLSWar], complete support families can be computed for large classes of ontologies. However, in general there might be exponentially many unfoldings produced at Step 2 (see below). Thus, to cope with exponentiality, one might often want to apply reasonable restrictions on the support families.

5.5.1 Partial Support Families

In this section we discuss restrictions on the size, structure and number of support sets, which are of interest for practical applications. We provide methods for effective computation of restricted support families and analyze conditions put on the TBox, under which the partial support family is complete.

In general, unlike for the *DL-Lite*$_\mathcal{A}$ case, due to possible cyclic dependencies of the form $\exists r.C \sqsubseteq C$ allowed in $\mathcal{EL}$, the explanations of an instance query can be of infinite size and so are the support sets for DL-atoms accessing an $\mathcal{EL}$ ontology. An analysis of a vast number of ontologies has revealed that in many realistic cases they do not contain (or imply) cyclic axioms [GTH06]; we thus assume that the TBox of the ontology in a given DL-program is acyclic (i.e., does not entail inclusion axioms of form $\exists r.C \sqsubseteq C$). However, even under this restriction support sets can be large in general.

Clearly, any support set $S' \supseteq S$ is a support set for d, if S is a support set for d. Therefore, when estimating the maximal support set size for a given DL-atom, we always consider only subset-minimal support sets.

Definition 5.45. Given a DL-atom $\mathrm{DL}[\lambda; Q](\vec{X})$ and an ontology $\mathcal{O}$, the *maximal support set size* $maxsup(d)$ for d is the size of the largest $\subseteq$-minimal support set for d over $\mathcal{O}$.

Example 5.46. Consider the TBox $\mathcal{T}$, which contains the following axioms:

$$\begin{array}{l}(1)\ \exists r.B_0 \sqcap \exists s.B_0 \sqsubseteq B_1\\ (2)\ \exists r.B_1 \sqcap \exists s.B_1 \sqsubseteq B_2\\ \quad\ldots\\ (\mathrm{n})\ \exists r.B_{n-1} \sqcap \exists s.B_{n-1} \sqsubseteq B_n\end{array}$$

For the DL-atom $d_1 = \mathrm{DL}[\lambda; B_1](X_1)$, the maximal support set size is 4, which is witnessed by

$$S_1 = \{r(X_1, X_2), B_0(X_2), s(X_1, X_3), B_0(X_3)\}.$$

For the DL-atom $d_2 = \mathrm{DL}[\lambda; B_2](X_1)$, it holds that $maxsup(d_1) = 10$.

$$\begin{array}{c}S_2 = \{r(X_1, X_2), r(X_2, X_3), B_0(X_3), s(X_2, X_4), B_0(X_4),\\ s(X_1, X_5), r(X_5, X_6), B_0(X_6), s(X_5, X_7), B_0(X_7)\}\end{array}$$

One can verify that the maximal support set size for $d_n = \mathrm{DL}[\lambda; B_n](X)$ is calculated as follows:

$$maxsup(d_n) = maxsup(d_{n-1}) \times 2 + 2.$$

Note that the maximal support set for d_n involves $n + 3$ predicates. Therefore, if the TBox contains only axioms of the form from above and $|\mathsf{sig}(\mathcal{T})| = k$, i.e. the TBox is over the signature of k predicates, we obtain that the lower bound for the worst case support set size for d is $2^{k-1} + 2$, i.e. $O(2^k)$, which is single exponential in the number of predicates in $\mathcal{T}$. □

It is difficult to control the size of support sets over an $\mathcal{EL}$ TBox in general, and it is not clear which TBox criteria guarantee that support sets for DL-atoms accessing a given ontology are of a limited size. These challenges motivate us to investigate reasonable restrictions on support families. In particular, we are interested in the following problem: provided that the computation of all support sets is computationally expensive, which exactly support sets should be computed and which methods should be exploited for that. Another issue that we investigate in this chapter is analysis of criteria on the TBox that ensure effective construction of complete support families.

A natural approach for computing a partial support family is the restriction of the target support set size. We may put a certain bound k on the size of support sets that we want to compute for a given DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ and proceed using, for example, limited program unfolding. That is we aim at controlling the rule unfoldings of the program $Prog_{Q,\mathcal{T}_{d_{norm}}}$, constructed in the Step 1 of our algorithm for support set computation. When a certain unfolding branch reaches the predefined size limit k, we stop its further expansion and choose a different branch. Similarly, we can compute a limited number k of support sets by stopping the rules unfolding of the program $Prog_{Q,\mathcal{T}_{d_{norm}}}$ once the k-th support set is identified.

An alternative approach for partial support family computation, and the one we exploit in what follows, is based on the TBox approximation techniques.

5.6 TBox Approximation Techniques for Partial Support Family Construction

Approximation of DL ontologies specified in a source language D using a less expressive target language D' is a well-known and important optimization technique in ontology management. Several approaches have been proposed in this regard. They can be divided into two major groups: *syntactic* and *semantic* approaches. Methods from the former group, e.g. [TRKH08, WGS05] focus on the syntactic form of the axioms of the original ontology, they appropriately rewrite the axioms not complying with the syntax of the target ontology language. Syntactic approaches are rather effective in general, but they can produce unsound answers [PT07]. Methods from the latter group focus on the entailments from the original ontology, rather then on its syntactic structure. They aim at preserving the maximal amount of these entailments by transforming the original ontology into the target language. Therefore, in general they are sound, but might require more computational power [CMR$^+$14].

Ontology approximation techniques are of relevance for our task of computing partial support families. More specifically, given an $\mathcal{EL}$ ontology one can approximate its TBox to *DL-Lite*$_\mathcal{A}$ DL, and construct complete support families for DL-atoms with respect to the latter, which is efficiently possible. If the approximation is sound then as a result we obtain partial support families over the original $\mathcal{EL}$ ontology in an effective way.

Most of the works on ontology approximation focused on rewriting ontologies over some expressive language into simpler DLs (often lightweight ones). E.g. the work [CFR$^+$14] deals with rewritings of $\mathcal{SHOIQ}$ to $\mathcal{EL}$, the work [CDR98] proposes a trans-

formation approach from $\mathcal{ALCFI}$ knowledge bases to $\mathcal{ALC}$. In our setting, however, we already have a TBox in the lightweight DL $\mathcal{EL}$, and we aim at approximating it to *DL-Lite*$_\mathcal{A}$, which has better properties for our particular problem of support set computation. Therefore, for the purpose of approximation in our setting, exploiting *logical difference* between $\mathcal{EL}$ TBoxes [KLWW12] seems suggestive.

The idea behind the logical difference is to decide whether two ontologies give the same answers to queries over a given vocabulary (called signature) Σ, and compute a succinct representation of the difference if it is not empty. Typical queries include subsumption queries between concepts, instance and conjunctive queries. For our setting of computing nonground support sets for a given DL-atom d, the concept subsumption queries are of a particular interest. More specifically, we exploit the following idea. After elimination of axioms that do not fall into *DL-Lite*$_\mathcal{A}$ DL from the $\mathcal{EL}$ TBox $\mathcal{T}_d$ we obtain a simplified TBox $\mathcal{T}'_d$. We then set the signature $\Sigma = \mathsf{sig}(\mathcal{A}_d \cup \mathcal{A} \cup Q)$, where $\mathcal{A}$ is an ABox of the original ontology, and Q is a DL-query. We then compute the logical difference between $\mathcal{T}$ and $\mathcal{T}'$ w.r.t. Σ. From the succinct representation of this difference we extract a set of axioms A that falls into the *DL-Lite*$_\mathcal{A}$ fragment, and then we add A to the TBox $\mathcal{T}'$. In such a way we reintroduce the redundantly eliminated axioms, and thus obtain an approximation $\mathcal{T}'$ of $\mathcal{T}$. By restricting the relevant vocabulary to Σ we minimize the computation of the logical difference only to those parts of the TBox that actually influence support sets for d. This approximation approach is particularly attractive, as the logical difference between $\mathcal{EL}$*-terminologies* was intensively studied in many works, e.g. [GHKS07, LWW07, KLWW12], and available effective algorithms can be reused for our needs.

$\mathcal{EL}$-terminologies are restricted forms of TBoxes, defined as follows.

Definition 5.47. An $\mathcal{EL}$*-terminology* is a general $\mathcal{EL}$ TBox $\mathcal{T}$, satisfying the following conditions:

(1) $\mathcal{T}$ consists of concept equivalences of the form $A \equiv C$ and $A \sqsubseteq C$, where A is atomic and C is an arbitrary $\mathcal{EL}$ concept;

(2) No concept name occurs more then once on the left hand side of axioms in $\mathcal{T}$.

To make use of the available tractable algorithms for computing logical difference we restrict ourselves in this section to $\mathcal{EL}$-terminologies, that is we assume that the ontology TBox satisfies the conditions (1) and (2) from above. Under the restriction to $\mathcal{EL}$-terminologies logical difference w.r.t. a given signature becomes tractable. To exploit the approach formally, let us first recall some of the existing notions in this context.

Definition 5.48 (cf. [KLWW12])**.** The Σ*-concept difference* between $\mathcal{EL}$-terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ is the set $\mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ of all $\mathcal{EL}$-inclusions α over Σ, such that $\mathcal{T}_1 \models \alpha$ and $\mathcal{T}_2 \not\models \alpha$.

Example 5.49. Let $\mathcal{T}_1 = \{B \sqsubseteq E; E \sqsubseteq \exists r.\top; C \sqsubseteq A \sqcap B\}$ and $\mathcal{T}_2 = \{C \sqsubseteq A; D \sqsubseteq B; D \equiv C\}$ be two terminologies. It is not difficult to verify that $\mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \emptyset$ for $\Sigma = \{A, B, C\}$, while $\mathsf{cDiff}_{\Sigma'}(\mathcal{T}_1, \mathcal{T}_2) = \{B \sqsubseteq \exists r.\top\}$ for $\Sigma' = \{B, r\}$. □

If two $\mathcal{EL}$-terminologies entail the same concept subsumptions built over the predicates from the given signature, then they may be considered as logically equivalent with respect to the relevant information regardless of their syntactic or structural form. Formally, such equivalent terminologies are defined as follows.

Definition 5.50 (cf. [KLWW12])**.** The terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-*concept inseparable*, in symbols $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$, if $\mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \mathsf{cDiff}_\Sigma(\mathcal{T}_2, \mathcal{T}_1) = \emptyset$

Example 5.51. Reconsider Example 5.49. The terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-concept-inseparable for $\Sigma = \{A, B, C\}$. □

The logical difference in terms of instance queries is defined as follows.

Definition 5.52 (cf. [KLWW12])**.** The Σ-*instance difference* between terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ is the set $\mathsf{iDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ of pairs of the form $(\mathcal{A}, \alpha)$, where $\mathcal{A}$ is a Σ-ABox and α a Σ-instance assertion, such that $\mathcal{T}_1 \cup \mathcal{A} \models \alpha$ and $\mathcal{T}_2 \cup \mathcal{A} \not\models \alpha$. We say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-*instance inseparable*, in symbols $\mathcal{T}_1 \equiv^i_\Sigma \mathcal{T}_2$ if $\mathsf{iDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \mathsf{iDiff}_\Sigma(\mathcal{T}_2, \mathcal{T}_1) = \emptyset$.

It is not difficult to verify that $\mathcal{T}_1 \equiv^i_\Sigma \mathcal{T}_2$ implies that $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$ holds. The other direction is not that obvious; however, it has also been proven in [LW10].

Theorem 5.53 (cf. [LW10])**.** *For any $\mathcal{EL}$-terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ and signature Σ, it holds that $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$ iff $\mathcal{T}_1 \equiv^i_\Sigma \mathcal{T}_2$.*

Before showing that a DL-atom has the same set of support sets under Σ-concept inseparable terminologies, we first prove the following preliminary lemma.

Lemma 5.54. *Let $d = \mathrm{DL}[\lambda; Q](\vec{t})$ be a DL-atom, let $\mathcal{O} = \langle \mathcal{T}_1, \mathcal{A} \rangle$ be an $\mathcal{EL}$ ontology, and let $\mathcal{T}_2$ be a TBox. Then if $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-concept inseparable, where $\Sigma = \mathit{sig}(\mathcal{A}) \cup \{Q\} \cup \{P \mid P \circ p \in \lambda\}$ then $\mathcal{T}_{1d}$ and $\mathcal{T}_{2d}$ are Σ'-concept inseparable w.r.t. $\Sigma' = \Sigma \cup \mathit{sig}(\mathcal{A}_d)$.*

Proof. Towards a contradiction, assume that $\mathcal{T}_{1d}$ and $\mathcal{T}_{2d}$ are not Σ'-concept inseparable. W.l.o.g. suppose that $\mathcal{T}_{1d} \models P_1 \sqsubseteq P_2$, but $\mathcal{T}_{2d} \not\models P_1 \sqsubseteq P_2$, where $P_1, P_2 \in \Sigma'$. Observe that the signatures Σ and Σ' differ only on predicates P_p, such that $P \circ p$ occurs in λ of d. Furthermore, the TBox $\mathcal{T}' = \mathcal{T}_{1d} \backslash \mathcal{T}_1 = \mathcal{T}_{2d} \backslash \mathcal{T}_2$ consists only of simple inclusions $P_p \sqsubseteq P$, such that P_p does not occur in $\mathcal{T}_1$ or $\mathcal{T}_2$.

There are two possibilities: either $P_1 \in \Sigma$ or $P_1 \in \Sigma' \backslash \Sigma$.

(i) Suppose that $P_1 \in \Sigma$. It holds that $P_2 \in \Sigma$, as otherwise $P_2 \notin \Sigma'$, contradicting our assumption. Due to Σ-concept inseparability of $\mathcal{T}_1$ and $\mathcal{T}_2$ we have that, if $\mathcal{T}_1 \models P_1 \sqsubseteq P_2$ then it must hold that $\mathcal{T}_2 \models P_1 \sqsubseteq P_2$. However, then $\mathcal{T}_{2d} \models P_1 \sqsubseteq P_2$ due to monotonicity, which leads to a contradiction.

Therefore, we have that $\mathcal{T}_1 \not\models P_1 \sqsubseteq P_2$, but $\mathcal{T}_{1d} \models P_1 \sqsubseteq P_2$. There are two possibilities: either $\mathcal{T}' \models P_1 \sqsubseteq P_2$ or $\mathcal{T}' \not\models P_1 \sqsubseteq P_2$. In the former case we get that $P_1 \notin \Sigma$, i.e. contradiction. In the latter case it must hold that $\mathcal{T}_1 \models P_1 \sqsubseteq P$ for some concept P and $\mathcal{T}' \models P \sqsubseteq P_2$, which means that $P \in \mathsf{sig}(\mathcal{A}_d)$, and thus

$P \in \Sigma \setminus \Sigma'$, but then $\mathcal{T}_1 \models P_1 \sqsubseteq P$ can not hold. Therefore, it must be the case that $\mathcal{T}_1 \models P_1 \sqsubseteq P_2$, and hence due to Σ-concept inseparability of $\mathcal{T}_1$ and $\mathcal{T}_2$, we have that $\mathcal{T}_2 \models P_1 \sqsubseteq P_2$, from which by monotonicity $\mathcal{T}_{2d} \models P_1 \sqsubseteq P_2$ follows.

(ii) Assume now that $P_1 \in \Sigma' \setminus \Sigma$. Then P_1 must be of the form P_p, where $P \circ p$ occurs in λ of d. We have that either $P_1 \sqsubseteq P_2 \in \mathcal{T}'$ or $P_1 \sqsubseteq P_2 \notin \mathcal{T}'$. The former case immediately leads to a contradiction, since $\mathcal{T}' \subseteq \mathcal{T}_{d2}$. In the latter case we have that $\mathcal{T}' \models P_1 \sqsubseteq P$ and $\mathcal{T}_1 \models P \sqsubseteq P_2$. As $\mathcal{T}' \models P_1 \sqsubseteq P$, we have that $P \in \Sigma$; moreover, $P_2 \in \Sigma$. Hence, due to Σ-concept inseparability of $\mathcal{T}_1$ and $\mathcal{T}_2$, it must be true that $\mathcal{T}_2 \models P \sqsubseteq P_2$, and consequently $\mathcal{T}_{2d} \models P \sqsubseteq P_2$.

□

Armed with the above results, we now establish the following:

Proposition 5.55. *Suppose that $d = \mathrm{DL}[\lambda; Q](\vec{X})$ is a DL-atom, and $\Sigma = \mathit{sig}(\mathcal{A} \cup \mathcal{A}_d\} \cup \{Q\} \cup \{P \mid P \circ p \in \lambda\})$, where $\mathcal{A}_d$ is the set of all input assertions for d. If $\mathcal{EL}$-terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ are Σ-concept inseparable, then complete nonground support families for d w.r.t. $\mathcal{O}_1 = \langle \mathcal{T}_1, \mathcal{A} \rangle$ and $\mathcal{O}_2 = \langle \mathcal{T}_2, \mathcal{A} \rangle$ coincide.*

Proof. Towards a contradiction, suppose that d has different complete nonground support families w.r.t. $\mathcal{O}_1$ and $\mathcal{O}_2$. Let us call them $\mathcal{S}_{\mathcal{T}_1}$ and $\mathcal{S}_{\mathcal{T}_2}$ respectively. Assume w.l.o.g. that there is some nonground support set S, such that $S \in \mathcal{S}_{\mathcal{T}_1}$, but $S \notin \mathcal{S}_{\mathcal{T}_2}$.

Since S is not in $\mathcal{S}_{\mathcal{T}_2}$, it holds that there exist some $\vec{c}$ and N_{gr}, such that $\gamma(\vec{c}, N_{gr}) = 1$, but N_{gr} is not a ground support set for $d(\vec{c})$ with respect to $\mathcal{O}_2$. Therefore, there exists an interpretation $I \supseteq N_{gr}$, such that $I \not\models^{\mathcal{O}_2} d(\vec{c})$, but $I \models^{\mathcal{O}_1} d(\vec{c})$. Applying the shifting lemma, we have $\mathcal{T}_{1d} \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\} \models Q(\vec{c})$, and $\mathcal{T}_{2d} \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\} \not\models Q(\vec{c})$.

By Lemma 5.54 we know that $\mathcal{T}_{2d}$ and $\mathcal{T}_{1d}$ are Σ-concept inseparable. Thus, by Theorem 5.53 they must be Σ-instance inseparable as well. By definition of Σ-instance inseparability, we have that for all Σ-ABoxes $\mathcal{A}'$ and Σ-assertions α, such that $\mathcal{T}_{1d} \cup \mathcal{A}' \models \alpha$, it holds that $\mathcal{T}_{2d} \cup \mathcal{A}' \models \alpha$. Hence, it must be true that $\mathcal{T}_{2d} \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\} \models Q(\vec{c})$, and therefore, $I \models^{\mathcal{O}_2} d$, leading to a contradiction. This shows that S is a nonground support set for d w.r.t. $\mathcal{O}_2$, which proves the statement. □

It has been shown in [KLWW12], that every inclusion $C \sqsubseteq D$ in the Σ-concept difference of $\mathcal{T}_1$ and $\mathcal{T}_2$ "contains" a basic witness inclusion that has a concept name either on the left or on the right hand side. More formally,

Theorem 5.56 (cf. [KLWW12])**.** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be $\mathcal{EL}$-terminologies and Σ a signature. If $\phi \in \mathit{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$, then either $C \sqsubseteq A$ or $A \sqsubseteq D$ is a member of $\mathit{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$, where $A \in \mathit{sig}(\phi)$ is a concept name and C and D are $\mathcal{EL}$-concepts occurring in ϕ. The sets of such inclusions are called left and right witnesses and denoted as $\mathit{cWTn}^{rhs}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ and $\mathit{cWTn}^{lhs}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ respectively.*

The logical difference between two $\mathcal{EL}$-terminologies in its compact representation consists only of inclusions which have an atomic concept name either on the left or on

the right hand side. Among them there might be some inclusions with atomic concepts on both sides or role restrictions of the form $\exists r.\top$, which fall into our target language of *DL-Lite*$_\mathcal{A}$ DL. This succinct representation of the logical difference is very beneficial for our needs, as it ensures that for some TBoxes the simplification of the $\mathcal{EL}$ axioms to *DL-Lite*$_\mathcal{A}$ can be expected.

To sum up, our approach of computing support families, for which each support set is of a bounded size is as follows: Given a DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{X})$ and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is an $\mathcal{EL}$-terminology,

(1). Construct the TBox $\mathcal{T}_d$;

(2). Construct a simplified TBox $\mathcal{T}_{simp}$ by removing from $\mathcal{T}_d$ all axioms of the form $C \sqsubseteq D$, where C or D is a concept not belonging to *DL-Lite*$_\mathcal{A}$ language;

(3). Compute right-hand side and left-hand side witnesses $RH = \mathsf{cWTn}^{rhs}_{\Sigma}(\mathcal{T}_d, \mathcal{T}_{simp})$ and $LH = \mathsf{cWTn}^{lhs}_{\Sigma}(\mathcal{T}_d, \mathcal{T}_{simp})$ between $\mathcal{T}_d$ and $\mathcal{T}_{simp}$, where $\Sigma = \mathsf{sig}(\mathcal{A}) \cup \{Q\} \cup \{P \mid P \circ p \in \lambda\}$;

(4). Obtain the TBox $\mathcal{T}'_{simp}$ by adding to the TBox $\mathcal{T}_d$ all witnesses of the form $A \sqsubseteq B$, where either (i) A and B are atomic, or A resp. B is a role restriction of the form $\exists r.\top$;

(5). Compute a complete support family $\mathbf{S}$ for d over $\mathcal{T}'_{simp}$. The family $\mathbf{S}$ is a partial support family for d over $\mathcal{T}_d$.

If in (3) we get that RH and LH are empty or they contain only inclusions falling into the *DL-Lite*$_\mathcal{A}$ language, then by the results of Proposition 5.55 the computed support family is guaranteed to be complete.

5.7 $\mathcal{EL}$ TBox Restrictions Ensuring Bounded Support Families

It is a relevant and an interesting quest to decide what are the syntactic conditions that ensure that maximal support set size for a given DL-atom is bounded by a certain constant n. Similarly, an important issue is to estimate the number of support sets in the smallest complete support family. In what follows we analyze these problems.

5.7.1 Support Set Size

To start with, recall from [KLWW12], that an atomic concept A is *primitive* in $\mathcal{T}$, if it does not occur on the left-hand side of axioms in $\mathcal{T}$. Moreover, A is *pseudo-primitive*, if it is either primitive or occurs on the left hand side of axioms of the form $A \sqsubseteq C$, where C is an arbitrary $\mathcal{EL}$ concept.

Proposition 5.57. *Let $d = \mathrm{DL}[\lambda; Q](\vec{t})$ be a DL-atom, and let $\mathcal{T}$ be an $\mathcal{EL}$-terminology. Then if Q is pseudo-primitive in $\mathcal{T}$, then $maxsup(d) = 1$.*

Proof. The proof of this proposition is based on the following theorem [KLWW12].

Theorem 5.58. *Let $\mathcal{T}$ be a normalized $\mathcal{EL}$-terminology, and let A be a concept name. Assume that $\mathcal{T} \models D \sqsubseteq A$, where $D = \bigsqcap_{1 \leq i \leq n} A_i \sqcap \bigsqcap_{1 \leq j \leq m} \exists r_j.C_j$ and A is a pseudo-primitive concept. Then some $1 \leq i \leq n$ exists, such that $\mathcal{T} \models A_i \sqsubseteq A$, where A_i is atomic.*

Intuitively, the theorem states that regardless of how complex the expression D in the inclusion $D \sqsubseteq A$ is, if A is pseudo primitive and $\mathcal{T} \models D \sqsubseteq A$, then there is always an atomic conjunct A' involved in D, such that $\mathcal{T} \models A' \sqsubseteq A$. This essentially means that all $\subseteq$-minimal support sets for A are then unary, which proves the statement. □

Proposition 5.57 puts some restrictions on the structure of the TBox (it must be an $\mathcal{EL}$-terminology) as well as on the DL-query Q for a DL-atom d. It exploits a specific case, in which the support set size bound is 1. For providing more liberal syntactic conditions on $\mathcal{T}$ that ensure a certain bound n on the size of support sets, we use the notion of an *ontology hypergraph* [NBM13, ELW13].

Ontology hypergraphs have been widely studied for extracting reachability based modules of ontologies [NBM13], determining concept difference between $\mathcal{EL}$ terminologies [ELW13]), efficient reasoning [LSS13a]in OWL 2 QL and other important tasks.

First let us recall the notion of a directed hypergraph, which is a natural generalization of the concept of a directed graph, first proposed in [ADS83] to represent functional dependencies in relational data base schemata. Directed hypergraphs are often used to model general types of functional relations, such as many-to-one (one-to-many) relations.

Definition 5.59. A *directed hypergraph* is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E}$ is a set of directed *hyperedges* of the form $e = (H, H')$, such that $H, H' \subseteq \mathcal{V}$ are nonempty sets called *hypernodes*.

Given a hyperedge $e = (H, H')$, we call H the *tail* of e, and H' the *head* of e, denoted by $\mathit{tail}(e)$ and $\mathit{head}(e)$ respectively. In this work we restrict ourselves to hypergraphs, which contain only hypernodes H, such that $|H| \leq 2$. We call a hypernode a *singleton*, if $|H| = 1$, and a *binary hypernode*, if $|H| = 2$.

We now recall the definition of an ontology hypergraph for DL $\mathcal{EL}$ introduced in [ELW13].

Definition 5.60 (cf. [ELW13])**.** Let $\mathcal{T}$ be an $\mathcal{EL}$ TBox in a normal form, and let $\Sigma \subseteq \mathbf{C} \cup \mathbf{R}$. The *ontology hypergraph* $\mathcal{G}_\mathcal{T}^\Sigma$ of $\mathcal{T}$ is a directed hypergraph $\mathcal{G}_\mathcal{T}^\Sigma = (\mathcal{V}, \mathcal{E})$ defined as follows:

$$\mathcal{V} = \{x_A \mid A \in \mathbf{C} \cap (\Sigma \cap \mathsf{sig}(\mathcal{T}))\} \cup \{x_r \mid r \in \mathbf{R} \cap (\Sigma \cap \mathsf{sig}(\mathcal{T}))\} \cup \{x_\top\}$$

$$\begin{aligned}\mathcal{E} = {} & \{(\{x_A\}, \{x_B\}) \mid A \sqsubseteq B \in \mathcal{T}, 1 \leq i \leq n\} \\ & \cup \{(\{x_A\}, \{x_r, x_Y\}) \mid A \sqsubseteq \exists r.Y \in \mathcal{T}, Y \in \mathbf{C} \cup \{\top\}\} \\ & \cup \{(\{x_r, x_Y\}, \{x_A\}) \mid \exists r.Y \sqsubseteq A \in \mathcal{T}, Y \in \mathbf{C} \cup \{\top\}\} \\ & \cup \{(\{x_{B_1}, x_{B_2}\}, \{x_A\}) \mid B_1 \sqcap B_2 \sqsubseteq A \in \mathcal{T}\}\end{aligned}$$

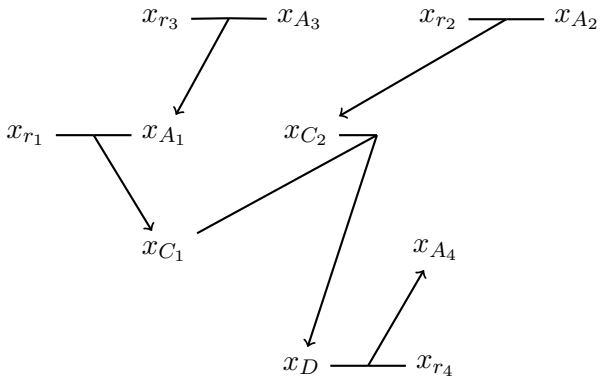


Figure 5.3: Hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$

A node of the form $\{x, y\}$ in a hypergraph corresponds to an unordered pair x, y. Thus $\{x, y\}$ and $\{y, x\}$ refer to the same node.

Example 5.61. Consider the following TBox in a normal form:

$$\mathcal{T} = \left\{ \begin{array}{ll} (1)\ \exists r_1.A_1 \sqsubseteq C_1 & (4)\ C_1 \sqcap C_2 \sqsubseteq D \\ (2)\ \exists r_2.A_2 \sqsubseteq C_2 & (5)\ A_3 \sqsubseteq A_2 \\ (3)\ \exists r_3.A_3 \sqsubseteq A_1 & (6)\ D \sqsubseteq \exists r_4.A_4 \end{array} \right\}$$

Suppose that $\Sigma{=}\mathsf{sig}(\mathcal{T})$. The ontology hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ is depicted in Figure 5.3. □

We now define the notion of an *incoming path* to a singleton node in an ontology hypergraph, which is a natural generalization of a path in a standard graph.

Definition 5.62. Suppose that $\mathcal{T}$ is an $\mathcal{EL}$ TBox in a normal form, $\mathcal{G}_{\mathcal{T}}^{\Sigma} = (\mathcal{V}, \mathcal{E})$ is an ontology hypergraph, and $x_Q \in \mathcal{V}$ is a singleton node occurring in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$. Then an *incoming path* to x_Q is a sequence $path = \langle e_1, e_2, \ldots, e_n \rangle$ of hyperedges of $\mathcal{G}_{\mathcal{T}}^{\Sigma}$, such that:

- $head(e_i) \neq head(e_{i'})$ and $tail(e_i) \neq tail(e_{i'})$ for all pairs e_i, $e_{i'}$, where $i \neq i'$,
- $head(e_n) = \{x_Q\}$,
- for every e_i, where $i < n$ one of the following holds:
  - $tail(e_i) = \{x_{P_1}\}$, $head(e_i) = \{x_{P_2}\}$, and either $tail(e_j) = \{x_{P_2}\}$ or $tail(e_j) = \{x_{P_2}, x_{P_3}\}$;
  - $tail(e_i) = \{x_{P_1}\}$, $head(e_i) = \{x_{P_2}, x_{P_3}\}$, and $tail(e_j) = \{x_{P_2}, x_{P_3}\}$;
  - $tail(e_i) = \{x_{P_1}, x_{P_2}\}$, $head(e_i) = \{x_{P_3}\}$, and either $tail(e_j) = \{x_{P_3}\}$ or $tail(e_j) = \{x_{P_3}, x_{P_4}\}$,

where $P_k \in \Sigma$, and j is some index, such that $i < j \leq n$.

Intuitively, hyperedges in an ontology hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ model inclusion relations between (complex) concepts over the signature Σ in the TBox $\mathcal{T}$. Consequently, an incoming path to a singleton node x_C in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ models a chain of inclusions that logically follow from $\mathcal{T}$, such that C is the most right element of the chain.

We now demonstrate this observation by the following example.

Example 5.63. Let us look at the ontology hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ depicted in Figure 5.3. As an example of an incoming path to x_{C_1} in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ consider the following sequence of edges:

$$path_1 = \langle(\{x_{r_3}, x_{A_3}\}, x_{A_1}), (\{x_{r_1}, x_{A_1}\}, x_{C_1})\rangle.$$

The path $path_1$ reflects inclusions:

- $\exists r_1.A_1 \sqsubseteq C_1$;
- $\exists r_1.(\exists r_3.A_3) \sqsubseteq C_1$.

For a singleton node x_D we have that an incoming path in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ to x_D is as follows:

$$path_2 = \langle(\{x_{r_3}, x_{A_3}\}, x_{A_1}), (\{x_{r_1}, x_{A_1}\}, x_{C_1}), (\{x_{r_2}, x_{A_2}\}, x_{C_2}), (\{x_{C_1}, x_{C_2}\}, x_D)\rangle.$$

The following set of inclusions can be extracted from $path_2$:

- $C_1 \sqcap C_2 \sqsubseteq D$;
- $\exists r_2.A_2 \sqcap C_1 \sqsubseteq D$;
- $\exists r_2.A_2 \sqcap \exists r_1.A_1 \sqsubseteq D$;
- $\exists r_2.A_2 \sqcap \exists r_1.(\exists r_3.A_3) \sqsubseteq D$.

□

We now introduce our novel notion of a support hypergraph for a DL-atom.

Definition 5.64. A *support hypergraph* $\mathcal{G}_{\mathcal{T},sup(d)}^{\Sigma}$ for a DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a hypergraph constructed in the following way:

- build an ontology hypergraph $\mathcal{G}_{\mathcal{T}_d}^{\Sigma}$, where $\Sigma = \mathsf{sig}(\mathcal{A} \cup \mathcal{A}_d) \cup \{Q\}$;
- leave nodes and edges participating in incoming paths to x_Q, and remove all other nodes and edges;
- for all x_C in the obtained graph, such that $C \notin \Sigma$ check if there are any incoming paths to x_C. If there are no such paths, then remove x_C from the graph together with the edges in which x_C participates. If there are incoming paths to x_C, then check whether at least one of the incoming paths contains a hypernode N, such that for all $x_P \in N$, it holds that $P \in \Sigma$; if this is the case, then leave the node x_C, otherwise remove it with its corresponding edges.

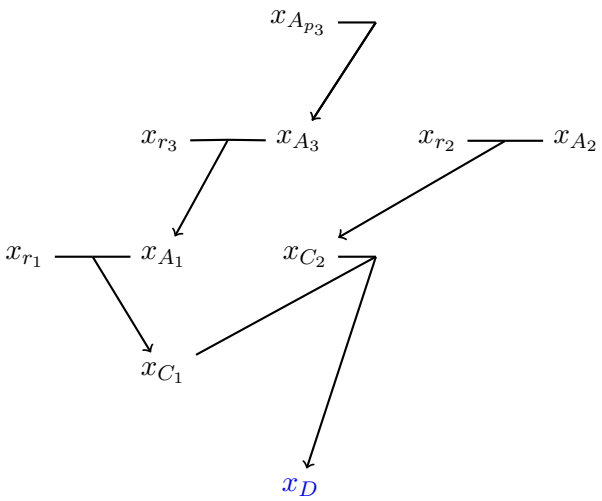


Figure 5.4: Support hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ from Example 5.65

Let us illustrate the notion of a support hypergraph on the following example:

Example 5.65. Let the TBox $\mathcal{T}$ from Example 5.61 be accessed by the DL-atom $d = \mathrm{DL}[A_3 \uplus p_3; D](\vec{X})$. The TBox $\mathcal{T}_d$ then contains axioms from $\mathcal{T}$ and the inclusion $A_{3_{p_3}} \sqsubseteq A_3$ mimicking the update of d. The support graph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ for d with $\Sigma = \mathsf{sig}(\mathcal{T}_d)$ is presented in Figure 5.4. The edge $(\{x_D\}, \{x_{r_4}, x_{A_4}\})$ is not present in $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$, since it does not lie on the incoming path to x_D. The node x_D is marked with a blue color, to highlight its correspondence to the DL-query of d. □

The support graph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)} = (\mathcal{V}, \mathcal{E})$ for a DL-atom $d = \mathrm{DL}[\lambda; Q](X)$ contains all incoming paths to x_Q, which start from nodes corresponding to predicates in $\mathcal{A} \cup \mathcal{A}_d$ by construction. Therefore, the graph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ reflects all inclusions with Q on the right-hand side and predicates over $\mathcal{A}_d \cup \mathcal{A}$ on the left hand-side that are entailed from $\mathcal{T}_d$. Hence, by traversing edges of every incoming path *path* to x_Q in $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)} = (\mathcal{V}, \mathcal{E})$ we can construct all query rewritings of Q over the TBox $\mathcal{T}_d$ which are of relevance for an ontology at hand, and which correspond to support sets. Thus, we obtain:

Lemma 5.66. *All nonground support sets over $\mathsf{sig}(\mathcal{A} \cup \mathcal{A}_d \cup Q)$ for a DL-atom d can be constructed from the hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ for $d = \mathrm{DL}[\lambda; Q](\vec{X})$ w.r.t. the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$.*

Proof (sketch). To prove the statement we provide an approach for constructing non-ground support sets from a given hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ for d w.r.t. the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$.

The approach is based on annotation of nodes of the hypergraph with variables X_i, such that $i \in \mathbb{N}$. We start from the node x_Q, which we annotate with X_0; then we

traverse the hypergraph backwards, going from a head of an edge to its tail. For every edge e that we encounter we annotate $\mathit{tail}(e)$ based on its form and on the annotation of $\mathit{head}(e)$, with variable names that occur in annotation of $\mathit{head}(e)$ and/or fresh variable names X_i, such that $i \in \mathbb{N}$ in the following way:

(1) if $|\mathit{tail}(e)| = 1$ then

(1.1) if $\mathit{head}(e) = \{x_{C_1}\}$, then $\mathit{tail}(e)$ is annotated with the same variable name as $\mathit{head}(e)$;

(1.2) if $\mathit{head}(e) = \{x_{r_1}, x_{C_1}\}$ and it is annotated with $\{\langle X_{i_1}, X_{i_2}\rangle, X_{i_3}\}$, then $\mathit{tail}(e)$ is annotated with X_{i_1};

(2) if $|\mathit{tail}(e)| = 2$ and $\mathit{head}(e) = \{x_C\}$, such that $\mathit{head}(e)$ is annotated with $\{X_i\}$, then

(2.1) if $\mathit{tail}(e) = \{x_{C_1}, x_{C_2}\}$, where $C_1, C_2 \in \mathbf{C}$, then the hypernode $\{x_{C_1}, x_{C_2}\}$ is annotated with $\{X_i, X_i\}$;

(2.2) if $\mathit{tail}(e) = \{x_{r_1}, x_{C_1}\}$, where $r_1 \in \mathbf{R}$, $C_1 \in \mathbf{C}$, then $\{x_{r_1}, x_{C_1}\}$ is annotated with $\{\langle X_i, X_{i'}\rangle, X_i\}$, where $X_{i'}$ is a fresh variable name not yet used in annotating hypernodes.

From every annotated hypernode N one can create a set of nonground atoms by identifying predicate names from labels of hypernodes and variable names from their annotations. The nonground support sets for $d = \mathrm{DL}[\lambda; Q](X_0)$ can then be extracted from all subpaths of incoming paths to x_Q. We pick some subpath path_1 to x_Q containing n edges. We start from the edge e_n, such that $\mathit{head}(e_n) = \{x_Q\}$. The support set S_1 for d is extracted from the annotated tail of e_n. We then pick an edge e_k, such that $\mathit{head}(e_k) \subseteq \mathit{tail}(e_n)$, and obtain further support sets by substituting nonground atoms that correspond to $\mathit{head}(e_k) \cap \mathit{tail}(e_n)$ in S_1 with the nonground atoms extracted from $\mathit{tail}(e_k)$. This process continues, and at each stage we pick a new edge from the subpath, such that its head intersects with a tail of at least one edge that has already been processed, and the new support sets are extracted accordingly.

The construction that we have presented mimics the DL-query unfolding over the TBox $\mathcal{T}_d$. We now formally show that (i) the sets extracted in the way as described above are indeed nonground support sets for d, and (ii) all nonground support sets for d can be constructed following our procedure.

We first prove (i) by induction on the length n of an incoming path, from which the support sets are extracted.

Base: n=1. Let us pick some path *path* in the hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$. Assume that there is a single (hyper-)edge e in *path*. By construction this hyperedge must have x_Q as a head node, i.e. $\mathit{head}(e) = x_Q$. There are four possibilities: (1) $\mathit{tail}(e) = \{x_C\}$, (2) $\mathit{tail}(e) = \{x_r, x_C\}$, (3) $\mathit{tail}(e) = \{x_C, x_D\}$ or (4) $\mathit{tail}(e) = \{x_r, \top\}$. We annotate the nodes of a path by variables as described above, and extract the nonground atoms from labels and annotations of the nodes. As a result for the case (1) we obtain $\{C(X_0)\}$,

for (2): $\{r(X_0, X_1), C(X_1)\}$, for (3): $\{C(X_0), D(X_0)\}$, and for (4): $\{r(X_0, X_1)\}$, where X_1 is a fresh variable. By construction of the hypergraph the edges of the forms (1)-(4) correspond to the TBox axioms $C \sqsubseteq Q$, $\exists r.c \sqsubseteq Q$, $C \sqcap D \sqsubseteq Q$ and $\exists r.\top \sqsubseteq Q$ respectively. Therefore, the sets that have been constructed in all of the considered cases naturally reflect the DL-query unfoldings of d, and hence they represent nonground support sets for d.

Induction hypothesis: Suppose that the statement is true for n, that is from a path of length n all sets extracted in the way as described above are nonground support sets for d. Consider a path *path* of length $n + 1$. Pick the first edge e on *path*. By the induction hypothesis, we have that all sets extracted from the path $path \setminus \{e\}$ following our approach are support sets for d. There are several possibilities for the form of e: (1) $tail(e) = \{x_C\}$ and $head(e) = \{x_D\}$, (2) $tail(e) = \{x_r, x_C\}$ and $head(e) = \{x_D\}$, (3) $tail(e) = \{x_C, x_D\}$ and $head(e) = \{x_B\}$, (4) $tail(e) = \{x_r, \top\}$ and $head(e) = \{x_C\}$, or (5) $tail(e) = \{x_C\}$ and $head(e) = \{x_r, x_D\}$.

For (1) it holds that by construction x_C and x_D are annotated with the same variable X_i. Let $\mathcal{S}$ be a family of sets extracted from $path \backslash \{e\}$. We pick a set S, in which $C(X_i)$ occurs. We substitute $C(X_i)$ in S with $D(X_i)$, and obtain a set S'. It must hold that S is a support set for d, since it has been extracted from a path of length n in $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$. However, then clearly S' is a support set too, as it mimics an additional unfolding step, which takes into account the rule $C(X) \leftarrow D(X)$ of the datalog rewriting of $\mathcal{T}_d$.

Let us look at (2). Assume that there is a set of nonground atoms S, such that $D(X_i) \subseteq S$, and such S has been constructed using our procedure. It must hold that X_i is an annotation for x_D. According to our construction $\{x_r, x_D\}$ is annotated with $\{\langle X_i, X_j\rangle, \{X_j\}\}$, where X_j is a fresh variable. The sets S' that we get from *path*, are obtained by substitution of $D(X_i)$ in some S by $\{r(X_i, X_j), C(X_j)\}$. The latter mimics the unfolding step for Q that takes into account the rule $D(X_i) \leftarrow r(X_i, X_j), C(X_j)$ of the rewriting $\mathcal{T}_d$. As S is a support set for d by our induction hypothesis, S' must be a support set for d as well. The cases (3)-(5) can be analyzed analogously. Therefore, all sets that are extracted from *path* of size $n + 1$ are support sets for d.

It is left to prove (ii). Towards a contradiction assume that some support set S for d exists, such that S was not identified during our procedure for extracting support sets from $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$. Since S is a support set, it must correspond to some query unfolding of Q, obtained from some inclusion $C \sqsubseteq Q$, which logically follows from $\mathcal{T}_d$, where C is a possibly complex concept. If we have that $C \sqsubseteq Q \in \mathcal{T}_d$ then by construction of a support hypergraph, there is a path *path* consisting of a single edge e in $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$, reflecting this inclusion. Since $head(e) = x_Q$, it must hold that some support set S' is extracted from e as a result of our procedure. Observe now that the inclusion that we consider corresponds only to a single query unfolding. Therefore, S and S' must coincide, leading to a contradiction.

Suppose now that $\mathcal{T}_d \models C \sqsubseteq Q$, but $C \sqsubseteq Q \notin \mathcal{T}_d$. The inclusion $C \sqsubseteq Q$ is reflected by some incoming path *path* to x_Q in $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$, such that for every $P \in \mathsf{sig}(C)$ there is a node $N \in path$ with x_P occurring in the label of N. Since our construction mimics unfolding steps for the query Q, there must be some set computed by our procedure,

which reflects the query unfolding to which S corresponds, i.e. contradiction. □

According to Lemma 5.66 one can determine all relevant nonground support sets for a DL-atom d w.r.t. ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ by just looking at the support hypergraph of d. We now illustrate the construction exploited in the proof of the lemma by the following example.

Example 5.67. Consider the hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ from Figure 5.4, which contains the following path to x_D:

$$path = \langle \underbrace{(x_{A_{3_{p_3}}}, x_{A_3})}_{e_1}, \underbrace{(\{x_{r_3}, x_{A_3}\}, x_{A_1})}_{e_2}, \underbrace{(\{x_{r_1}, x_{A_1}\}, x_{C_1})}_{e_3}, \underbrace{(\{x_{r_2}, x_{A_2}\}, x_{C_2})}_{e_4}, \underbrace{(\{x_{C_1}, x_{C_2}\}, x_D)}_{e_5} \rangle,$$

We annotate x_D with X_0 and traverse the path backwards starting from x_D. We consider the edges of *path* in the reverse order as follows: e_5, e_4, e_3, e_2, e_1 and arrive at the following annotation of *path*:

$$\langle \underbrace{(\overbrace{x_{A_{3_{p_3}}}}^{\{X_3\}}, \overbrace{x_{A_3}}^{\{X_3\}})}_{e_1}, \underbrace{(\overbrace{\{x_{r_3}, x_{A_3}\}}^{\{\langle X_2, X_3\rangle, X_3\}}, \overbrace{x_{A_1}}^{\{X_2\}})}_{e_2}, \underbrace{(\overbrace{\{x_{r_1}, x_{A_1}\}}^{\{\langle X_0, X_2\rangle, X_2\}}, \overbrace{x_{C_1}}^{\{X_0\}})}_{e_3}, \underbrace{(\overbrace{\{x_{r_2}, x_{A_2}\}}^{\{\langle X_0, X_1\rangle, X_1\}}, \overbrace{x_{C_2}}^{\{X_0\}})}_{e_4}, \underbrace{(\overbrace{\{x_{C_1}, x_{C_2}\}}^{\{X_0, X_0\}}, \overbrace{x_D}^{\{X_0\}})}_{e_5} \rangle.$$

The nonground support sets for d can now be read from the annotated path.

- We start with the subpath $path_1 = \langle e_5 \rangle$ and extract from $path_1$ the first support set $S_1 = \{C_1(X_0), C_2(X_0)\}$.
- We pick the next subpath $path_2 = \langle e_4, e_5 \rangle$, as $head(e_4) \subseteq tail(e_5)$, and obtain the following additional support set: $S_2 = \{r_2(X_0, X_1), A_2(X_0, X_1), C_2(X_0)\}$.
- $S_3 = \{C_1(X_0), r_1(X_0, X_2), A_1(X_2)\}$ is then extracted from $path_3 = \langle e_3, e_5 \rangle$.
- $path_4 = \langle e_5, e_4, e_3 \rangle$ yields $S_4 = \{r_2(X_0, X_1), A_2(X_1), r_1(X_0, X_2), A_1(X_2)\}$.
- We get $S_5 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_3(X_3), C_2(X_0)\}$ from $path_5 = \langle e_2, e_3, e_5 \rangle$.
- $S_6 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_3(X_3), r_2(X_0, X_1), A_2(X_1)\}$ is then extracted from $path_6 = \langle e_2, e_3, e_4, e_5 \rangle$.
- The support set $S_7 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_{3_{p_3}}(X_3), C_2(X_0)\}$ is determined from $path_7 = \langle e_1, e_2, e_3, e_5 \rangle$.
- Finally, $S_8 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_{3_{p_3}}(X_3), r_2(X_0, X_1), A_2(X_1)\}$ is constructed from the subpath $path_8 = \langle e_1, e_2, e_3, e_4, e_5 \rangle$.

□

Lemma 5.66 allows one to reason about the structure and size of support sets by analyzing only parameters of the hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$. As a possible such parameter, one can consider, for instance, the number of hyperedges with a singleton head node. Then we obtain:

Proposition 5.68. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{EL}$ ontology, where $\mathcal{T}$ is in normal form, and let $d = \mathrm{DL}[\lambda; Q](\vec{X})$ be a DL-atom with a support hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$. Then $maxsup(d) \leq n + 1$, where n is the maximal number of hyperedges with a binary tail node and a singleton head node located on an incoming path to x_Q, excluding hyperedges of the form $(\{x_r, \top\}, x_C)$.*

Proof. We prove the statement by induction on the number n of hyperedges with a singleton head node in the support graph.

Base: $n = 0$. We show that if there are no hyperedges of the required form in the hypergraph, then $maxsup(d) = 1$. By definition there are several possibilities for $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$: (i) either it does not contain any hyperedges, or (ii) all hyperedges that occur in $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ are of the form $(\{x_r, \top\}, x_C)$, or (iii) there are hyperedges of the form $(x_C, \{x_r, x_D\})$ in the hypergraph.

For (i) and (ii) we have that by construction the TBox $\mathcal{T}$ contains only concept inclusions $C \sqsubseteq D$, such that C, D are either atomic or of the form $\exists r.\top$. These axioms fall into the *DL-Lite*$_{\mathcal{A}}$ fragment, for which support sets are either unary or binary. Observe, that according to Definition 5.24 all binary support sets in the *DL-Lite*$_{\mathcal{A}}$ case can only occur, if inconsistency arises in the updated ontology. Since the negation is neither allowed nor expressible in $\mathcal{EL}$, we get that the support sets must be maximum of size 1.

Finally, it is left to analyze the case (iii). Consider a hyperedges e_i in $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$, such that $head(e_i) = \{x_r, x_C\}$. By definition of an incoming path we have that some e_j must exist such that $tail(e_j) = head(e_i)$. The latter, however means that e_j is a hyperedge with a singleton head node, which contradicts our assumption.

Induction Step: Suppose that the statement is true for n, we prove it for $n+1$. Let $p = e_1, \ldots, e_n, e_{n+1}$ be an incoming path to x_Q in the hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ with a maximal number $n + 1$ of hyperedges with singleton head nodes. We have that $head(e_{n+1}) = x_Q$. Assume that e_i is the first hyperedge of the required form occurring on the path p. Let us split the path p into two parts: $e_1, \ldots, e_i$ and $e_{i+1}, \ldots, e_n$. Consider the graph $\mathcal{G}'' = \{\mathcal{V}, \mathcal{E}''\}$, where $\mathcal{E}'' = \mathcal{E} \setminus \{e_1, \ldots, e_i\}$. The maximal number of hyperedges occurring on an incoming path to x_Q in $\mathcal{G}''$ is n by construction. Therefore, by the induction hypothesis, $maxsup(d)$ w.r.t. $\mathcal{O}'' = \langle \mathcal{T}'', \mathcal{A} \rangle$ is bounded by $n + 1$. Consider the graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$, where $\mathcal{E}' = \mathcal{E} \cup \{e_i\}$, and let $\mathcal{T}'$ be a TBox corresponding to $\mathcal{G}'$. We have that $head(e_i) = x_A$, since by our assumption e_i is a hyperedge with a singleton node. There are two possibilities: either $A = Q$ or $A \neq Q$. In the former case we have that e_i is a single hyperedge of the required form on the path p. It either corresponds to $C \sqcap D \sqsubseteq Q$ or to $\exists r.C \sqsubseteq Q$. In both cases the support sets for d are bounded by 2.

If $A \neq Q$, then by definition of an incoming path there exists a single (hyper)edge e_j, such that either (i) $head(e_i) = tail(e_j)$ or (ii) $head(e_i) \subseteq tail(e_j)$ for some $j > i$. If $head(e_i) = tail(e_j)$ then e_j must correspond to an axiom with an atomic concept on the left hand side, i.e. some inclusion of the form $A \sqsubseteq \ldots$ occurs in $\mathcal{T}'$. If $head(e_i) \subseteq tail(e_j)$ for some $j > i$ then e_j corresponds either to $A \sqcap B \sqsubseteq \ldots$ or to $\exists r.A \sqsubseteq \ldots$. In all of

these cases, the query unfolding for Q w.r.t. $\mathcal{T}''$, which involves atoms over A is of size at most $n+1$ by our induction hypothesis. We can obtain a query unfolding U for Q w.r.t. $\mathcal{T}'$ by rewriting atoms over A to the respective atoms over C and D, if $e_i = \{x_C, x_D\}$ or over r and C, if $e_i = \{x_r, x_C\}$. This will result in a query unfolding of size at most $n+2$. This means that the support set size in $\mathcal{G}'$ is bounded by $n+2$.

Finally, let us look at $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)} = \mathcal{G}' \cup \{e_1, \ldots, e_{i-1}\}$. We know that the rewritings for all of the concepts in $\mathcal{T}$ corresponding to the hypernodes of the path $e_1, \ldots, e_{i-1}$ are of size 1, which follows from the base case. That means that the support sets for d w.r.t. $\mathcal{T}$ can not be larger then those with respect to $\mathcal{T}'$, and hence they will be bounded by $n+2$.

□

The bound provided by the above proposition is not tight, which is obvious from the following example.

Example 5.69. Consider the DL-atom $d(X) = \mathrm{DL}[;Q](X)$ accessing the TBox $\mathcal{T}_d$:

$$\mathcal{T}_d = \left\{ \begin{array}{ll} (1)\ A \sqcap D \sqsubseteq F & (4)\ E \sqcap F \sqsubseteq L \\ (2)\ A \sqcap C \sqsubseteq K & (5)\ E \sqcap K \sqsubseteq M \\ (3)\ A \sqcap B \sqsubseteq E & (6)\ M \sqcap L \sqsubseteq Q \end{array} \right\}$$

Let $\Sigma = \mathsf{sig}(\mathcal{T}_d)$. The support hypergraph for d is depicted in Figure 5.5. There are 6 hyperedges with single head nodes, but the maximal support set size for $d(X)$ is 4, e.g. $S = \{A(X), B(X), D(X), K(X)\}$. □

Before providing a more accurate upper bound on the support set size, we first define the notion of a *hyper outdegree* for a node as follows:

Definition 5.70. Given a support hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ and a node x, we say that x has a *hyper out-degree* k, denoted by $hd^+(x) = k$, if it has k outgoing hyperedges of the form $(\{x, y\}, z)$. We denote by $HN(\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)})$ the set of nodes x in a graph, such that $hd^+(x) \geq 1$.

Example 5.71. All nodes $X \subseteq \mathcal{V} \backslash \{x_{A_{3_p}}, x_{A_3}\}$ in the graph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ depicted in Figure 5.4 have hyper out-degree 1. For $\mathcal{G}^{\Sigma}_{\mathcal{T},supp(d)} \cup (\{x_{C_2}, x_{A_2}\}, x_D)$, we have $hd^+(x_{C_2}) = hd^+(x_{A_2}) = 2$. For the graph in Figure 5.5 one can see that $hd^+(x_E) = 2$ and $hd^+(x_A) = 3$. □

Let us now denote a set of incoming paths to a node x in a hypergraph $\mathcal{G}$ by $Inpaths(x)$, and let $s_{max}(x) = max_{path \in Inpaths(x)}(n(path) - m + 1)$, where

- $n(path)$ is the number of hyperedges on $path$ with a singleton head node excluding those of the form $(\{\top, x_r\}, x_C)$, and
- $m = \sum_{x_C \in HN(path)}(hd^+(x_C) - 1)$, where x_C lies on $path$, and $hd^+(x_C)$ is the number of hyperedges of the form $(\{x_C, x_{C'}\}, x_{C''})$, which are outgoing from x_C and located on $path$.

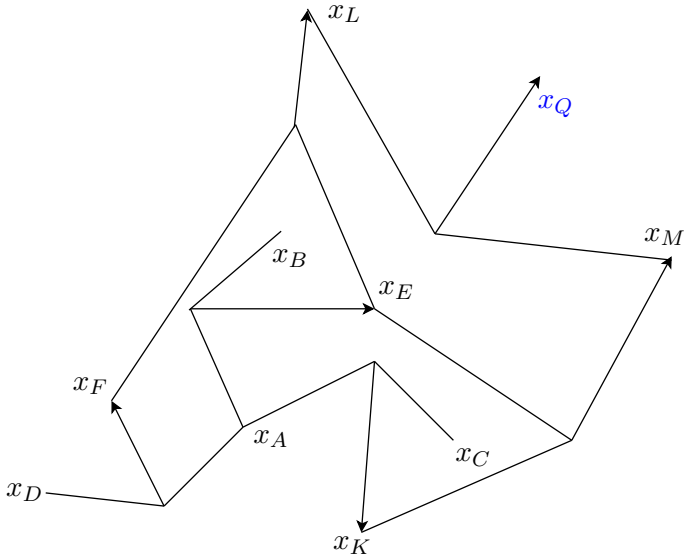


Figure 5.5: Support hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},supp(d)}$ from Example 5.69

Example 5.72. Consider the hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ from Figure 5.4. The set $Inpaths(x_D)$ contains a single path: $path = \langle(\{x_{r_2}, x_{A_2}\}, x_{C_2}), (\{x_{r_1}, x_{A_1}\}, x_{C_1}), (\{x_{C_1}, x_{C_2}\}, x_D)\rangle$. There are 3 hyperedges with a singleton head node occurring in $path$, hence $n(path) = 3$. Furthermore, $m = 0$, as all nodes have a hyper outdegree at most 1, therefore $s_{max}(x_Q) = 3 - 0 + 1 = 4$. The hypergraph in Figure 5.5 contains also a single incoming $path$ to Q. It holds that $n(path) = 6$, $m = (hd^+(A) - 1) + (hd^+(E) - 1) = 3$, and thus $s_{max}(Q) = 6 - 3 + 1 = 4$. □

Using the defined parameter $s_{max}(x)$ for a node x corresponding to the DL-query of a DL-atom d, we tighten the bound on the maximal size of support sets for d from Proposition 5.68 as follows:

Proposition 5.73. *Let $\mathcal{O} = \langle\mathcal{T}, \mathcal{A}\rangle$ be an $\mathcal{EL}$ ontology, where $\mathcal{T}$ is in normal form, and let $d = \mathrm{DL}[\lambda; Q](\vec{t})$ be a DL-atom. Let, moreover, $\mathcal{G}^{\Sigma}_{\mathcal{T},sup(d)}$ be its support hypergraph. Then the maximal support set size of d is bounded by $s_{max}(x_Q)$, i.e. $maxsup(d) \leq s_{max}(x_Q)$.*

Proof (sketch). We sketch a general idea of the proof. Consider a path $path$ of the support graph, for which $n(path) + m - 1$ is maximal and yields $s_{max}(x_Q)$. Let us look at some node x_{C_n} on $path$, such that $hd^+(x_{C_n}) = k$, where $k > 1$, i.e, there are k hyperedges outgoing from x_{C_n}: $(\{x_{C_{n_1}}, x_{C_n}\}, x_{C_{n'_1}}) \ldots (\{x_{C_{n_k}}, x_{C_n}\}, x_{C_{n_{k'}}})$.

These hyperedges lie on $path$. That means that the subgraphs induced by the nodes $x_{C_{n'_1}}$ and $x_{C_{n_{k'}}}$ at some point must be connected by a hyperedge. We illustrate our observation in Figure 5.6.

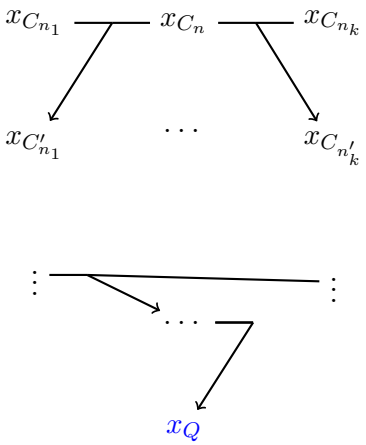


Figure 5.6: Fragment of a hypergraph used for illustration in proof of Proposition 5.73

According to the hypergraph, there must be the following axioms present in the TBox $\mathcal{T}$: $C_{n_1} \sqcap C_n \sqsubseteq C_{n'_1}$, ..., $C_{n_k} \sqcap C_n \sqsubseteq C_{n'_k}$. It must be the case that all of $C_{n'_1} \dots C_{n'_k}$ participate in a certain support set of size m, where m is the number of hyperedges in the graph below the elements $x_{C_{n_i}}, x_{C_n}, x_{C'_{n_i}}$. The intuitive meaning of the hyperedges in which x_{C_n} participates is as follows: if there is a support set S for d that contains $C_{n'_1}$, then there is a support set S' where $C_{n'_1}$ is substituted by C_n and C_{n_1}. Note that since we consider a single path of a hypergraph, there must be a support set in which all $C_{n'_1}$ and $C_{n'_k}$ appear together. The size of S is at most m, where m is the number of hyperedges in the graph below $x_{C_{n'_1}}, \dots, x_{C'_{n_k}}$. By Proposition 5.68 for the size of S' we get $m+k+1$, due to the hyperedges k hyperedges: $(\{x_{C_{n_1}}, x_{C_n}\}, x_{C_{n'_1}})$, ..., $(\{x_{C_{n_k}}, x_{C_n}\}, x_{C_{n'_k}})$. Now observe that the support set S' involves C_n k times, thus the size of S' in fact should be $m+1$ instead, this is exactly due to $hd^+(x_{C_n}) = k$. The generalization of the above argument to multiple nodes like x_{C_n} brings us to the result. □

Example 5.74. Reconsider the hypergraph in Figure 5.4. It has 3 hyperedges, and for all nodes $x \in \mathcal{V}$, $hd^+(x) \leq 1$. Thus, $s_{max}(Q) = 4$, which coincides with the maximal size of support sets for d. For the graph in Figure 5.5 we have $s_{max}(Q) = 4$, and 4 is indeed the maximal support set size. □

Observe that it is important that in Proposition 5.73 we considered the outdegree of nodes x_C that participate in hyperedges of the form $(\{x_C, x_D\}, x_E)$, where C, D, E are concepts. Indeed, if there is a role involved in some of the hyperedges, then the situation changes. For instance, let $(\{x_r, x_C\}, x_D)$, $(\{x_C, x_s\}, x_M)$, $(\{x_D, x_M\}, x_Q)$ be present in a support hypergraph for a DL-atom $d = \mathrm{DL}[\lambda; Q](X)$. The hyperedges reflect the axioms $\exists r.C \sqsubseteq D$, $\exists s.C \sqsubseteq M$ and $M \sqcap D \sqsubseteq Q$. The largest support set for d is $S = \{r(X,Y), C(Y), s(X,Z), C(Z)\}$, i.e. it is of size 4, and corresponds to $n+1$, where n is the number of hyperedges. This means that the number of nodes with the outdegree

greater then 1 plays a role on the size of support sets only when the hyperedges in which these nodes participate involve concepts.

5.7.2 Number of Support Sets

Another restriction relevant in practice regards the number of support sets. In general, determining the exact number of support sets that is needed to form a complete family for a DL-atom is a hard problem. It is tightly related to counting minimal explanations for an abduction problem, which was analyzed in [HP10] for propositional theories under various restrictions; there it was shown that counting all smallest solutions (explanations) for an abduction problem over a Horn theory is $\#Opt_P[log_n]$-complete. $\#Opt_p[log_n]$ is a relatively new complexity class for counting optimal solutions to counting problems introduced in [HP08]; we refer the reader to this work for further details. Moreover, meaningful conditions such that a fixed number n of support sets suffices to obtain a complete family are non-obvious (bounded tree-width [GPW07] might be useful, as for efficient datalog abduction); a careful analysis of real world ontologies is needed to ensure practical relevance.

Like for the size of support sets, for estimating the maximal number of support sets for a given DL-atom, the support hypergraph might be of relevance. Given a DL-atom $\mathrm{DL}[\lambda; Q](X)$ accessing an ontology with the TBox $\mathcal{T}$, the number of support sets for d can be accurately measured in terms of the size of the hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T},supp(d)}$. Intuitively, the bound on the number of support sets should be obtained by summing up all lengths of paths leading to x_Q in the hypergraph. Each path in the hypergraph, corresponds to a certain set of support sets, while every edge symbolizes a transition from one support set to another, obtained by substituting the predicate in the head node by the tail predicate(s). Therefore, summing up the lengths of paths in the hypergraph should be sufficient for measuring the maximal number of support sets. Extensive investigation of this issue remains for future work.

5.8 Repair Computation Based on Partial Support Families

In this section we present our algorithm *SoundRAnsSet* (see Algorithm 5.5) for computing deletion repair answer sets. As in *SupRAnsSet* we also exploit support families, but do not require that they are complete. If the families are complete (which may be known), then *SoundRAnsSet* is guaranteed to be complete; otherwise, it may miss repair answer sets (an easy extension ensures completeness though).

We start at (a) by computing a family $\mathbf{S}$ of nonground support sets for each DL-atom. Next the replacement program $\hat{\Pi}$ is created, whose answer sets $\hat{I}$ are computed one by one in (b). For $\hat{I}$, we first determine the sets D_p (resp. D_n) of DL-atoms that are guessed true (resp. false) in it and then use the function $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$ which instantiates $\mathbf{S}$ for the DL-atoms in $D_p \cup D_n$ to relevant ground support sets, i.e., those compatible with $\hat{I}$. In (d) we check whether some DL-atom in D_n has a support set S consisting

Algorithm 5.5: *SoundRAnsSet*: compute deletion repair answer sets

Input: $\Pi{=}\langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$

Output: a set of deletion repair answer sets of Π

(a) compute a set $\mathbf{S}$ of nongr. supp. sets for the DL-atoms in Π

(b) **for** $\hat{I} \in AS(\hat{\Pi})$ **do**

(c) $\quad D_p \leftarrow \{a \mid e_a \in \hat{I}\}$; $D_n \in \{a \mid ne_a \in \hat{I}\}$; $\mathbf{S}^{\hat{I}}_{gr} \leftarrow Gr(\mathbf{S}, \hat{I}, \mathcal{A})$;

(d) $\quad$ **if** *every* $S \in \mathbf{S}^{\hat{I}}_{gr}(a')$ *for* $a' \in D_n$ *fulfills* $S \cap \mathcal{A} \neq \emptyset$ **then**

(e) $\quad\quad$ **for** *all min. hitting sets* $H \subseteq \mathcal{A}$ *of* $\bigcup_{a' \in D_n} \mathbf{S}^{\hat{I}}_{gr}(a')$ **do**

(f) $\quad\quad\quad$ **if** *for every* $a \in D_p$ *some* $S \in \mathbf{S}^{\hat{I}}_{gr}(a)$ *exists s.t.* $S \cap H = \emptyset$*;*
$\quad\quad\quad$ **then** $rep \leftarrow eval_n(D_n, \hat{I}, \mathcal{T} \cup \mathcal{A} \backslash H)$ **else**
$\quad\quad\quad$ $rep \leftarrow eval_n(D_n, \hat{I}, \mathcal{T} \cup \mathcal{A} \backslash H) \wedge eval_p(D_p, \hat{I}, \mathcal{T} \cup \mathcal{A} \backslash H)$

(g) $\quad\quad\quad$ **if** *rep and flpFND*$(\hat{I}, \langle \mathcal{T} \cup \mathcal{A} \backslash H, \mathcal{P} \rangle)$ **then** output $\hat{I}|_{\Pi}$

$\quad\quad$ **end**

$\quad$ **end**

end

just of input assertions; if so we move to the next answer set $\hat{I}$ of $\hat{\Pi}$. Otherwise, we (e) loop over all minimal hitting sets $H \subseteq \mathcal{A}$ of the support sets for DL-atoms in D_n, formed by ABox assertions only. For each H we check whether every atom in D_p has at least one support set disjoint from H. If yes (f), i.e. removing H from $\mathcal{A}$ does not affect the values of DL-atoms in D_p, then we evaluate in a *postcheck* the atoms from D_n over $\mathcal{T} \cup \mathcal{A} \backslash H$ w.r.t. $\hat{I}$. Otherwise, we evaluate the DL-atoms from D_n and D_p. A Boolean flag *rep* stores the evaluation result of a function $eval_n$(resp. $eval_p$). More specifically, given D_n (resp. D_p), $\hat{I}$ and $\mathcal{T} \cup \mathcal{A} \backslash H$, the function $eval_n$ (resp. $eval_p$) returns *true*, if all atoms in D_n (resp. D_p) evaluate to false (resp. true). If *rep* is *true* and the foundedness check *flpFND*$(\hat{I}, \mathcal{A} \setminus H, \mathcal{P})$ succeeds, then in (g) $\hat{I}|_{\Pi}$ is output as repair answer set.

We remark that in many cases, the foundedness check might be trivial [EFK$^+$14]; if we would consider weak FLP-answer sets [EFS13a], it can be skipped.

Example 5.75. Consider Π from Example 5.7 with equivalence ($\equiv$) in axioms (2) and (3) substituted by $\sqsupseteq$. Let $\hat{I} = \{projfile(p1), hasowner(p1, john), chief(john), e_a, ne_b\}$ be returned at (b), where $a = \mathrm{DL}[Project \uplus projfile; Staffrequest](r1)$ and $b = \mathrm{DL}[Staff \uplus chief; BlacklistedStaffRequest](r1)$. At (c) we obtained

- $\mathbf{S}^{\hat{I}}_{gr}(a) = \{S_1, S_2\}$, where $S_1 = \{hasAction(r1, read), hasSubject(r1, john), Action(read), Staff(john), hasTarget(r1, p1), Project_{projfile}(p1)\}$ and $S_2 = \{StaffRequest(r1)\}$;
- $\mathbf{S}^{\hat{I}}_{gr}(b) = \{S'_1, S'_2\}$ with $S'_1 = \{StaffRequest(r1), hasSubject(r1, john), Blacklisted(john)\}$ and $S'_2 = \{BlacklistedStaffRequest(r1)\}$.

At (e) we got a hitting set $H = \{StaffRequest(r1), BlacklistedStaffRequest(r1)\}$, which is disjoint with S_1. Thus we get to the if branch of (f) and check whether b is false under $\mathcal{A}\backslash\{StaffRequest(r1)\}$. This is not true, hence $rep = false$ and we pick a different hitting set H', e.g $\{Blacklisted(john), BlacklistedStaffRequest(r1)\}$. Proceeding with H', we get to (g), since at (f) $eval_n(b, \hat{I}, \mathcal{T} \cup \mathcal{A} \cap H) = true$. □

Proposition 5.76. *SoundRAnsSet is sound, i.e. it outputs only deletion repair answer sets.*

Proof. Suppose *SupRAnsSet* outputs $I = \hat{I}|_\Pi$. We can get to (g) only if $\hat{I}$ is an answer set of $\hat{\Pi}$, and if the foundness check of I with respect to the ontology $\mathcal{T} \cup \mathcal{A}'$, where $\mathcal{A}' = \mathcal{A}\backslash H$ succeeded. It is thus left to show that $\hat{I}$ is a compatible set for $\mathcal{T} \cup \mathcal{A}'$. Towards a contradiction, suppose that this is not the case. In (c) we partitioned the DL-atoms into two sets: D_p and D_n, corresponding to DL-atoms a guessed to be true and false in $\hat{I}$ respectively and set $\mathbf{S}^{\hat{I}}_{gr}$ to $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$. Since we assume that $\hat{I}$ is not compatible, one of the following must hold:

(1) There is a DL-atom a in D_n, such that $I \models^{\mathcal{O}'} a$. There are two possibilities: either there is a support set $S \in \mathbf{S}^{\hat{I}}_{gr}$ for a or no support sets for a were identified. If the former is the case, then by the check (d) we are guaranteed that all support sets S for a are such that $S \cap \mathcal{A} \neq \emptyset$. Hence there must exist a support set which has a nonempty ABox part, which is in $\mathcal{A}'$. However, according to (e) some minimal hitting set H of all support sets for DL-atoms in D_n is not in $\mathcal{A}'$, thus $S \notin Supp_a(\mathcal{O}')$. Now since $rep = true$ at (g), postevaluation of a must have succeeded in (f), i.e. $I \not\models^{\mathcal{O}'} a$ must hold. This is a contradiction.

(2) There is a DL-atom a in D_p, such that $I \not\models^{\mathcal{O}'} a$. Therefore, there are no support sets for a in $\mathbf{S}^{\hat{I}}_{gr}$. We thus get to the else part of (f), where post-evaluation is performed. The latter, however, must have succeeded, as $rep = true$ at (g), leading to a contradiction. Hence $\hat{I}$ is a compatible for Π', and thus a deletion repair answer set of Π.

□

If we know in addition that the support families are complete, then the postchecks at (f) are redundant. In case the if-condition of (f) is satisfied, we set $rep = true$, otherwise $rep = false$.

Proposition 5.77. *If for all DL-atoms in* Π *the support families in* $\mathbf{S}$ *are complete, then SoundRAnsSet is complete, i.e., it outputs every deletion repair answer set.*

Proof. Suppose I is a deletion repair answer set. That is, for some $\mathcal{A}' \subseteq \mathcal{A}$, we have that I is an answer set of $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle$. This implies that $\hat{I}$ is an answer set of $\hat{\Pi}$ and thus will be considered in (b), with D_p and D_n reflecting the (correct) guess for $I \models^{\mathcal{O}'} a$ for each DL-atom a, where $\mathcal{O}' = \mathcal{T} \cup \mathcal{A}'$. From Proposition 5.28 and completeness of $\mathbf{S}$, we

obtain that each $a \in D_p$ has $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \neq \emptyset$ and each $a \in D_n$ has $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) = \emptyset$. The initial $\mathbf{S}^{\hat{I}}_{gr}$ is such that $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \subseteq \mathbf{S}^{\hat{I}}_{gr} = Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a)$ holds for each DL-atom a; in further steps, the algorithm removes all support sets $S \in Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a)$ for $a \in D_p$ from $\mathbf{S}^{\hat{I}}_{gr}(a)$ such that $S \cap S' \cap \mathcal{A} \neq \emptyset$ for some support set $S' \in Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a')$ and $a' \in D_n$, and removes all assertions in $S' \cap \mathcal{A}$ from $\mathcal{A}$. Importantly no removed S is in $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a)$, since by the assertion that $\mathcal{T} \cup \mathcal{A}$ is consistent, $|S' \cap \mathcal{A}| = 1$ must hold. Thus step (g) will be reached, and the variable $\mathcal{A}'$ is assigned an ABox $\mathcal{A}''$ such that $\mathcal{A}' \subseteq \mathcal{A}'' \subseteq \mathcal{A}$. Since $\hat{I}$ is a compatible set for $\Pi'' = \langle \mathcal{T} \cup \mathcal{A}'', \mathcal{P} \rangle$ and I is an answer set of Π', by Proposition 5.36 I is also an answer set of Π'', and thus I is a minimal model of $\Pi''^{I,\mathcal{O}''}_{flp} = \langle \mathcal{T} \cup \mathcal{A}'', \mathcal{P}^{I,\mathcal{O}''}_{flp} \rangle$. Hence, the test $\mathit{flpFND}(\hat{I}, \mathcal{T} \cup \mathcal{A}', \mathcal{P})$ in step (h) (where $\mathcal{A}'$ has value $\mathcal{A}''$) succeeds, and $\hat{I}_\Pi$, i.e, I is output. □

We easily can turn *SoundRAnsSet* into a complete algorithm, by modifying (e) to consider all hitting sets and not only minimal ones. In the worst case, this means a fallback to almost the naive algorithm (note that all hitting sets can be enumerated efficiently relative to their number).

5.9 Optimization Techniques

The algorithms provided in this chapter admit various optimizations. Research in repairing databases (see [Ber11] for overview) suggests several techniques, which could be of potential interest for DL-program repairs. One of such techniques deals with *localization* of repairs [EFGL07]. The set of repair candidates can be narrowed to a part which is touched by inconsistency of the DL-program. The ontology ABox can be split into the two parts: the one that is safe, will not be touched by any repair and the one that is the matter of change. The general approach of repair localization is to isolate the part of the ABox which should not be changed, and to aim at finding the repair by eliminating assertions from the rest of the ABox. After following this procedure, the obtained repair can be combined with the safe part for getting the final result. The important task that naturally arises in this context is related to effective identification of the safe and affected parts of the ABox. This is clearly a difficult problem in general; however, the meta knowledge about the ontology (e.g. modules, domain knowledge), if available, can be fruitfully exploited for solving it.

Another common approach for tackling an inconsistency problem, which proved to be effective for databases, is called *decomposition* [EFGL07]. This approach tries to decompose the available knowledge into parts, such that the reasons for inconsistency can be identified in each part separately, and then the repairs for each of the parts can be conveniently combined. Even for databases decomposition is natural, for DL-programs it is far nontrivial whether there are possibilities to effectively exploit this technique. One way to approach this problem is by analyzing the values of the DL-atoms. Given the program $\hat{\Pi}$ and a replacement atom e_a, one might perform brave reasoning to identify whether all answer sets of the replacement program must entail e_a or ne_a. Given a set of DL-atoms which must have the same value under all answers sets, we can first try to

find repairs that satisfy these values, and then aim at extending the solution to obtain the final repair. Finally, modules of the DL-program can be exploited for decomposing the repair problem.

5.10 Conclusion, Related Work and Outlook

We have provided algorithms and optimization techniques for repairing inconsistent DL-programs over *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ ontologies. It turns out that the naive algorithm, that traverses all possible ABoxes that satisfy a certain σ selection criteria and aims at finding an ABox which is suitable as a repair is not effective in practice. For that reason we introduced the notion of support sets. We exploited support sets for optimizing the naive repair approach. Support sets reduce the evaluation of DL-atoms to constraint matching, providing a base for effective deletion repair answer set computation under *weak* or *flp*-semantics. Our results, and in particular algorithms *SupRAnsSet* and *SoundRAnsSet*, can be extended to other semantics, e.g. well-founded, and to other notions of repairs, e.g. bounded addition.

Note that algorithm *SupRAnsSet* for *DL-Lite*$_\mathcal{A}$ ontologies constructs in its search all maximal deletion repairs, i.e., $\subseteq$-maximal repairs $\mathcal{A}' \subseteq \mathcal{A}$ that admit an answer set (however, it also may construct non-maximal repairs). In this regard it is similar to work on ABox cleaning [MRR11, RRGM12]. There, an inconsistent ontology (with consistent TBox) is repaired by identifying and eliminating minimal conflict sets causing, i.e., explaining, the inconsistency, thus resulting in maximal deletion repairs. However, our setting and work differ fundamentally: (i) the ontology is consistent and inconsistency arises only through the interface of DL-atoms, and (ii) several DL-atom queries have to be considered (entailment or non-entailment) under different potential ABox updates.

More remotely related to our work are approaches for explaining positive and negative answers to conjunctive queries in *DL-Lite* [COSS12a,BCRM08], as they apply the perfect reformulation algorithm [CLLR07] and then extract explanations in a nontrivial way. Unary explanations for instance queries relate to our support sets for DL-atoms without updates. For arbitrary DL-atoms our support sets are more general, since they can also represent minimal set of assertions that make the updated ontology inconsistent, and thus the DL-query trivially derivable from it.

For DL-programs over $\mathcal{EL}$ ontologies, we generalized the support set approach for *DL-Lite*$_\mathcal{A}$ to work with incomplete families of supports sets; this advance is needed since in $\mathcal{EL}$ complete support families can be large or even infinite. We discussed how to generate support families, by exploiting query rewriting over ontologies to datalog [LTW08,Ros07,SMH12], which is in contrast to our approach for computing support sets in the *DL-Lite*$_\mathcal{A}$ case, where TBox classification is invoked. We presented an algorithm to compute deletion repair answer sets which trades answer completeness for scalability.

Our support sets are related to solutions of abduction problems for $\mathcal{EL}$ [Bie08], and correspond in the ground case to support sets for query answering over first-order rewritable ontologies [BR13]; nonground computation naturally links to TBox classification [KKS13]. Abduction had been studied for *DL-Lite* in [BCRM08] and for datalog

e.g. in [GPW07, HP08]. The use of incomplete support families for DL-atoms is related in spirit to approximate inconsistency-tolerant reasoning on DLs using restricted support sets [BR13]; however, we target repair computation while [BR13] targets inference from all repairs.

Furthermore, support sets are used in a companion work to compute answer sets of HEX-programs [EFRS14]. As external atoms lack an explicit data part (ABox), support sets ought to be more abstract, which makes direct efficient usage less clear; repair is an open issue.

It remains an issue for further research to extend the work to other members of the $\mathcal{EL}$ family. To increase usability in practice, real world ontologies need to be analyzed to develop good heuristics and strategies for computing incomplete support families. The work on determining syntactic conditions put on the TBox that ensure that all support sets are of bounded size can be further extended by considering various parameters in a hypergraph, that can influence the support set size and number. We have studied acyclic $\mathcal{EL}$ TBoxes, but it is also possible to look at cyclic TBoxes and consider for instance such parameter, as hypertree width [HOSG08].

CHAPTER 6

Optimization Techniques for DL-programs

In this chapter we propose optimization techniques that allow one to simplify the rule part $\mathcal{P}$ of a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ before computing its answer sets if they exist, and repair answer sets otherwise.

For computing answer sets of DL-programs over ontologies in arbitrary DLs in practice individual DL-atoms may often need to be evaluated under varying input in general. Thus, the possible ontology updates specified by a DL-atom define respective contexts for their evaluation, and many different such contexts need to be considered. Moreover, even for one context evaluating the query specified by the DL-atom in this context may be costly. Therefore, developing optimization techniques, e.g. caching techniques [EIST04], partial evaluation and atom merging [EIKS08], is necessary for the development of effective solvers.

Caching techniques, for instance, aim at memorizing the value of a DL-atom for some inputs, such that its value on a new input can be concluded. However, the very question whether its value is on all inputs the same has not been considered so far; we call DL-atoms with this property *independent*. The identification of independent DL-atoms has immediate applications in optimizations, as such atoms respectively rules involving them can be removed from the DL-program.

Information about independent DL-atoms is certainly also of use in repair answer set computation. Once the repair answer set candidate is identified together with the guesses on the values of DL-atoms, we aim at modifying the ontology data part in such a way that the real values of DL-atoms coincide with the guessed ones. However, the value of a DL-atom can be independent of the underlying ontology; thus some candidate interpretations can never become repair answer sets regardless of the changes applied to the ontology at hand. Identifying independent DL-atoms and subsequently faulty answer set candidates at early stages of the repair answer set computation should bring potential performance gains.

However, it is not always obvious that a certain DL-atom is independent, for instance tautologic. Let us illustrate this by an example.

Example 6.1. Consider the following DL-program representing information in the fruit domain: $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ with underlying ontology $\mathcal{O}$ and the rules part

$$\mathcal{P} = \left\{ \begin{array}{llll} (1) & so(pineapple, chile); & (2) & vi(X) \leftarrow ex(X); \\ (3) & sw(X) \leftarrow ex(X), not\ bi(X); & (4) & ex(X) \leftarrow so(X, Y); \\ (5) & no(X) \leftarrow \mathrm{DL}[H \uplus vi, H \uplus sw, A \cap ex; \neg A](X). & & \end{array} \right\},$$

where predicate *so* stands for Southern fruit with its country of origin, *vi* for vitamin, *ex* for exotic, *bi* for bitter, *sw* for sweet, and *no* for non-African fruit, respectively. Moreover, H stands for the concept of healthiness and A for the concept of African fruit. Here (1) is the fact that pineapple is a Southern fruit possibly from Chile, rule (2) states that exotic fruits are rich of vitamin and rule (3) that exotic fruits are sweet, unless they are known to be bitter. Rule (4) says that Southern fruits are exotic. Finally, rule (5) contains a DL-atom in its body. Informally, it selects all objects o into *no* such that $\neg A(o)$, i.e., o is proved to be not an African fruit from the ontology $\mathcal{O}$, upon the (temporary) assertions that vitamin objects are healthy, sweet ones are unhealthy, and the restriction that only fruit known to be exotic may be African.

It is not straightforward for this DL-atom, nor for any of its instances, that it is tautologic; this however will be shown in Section 6.2.

If we adopt the reasonable assumption that the underlying ontology is satisfiable, another kind of independence is possible: DL-atoms which are *contradictory*, i.e., always evaluate to false.

Our contributions on identifying independent DL-atoms briefly are as follows:

- Based on a semantic notion of independence, we provide a syntactic characterization of independent DL-atoms in Section 6.1. While tautologic DL-atoms have a rich structure, contradictory DL-atoms are simple and only possible without ontology update prior to query evaluation.
- We also consider relaxed forms of tautologies, relative to additional information on rule predicates (acting as constraints on the possible updates to the ontology). In particular, we study inclusion among rule predicates in Section 6.2.
- We develop a complete axiomatization for deriving all tautologies by means of simple rules of inference, in the general case as well as under separable inclusion constraints, i.e., without projective input inclusions.
- We determine the complexity of the calculus in Section 6.3. It turns out that tautology checking is feasible in polynomial time (more precisely, in NLogSpace in general, and in LogSpace, in fact it is first-order expressible, for non-negative queries), also relative to separable inclusion constraints (in this case, it is NLogSpace-complete). Thus, we establish that checking whether a given DL-atom is independent can be done efficiently.

Our results provide further insight into the nature of DL-programs. In particular, they might be useful for DL-programs that are automatically constructed (like the ones encoding a fragment of Baader and Hollunder's terminological default logic over ontologies [DTEK09]). They can be applied to simplify DL-programs, as well as to optimize the algorithms for repair answer set computation that we have proposed in Chapter 5.

6.1 Independent DL-atoms

We call a DL-atom a *independent*, if it always has the same truth value, regardless of the underlying ontology and the context in which it is evaluated, i.e, the interpretation I of the rules. This means that a amounts to one of the logical constants $\bot$ (false, i.e., is a contradiction) or $\top$ (true, i.e., is a tautology).

In formalizing this notion, we take into account that independence trivializes for unsatisfiable underlying ontologies, and thus restrict to satisfiable ones.

Definition 6.2 (independent DL-atom)**.** A ground DL-atom a is *independent*, if for all satisfiable ontologies $\mathcal{O}, \mathcal{O}'$ and all interpretations I, I' it holds that $I \models^{\mathcal{O}} a$ iff $I' \models^{\mathcal{O}'} a$.

Furthermore, we call a *tautologic* (resp., *contradictory*), if for all satisfiable ontologies $\mathcal{O}$ and all interpretations I, it holds that $I \models^{\mathcal{O}} a$ (resp., $I \not\models^{\mathcal{O}} a$).

Example 6.3. A DL-atom of the form $a = \mathrm{DL}[; \neg(C \sqsubseteq C)]()$ is contradictory. Indeed, the query $\neg(C \sqsubseteq C)$ is unsatisfiable, hence there does not exist any satisfiable ontology $\mathcal{O}$, such that $\mathcal{O} \models \neg(C \sqsubseteq C)$. Therefore, regardless of I, always $I \not\models^{\mathcal{O}} \neg(C \sqsubseteq C)$.

On the other hand consider a DL-atom $b = \mathrm{DL}[C \uplus p, C \uplus p; \neg C](c)$. It is tautologic, because under any interpretation I of p, it holds that $\neg C(c) \in \lambda^I(b)$. Hence, it is true that $I \models^{\mathcal{O}} \neg C(c)$ for any ontology $\mathcal{O}$ (and any interpretation I).

In the following, we aim at a characterization of independent DL-atoms.

6.1.1 Contradictory DL-atoms

We defined above contradictory DL-atoms relative to satisfiable ontologies (otherwise, trivially no contradictory DL-atoms exist).

An obvious example of a contradictory DL-atoms is $\mathrm{DL}[; \top \sqsubseteq \bot]()$, where $\bot$ and $\top$ are the customary empty and full concept, respectively. Indeed, the DL-query $\bot \sqsubseteq \top$ is false in every interpretation, i.e., a logical contradiction. As it turns out, contradictory DL-atoms are characterized by such contradictions, and have a simple input signature.

We call a DL-query $Q(\vec{t})$ *satisfiable*, if there exists some satisfiable ontology $\mathcal{O}$ such that $\mathcal{O} \models Q(\vec{t})$, and *unsatisfiable* otherwise. Then we have the following result.

Proposition 6.4. *A ground DL-atom $a = \mathrm{DL}[\lambda; Q](\vec{t})$ is contradictory if and only if $\lambda = \epsilon$ and $Q(\vec{t})$ is unsatisfiable.*

Proof. (if) If $\lambda = \epsilon$, then for every I, $I \models^{\mathcal{O}} a$ iff $\mathcal{O} \models Q(\vec{t})$; as $Q(\vec{t})$ is unsatisfiable, we have for every satisfiable $\mathcal{O}$ that $\mathcal{O} \not\models Q(\vec{t})$. Thus a is contradictory.

(Only If). Suppose a is contradictory, i.e., $I \not\models^{\mathcal{O}} a$ for every satisfiable ontology $\mathcal{O}$ and every interpretation I, i.e., $\mathcal{O} \cup \lambda^I(a) \not\models Q(\vec{t})$. It follows that $Q(\vec{t})$ is unsatisfiable. To show $\lambda = \epsilon$, assume towards a contradiction that $\lambda \neq \epsilon$. Then there exists some interpretation I_0 such that $\lambda^{I_0}(a) \neq \emptyset$, i.e., contains some assertion B. Consider an arbitrary satisfiable ontology L. As $L \cup \lambda^{I_0}(a) \not\models Q(\vec{t})$, it follows that $L \not\models \neg.B$, where $\neg.B$ is the opposite of B. However, it is not difficult to see that satisfiable ontologies L_0 exist such that $L_0 \models \neg.B$.[1] This, however, raises a contradiction. Thus $\lambda = \epsilon$. □

By this result, contradictory DL-atoms have a simple form. As concept and role instance queries are always satisfiable, Q must be a (possibly negated) concept inclusion query and of the form $\neg(C \sqsubseteq C)$, $\neg(C \sqsubseteq \top)$, $\neg(\bot \sqsubseteq C)$, $\neg(\bot \sqsubseteq \top)$, or $\top \sqsubseteq \bot$.

6.1.2 Tautologic DL-atoms

For tautologic DL-atoms, the situation is more complex. First of all, clearly a DL-atom is tautologic if it has a tautologic query (i.e., it is satisfied by the empty ontology). This is, however, only possible for concept inclusion queries; instance queries $(\neg)C(t)$, resp. $(\neg)R(t_1, t_2)$, are clearly not tautologic.

DL-atoms with tautologic queries are of the form DL[λ; $C \sqsubseteq \top$](), DL[λ; $\bot \sqsubseteq C$](), DL[λ; $C \sqsubseteq C$](), or DL[λ; $\top \not\sqsubseteq \bot$](), where λ is an arbitrary input signature.

However, there are also tautologic DL-atoms whose query is not tautologic.

Example 6.5. Consider in the fruit scenario the DL-atom

$$a = \mathrm{DL}[EF \uplus fr, S \cup fr, S \uplus fr; \neg EF](c),$$

where EF stands for exotic fruit, S for sweet, fr for fruit.

Intuitively, we restrict here the concept $\neg EF$ and extend the concepts S and $\neg S$ by the predicate fr. Then we ask whether c is not an exotic fruit. No matter which interpretation I of the DL-program we consider and irrespective of $\mathcal{O}$, we will always get that $\mathcal{O} \cup \lambda^I(a) \models \neg EF(c)$. Indeed, if $fr(c) \in I$, then $\lambda^I(a)$ is unsatisfiable; otherwise $\neg EF(c)$ is explicitly present in $\lambda^I(a)$. Hence in both cases, $\lambda^I(a) \models \neg EF(c)$. This means that a is tautologic.

In the rest of this section, we identify for each query type those forms of the input signature for which the DL-atom is tautologic, or prove nonexistence of such forms. We first consider concept queries, i.e., queries $(\neg)C(\vec{t})$ and $(\neg)(C \sqsubseteq D)$, and then role queries, for which similar results hold.

[1] If B is a negative (resp., positive) assertion, then $\neg.B$ is a positive (resp. negative) assertion and we can take $L_0 = \{\neg B\}$. If $\neg B$ is not an admissible assertion, we can effect $\neg B$ by a set of possible more restrictive axioms (e.g. we can enforce a negative role assertion $\neg R(a, b)$ in basic *DL-Lite* e.g. by $L_0 = \{\exists R \sqsubseteq C, \exists R \sqsubseteq \neg C\}$ and in $\mathcal{EL}^{++}$ ($\mathcal{EL}$ with some additional constructs and $\bot$ [BBL05]) by $L_0 = \{\exists R \sqsubseteq \bot\}$). Note that if negative assertions were not explicitly available in the DL and the operators $\cup, \uplus$ disallowed in DL-atoms, still the above construction may be used as e.g. in case of *DL-Lite* and $\mathcal{EL}^{++}$, and thus the same characterization of contradictions holds.

Concept queries

Concept instance. To start with, let us consider the query $C(\vec{t})$. No matter what input signature is considered for this type of the DL-atom, it can never be tautologic.

Proposition 6.6. *For no input signature λ, a ground DL-atom a of the form* $\mathrm{DL}[\lambda; C](\vec{t})$ *is tautologic.*

Proof. Consider a ground DL-atom $a = \mathrm{DL}[\lambda; C](\vec{t})$. Towards a contradiction, suppose that λ is a signature such that a is tautologic. Thus by definition, for all ontologies $\mathcal{O}$ and for all interpretations I it holds that $\mathcal{O} \cup \lambda^I(a) \models C(\vec{t})$. Thus in particular, for $L = \emptyset$ it holds that $\lambda^I(a) \models C(\vec{t})$.

We consider two cases, according to the satisfiability of $\lambda^I(a)$. (1) Suppose $\lambda^I(a)$ is unsatisfiable. Then there must exist some S, such that $S(\vec{t}) \in \lambda^I(a)$ and $\neg S(\vec{t}) \in \lambda^I(a)$. The presence of $S(\vec{t})$ in $\lambda^I(a)$ can only be ensured if some $S \uplus p$ occurs in the input signature λ of a for some p. Now consider the interpretation $I = \emptyset$. As $p(\vec{t}) \notin I$, we can not get $S(\vec{t}) \in \lambda^I(a)$, which leads to contradiction.

(2) Now suppose $\lambda^I(a)$ is satisfiable. Then $C(\vec{t})$ must be in $\lambda^I(a)$. Similar to the previous case, this requires that $C \uplus p$ occurs in λ for some p. Again $I = \emptyset$ does not allow us to obtain $C(\vec{t}) \in \lambda^I(a)$, hence $\lambda^I(a) \not\models C(\vec{t})$. This contradicts our assumption. □

Concept inclusion. For DL-atoms with concept queries of the form $C \sqsubseteq D$ and $C \not\sqsubseteq D$, where $C \neq D$ and neither concept is $\top$ or $\bot$, we get the same result as for positive instance queries.

Proposition 6.7. *For no input signature λ, a ground DL-atom of the form* $\mathrm{DL}[\lambda; C \sqsubseteq D]()$ *or* $\mathrm{DL}[\lambda; C \not\sqsubseteq D]()$*, where $C \neq D$ are different concept names, is tautologic.*

Proof. Consider a ground DL-atom $a = \mathrm{DL}[\lambda; C \sqsubseteq D]()$, and suppose a is tautologic. Then for every ontology $\mathcal{O}$ and interpretation I, it holds that $\mathcal{O} \cup \lambda^I(a) \models C \sqsubseteq D$. Let $\mathcal{O} = \emptyset$ and $I = \emptyset$. Observe that $\lambda^I(a)$ is satisfiable, as it contains only negative assertions. Let c be a fresh constant; then $\mathcal{O}' = \mathcal{O} \cup \lambda^I(a) \cup \{C(c), \neg D(c)\}$ is satisfiable, and $\mathcal{O}' \not\models C \sqsubseteq D$. By monotonicity of $\models$, it follows $\mathcal{O} \cup \lambda^I(a) \not\models C \sqsubseteq D$. Thus a is not tautologic, which is a contradiction.

The proof for $a = \mathrm{DL}[\lambda; C \not\sqsubseteq D]()$ is similar. □

Out of the remaining concept queries, only the following (straightforwardly) give rise to tautologic DL-atoms.

Proposition 6.8. *A ground DL-atom of the form* $\mathrm{DL}[\lambda; Q]()$ *is a tautology iff $Q = C \sqsubseteq C$, $Q = C \sqsubseteq \top$, or $Q = \top \not\sqsubseteq \bot$, for any $C \in \mathbf{C} \cup \{\bot, \top\}$.*

Negative concept instance. Finally, we investigate the forms of tautologic DL-atoms with a query $\neg C(\vec{t})$.

Proposition 6.9. *A ground DL-atom a with the query $\neg C(\vec{t})$ is tautologic if and only if it has one of the following forms:*

c1. $\mathrm{DL}[\lambda,\ C \sqcap p, C \uplus p;\ \neg C](\vec{t})$,

c2. $\mathrm{DL}[\lambda,\ C \sqcap p, D \uplus p, D \uplus p;\ \neg C](\vec{t})$,

c3. $\mathrm{DL}[\lambda, C{\sqcap}p_0, C^0 \uplus p_0, C^0{\sqcap}p'_0,$
$C^1 \uplus p_1, C^1{\sqcap}p'_1, \ldots, C^n \uplus p_n, C^n{\sqcap}p'_n, C{\uplus}p_{n+1};\ \neg C](\vec{t})$,

c4. $\mathrm{DL}[\lambda, C{\sqcap}p_0, C^0 \uplus p_0, C^0{\sqcap}p'_0,$
$C^1 \uplus p_1, C^1{\sqcap}p'_1, \ldots, C^n \uplus p_n, C^n{\sqcap}p'_n, D \uplus p_{n+1} D{\uplus}p'_{n+1};\ \neg C](\vec{t})$,

where for every $i = 0, \ldots, n+1$, $p_i = p'_j$ *for some* $j < i$ *or* $p_i = p_0$, *and* $p'_{n+1} = p'_{i_j}$ *for some* $j \leq n$ *or* $p'_{n+1} = p_0$.

Proof. Informally, the lists of ($c3$) and ($c4$) include a "chain" $p = p_0 \subseteq p_{j_1} \subseteq p_{j_2} \subseteq p_{j_k} = p_{n+1}$ resp. $p = p_0 \subseteq p_{j'_1} \subseteq p_{j'_2} \subseteq p_{j'_{k'}} = p'_{n+1}$.

(Only If) Consider a tautologic ground DL-atom a of the form (2.3) where $Q = \neg C$. By definition of tautologic DL-atom, for all ontologies $\mathcal{O}$ and for all interpretations I it holds that $\mathcal{O} \cup \lambda^I(a) \models \neg C(\vec{t})$. In particular, it holds for $\mathcal{O} = \emptyset$ and $I_0 = \emptyset$. Note that $\lambda^{I_0}(a)$ must be satisfiable. Indeed for unsatisfiability, some concept D or role R must occur both positively and negatively in $\lambda^{I_0}(a)$. However, as all predicates in I_0 are empty, $\lambda^{I_0}(a)$ contains only negative assertions, and thus is satisfiable. As $\lambda^{I_0}(a) \models \neg C(\vec{t})$, it follows that (*) $C \sqcap p$ occurs in λ for some $p \in \mathcal{P}_p$, (i.e., $S_i = C$, $op_i = \sqcap$ and $p_i = p$). Now consider the interpretation $I' = \{p(\vec{t})\}$ and whether $\lambda^{I'}(a)$ is satisfiable or not.

1.) $\lambda^{I'}(a)$ is satisfiable. Then $C \uplus p \in \lambda$; therefore a is of the form ($c1$).

2.) $\lambda^{I'}(a)$ is unsatisfiable. Hence $S(\vec{t}), \neg S(\vec{t}) \in \lambda^{I'}(a)$ for some $S \in \mathcal{P}_o$. As $S(\vec{t}) \in \lambda^{I'}(a)$, it follows that (**) $S \uplus p$ occurs in λ. The presence of $\neg S(\vec{t})$ in $\lambda^{I'}(a)$ is due to some $S_i\ op_i\ p_i$ in λ where $S_i = S$, for which we have the following cases.

2.a $op_i = \uplus$. Then $p_i = p$ and (***) $S_i = C$, and a is of the form $c1$.

2.b $op_i = \sqcap$. As $p_i(\vec{t}) \notin I'$, we have $p_i \neq p$. Consider now $I'' = \{p(\vec{t}), p_i(\vec{t})\}$. If $\lambda^{I''}(a)$ is satisfiable, then $C \uplus p_i \in \lambda$. As we have $S{\uplus}p$, $S{\sqcap}p_i$ in λ, we conclude that a is of the form ($c3$) (where $n = 0$, $p_0 = p$, $C^0 = S$, $p'_0 = p_i$ and $p_1 = p_i$).

Otherwise, $\lambda^{I''}(a)$ is unsatisfiable, i.e., $S'(\vec{t}), \neg S'(\vec{t}) \in \lambda^{I''}(a)$ for some S'. That is, $S' \uplus p' \in \lambda$ for some $p' \in \{p, p_i\}$ and either

(i) $S' \uplus q' \in \lambda$ for some $q' \in \{p, p_i\}$ or

(ii) $S' \sqcap q' \in \lambda$ for some q' $(\notin \{p, p_i\})$.

In case (i), the DL-atom a is of the from ($c4$) where $n = 0$, $p_0 = p$, $C^0 = S$, $p'_0 = p_i$ and $p_1 = p'$ and $p'_1 = q'$.

In case (ii), we can continue the argument, by considering the increased interpretation $I'' = I' \cup \{q'(\vec{t})\}$, such that either $\lambda^{I''}(a)$ is satisfiable and the query is satisfied

by some element $C \uplus q'$ in λ, or $\lambda^{I''}(a)$ is unsatisfiable and contains some contradictory pair $S''(\vec{t}), \neg S''(\vec{t})$ etc. Eventually, this case is not encountered again, and we must have arrived at a list either of the form

$$C \sqcap p, S^0 \uplus p_0, S^0 \sqcap p'_0, S^1 \uplus p_1, S^1 \sqcap p'_1, \ldots, S^n \uplus p_n, S^n \sqcap p'_n, C \uplus p_{n+1} \tag{6.1}$$

or of the form

$$C \sqcap p, S^0 \uplus p_0, S^0 \sqcap p'_0, \ldots, S^n \uplus p_n, S^n \sqcap p'_n, S^{n+1} \uplus p_{n+1}, S^{n+1} \uplus p'_{n+1} \tag{6.2}$$

where $p_0 = p$, $n \geq 0$, and each $p_i = p'_j$ for some $j < i$ or $p_i = p_0$ and $p'_{n+1} = p'_j$ for some $j \leq n$ or $p'_{n+1} = p_0$ as in $(c3)$ resp. $(c4)$.

(If) Now we show that the forms $(c1)$ – $(c4)$ are indeed tautologic. For any interpretation I, there are two possible cases, where $p_0 = p$: either $p(\vec{t}) \notin I$ or $p(\vec{t}) \in I$. In the former case, as $C \sqcap p$ is in the input signature of each form $(c1)$ – $(c4)$, $\neg C(\vec{t})$ is in $\lambda^I(a)$ and thus $I \models^{\mathcal{O}} a$ holds. In the case $p(\vec{t}) \in I$, consider the forms:

c1. $a = \mathrm{DL}[\lambda,\, C \sqcap p, C \uplus p;\, \neg C](\vec{t})$. As $C \uplus p$ occurs in the input signature, $\neg C(\vec{t}) \in \lambda^I(a)$ and thus $I \models^{\mathcal{O}} a$.

c2. $a = \mathrm{DL}[\lambda,\, C \sqcap p, D \uplus p, D \uplus p;\, \neg C](\vec{t})$. Then $\lambda^I(a)$ is unsatisfiable as $D(\vec{t}), \neg D(\vec{t}) \in \lambda^I(a)$.

c3. $a = \mathrm{DL}[\lambda, C \sqcap p_0, C^0 \uplus p_0, C^0 \sqcap p'_0,$
$C^1 \uplus p_1, C^1 \sqcap p'_1, \ldots, C^n \uplus p_n, C^n \sqcap p'_n, C \uplus p_{n+1};\, \neg C](\vec{t})$

where for every $i = 0, \ldots, n+1$, $p_i = p'_j$ for some $j < i$ or $p_i = p_0$. If $\lambda^I(a)$ is unsatisfiable, then $I \models^{\mathcal{O}} a$; so suppose that $\lambda^I(a)$ is satisfiable. This implies that, for every $i = 0, \ldots, n$, it holds that $p_i^I \subseteq {p'_i}^I$ which is enforced by $C^i \uplus p_i, C^i \sqcap p'_i$ in the input signature. It then follows that there is a chain $q_0^I \subseteq q_1^I \subseteq \cdots \subseteq q_k^I$ such that $q_0 = p_0$, $q_k = p_{n+1}$, and for each $1 \leq h \leq k$, if $q_h = p_i$ for some i then q_{h-1} is the 'witness' p'_{i_j} of p_i, and otherwise, i.e., if $q_h = p'_i$ for some i, then $q_{h-1} = p_i$; in fact, the same chain exists in every I such that $\lambda^I(a)$ is satisfiable, so in abuse of notation we write $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_k$.

As $p_0(\vec{t}) \in I$, it follows that $p_{n+1}(\vec{t}) \in I$, and therefore $\neg C(\vec{t}) \in \lambda^I(a)$. Thus, $I \models^{\mathcal{O}} a$.

c4. $a = \mathrm{DL}[\lambda, C \sqcap p_0, C^0 \uplus p_0, C^0 \sqcap p'_0, C^1 \uplus p_1, C^1 \sqcap p'_1, \ldots,$
$C^n \uplus p_n, C^n \sqcap p'_n, D \uplus p_{n+1} D \uplus p'_{n+1};\, \neg C](\vec{t})$

where for every $i = 0, \ldots, n+1$, $p_i = p'_j$ for some $j < i$ or $p_i = p_0$ and $p'_{n+1} = p'_{i_j}$ for some $j \leq n$ or $p'_{n+1} = p_0$. Similarly as for the form $(c3)$, assuming that $\lambda^I(a)$ is satisfiable we obtain that there is a chain $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_k$ (using the notation from above) where $q_0 = p_0$, $q_k = p_{n+1}$, such that $p_{n+1}(\vec{t}) \in I$, and thus $D(\vec{t}) \in \lambda^I(a)$. Furthermore, there is a chain $r_0 \subseteq r_1 \subseteq \cdots \subseteq r_{k'}$ such that $r_0 = p_0$, $r_{k'} = p'_{n+1}$, and $p'_{n+1}(\vec{t}) \in I$, thus $\neg D(\vec{t}) \in \lambda^I(a)$; hence $\lambda^I(a)$ is unsatisfiable, which is a contradiction.

As in all cases either $\neg C(\vec{t}) \in \lambda^I(a)$ or $\lambda^I(a)$ is unsatisfiable, it follows $I \models^{\mathcal{O}} a$. As $\mathcal{O}$ and I are arbitrary, a is tautologic. □

Example 6.10 (cont'd)**.** The DL-atom $a = \mathrm{DL}[EF \cap fr, S \uplus fr, S \uplus fr; \neg EF](c)$ is an example of the tautologic form (c2). However, the DL-atom in the program of Example 6.1 is not of any form (c1)–(c4), and thus in general not tautologic.

Role queries

A careful analysis reveals that the result for tautologic DL-atoms with concept instance queries carries over to the case when the query $Q(\vec{t})$ is a role instance query. The same holds for negative concept and role instance queries, when the concept names C, D are replaced with names R_1, R_2 (and the predicates p, q are binary). For the latter consider $a = \mathrm{DL}[\lambda; \neg R](\vec{t})$ that is tautologic. Following the analysis in Proposition 6.9, which is generic in the arity of the tuple $\vec{t}$, necessarily the existence of roles R_1 and R_2 instead of C resp. D, and binary instead of unary input predicates p and q can be concluded, For example, the form (c3) above for the role query $\neg R_1$ results in $\mathrm{DL}[\gamma, R_1 \cap p, R_2 \cap q, R_2 \uplus p, R_2 \uplus q; \neg R_1](\vec{t})$, where R_1, R_2 are roles and p, q are binary predicates. More formally, the following is obtained.

Proposition 6.11. *Propositions 6.6 and 6.9 hold if C and D are replaced by role names R_1 and R_2, respectively (and p and q are binary instead of unary).*

Thus, as an interesting consequence, there is no interference of concept and role names in tautologic DL-atoms.

Axiomatization

Based on the results above, we obtain a calculus for the derivation of all tautologic DL-atoms as follows.

The axioms are:

a0. $\mathrm{DL}[; Q]()$,

a1. $\mathrm{DL}[S \cap p, S \uplus p; \neg S](\vec{t})$,

a2. $\mathrm{DL}[S \cap p, S' \uplus p, S' \uplus p; \neg S](\vec{t})$,

where $Q = S \sqsubseteq S$, $Q = S \sqsubseteq \top$, or $Q = \top \not\sqsubseteq \bot$, S, S' are either distinct concepts or distinct roles, and p is a unary resp. binary predicate.

The first rule of inference is reflecting the monotonicity of DL-atoms wrt. increasing input signatures:

Expansion

$$\frac{\mathrm{DL}[\lambda; Q](\vec{t})}{\mathrm{DL}[\lambda, \lambda'; Q](\vec{t})}\ (e).$$

The following rules state that an update predicate p may be replaced by q, if the latter has in case of consistent update $\lambda^I(a)$ a larger value than p:

Increase

$$\frac{\mathrm{DL}[\lambda, S \uplus q, S' \uplus p, S' \cap q;\ Q](\vec{t})}{\mathrm{DL}[\lambda, S \uplus p;\ Q](\vec{t})}\ (in_{\uplus}), \qquad \frac{\mathrm{DL}[\lambda, S \cup q, S' \uplus p, S' \cap q;\ Q](\vec{t})}{\mathrm{DL}[\lambda, S \cup p;\ Q](\vec{t})}\ (in_{\cup}).$$

Let $\mathcal{K}_{taut}$ denote the respective calculus.

Lemma 6.12. *Every ground DL-atom a of form $(c1)$ – $(c4)$ is derivable from axioms **a**1 and **a**2 using the rules (e), $(in_{\uplus})$, and $(in_{\cup})$.*

Proof. Forms $(c1)$ and $(c2)$ can be obviously obtained from **a**1 and **a**2 (where $(c2)$ with $D = C$ is obtained from **a**1 and rule (e)).

Form $(c3)$ is obtained from the instance $\mathrm{DL}[C \cap p, C \cup p;\ \neg C](\vec{t})$ of **a**1 by repeated application of the rule $(in_{\cup})$ that replaces, along the chain $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_k$ for a as in the proof of Proposition 6.9, $C \cup q_h$, $0 \leq h < k$ with $C \cup p'_i, C^i \uplus p_i, C^i \cap p'_i$ given that $q_h = p_i$, and by then applying rule (e) to fill up the form.

Form $(c4)$ is obtained from the instance $\mathrm{DL}[C \cap p, D \uplus p, D \cup p;\ \neg C](\vec{t})$ of **a**2 by similarly repeated application of the rule $(in_{\uplus})$ that replaces, along the chain $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_k$ for a as in the proof of Proposition 6.9 $D \uplus q_h$, $0 \leq h < k$ with $D \uplus p'_i, C^i \uplus p_i, C^i \cap p'_i$ given that $q_h = p_i$, and along the chain $r_0 \subseteq r_1 \subseteq \cdots \subseteq r_{k'}$ for a as in the proof of Proposition 6.9 $D \cup r_h$, $0 \leq h < k'$ with $D \cup p'_i, C^i \uplus p_i, C^i \cap p'_i$ given that $r_h = p_i$, and by then applying rule (e) to fill up the form. □

Since also correctness of the rules is easily established we have:

Theorem 6.13. *The calculus $\mathcal{K}_{taut}$ is sound and complete for the theory of tautologic ground DL-atoms.*

Proof. Soundness. It is easily seen that the rules (e), $(in_{\cup})$, and $(in_{\uplus})$ are sound. Indeed if a' results from a by rule (e), then $\lambda^I(a') \supseteq \lambda^I(a)$; if a' results from a by rule $(in_{\uplus})$ resp. $(in_{\cup})$, then either $\lambda^I(a')$ is unsatisfiable or again $\lambda^I(a') \supseteq \lambda^I(a)$ (and in case of satisfiability $p^I \subseteq q^I$ must hold).

Completeness. The completeness of the theory follows, as regards concept queries, from Propositions 6.6–6.9, and Lemma 6.12, and as regards roles from Proposition 6.11. □

Notice that in fact $\mathcal{K}_{taut}$ is minimal, i.e., no axiom scheme or inference rule is redundant.

6.2 Independence under Inclusion

In the previous section, we considered the existence of contradictory and tautologic DL-atoms in DL-programs in the general case, assuming that the rules of the DL-program are arbitrary. However, by simple analysis or by assertions, we might have information about the relationship between rule predicates that must hold in any model or answer set.

For example, suppose that a DL-program contains the rule

$$q(X) \leftarrow p(X). \tag{6.3}$$

It imposes an inclusion constraint on the predicates p and q, i.e., for every model I of the program, $p^I \subseteq q^I$ must hold. If p and q are input predicates for DL-atoms, then the rule (6.3) might affect the independence behavior of a DL-atom in the program: relative to the inclusion constraint, it might be tautologic. Similar rules might state inclusions between binary input predicates, e.g.

$$q(X,Y) \leftarrow p(X,Y), \tag{6.4}$$

$$q(Y,X) \leftarrow p(X,Y); \tag{6.5}$$

also projections, e.g.

$$r(X) \leftarrow p(X,Y), \text{ or } r(Y) \leftarrow p(X,Y), \tag{6.6}$$

(of p on r) might occur. An interesting question is how the presence of such predicate constraints influences the independence behavior, which we address in this section.

We call any rule

$$q(Y_1,\ldots,Y_n) \leftarrow p(X_1,\ldots,X_m) \tag{6.7}$$

where $n \leq m$ and the Y_i are pairwise distinct variables from $X_1,\ldots,X_m$ an *inclusion constraint (IC)*; if $n=m$, we also write $p \subseteq q$ if $Y_i=X_i$ and $p \subseteq q^-$ if $Y_i=X_{n-i+1}$, for all $i=1,\ldots,n$. Moreover, for the calculus for tautologic DL-atoms under inclusion constraints as developed in this section, we consider an extended language including p^- as a name, representing for every $p \in \mathcal{P}_o$ its inverse as defined above. By $Cl(\mathcal{C})$ we denote the closure of $\mathcal{C}$, i.e., the set of all ICs which are satisfied by every interpretation I such that $I \models \mathcal{C}$. In particular note that $p \subseteq q^- \models p^- \subseteq q$.

Let us now consider the impact of a set $\mathcal{C}$ on independence. To this end, we consider independence *relative to* $\mathcal{C}$, i.e., the interpretations I, I' in Definition 6.2 must satisfy $\mathcal{C}$.

Example 6.14 (cont'd)**.** Reconsider P in Example 6.1. We can include rule (2) (also written $ex \subseteq vi$) as an inclusion constraint to the set $\mathcal{C}$, and also rule (4). Moreover, as none of the fruits is known to be bitter in our context, we additionally include $ex \subseteq sw$ in $\mathcal{C}$. The closure $Cl(\mathcal{C})$ moreover contains the ICs $vi(X) \leftarrow so(X,Y)$ and $sw \leftarrow so(X,Y)$.

6.2.1 Contradictory DL-atoms

In what follows, we show that the presence of inclusion constraints $\mathcal{C}$ does not change the result regarding contradictory DL-atoms as obtained for the general case.

Proposition 6.15. *Let $a = \mathrm{DL}[\lambda; Q](t)$ be a ground DL-atom. Then a is contradictory relative to a set $\mathcal{C}$ of inclusion constraints iff $\lambda = \epsilon$ and $Q(t)$ is unsatisfiable.*

Proof. (If) Identical to the if-part of the proof of the Proposition 6.4.

(Only If) We use the same reasoning as in Proposition 6.4. If $\lambda \neq \epsilon$ then we can always find an interpretation I_0 such that $\lambda^{I_0}(a) \neq \emptyset$; indeed, we can use $I_0 = \emptyset$, if $\cap$ occurs in λ, and use $I_0 = \mathcal{HB}_\Pi$, i.e., the set of all ground atoms, otherwise. □

6.2.2 Tautologic DL-atoms

Next we investigate how the list of tautologies is modified when inclusion constraints are put on the predicates involved in them.

As we have noted above, the minimal forms of tautologic DL-atoms with concept (resp., role) queries involve only concepts and unary input predicates (resp., roles and binary input predicates).

An inclusion constraint of the form (6.6) in $\mathcal{C}$ (or the DL-program) will not allow us to get any further tautologic forms. E.g., consider the tautologic DL-atom $\mathrm{DL}[R \cap p, R \uplus p; \neg R](\vec{t})$ we intuitively should get that $\mathrm{DL}[R \cap r, R \uplus p; \neg R](\vec{t})$ is also tautologic. However, this is not a legal DL-atom, as the role R is extended by the unary predicate r.

Dependencies of the form (6.5) do not allow us to obtain new tautologic DL-atoms either. For example, consider a ground DL-atom $\mathrm{DL}[R \uplus p, R \cap p; \neg R](a, b)$, which has the form of axiom **a1**. If we replace the first occurrence of p by q, the resulting DL-atom $\mathrm{DL}[R \uplus q, R \cap p; \neg R](a, b)$ is not tautologic. However, for a constraint (6.4), it is tautologic; it also would be in the former case if the query argument is (a, a).

The following can be shown. For any DL-atom $a = \mathrm{DL}[\lambda; Q](\vec{t})$ and set $\mathcal{C}$ of ICs, let $inp_a(\mathcal{C})$ denote the set of all $q(\vec{Y}) \leftarrow p(\vec{X})$ in $\mathcal{C}$ such that p and q occur in λ. We call $\mathcal{C}$ *separable* for a, if every $ic \in inp_a(Cl(\mathcal{C}))$ involves predicates of the same arity.

Proposition 6.16. *Let $a = \mathrm{DL}[\lambda; Q](\vec{t})$ be a ground DL-atom and $\mathcal{C}$ a separable set of ICs for a. Then a is tautologic relative to $\mathcal{C}$ iff it is tautologic relative to $\mathcal{C}'$ which contains, depending on the type of $Q(\vec{t})$, the following constraints: (1) $\mathcal{C}' = \emptyset$, in case of a (negated) concept inclusion; (2) every $p \subseteq q$ in $inp_a(Cl(\mathcal{C}))$ where p, q are unary, in case of a (negated) concept instance; (3) every $p \subseteq q$ and $p \subseteq q^-$ in $inp_a(Cl(\mathcal{C}))$ where p, q are binary, in case of a (negated) role instance.*

Proof (sketch). Every model I of $\mathcal{C}$ is a model of $\mathcal{C}'$. On the other hand, by the form of the ICs, every model I' of $\mathcal{C}'$ can be extended to an interpretation I such that $I \models \mathcal{C}$. In general, the intersection I of all models $I'' \supseteq I'$, which is given by the answer set of $\mathcal{C} \cup I'$, fulfills the claim. Indeed, a fact $a = q(\vec{c})$ can be in I iff it is provable from I' using a sequence $r_1, r_2, \ldots, r_k$ of rules from $\mathcal{C}$. As all rules are unary, a can be proved

from some fact $a' = p(\vec{c'})$ in $\mathcal{C}$; unfolding the rules, we obtain a rule r of the form $q(\vec{Y}) \leftarrow p(\vec{X})$, where $\vec{Y} = Y_1, \ldots, Y_m$ are distinct variables from $\vec{X} = X_1, \ldots, X_n$. As $\mathcal{C} \models r$ and $\mathcal{C}$ is separable for a, it follows that $m = n$ and thus $r \in \mathcal{C}'$, which implies $a' \in I'$. Consequently, I is an extension of I' as claimed.

□

That is, for negative role queries we must in general take inverse predicate inclusions into account. To this end, we consider a language including for every $p \in \mathcal{P}_p$ a name p^- for its inverse (as defined in the paper). Such an inverse can be also effected by means of inclusions $q(Y, X) \leftarrow p(X, Y)$ and $p(Y, X) \leftarrow q(X, Y)$ in the set $\mathcal{C}$ of inclusion constraints (where q is then p^- and p is q^-).

Each rule $q(Y_1, Y_2) \leftarrow p(X_1, X_2)$ in $\mathcal{C}$ is then either an inclusion $p \subseteq q$ or an inclusion $p \subseteq q^-$. Note that $p \subseteq q$ iff $p^- \subseteq q^-$ and that for unary predicates, $p^- = p$ and is thus immaterial; furthermore, viewing $\cdot^-$ as an operator, $(p^-)^- = p$. We let $\mathcal{P}_p^{(-)} = \mathcal{P}_p \cup \{p^- \mid p \in \mathcal{P}_p\}$. To see some examples, consider the tautologic form (c1) in Proposition 6.9. Taking the inclusion constraint $p \subseteq q$ into account, we obtain the following new tautologic form:

- $\mathrm{DL}[\lambda, S \cap p, S' \uplus q;\ \neg S](\vec{t})$.

The form (c2) yields

- $\mathrm{DL}[\lambda, S \cap p, S' \uplus q, S' \uplus p;\ \neg S](\vec{t})$, where $p \neq q, S' \neq S$;
- $\mathrm{DL}[\lambda, S \cap p, S' \uplus p, S' \uplus q;\ \neg S](\vec{t})$, where $p \neq q, S' \neq S$;
- $\mathrm{DL}[\lambda, S \cap p, S' \uplus q, S' \uplus q;\ \neg S](\vec{t})$, where $p \neq q, S' \neq S$.

From the tautological DL-atom we get

- $\mathrm{DL}[\lambda, S \cap p, S' \uplus q, S' \cap r, S' \uplus r;\ \neg S](\vec{t})$, where $S' \neq S, p \neq r, q \neq r, p \neq q$.

Tautologic forms emanating from (c4) are redundant here, because its modification for the considered case is already included above. For the cases when the DL-query has any of the forms $S(\vec{c})$, $C \sqsubseteq D$ or $C \not\sqsubseteq D$, where S is either a concept or a role and C, D are concepts, there are no new tautologies.

6.2.3 Axiomatization for Tautologies

The results presented above allow us to define rules of inference for deriving tautologies when inclusion constraints are put on the input predicates of a DL-atom.

Inclusion

$$\frac{\mathrm{DL}[\lambda, S \uplus p;\ Q](\vec{t}) \quad p \subseteq q}{\mathrm{DL}[\lambda, S \uplus q;\ Q](\vec{t})}\ (i_1), \tag{6.8}$$

$$\frac{\mathrm{DL}[\lambda, S \uplus p;\ Q](\vec{t}) \quad p \subseteq q}{\mathrm{DL}[\lambda, S \uplus q;\ Q](\vec{t})}\ (i_2). \tag{6.9}$$

The "increase" rules are slightly adapted, in comparison to the general case by taking into account that $p \subseteq q$ iff $p^- \subseteq q^-$:

Increase

$$\frac{\mathrm{DL}[\lambda, S \uplus p;\, Q](\vec{t})}{\mathrm{DL}[\lambda, S \uplus q, S' \uplus p, S' \cap q;\, Q](\vec{t})}\ (in_{\uplus}), \quad \frac{\mathrm{DL}[\lambda, S \cup p;\, Q](\vec{t})}{\mathrm{DL}[\lambda, S \cup q, S' \uplus p, S' \cap q;\, Q](\vec{t})}\ (in_{\cup}),$$

$$\frac{\mathrm{DL}[\lambda, S \uplus p;\, Q](\vec{t})}{\mathrm{DL}[\lambda, S \uplus q, S' \uplus p^-, S' \cap q^-;\, Q](\vec{t})}\ (in^-_{\uplus}), \quad \frac{\mathrm{DL}[\lambda, S \cup p;\, Q](\vec{t})}{\mathrm{DL}[\lambda, S \cup q, S' \uplus p^-, S' \cap q^-;\, Q](\vec{t})}\ (in^-_{\cup}),$$

where $p, q \in \mathcal{P}_p^{(-)}$ are of the same arity.

We consider the following extended set of axioms compared to the case without inclusion constraints:

a0. $\mathrm{DL}[;\, Q]()$,

a1. $\mathrm{DL}[S \cap p, S \cup p;\, \neg S](\vec{t})$,

a2. $\mathrm{DL}[S \cap p, S' \uplus q, S' \cup q;\, \neg S](\vec{t})$, where $q \in \{p, p^-\}$,

and $Q = S \sqsubseteq S$, $Q = S \sqsubseteq \top$, or $Q = \top \not\sqsubseteq \bot$, S, S' are either distinct concepts or distinct roles; moreover, p is a unary or binary predicate.

The described axioms and rules together with the expansion rule defined above, form a calculus for the derivation of tautologic DL-atoms, which we denote by $\mathcal{K}^{\subseteq}_{taut}$.

Lemma 6.17. *Let $a = \mathrm{DL}[\lambda;\, Q](\vec{t})$ be a ground tautologic DL-atom relative to a separable set $\mathcal{C}$ of inclusion constraints for a. Then $a' = \mathrm{DL}[\lambda';\, Q](\vec{t})$ where λ' contains all $S_i\ op_i\ p_i$ from λ such that p_i has the same arity as the predicates in Q is also tautologic (and $\mathcal{C}$ is separable for a').*

Proof. Consider the DL-atoms a and a' defined as above. By Proposition 6.16, without loss of generality we may assume that $\mathcal{C} = inp_a(Cl(\mathcal{C}))$ and each predicate p occurring in $\mathcal{C}$ has the arity of the predicates occurring in Q; note that p occurs in λ'. Towards a contradiction, suppose that a' is not tautologic. Hence there exists some $\mathcal{O}$ and I such that $I \not\models^{\mathcal{O}} a'$. That is, $\mathcal{O} \cup \lambda^I(a') \not\models Q(\vec{t})$. Hence, by monotonicity $\lambda^I(a') \not\models Q(\vec{t})$, so without loss of generality $\mathcal{O} = \emptyset$.

Let I' be the extension of I such that all predicates not occurring in λ' are empty. Note that $I' \models \mathcal{C}$. As $\lambda' \subseteq \lambda$, it follows that $\lambda^{I'}(a)$ is of the form $\lambda^{I'}(a) = \lambda^{I'}(a') \cup S = \lambda^I(a') \cup S$, where S is set of negative literals whose predicates do neither occur in $\lambda^I(a')$ nor in Q (they have different arity). Hence, as $\lambda^{I'}(a)$ is satisfiable and $\lambda^I(a') \not\models Q(\vec{t})$, it follows that $\lambda^{I'}(a) \not\models Q(\vec{t})$. This, however means that for $\mathcal{O} = \emptyset$, $I \not\models^{\mathcal{O}} Q(\vec{t})$, which means that a is not tautlogic relative to $\mathcal{C}$. This is a contradiction. □

As an immediate consequence, we obtain:

Corollary 6.18. *Every minimal form of a tautologic ground DL-atom $a = \mathrm{DL}[\lambda; Q](\vec{y})$ (i.e., λ can not be replaced with $\lambda' \subset \lambda$) relative to a separable set $\mathcal{C}$ of inclusion constraints for a, involves only predicates of the same arity as the DL-query of a.*

The main result of this section, following next, is its soundness and completeness.

Theorem 6.19. *The calculus $\mathcal{K}^{\subseteq}_{taut}$ is sound and complete for tautologic ground DL-atoms a relative to any closed set of inclusion constraints $\mathcal{C}$ (i.e., such that $\mathcal{C} = Cl(\mathcal{C})$) that is separable for a.*

Proof. Soundness. From Theorem 6.13, it follows that all instances of the axiom schemata **a**0–**a**2 are sound relative to $\mathcal{C}$. Furthermore, it is not difficult to establish that its inference rules are sound. Indeed, whenever a DL-atom a' results from a DL-atoms $a = \mathrm{DL}[\lambda; Q](\vec{t})$ by any of the rules, then for every interpretation I, either $\lambda^I(a')$ is unsatisfiable or $\lambda^I(a') \supseteq \lambda^I(a)$, and thus as $\mathcal{O} \cup \lambda^I(a) \models Q(\vec{t})$, it holds that $\mathcal{O} \cup \lambda^I(a') \models Q(\vec{t})$ by monotonicity, for every ontology $\mathcal{O}$.

Completeness. Assume that $a = \mathrm{DL}[\lambda; Q](\vec{t})$ of the form (2.3) is a tautologic relative to a closed set $\mathcal{C}$ of constraints that is separable for a. Without loss of generality, by Corollary 6.18 and the expansion rule a is minimal, and thus contains only predicates of the same arity as the DL-query Q of a, and by Proposition 6.16 we may assume that $\mathcal{C} = inp_a(\mathcal{C})$. We consider different cases according to Q.

(I) $Q = C \sqsubseteq D$ ($Q = \neg(C \sqsubseteq D)$). By Proposition 6.16, a is a tautologic relative to $\mathcal{C}$ iff it is tautologic relative to $\mathcal{C}' = \emptyset$. By the completeness of $\mathcal{K}_{taut}$ for $\mathcal{C}' = \emptyset$ and since $\mathcal{K}^{\subseteq}_{taut}$ subsumes $\mathcal{K}_{taut}$, a is provable in $\mathcal{K}^{\subseteq}_{taut}$.

(II) $Q = C$ or $Q = \neg C$. By assumption, $\mathcal{C}$ contains only inclusions $p \subseteq q$, where p and q are unary.

Next, we argue for the DL-atoms with query $C(\vec{t})$ and $\neg C(\vec{t})$ separately.

- $Q = C(\vec{t})$. According to Proposition 6.6, for no λ a DL-atom $\mathrm{DL}[\lambda; C](\vec{t})$ is tautologic if $\mathcal{C} = \emptyset$. In fact the situation does not change even if $\mathcal{C} \neq \emptyset$. Indeed, the presence of inclusion constraints $p \subseteq q$ in $\mathcal{C}$ still allows us to consider interpretation $I_0 = \emptyset$ for which, as shown in the proof of Proposition 6.6, we can never find an input signature λ of a such that $\lambda^{I_0}(a) \models C(\vec{t})$. So a can not be of this form.

- $Q = \neg C(\vec{t})$. Similar as in the proof of Proposition 6.9, we conclude that λ must contain some (*) $C \sqcap p$, by considering $I_0 = \emptyset$, as $I_0 \models \mathcal{C}$. Now let $I' \supseteq \{p(\vec{t})\}$ be the least interpretation such that $I' \models \mathcal{C}$. We distinguish whether $\lambda^{I'}(a)$ is satisfiable or not.

  (a) $\lambda^{I'}(a)$ is satisfiable. Then $\neg C(t) \in I'$, which implies that $C \sqcup q \in \lambda$ and that $p \subseteq q$. But then a is derivable from the instance $\mathrm{DL}[C \sqcap p, C \sqcup p; \neg C](\vec{t})$ of axiom **a**1 using rule (i_1).

(b) $\lambda^{I'}(a)$ is unsatisfiable. Hence $S(\vec{t}), \neg S(\vec{t}) \in \lambda^{I'}(a)$ for some $S \in \mathcal{P}_o$. As $S(\vec{t}) \in \lambda^{I'}(a)$, it follows that (**) $S \uplus p'$ occurs in λ such that $p \subseteq p'$.

The presence of $\neg S(\vec{t})$ in $\lambda^{I'}(a)$ is due to some $S_i\ op_i\ p_i$ in λ where $S_i = S$, for which we consider the following possible cases:

(b.1) $op_i = \uplus$. Then $S \uplus q \in \lambda$ such that $p \subseteq q$. But then a is derivable if $S = C$, from the instance $\mathrm{DL}[C{\cap\!\!\!\cdot}p, C{\uplus}p;\ \neg C](\vec{t})$ of $\mathbf{a}1$, using (i_1) and (e), and if $S \neq C$ from the instance $\mathrm{DL}[C{\cap\!\!\!\cdot}p, S \uplus p, S{\uplus}p;\ \neg C](\vec{t})$ of $\mathbf{a}2$ by applying the rules (i_2) on $S \uplus p$ and (i_1) on $S{\uplus}p$, respectively.

(b.2) $op_i = \cap\!\!\!\cdot$. I.e., we have $S{\cap\!\!\!\cdot}p_i \in \lambda$, and as $\neg S(\vec{t}) \in \lambda^{I'}(a)$, it follows that $p \not\subseteq p_i$. Let now $I'' \supseteq \{p(\vec{t}), p_i(\vec{t})\}$ be the least interpretation such that $I'' \models \mathcal{C}$.

If $\lambda^{I''}(a)$ is satisfiable, then some $C{\uplus}q' \in \lambda$ such that $\mathcal{C} \models p_i \subseteq q'$. In this case, a is derivable from an instance $\mathrm{DL}[C{\cap\!\!\!\cdot}p, C{\uplus}p;\ \neg C](\vec{t})$ of $\mathbf{a}1$ by applying

- rule (i_1) on $C{\uplus}p$ for $p \subseteq p'$, which yields $\mathrm{DL}[C{\cap\!\!\!\cdot}p, C{\uplus}p';\ \neg C](\vec{t})$; then
- rule $(in_{\uplus})$ on $C{\uplus}p'$ for introducing $S{\cap\!\!\!\cdot}p_i$, which yields
  $\mathrm{DL}[C{\cap\!\!\!\cdot}p, C{\uplus}p_i, S{\uplus}p', S{\cap\!\!\!\cdot}p_i;\ \neg C](\vec{t})$;
- rule (i_1) on $C{\uplus}p_i$ for $p_i \subseteq q'$, which yields
  $\mathrm{DL}[C{\cap\!\!\!\cdot}p, C{\uplus}q', S{\uplus}p', S{\cap\!\!\!\cdot}p_i;\ \neg C](\vec{t})$;

and filling up λ with rule (e).

Otherwise, $\lambda^{I''}(a)$ is unsatisfiable, i.e., $S'(\vec{t}), \neg S'(\vec{t}) \in \lambda^{I''}(a)$ for some S'. This implies that $S'{\uplus}\ p'' \in \lambda$ for some p'' such that either $p \subseteq p''$ or $p_i \subseteq p''$, and either

(i) $S' \uplus q \in \lambda$, such that $p \subseteq q$ or $p_i \subseteq q$, or

(ii) $S' \cap\!\!\!\cdot\, q \in \lambda$ such that $p \not\subseteq q$ and $p_i \not\subseteq q$.

In case (i), we thus have $C{\cap\!\!\!\cdot}p, S{\uplus}p', S{\cap\!\!\!\cdot}p_i, S'{\uplus}p'', S'{\uplus}q$ in λ such that $p \subseteq p'$, and either (a) $p \subseteq p''$ or (b) $p_i \subseteq p''$, as well as either (1) $p \subseteq q$ or (2) $p_i \subseteq q$, holds. We can derive a from the instance $\mathrm{DL}[C{\cap\!\!\!\cdot}p, S'{\uplus}p, S'{\uplus}p;\ \neg C](\vec{t})$ of axiom $\mathbf{a}1$ by applying, depending on the emerging four possible combinations, the following rules:

- (a1): $p \subseteq p''$ and $p \subseteq q$
  - (i_2) on $S'{\uplus}p$ for $p \subseteq p''$,
  - (i_1) on $S'{\uplus}p$ for $p \subseteq q$, and
  - fill up with rule (e);
- (b2): $p_i \subseteq p''$ and $p_i \subseteq q$
  - (i_2) on $S'{\uplus}p$ for $p \subseteq p'$,
  - (i_1) on $S'{\uplus}p$ for $p \subseteq p'$,
  - $(in_{\uplus})$ on $S'{\uplus}p'$ to introduce $S'{\uplus}p_i$, $S{\uplus}p'$, $S{\cap\!\!\!\cdot}p_i$,

– $(in_{\sqcup})$ on $S' \uplus p'$ to introduce $S \uplus p_i$, $S \uplus p'$, $S \sqcap p_i$,
– (i_2) on $S' \uplus p_i$ for $p_i \subseteq p''$,
– (i_1) on $S' \uplus p_i$ for $p_i \subseteq q$, and finally,
– fill up with rule (e).

- (a2): $p \subseteq p''$ and $p_i \subseteq q$
  – (i_2) on $S' \uplus p$ for $p \subseteq p''$,
  – (i_1) on $S' \uplus p$ for $p \subseteq p'$,
  – $(in_{\sqcup})$ on $S' \uplus p'$ to introduce $S' \uplus p_i$, $S \uplus p'$, $S \sqcap p_i$,
  – (i_1) on $S' \uplus p_i$ for $p_i \subseteq q$, and finally,
  – fill up with rule (e).
- (b1): $p_i \subseteq p''$ and $p \subseteq q$
  – (i_1) on $S' \uplus p$ for $p \subseteq q$,
  – (i_2) on $S' \uplus p$ for $p \subseteq p'$,
  – $(in_{\uplus})$ on $S' \uplus p'$ to introduce $S' \uplus p_i$, $S \uplus p'$, $S \sqcap p_i$,
  – (i_2) on $S' \uplus p_i$ for $p_i \subseteq p''$, and
  – fill up with rule (e).

In case (ii) we can continue the argument, by considering the least interpretation $I''' \supseteq I'' \cup \{q(\vec{t})\}$ such that $I''' \models \mathcal{C}$. This I''' exists and either $\lambda^{I'''}(a)$ is satisfiable and the query is satisfied by some element $C \uplus q'$ in λ, or $\lambda^{I'''}(a)$ is unsatisfiable and contains some contradictory pair $S''(\vec{t}), \neg S''(\vec{t})$ etc. Eventually, we must arrive at a list of form similar to (6.1), viz.

$$C \sqcap p, S^0 \uplus p_0, S^0 \sqcap p'_0, S^1 \uplus p_1, S^1 \sqcap p'_1, \ldots, S^n \uplus p_n, S^n \sqcap p'_n, C \uplus p_{n+1} \tag{6.10}$$

or to (6.2), viz.

$$C \sqcap p, S^0 \uplus p_0, S^0 \sqcap p'_0, \ldots, S^n \uplus p_n, S^n \sqcap p'_n, S^{n+1} \uplus p_{n+1} S^{n+1} \uplus p'_{n+1} \tag{6.11}$$

where $p_0 \supseteq p$, $n \geq 0$, and each $p_i \supseteq p'_j$ for some $j < i$ or $p_i \supseteq p_0$ and $p'_{n+1} \supseteq p'_j$ for some $j \leq n$ or $p'_{n+1} \supseteq p_0$; that is, instead of equalities "=" between predicates we have inclusion "$\supseteq$".

Now in case of form (6.10), similarly as for form (6.1) in the proof of the if-part of Proposition 6.9, we conclude that there is a chain $q_0^I \subseteq q_1^I \subseteq \cdots \subseteq q_k^I$ for the eventual interpretation I such that $q_0 = p$, $q_k = p_{n+1}$, and for each $1 \leq h \leq k$, (*) if $q_h = p_i$ for some i and $h > 1$, then q_{h-1} is the 'witness' p'_{i_j} of p_i, and otherwise, i.e., (**) if $q_h = p'_i$ for some i, then $q_{h-1} = p_i$; again, the same chain exists in every I such that $\lambda^I(a)$ is satisfiable, so we simply write $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_k$.

We then can derive a from the instance $\mathrm{DL}[C \sqcap p, C \uplus p; \neg C](\vec{t})$ of axiom **a**1 by repeated application of the rules (i_1) and $(in_{\sqcup})$ along the chain $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_k$, where we replace $C \uplus p$ by $C \uplus p_0$ applying (i_1) and then replace $C \uplus q_h$, $1 \leq h < k$,

in case (*) with $C \uplus p_i$ applying (i_1) and in case (**) with $C \uplus p'_i, C^i \uplus p_i, C^i \cap p'_i$; after that, we apply rule (e) to fill up the form.

In case of form (6.11), we conclude that similarly as for (6.2) and (6.10), that there is a chain $q_0^I \subseteq q_1^I \subseteq \cdots \subseteq q_k^I$ for the eventual interpretation I such that $q_0 = p$, $q_k = p'_{n+1}$, and for each $1 \leq h \leq k$, (*) if $q_h = p_i$ for some i and $h > 1$, then q_{h-1} is the 'witness' p'_{i_j} of p_i, and otherwise, i.e., (**) if $q_h = p'_i$ for some i, then $q_{h-1} = p_i$, and a chain $r_0^I \subseteq r_1^I \subseteq \cdots \subseteq r_{k'}^I$ such that $r_0 = p$, $r_{k'} = p_{n+1}$, and for each $1 \leq h \leq k$, (*) if $r_h = p_i$ for some i and $h > 1$, then r_{h-1} is the 'witness' p'_{i_j} of p_i, and otherwise, i.e., (**) if $r_h = p'_i$ for some i, then $r_{h-1} = p_i$.

We can then derive a from the instance $\mathrm{DL}[C \cap p, S \uplus p, S \uplus p; \neg C](\vec{t})$ of **a2** along these chains, by repeated application of the rules (i_1), (i_2), $(in_{\uplus})$, and $(in_{\cup})$ along the chains $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_k$, and $r_0 \subseteq r_1 \subseteq \cdots \subseteq r_{k'}$, where we first replace $S \uplus p$ by $S \uplus p_0$ applying (i_2) and $S \uplus p$ by $S \uplus p_0$ applying (i_1), and then process the chain $q_0 \subseteq q_1 \subseteq q_k$ using (i_2) and $(in_{\cup})$, and the chain $r_0 \subseteq r_1 \subseteq \cdots \subseteq r_{k'}$ using (i_1) and $(in_{\uplus})$, where the application of $(in_{\uplus})$ and $(in_{\cup})$ introduces the terms $S^i \uplus p_i, S^i \cap p'_i$; finally, rule (e) is applied to fill up the form.

- $Q = R(\vec{t})$. Similar as the case of $Q = C(\vec{t})$ (cf. Propositions 6.6 and 6.11), where all roles are binary instead of unary).

- $Q = \neg R(\vec{t})$. The proof for this case is similar as for the case where $Q = \neg C(\vec{t})$, but here also inverse predicates p^- come into play. Note that in the argument for the concept query case, only interpretations I, I' etc with facts involving the query tuple $\vec{t}$ play a role. For role queries, we can see that similarly interpretations with facts over $\vec{t} = (t_1, t_2)$ and its inverse $\vec{t^-} = (t_2, t_1)$ are sufficient: indeed, from any interpretation I such that $I \not\models^{\mathcal{O}} a$ we can remove all facts $S(\vec{t'})$, $\vec{t'} \neq \vec{t}, \vec{t^-}$ and the resulting interpretation I' satisfies $I' \not\models^{\mathcal{O}} a$. In fact, by taking into account that $I \models p(\vec{t^-})$ iff $I \models p^-(\vec{t})$ it is sufficient to consider interpretations of $\mathcal{P}_p$ on the query tuple $\vec{t}$, coherently extended to the vocabulary $\mathcal{P}_p^{(-)}$.

  The line of argumentation is then analogous as for a concept query $\neg C(\vec{t})$, where one takes also inverse predicates into account. Note that a new tautologic form emerges, viz. $\mathrm{DL}[S \cap p, S' \uplus p^-, S' \cup p^-; \neg S](\vec{t})$ as in axiom **a2**, and that by the equivalence of $p \subseteq q$ and $p^- \subseteq q^-$ respective variants $(in_{\uplus}^-)$ and $(in_{\cup}^-)$ of $(in_{\uplus})$ and $(in_{\cup})$ are added. The given calculus is in fact complete for the extended language.

□

We use our running example to illustrate the application of $\mathcal{K}_{taut}^{\subseteq}$.

Example 6.20 (cont'd)**.** Reconsider the DL-program in Example 6.1, and recall that no ground instance of its DL-atom, in particular

$$a = \mathrm{DL}[H \uplus vi, H \cup sw, A \cap ex; \neg A](pineapple)$$

is tautologic.

Now let us take the predicate constraints in P into account. Recall that essentially by the rules (2) and (3), we have that $\{ex \subseteq vi, ex \subseteq sw\} \subseteq Cl(\mathcal{C})$ (which is also separable for a).

We thus can derive a in $\mathcal{K}^{\subseteq}_{taut}$ given $\mathcal{C}$ as follows:

$$\dfrac{\dfrac{\dfrac{\mathrm{DL}[H \uplus ex, H \uplus ex, A \cap ex;\, \neg A](pineapple)}{\mathrm{DL}[H \uplus ex, H \uplus ex, A \cap ex;\, \neg A](pineapple) \quad ex \subseteq vi}\,(i_2)}{\mathrm{DL}[H \uplus vi, H \uplus ex, A \cap ex;\, \neg A](pineapple) \quad ex \subseteq sw}\,(i_2)}{\mathrm{DL}[H \uplus vi, H \uplus sw, A \cap ex;\, \neg A](pineapple)}\,(i_1)$$

The leaf of the proof tree is a DL-atom $\mathrm{DL}[H \uplus ex, H \uplus ex, A \cap ex;\, \neg A](pineapple)$. It has the form of axiom **a2**. Hence the initial DL-atom a is, by virtue of Theorem 6.19, tautologic relative to $\mathcal{C}$.

The results of this section can be readily used for optimization or reasoning tasks on DL-programs that involve ground DL-atoms, especially DL-program repair computation. They can moreover be exploited for dealing with non-ground DL-atoms. We may call such a DL-atom $a = \mathrm{DL}[\lambda;\, Q](\vec{X})$ independent (resp. contradictory, tautologic), if each of its ground instances has this property. From the results above, we obtain that there are no contradictory nonground DL-atoms, and that for proving that a is tautologic, it is sufficient to consider a single instance a (particular constants do not matter, and for role queries $(\neg)R(t_1, t_2)$, considering different constants if possible).

Example 6.21. In our running example, e.g., the instance of a for $X = pineapple$ is tautologic relative to the constraints; hence a is tautologic and can be removed from rule (5).

6.3 Complexity

Let us now consider the complexity of determining whether a DL-atom a is independent. To determine whether a is contradictory is trivial, given the simple forms of unsatisfiable DL-queries. For determining whether a is tautologic, we can use the calculus $\mathcal{K}^{\subseteq}_{taut}$ established above, and aim at a derivation of a. In the search, we need an oracle for deciding whether $ic \in Cl(\mathcal{C})$, for a given IC ic and $\mathcal{C}$, to see whether a rule is applicable.

The complexity of this oracle is in fact the dominating factor for the search. Indeed, the inclusion rules of $\mathcal{K}^{\subseteq}_{taut}$ work strictly local, in the sense that they only replace one occurrence of an input predicate by another one, and few independent rule applications are needed to arrive at an axiom (see below).

The complexity of deciding, given an IC ic and a set $\mathcal{C}$ of ICs, whether $ic \in Cl(\mathcal{C})$, depends on the form of the ICs. In general, the problem is decidable in polynomial space, and it is NLogSpace-complete if the arities of the predicates in $\mathcal{C}$ are bounded by a constant k. In particular, for $k = 2$ deciding $ic \in Cl(\mathcal{C})$ if all predicates in ic have the

same arity, is possible using the following inference rules:

$$\frac{X \subseteq Y \quad Y \subseteq Z}{X \subseteq Z} \qquad \frac{X \subseteq Y}{X^- \subseteq Y^-} \qquad \frac{X^- \subseteq Y^-}{X \subseteq Y} \tag{6.12}$$

where X, Y, Z are meta variables which denote unary (binary) predicates. On the other hand, the problem is NLogSpace-hard for every $k \geq 1$ as it subsumes graph reachability.

We have the following result.

Theorem 6.22. *Given a DL-atom a and a separable set $\mathcal{C}$ of ICs for a, deciding whether a is tautologic relative to $\mathcal{C}$ is (i)* NLogSpace*-complete and* NLogSpace*-hard even if $\mathcal{C} = \emptyset$, and is (ii) in* LogSpace*, and in fact expressible by a fixed first-order formula (hence in* AC^0*), if the DL query Q of a is not a negative concept resp. role query.*

Proof (sketch). By the above results on $\mathcal{K}^{\subseteq}_{taut}$, we need an oracle for $ic \in Cl(\mathcal{C})$, where ic involves only unary resp. binary predicates. Due to the special form of ICs, $ic \in Cl(\mathcal{C})$ iff $ic \in Cl([\mathcal{C}]_2)$, where $[\mathcal{C}]_2$ is the set of all ICs in $\mathcal{C}$ that involve only unary and/or binary predicates. Thus, by the observation above, an NLogSpace oracle is sufficient.

To prove that $a = \mathrm{DL}[\lambda; Q](\vec{t})$ is tautologic, we can guess an instance of an axiom **ai** from which we want to arrive at a by application of rules in $\mathcal{K}^{\subseteq}_{taut}$. Checking that $Q(\vec{t})$ matches the query of **ai** is easy, and we can check in case of **a**1, **a**2 that $S \barwedge p$ occurs in λ; we then can check whether $S \veebar p$ resp. $S' \uplus p^{(-)}, S' \veebar p^{(-)}$ occur in λ, and if not, in case of **a**1 build nondeterministically a "chain" $q_0 (= p) \subseteq q_1 \subseteq \cdots \subseteq q_k$ such that $S' \veebar q_k \in \lambda$ and in case of **a**2 also a "chain" $r_0 (= p) \subseteq r_1 \subseteq \cdots \subseteq r_{k'}$ such that $S' \uplus q_{k'} \in \lambda$, where for every q_i, we have that either $q_{i-1} \subseteq q_i$ (which can be checked with the oracle), or some pair $S'' \uplus q_{i-1}^{(-)}, S'' \barwedge q_i^{(-)}$ occurs in λ and similarly, for every r_j we have that either $r_{j-1} \subseteq r_j$ (an oracle check), or some pair $S'' \uplus r_{j-1}^{(-)}, S'' \barwedge r_j^{(-)}$ occurs in λ; building a chain stops as soon as $S' \veebar q_i \in \lambda$ resp. $S' \uplus r_i \in \lambda$ is found (it may else stop after a certain number of steps, but this is irrelevant here).

A simple analysis reveals that this overall algorithm is feasible, relative to the oracle, in logarithmic space (one can cycle through the few guesses with constantly many variables, and building chains as above is feasible in nondeterministic logarithmic space, as we just need to memorize q_i, p_0, p_{n+1} resp. p'_{n+1}, and S'). It follows that in general, the problem is in NLogSpace.

The problem is shown to be NLogSpace-hard via a reduction from the canonical graph reachability problem. Let $G = (V, E)$ be a directed graph and let $s, t \in V$ be nodes. We view each node $v \in V$ as a unary predicate, and define the DL-atom $a = \mathrm{DL}[C \barwedge s, \lambda, C \veebar t; \neg C](a)$ where λ contains for each edge $(v, w) \in E$ the elements $C^{(v,w)} \uplus v, C^{(v,w)} \barwedge w$, where $C^{(v,w)}$ does not occur elsewhere. Then it holds that a is tautologic (wrt. $\mathcal{C} = \emptyset$) iff t is reachable from s in G. Indeed, note that by its form, a must be derived from an instance $\mathrm{DL}[C \barwedge s, C \veebar t; \neg C](a)$ of **a**1, and that for this a chain $q_0 = s \subseteq q_1 \subseteq \cdots \subseteq q_k = t$ must be built to obtain $C \veebar t$, and only the rule $(in_{\veebar})$ is applicable. This chain corresponds to a path in G from s to t. Conversely from any path $s = v_0, v_1, \ldots v_k = t$ in G, we can build a corresponding chain with elements $C^{(v,w)} \uplus v, C^{(v,w)} \barwedge w$ in λ using the rule $(in_{\veebar})$.

Finally, if Q is not a negative concept resp. role query, then for a to be tautologic it must be an instance of $\mathbf{a}0$, which is checkable in logarithmic space and also expressible by a FOL formula ϕ over a relational structure (roughly, a plain SQL query over a database) that stores in suitable relations: all triples $S_i\ op_i\ p_i$ in λ, using S_i, op_i, and p_i as constants; the query $Q(\vec{t})$; and all inclusions $p \subseteq q^{(-)}$ from $Cl(\mathcal{C})$. In fact, ϕ can be fixed, and the relations are easily assembled from a and $\mathcal{C}$. As evaluating a fixed FOL formula over relational structures is in AC^0, we obtain the result. □

6.4 Discussion and Outlook

To the best of our knowledge, the notion of independent DL-atom has not been considered before, which is of use in optimization and for reasoning tasks on DL-programs. We investigated the forms of tautologic and contradictory ground DL-atoms in the general case, as well as in the case when inclusion constraints on the input predicates are known. We showed that contradictory DL-atoms have a simple form, and we presented a sound and complete calculus for determining tautologic DL-atoms. Based on it, we determined the complexity of deciding this problem, and showed that the problem is very efficiently solvable in general, as well as relative to the predicate constraints. Furthermore, the results for ground DL-atoms can be easily lifted to deal with nonground DL-atoms, and an implementation of the calculus using logic programming is rather straightforward. Incorporation of the latter into the dlvhex system remains for future work.

Several issues remain for further investigation. A possible extension is to consider DL queries which allow for non-atomic concepts, respectively roles. Some of our results can be readily extended to such queries (e.g., to conjunctive concept/role queries), but to get a clear picture further work is needed. As an alternative, or in addition to ICs, further information about the DL-program might be available relative to which independence of a DL-atom can be established. Regarding predicate constraints, one issue is non-separable sets of inclusion constraints, i.e., to permit projections among input predicates of DL-atoms, for which the presented calculus is sound but not complete. One can also imagine more general inclusion constraints, by relaxing the conditions to allow e.g. repetition of arguments, or inclusion of intersections. Other possibilities are to consider exclusion constraints, or (non-)emptiness constraints on predicates. Adopting a technical view, we could consider arbitrary sets of constraints that describe an envelope of the set of answer sets of the underlying DL-program. The study of different forms of constraints remains to be done. Orthogonal to rules, one may exploit information about the ontology $\mathcal{O}$. So far, no information about the concepts (roles) in $\mathcal{O}$ was assumed to be available, viewing $\mathcal{O}$ as blackbox under full information hiding. However, information about $\mathcal{O}$ may lead to further independent DL-atoms. For example, knowing that $\mathcal{O} \models C \sqsubseteq D$ and that $\mathrm{DL}[\lambda, C \uplus p; Q](\vec{t})$ is tautologic, we can infer that $\mathrm{DL}[\lambda, D \uplus p; Q](\vec{t})$ is also tautologic. Incorporating such and further information into the calculus remains for future work.

CHAPTER 7

Implementation and Evaluation

In this chapter we focus on the practical part of this thesis. We discuss the key implementation issues involved in deploying the repair answer set computation approaches seen in Chapter 5, and present results demonstrating the performance and practical relevance of the developed algorithms. The chapter is divided into two main sections. In Section 7.1 we describe the system architecture and usage of the developed software components for the deletion repair answer set computation. In Section 7.2 we analyze their performance by discussing the results of the conducted experiments.

7.1 Implementation

We have implemented the repair answer set computation algorithms within the dlliteplugin library of the dlvhex framework, thus providing means to effectively compute deletion repair answer sets for DL-programs over *DL-Lite*$_{\mathcal{A}}$ and $\mathcal{EL}$ ontologies.

The dlvhex framework [1] is a system for evaluating Answer Set Programs with external computations. The system is written in C++ and it is an open source software[2]. The underpinning functions for external sources can be conveniently implemented within the plugins of the dlvhex system, which distinguishes it from other ASP solvers. There is a wide range of such plugins already at avail, ranging from string manipulation functions to complex plugins implementing Equilibrium-semantics of Multi-Context Systems.

The first plugin for DL-atoms[3] was developed in 2007 as part of the master thesis of T. Krennwallner [Kre07] and the PhD thesis of R. Schindlauer [Sch06]. Evaluation of DL-atoms within this plugin is done by means of calls to the RacerPro system[4]. Technical peculiarities of the communication process between the dlvhex and the RacerPro did not

[1] http://www.kr.tuwien.ac.at/research/systems/dlvhex
[2] https://github.com/hexhex
[3] https://github.com/hexhex/dlplugin
[4] http://racer.sts.tuhh.de

allow for a smooth extension of the available plugin to repair answer set computation. This called for the development of a new separate dlliteplugin which was created during this thesis. It effectively evaluates and repairs DL-programs over light-weight ontologies.

The source code of the new plugin is available at `https://github.com/hexhex/dlliteplugin`. The dlliteplugin uses the owlcpp[5] [LRMC11] library for ontology parsing and invokes the fact++[6] system as a back-end for ontology reasoning tasks. When repairing DL-programs over $\mathcal{EL}$ ontologies it also communicates with the REQUIEM reasoner[7] for support set construction.

In what follows, we present an architectural overview of the dlliteplugin, its implementation details, installation and usage including description of available command-line options.

7.1.1 Architectural Overview

Plugins for the dlvhex system are responsible for evaluation of the respective external atoms, for which they are designed. In other words, plugins can be viewed as query evaluation units. The actual answer set computation is implemented in model generators located in the main part of the dlvhex system. Model generators exploit different optimization heuristics, and they perform particularly well for certain types of programs. Based on the input instance analysis the dlvhex evaluation framework invokes a suitable model generator, which is more likely to be effective on the given instance. The model generators are generic and normally they can be used within any plugin.

The main practical goal of this work was the development of a model generator, capable of computing repair answer sets for DL-programs. By the time this thesis project was started there were no comparable model generators in the dlvhex system. Besides being nontrivial, development of a generic repair model generator suitable for any HEX-program is not necessarily practical, as useful repairs are often very much external source/domain dependent. For these reasons, it has been decided to include the repair model generator for DL-programs as a part of the dlliteplugin which is in contrast the standard plugin structure.

The architectural overview of the dlliteplugin is given in Figure 7.1, where arcs model both control and data flow of the system. The repair answer set computation proceeds as follows:

- The user provides an input DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ as a pair of files: `program.hex` $\mathcal{P}$ and `ontology.owl` $\mathcal{O}$, storing the rule part and the ontology part of the DL-program Π respectively. The rule part $\mathcal{P}$ of Π is passed to the solver in (1), where the rules are analyzed; the DL-atoms are identified, and the dlliteplugin for treating them is initialized. Then in (2) the replacement program $\hat{\Pi}$ is constructed from $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$.

[5] `http://owl-cpp.sourceforge.net`
[6] `https://code.google.com/p/factplusplus`
[7] `http://www.cs.ox.ac.uk/isg/tools/Requiem`

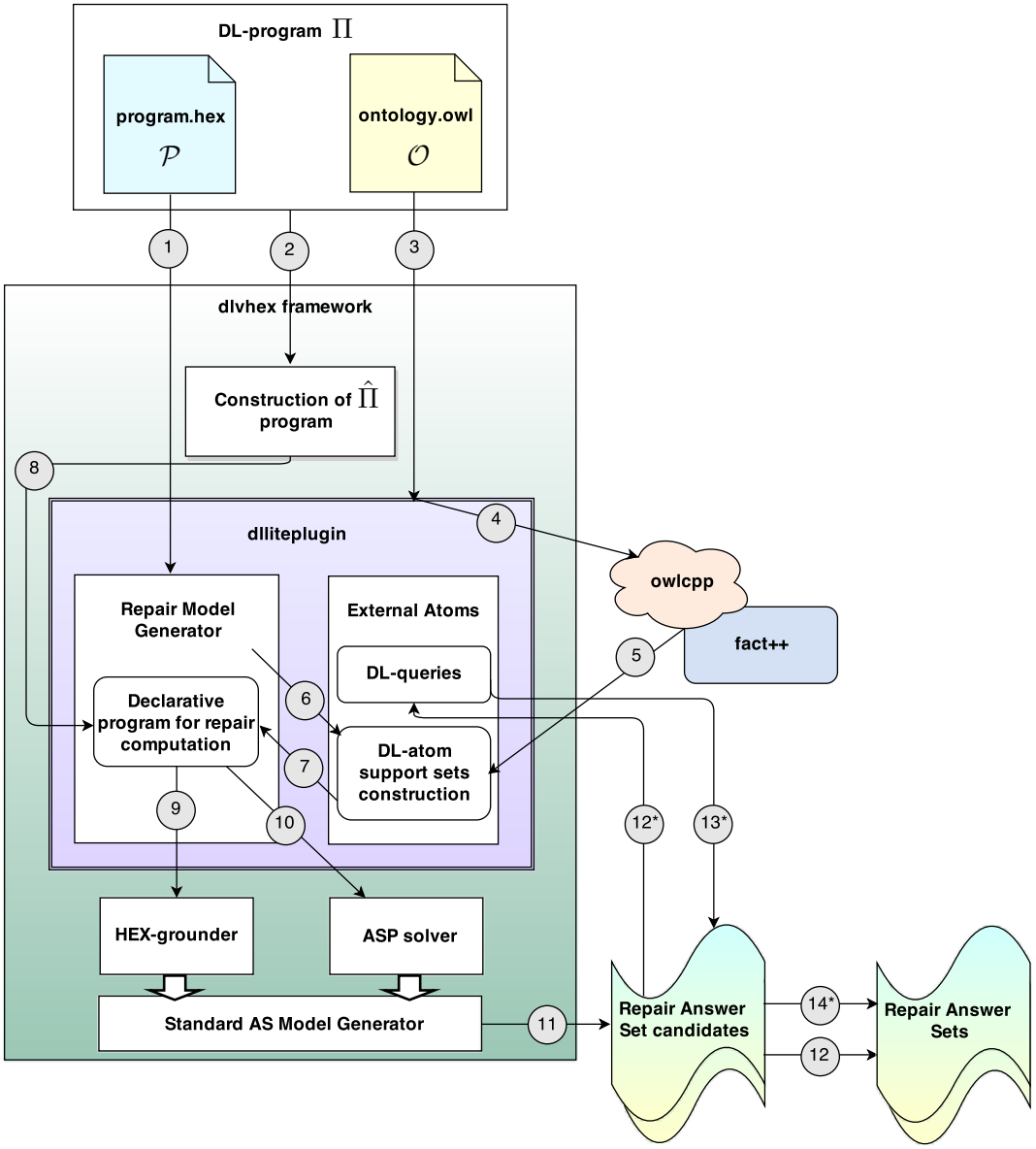


Figure 7.1: System architecture of the dlliteplugin for Repair Answer Set Computation

- In (3) and (4), the dlliteplugin parses the `ontology.owl` file using the owlcpp library, which is an open-source `C++` library for parsing, querying, and reasoning with OWL-2 ontologies; owlcpp provides a convenient interface for communicating with the fact++ system, where the actual ontology reasoning tasks can be performed. In our plugin fact++ is used for the ontology consistency check, which is a necessary condition to ensure correctness of our algorithms. We did not use the widely known OWL API[8], since the latter is written in Java and for purely technical reasons could not effectively serve our needs. Then the repair model generator is instantiated. In the standard answer set computation approaches, the program evaluation can be optimized by using a certain heuristics, which appropriately splits the program into components and evaluates each component separately. Our repair answer set computation algorithms do not support this program split and extensions are nontrivial. Therefore, we assume that the program is evaluated as a whole, which is ensured by the monolithic heuristics used in our repair setting by default.

- In (5) and (6), the ontology and all DL-atoms of the input program are passed to the External Atoms block, where the support set computation is performed. The latter is DL dependent, and consequently it differs for the *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ ontologies. We discuss the details in the next subsection.

- Once all support sets are obtained, they are represented in (7) using declarative means. When extended with a proper representation of the ontology ABox and $\hat{\Pi}$ in (8), the declarative program is obtained, whose models encode the repair answer sets and repairs of the DL-program Π. For evaluating this declarative program, in (8) and (9) the back-end grounder and the solver of the dlvhex system are invoked.

- Finally, in (11) the repair answer set candidates of Π are extracted. If the ontology is in *DL-Lite*$_\mathcal{A}$, and candidate model is minimal with respect to the FLP reduct, which is checked in (12), then it is output to the user. If the ontology is in $\mathcal{EL}$, and the support set information based on which the repair answer set candidates are obtained is incomplete, then some DL-atoms might need an evaluation postcheck, which is performed in (12*), (13*). In case the check succeeds, and, moreover, the candidate is a minimal model of the *flp*-reduct (check in (14*)), then it is returned in the output.

The dlliteplugin can also be used for standard answer set computation; in this case a standard model generator is used instead of the repair model generator (see [Red14], [Sch06] for overviews of available model generators).

[8] `http://owlapi.sourceforge.net`

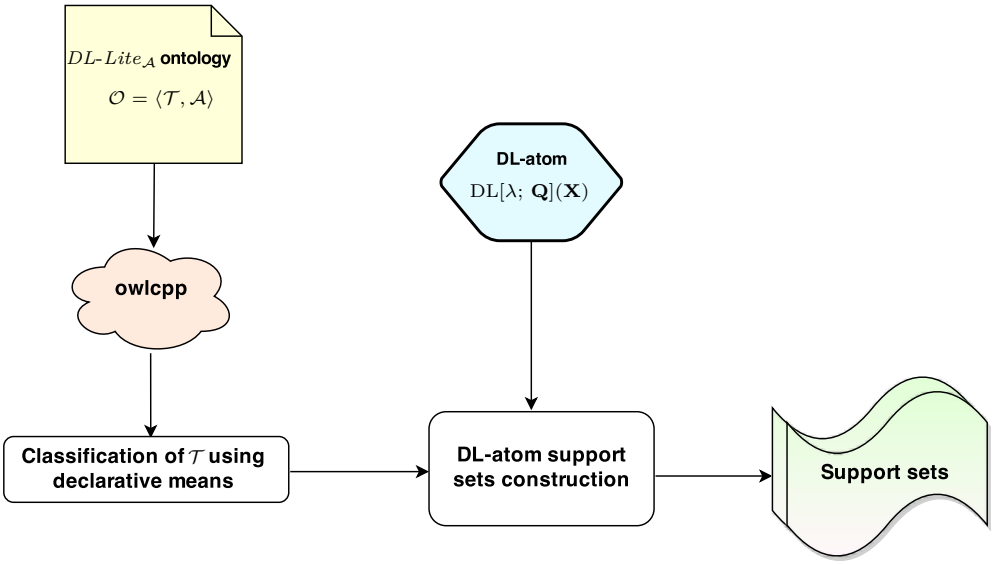


Figure 7.2: Support set construction for DL-atoms over $DL\text{-}Lite_{\mathcal{A}}$ ontologies

7.1.2 Implementation Details

In order to profit from existing dlvhex data structures (e.g. for parsing) and optimization methods (such as nogood learning, etc.), we pursued a declarative ASP approach to realize the algorithms for repair answer set computation over ontologies in $DL\text{-}Lite_{\mathcal{A}}$ and $\mathcal{EL}$, which we have seen in Chapter 5. We now have a closer look at the exact implementation details of the support set generation and the repair answer set search.

DL-programs over ontologies in $DL\text{-}Lite_{\mathcal{A}}$. Our declarative implementation of the algorithms for DL-programs over $DL\text{-}Lite_{\mathcal{A}}$ ontologies comprises both computing complete nonground support families and searching candidate deletion repair answer sets and deletion repairs.

First we describe our approach to computing the support families, which is depicted in Figure 7.2. The routine for computing support families gets a $DL\text{-}Lite_{\mathcal{A}}$ ontology and a nonground DL-atom as input. After parsing the ontology $\mathcal{O}$ using the owlcpp library, we compute its TBox classification. The latter is done in a declarative manner using the program $Prog_{\mathcal{T}_{class}}$, shown in Figure 7.3.

The program $Prog_{\mathcal{T}_{class}}$ reifies concepts (roles, existential restrictions on roles), as well as positive replacements of their negations. Facts express subsumptions in $Pos(\mathcal{T})$ using *sub* predicate, role inverses using *inv*, role functionalities with $funct$, and the duality of concepts (roles, etc.) and their opposites with *op*. The rule (1) of $Prog_{\mathcal{T}_{class}}$ transitively closes the subsumption relation, while (2) expresses contraposition for subsumption. The rules (3)-(5) mimic the construction of binary and unary conflict sets that are then stored in the predicates $conf$ and $confref$ respectively. These are based

$$Prog_{\mathcal{T}_{class}} = \left\{ \begin{array}{l} (1)\ sub(X,\, Y) \leftarrow sub(X,\, Y), sub(\, Y,\, Z); \\ (2)\ sub(\, Y',\, X') \leftarrow sub(X,\, Y), op(X,\, X'), op(\, Y,\, Y'); \\ (3)\ conf(X,\, Y') \leftarrow sub(X,\, Y), op(\, Y,\, Y'); \\ (4)\ inv(X',\, X) \leftarrow inv(X,\, X'); \\ (5)\ op(X,\, Y) \leftarrow op(\, Y,\, X); \\ (6)\ confref(X) \leftarrow conf(X,\, Y), op(Y, Z), inv(X, Z); \end{array} \right\}$$

Figure 7.3: Program $Prog_{\mathcal{T}_{class}}$ for computing classification of $\mathcal{T}$

on the theoretical results from [RRGM12]. Since the program $Prog_{\mathcal{T}_{class}}$ is positive, it has a single answer set $M_{\mathcal{T}_{class}}$, from which the support family $\mathcal{S}$ for the DL-atom $d = \mathrm{DL}[\lambda;\, Q](X)$ is conveniently extracted in the following way:

- for every $sub(P, Q) \in M_{\mathcal{T}_{class}}$, where P is a positive ontology predicate, we add $S = \{P(\vec{X})\}$ to $\mathcal{S}$;
- for every $sub(P, Q) \in M_{\mathcal{T}_{class}}$, where P is a replacement for an existential restriction $\exists R$, we add $S = \{R(X, Y)\}$ to $\mathcal{S}$;
- for every $conf(P, P') \in M_{\mathcal{T}_{class}}$, we add $\{P_p(\vec{Y}), P'(\vec{Y})\}$ to $\mathcal{S}$, if $P_p(\vec{c}) \in \mathcal{A}_d$ for some $c \in \mathcal{C}$ and there is no $S' \subset S$, such that $S' \in \mathcal{S}$;
- for every $conf(P, P') \in M_{\mathcal{T}_{class}}$, we add $\{P_p(\vec{Y}), P'_p(\vec{Y})\}$ to $\mathcal{S}$, if $P_p(\vec{c}), P'_p(\vec{d}) \in \mathcal{A}_d$ for some $\vec{c}, \vec{d} \in \mathcal{C}$ and there is no $S' \subset S$, such that $S' \in \mathcal{S}$;
- for every $confref(P) \in M_{\mathcal{T}_{class}}$, we add $\{P_p(Y, Y)\}$ to $\mathcal{S}$, if $P_p(c, d) \in \mathcal{A}_d$ for some $c, d \in \mathcal{C}$;
- for every $funct(P) \in M_{\mathcal{T}_{class}}$, we add $\{P_p(Y, Z)$,$P_p(Y, Z')\}$ to $\mathcal{S}$, if $P_p(c, d) \in \mathcal{A}_d$ for some $c, d \in \mathcal{C}$, and there is no $S' \subset S$, such that $S' \in \mathcal{S}$.

Now from the definition of support sets, complete support families and the results in [RRGM12], the following proposition is obtained:

Proposition 7.1. *The support family constructed from the model M of $Prog_{\mathcal{T}_{class}}$ is complete.*

Proof. The program $Prog_{\mathcal{T}_{class}}$ intuitively computes all inclusions that follow from the TBox (*sub* predicate) as well as the unary and binary conflict sets. For *DL-Lite*$_{\mathcal{A}}$ ontologies classification can be modeled declaratively as a reachability problem, this is exactly what the rules (1) and (2) represent. The conflict sets in turn are found by means of the rules (3)-(6) and analysis of functional roles. The approach that we used for their computation is proved to be complete in [RRGM12]. Given a DL-atom and all assertions coming from its update, the computed conflict sets and subsumptions, the complete support family is straightforwardly obtained as described above. Formal elaboration of the proof is easily possible, but we omit its details here. □

We now turn to determining the repair answer sets, for which the declarative approach is used as well. More specifically, the non-ground rules (see below) are added to $\hat{\Pi}$ for the purpose of filtering candidate deletion repair answer sets as done by the algorithm *SupRAnsSet* described in Chapter 5. The language of $\hat{\Pi}$ is extended to include support set information. For every nonground DL-atom $a(\vec{X})$ and its nonground support set $S_a(\vec{Y})$ with $\vec{Y} = \vec{X}\vec{X}'$, the following rules are added to the replacement program $\hat{\Pi}$:

$$(r_1)\ \ \bot \leftarrow e_a(\vec{X}), \mathit{not}\ \mathit{Sup}_a(\vec{X}) \qquad (r_3)\ \ \mathit{Sup}_a(\vec{X}) \leftarrow r(S_a(\vec{Y})), \mathit{not}\ \bar{S}_a^{\mathcal{A}}(\vec{Y})$$
$$(r_2)\ \ \bot \leftarrow ne_a(\vec{X}), \mathit{Sup}_a(\vec{X}) \qquad (r_4)\ \ \bar{S}_a^{\mathcal{A}}(\vec{Y}) \leftarrow ne_a(\vec{X}), r(S_a(\vec{Y}))$$

where $r(S)$ is a suitable representation of a support set S using predicates $p(\vec{X})$ for input assertions $P_p(\vec{X})$, resp. $p_P(\vec{X})$ ($np_P(\vec{X})$) for ABox assertions $P(\vec{X})$ ($\neg P(\vec{X})$). $\bar{S}_a^{\mathcal{A}}$ states that the ABox part of S_a is marked for deletion if $S_a \cap \mathcal{A} \neq \emptyset$, otherwise it is void. Furthermore, Sup_a is a fresh predicate not in $\mathcal{P}$, that intuitively says that a has an applicable support set, i.e. its ABox part is either empty or not marked for deletion. The resulting program intuitively prunes candidates $\hat{I}$, resp. encodes deletion repair candidates, according to the *SupRAnsSet* algorithm presented in Chapter 5.

We now formally prove the soundness and correctness of the described declarative implementation.

Proposition 7.2. *Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a ground DL-program, where $\mathcal{O}$ is a DL-Lite$_{\mathcal{A}}$ ontology, and let $a_1, \ldots, a_n$ be DL-atoms of Π. Let, moreover, $\mathcal{S}_1, \ldots, \mathcal{S}_n$ be complete nonground support families for $a_1, \ldots, a_n$ w.r.t. $\mathcal{O}$, and let $\mathcal{R}$ be the set of rules of the forms (r_1)-(r_4) constructed for each support set from $\mathcal{S}_i$ covering a_i, $1 \leq i \leq n$. Then $AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))|_{\Pi} = RAS_{weak}(\Pi)$,[9] where $facts(\mathcal{A}) = \{p_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A}\} \cup \{np_P(\vec{c}) \mid \neg P(\vec{c}) \in \mathcal{A}\}$ is the set of facts corresponding to the assertions from $\mathcal{A}$.*

Proof. We separately prove inclusions $AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))|_{\Pi} \subseteq RAS_{weak}(\Pi)$ and $AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))|_{\Pi} \supseteq RAS_{weak}(\Pi)$, which respectively reflect the correctness and completeness of the provided implementation.

($\subseteq$) Assume towards a contradiction that $AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))|_{\Pi} \not\subseteq RAS_{weak}(\Pi)$. Then there exists an element $I \in AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))$, such that $I|_{\Pi} \notin RAS_{weak}(\Pi)$. This means that for all $\mathcal{A}' \subseteq \mathcal{A}$, it holds that $I|_{\Pi} \notin AS_{weak}(\Pi')$ with $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. Consider the ABox $\mathcal{A}'' = \{P(\vec{c}) \mid p_P(\vec{c}) \in I|_{facts(\mathcal{A})},\ \bar{p}_P(\vec{c}) \notin I\}$[10], which is a particular subset of $\mathcal{A}$. We have that $I|_{\Pi} \notin AS_{weak}(\Pi'')$ with $\Pi'' = \langle \mathcal{T}, \mathcal{A}'', \mathcal{P} \rangle$. Thus one of the following must be true: (i) no extension of $I|_{\Pi}$ with guessed values of replacement atoms is a model of $\hat{\Pi}''$, (ii) no model of $\hat{\Pi}''$ is a compatible set for Π'' or (iii) there exists $I' \subset I|_{\Pi}$, which is a model of $\mathcal{P}_{weak}^{I|_{\Pi}, \mathcal{O}''}$.

The case (i) is irrelevant, as $I|_{\hat{\Pi}}$ satisfies all rules of $\hat{\Pi}$ due to $I \in AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))$ and $\hat{\Pi}'' = \hat{\Pi}$. We next show that (ii) can not hold by deriving a contradiction. Indeed, assume that (ii) holds, then as $I|_{\hat{\Pi}}$ is a model of $\hat{\Pi}$, it is not a compatible set for Π.

[9] $AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))|_{\Pi} = \{I|_{\Pi} \mid I \in AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))\}$.

[10] $\bar{p}_P$ correspond to the respective $\bar{S}_a^{\mathcal{A}}$

Therefore there exists a DL-atom a_i in Π'', such that its real value is different from the guessed value in $I|_{\hat{\Pi}}$. Suppose first that $I|_\Pi \models a_i$, but $ne_{a_i} \in I|_{\hat{\Pi}}$. By Proposition 5.28 there must exist a support set $S \in \mathcal{S}_i$, such that S is coherent with $I|_\Pi$ and its ABox part $S^\mathcal{A}$ is in $\mathcal{A}''$. If $S^\mathcal{A}$ is nonempty, then due to the rule of the form (r_4) of $\mathcal{R}$ we get that $\bar{S}^\mathcal{A}$ must be in I, but then $S_\mathcal{A}$ is not present in $\mathcal{A}''$. Therefore, $S^\mathcal{A}$ must be empty, i.e. S must contain only input assertions. However, then the body of the constraint (r_2) of $\mathcal{R}$ is satisfied, contradicting $I \in AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))$. In conclusion, this shows that (ii) does not hold, and in particular that $I|_{\hat{\Pi}}$ is a compatible set for Π.

Finally, the last possibility is that (iii) holds, meaning that there is an interpretation $I' \subset I|_\Pi$ which is a model of $\mathcal{P}_{weak}^{I|_\Pi, \mathcal{O}''}$. The interpretations $I|_\Pi$ and I' differ on the set $M = I|_\Pi \backslash I'$, containing only ground atoms from the language of Π. Let us now look at the interpretation $I'' = I \backslash M$. We know that I is an answer set of $\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A})$, i.e. it is a minimal model of $\hat{\Pi}_{gl}^I \cup \mathcal{R}_{gl}^I \cup facts(\mathcal{A})$. Therefore, there must exist some rule r_{gl}^I either in (1) $\hat{\Pi}_{gl}^I$ or in (2) $\mathcal{R}_{gl}^I$, which I'' does not satisfy, i.e. $I'' \models B(r_{gl}^I)$ and $I'' \not\models H(r_{gl}^I)$.

Assume that (1) holds. Then the rule r_{gl}^I must involve some replacement atoms e_a occurring positively. Otherwise $I' \not\models r_{gl}^I$, and since this rule is also in $\mathcal{P}_{weak}^{I|_\Pi, \mathcal{O}''}$, we have that I' is not a model of $\mathcal{P}_{weak}^{I|_\Pi, \mathcal{O}''}$, leading to a contradiction. Furthermore, we know that $I|_{\hat{\Pi}}$ is a compatible set. Therefore, $r_{weak}^{I, \mathcal{O}''}$ is the rule r_{gl}^I without replacement atoms in its body; but then $I' \not\models r_{weak}^{I, \mathcal{O}''}$, and hence I' is not a model of $\mathcal{P}_{weak}^{I|_\Pi, \mathcal{O}''}$.

Now assume that (2) holds, i.e. there is a rule $r_{gl}^I \in \mathcal{R}_{gl}^I$, such that $I'' \models B(r_{gl}^I)$, but $I'' \not\models H(r_{gl}^I)$. The rule r_{gl}^I can not be a constraint of the forms r_1, r_2, since then $I \supset I''$ is not an answer set of $\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A})$, leading to a contradiction. Therefore, r must be of the form r_3 or r_4. However, the latter is not possible either, since the set of atoms M on which I and I'' differ contains only atoms from the signature of Π, and $H(r_{gl}^I)$ does not fall into this set, meaning that $I \not\models r_{gl}^I$, which contradicts to $I \in AS(\hat{\Pi} \cup facts(\mathcal{A}) \cup \mathcal{R})$.

($\supseteq$) Suppose that $I \in RAS_{weak}(\Pi)$, but there is no $I' \supseteq I$, such that $I' \in AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))$. By definition of repair answer sets, some $\mathcal{A}' \subset \mathcal{A}$ exists, such that $I \in AS_{weak}(\Pi')$, where $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. We construct the interpretation I' by extending I with

- $\{e_a \mid I \models^{\mathcal{O}'} a\} \cup \{ne_a \mid I \not\models^{\mathcal{O}'} a\}$, i.e. facts reflecting the values of the replacement atoms under I and $\mathcal{A}'$;
- $facts(\mathcal{A})$;
- $\{\bar{p}_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A} \backslash \mathcal{A}'\}$;
- $Sup_{a_i}(\vec{c})$ encoding information about support sets of $a_i(\vec{c})$ coherent with I.

We now show that the constructed interpretation I' is an answer set of $\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A})$, i.e. it is a minimal model of the GL-reduct $(\hat{\Pi} \cup facts(\mathcal{A}') \cup \mathcal{R})_{gl}^{I'}$. Assume towards a contradiction that this is not the case. There are two possibilities: (i) either I' does not satisfy some rules of the reduct, or (ii) some smaller model of the reduct exist.

First consider (i). I' immediately satisfies all facts as well as all rules in $\hat{\Pi}^{I'}_{gl}$. This means that there must be some rule $r^{I'}_{gl}$ in $\mathcal{R}^{I'}_{gl}$ that is not satisfied, i.e. $I' \models B(r^{I'}_{gl})$, but $I' \not\models H(r^{I'}_{gl})$. By construction of I' and Proposition 5.28, if $e_a \in I'$ (resp. $ne_a \in I'$) then $Sup_a \in I'$ (resp. $Sup_a \notin I'$), therefore r can not be of the form (r_1) or (r_2). Suppose that r is of the form (r_3). We have that some DL-atom a has a support set whose ABox part is in $\mathcal{A}'$ or empty. Then by construction of I' the head of the rule $r^{I'}_{gl}$ has to be satisfied. Therefore, the rule r must be of the form (r_4). Then $I \not\models^{\mathcal{O}'} a$ for some DL-atom a, such that there is a support set for a which is coherent with I and its ABox part is either empty or present in $\mathcal{A}$. In both cases by Proposition 5.28 we get that $I \models^{\mathcal{O}'} a$, which leads to a contradiction.

Let us now look at (ii), i.e. some interpretation $I'' \subset I'$ exists, such that I'' is a model of $(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))^{I'}_{gl}$. Note that I'' and I' can not differ only on replacement atoms, since for each DL-atom a, either e_a or ne_a must be in I''. As I' already contains the corresponding replacement atoms, removal of any such atom will violate the satisfaction of some guessing rule $e_a \vee ne_a$ in $\hat{\Pi}^{I'}_{gl}$. Suppose that $I''\backslash I'$ contains some atoms from Π. Consider $I''|_\Pi$, which is a subset of I. Observe that $I''|_\Pi$ can not be a model of $\mathcal{P}^{I,\mathcal{O}'}_{weak}$, because $I \supset I''|_\Pi$ is its minimal model. Therefore, some rule $r^{I,\mathcal{O}'}_{weak}$ must exist in the reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$ which is not satisfied by $I''|_\Pi$, i.e. $I''|_\Pi \models B(r^{I,\mathcal{O}'}_{weak})$ but $I''|_\Pi \not\models H(r^{I,\mathcal{O}'}_{weak})$. By construction of the weak reduct this rule does not contain any DL-atoms. Let us look at the corresponding rule in the reduct $\hat{\Pi}^{I''}_{gl}$. The rule $r^{I''}_{gl}$ either does not contain any replacement atoms or contains only positive atoms e_a such that $e_a \in I''$ (by construction og the GL-reduct). Therefore $I'' \models B(r^{I'}_{gl})$, but $I'' \not\models H(r^{I'}_{gl})$, contradicting $I'' \models \hat{\Pi}^{I'}_{gl}$.

Suppose that the interpretations I' and I'' differ only on the facts over predicates in $\mathcal{R}$. We know that the rule $r^{I'}_{gl}$, where r is of the form (r_1) is not present in $\mathcal{R}^{I'}_{gl}$, moreover, $I'' \not\models r'^{I'}_{gl}$ for r' of the form (r_2). If the difference $I'\backslash I''$ contains Sup_a, then it must contain some atoms from $r(S_a)$ too. Moreover, these atoms must be related to the ABox facts, which are present in I'. This, however, means that some fact in $facts(\mathcal{A})$ is not satisfied, contradicting $I'' \models (\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}))^{I'}_{gl}$. Finally, $I'' \setminus I'$ can not contain elements $\bar{S}^{\mathcal{A}}_a$, as then the rule $r^{I'}_{gl}$ for r of the form (r_4) is not satisfied by I''.

□

Observe, that our declarative implementation computes exactly the weak repair answer sets. Thus, in some cases rarely met in practice [EFK+12] an additional minimality check is needed to ensure that the weak repair answer set identified is also an *flp*-repair answer set. It may happen in case of cyclic support, i.e. recursion through a DL-atom that makes an atom true [EIL+08]. We ilustrate this by the following example:

Example 7.3. Consider a variation of Example 2.44: $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where

$$\mathcal{O} = \{Student(pat)\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ man(pat) \leftarrow \mathrm{DL}[Male \uplus man;\ Male](pat); \\ (2)\ \bot \leftarrow \mathrm{DL}[;\ Student](pat) \end{array} \right\}.$$

$$\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) = \left\{ \begin{array}{l} (1)\ man(pat) \leftarrow e_{a_1}; \\ (2)\ \bot \leftarrow e_{a_2}; \\ (3)\ e_{a_1} \vee ne_{a_1}; \\ (4)\ e_{a_2} \vee ne_{a_2}; \\ (5)\ Sup_{a_1} \leftarrow man(pat); \\ (6)\ \bot \leftarrow e_{a_1}, not\ \ Sup_{a_1}; \\ (7)\ \bot \leftarrow ne_{a_1}, Sup_{a_1}; \\ (8)\ Sup_{a_2} \leftarrow p_{Student}(pat), not\ \ \bar{p}_{Student}(pat); \\ (9)\ \bar{p}_{Student}(pat) \leftarrow ne_{a_2}, p_{Student}(pat); \\ (10)\ \bot \leftarrow e_{a_2}, not\ \ Sup_{a_2}; \\ (11)\ \bot \leftarrow ne_{a_2}, Sup_{a_2}; \\ (12)\ p_{Student}(pat). \end{array} \right\}$$

Figure 7.4: Program $\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A})$ from Example 7.3

The respective logic program $\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A})$ constructed for Π is shown on Figure 7.4, where $a_1 = \text{DL}[Male \uplus man; Male](pat)$, and $a_2 = \text{DL}[; Student](pat)$.

For the interpretation $I = \{man(pat), e_{a_1}, ne_{a_2}, p_{Student}(pat), \bar{p}_{Student}(pat), Sup_{a_1}\}$ we have that $\mathcal{P}^I_{gl}$ contains the rules (1)-(5), (7), (9), (10'), (11) and (12), where (10') is the rule $\bot \leftarrow e_{a_2}$. It holds that I is a minimal model of this reduct. We extract a repair $\mathcal{A}'$ from I. Since $\bar{p}_{Student}(pat) \in I$, we set $\mathcal{A}' = \mathcal{A} \setminus \{Student(pat)\}$. Let us now look at $I|_\Pi = \{man(pat)\}$. We check whether $I|_\Pi$ is a minimal model of

$$\mathcal{P}^{I|_\Pi, \mathcal{O}'}_{\mathit{flp}} = \{man(pat) \leftarrow \text{DL}[Male \uplus man; Male](pat)\},$$

where $\mathcal{O}' = \emptyset$. Clearly, $I|_\Pi$ is a model of the reduct, but its smaller model exists, namely $I' = \emptyset$. Thus $I|_\Pi$ is not an *flp*-repair answer set of Π. Observe, however, that $I|_\Pi$ is a minimal model of the weak reduct $\mathcal{P}^{I|_\Pi, \mathcal{O}'}_{weak} = \{man\}$, and hence $I|_\Pi$ is a *weak*-repair answer set of Π. □

DL-programs over ontologies in $\mathcal{EL}$ DL.

Unlike in the *DL-Lite*$_\mathcal{A}$ DL, the support sets for the $\mathcal{EL}$ ontologies are of a rich structure, and thus for their computation classification of $\mathcal{T}$ is insufficient. Apart from inclusions $A \sqsubseteq B$, where A and B are atomic, we need to identify also all inclusions of the form $C \sqsubseteq B$, where C is an arbitrarily complex concept. Thus we exploit the REQUIEM tool [PUMH10], which produces the rewritings of the target query over the given TBox using datalog rewriting techniques. The general workflow of the support set computation in $\mathcal{EL}$ is given in Figure 7.5. As discussed in Chapter 5, there might be exponentially many such rewritings, and each of them might be of exponential size. In the dlliteplugin there is a possibility to limit the number and size of the rewritings that are to be computed by REQUIEM.

After the support sets are computed we then use a declarative approach for determining repair answer sets, in which the minimal hitting set computation is accomplished by rules. To this end, for each DL-atom $a(\vec{X})$ fresh predicates $Sup_a(\vec{X})$, $S_a^{\mathcal{P}}(\vec{Y})$ and $S_a^{\mathcal{A},\mathcal{P}}(\vec{Y})$ are introduced, where $\vec{Y} = X\vec{X}'$, which intuitively say that $a(\vec{X})$ has some support set, some support set with only logic program predicates, and some mixed support set, respectively (for simplicity we superficially use uniform variables). Furthermore, rules of the following form are added to the replacement program $\hat{\Pi}$:

$$(r_1^*)\quad Sup_a(\vec{X}) \leftarrow S_a^{\mathcal{P}}(\vec{Y}) \qquad (r_5^*)\quad \bot \leftarrow ne_a(\vec{X}), S_a^{\mathcal{P}}(\vec{Y})$$
$$(r_2^*)\quad Sup_a(\vec{X}) \leftarrow S_a^{\mathcal{A},\mathcal{P}}(\vec{Y}) \qquad (r_6^*)\quad \bar{P}_{1a}(\vec{Y}) \vee \ldots \vee \bar{P}_{na}(\vec{Y}) \leftarrow ne_a(\vec{X}), S_a^{\mathcal{A},\mathcal{P}}(\vec{Y})$$

$$(r_3^*)\quad S_a^{\mathcal{P}}(\vec{Y}) \leftarrow rb(S_a^p(\vec{Y})) \qquad (r_7^*)\quad eval_a(\vec{X}) \leftarrow e_a(\vec{X}), not\ C_a, not\ Sup_a(\vec{X})$$
$$(r_4^*)\quad S_a^{\mathcal{A},\mathcal{P}}(\vec{Y}) \leftarrow rb(S_a^{\mathcal{A},\mathcal{P}}(\vec{Y})), nd(S^{\mathcal{A},\mathcal{P}}(\vec{Y})) \qquad (r_8^*)\quad eval_a(\vec{X}) \leftarrow ne_a(\vec{X}), not\ C_a$$

$$(r_9^*)\ \bot \leftarrow e_a(\vec{X}), C_a, not\ Sup_a(\vec{X})$$

Here the fact C_a says that the support family for $a(\vec{X})$ is known to be complete; such information can be added by facts; we denote by $\mathcal{COMP}$ a set of such facts. The rules (r_1^*)-(r_4^*) derive information about support sets of $a(\vec{X})$ under a potential repair; $rb(S)$ stands for a rule body rendering of a support set S; $nd(S) = not\ \bar{p}_{P_{1a}}(\vec{Y}), \ldots, not\ \bar{p}_{P_{na}}(\vec{Y})$, where $\{p_{P_{1a}}(\vec{Y}), \ldots, p_{P_{na}}(\vec{Y})\}$ encodes the ontology part of S and $\bar{p}_{P_{ia}}(\vec{Y})$ states that the assertion $P_{ia}(\vec{Y})$ is marked for deletion. The constraint (r_5^*) forbids $a(\vec{X})$, if guessed false, to have a matching support set with only input assertions; (r_6^*) means that if $a(\vec{X})$ has instead a matching mixed support set, then some assertion from its ontology part must be eliminated. The rule (r_7^*) says that if $a(\vec{X})$ is guessed true and completeness of its support family is not known, then an evaluation postcheck must be performed ($eval_a(\vec{X})$) if no matching support set is available; rule (r_8^*) is similar for $a(\vec{X})$ guessed false. The rule (r_9^*) states a DL-atom guessed true must have some support set, if its support family is known to be complete.

The set of facts $facts(\mathcal{A}) = \{p_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A}\} \cup \{np_P(\vec{c}) \mid \neg P(\vec{c}) \in \mathcal{A}\}$ encoding the ABox assertions and $\mathcal{COMP} = \{C_a \mid \mathcal{S}_a \text{ is a complete support family for } a\}$ are added to the program, and then its answer sets are computed. For each such answer set I, we proceed with an evaluation postcheck for all atoms $a(\vec{c})$ for which the fact $eval_a(\vec{c})$ is in the answer set. If the evaluation postcheck succeeds for all external atoms, then we extract the repair answer sets of the original program from I.

We now formally show that the described approach indeed correctly computes the repair answer sets.

Proposition 7.4. *Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a ground DL-program, where $\mathcal{O}$ is an $\mathcal{EL}$ ontology. Let, moreover, $a_1, \ldots, a_n$ be DL-atoms of Π with nonground support families $\mathcal{S}_1, \ldots, \mathcal{S}_n$, and let $\mathcal{R}$ be the set of rules of the form (r_1^*)-(r_9^*) constructed for each DL-atom. Consider the set $\mathcal{M} \subseteq AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})$, where $facts(\mathcal{A}) = \{p_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A}\} \cup \{np_P(\vec{c}) \mid \neg P(\vec{c}) \in \mathcal{A}\}$ is the set of facts corresponding to the assertions from $\mathcal{A}$, and $\mathcal{COMP} = \{C_{a_i} \mid \mathcal{S}_i \text{ is a complete support family for } a_i\}$. Suppose*

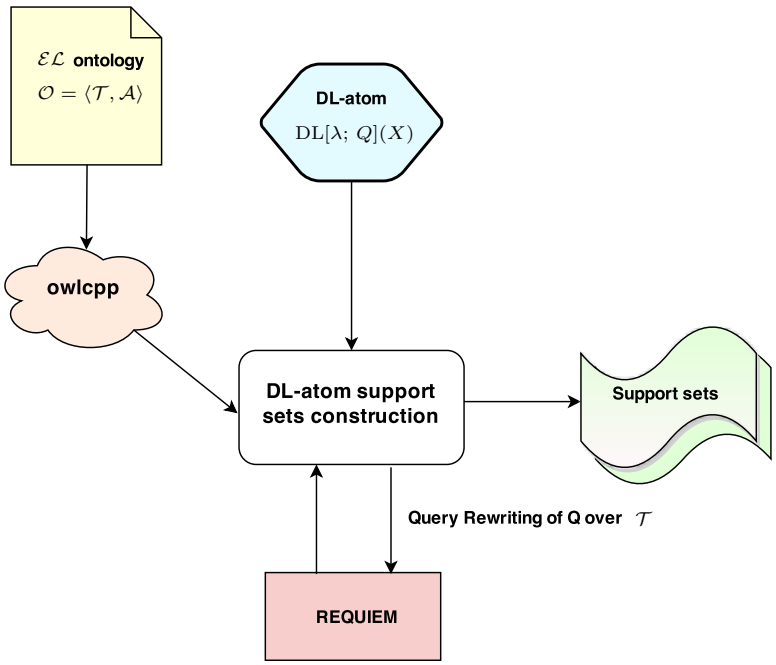


Figure 7.5: Support set construction for DL-atoms over $\mathcal{EL}$ ontologies

that all evaluation postchecks for every $I \in \mathcal{M}$ succeeded. Then $\mathcal{M}|_\Pi \subseteq RAS_{weak}(\Pi)$, and $\mathcal{M}|_\Pi = RAS_{weak}(\Pi)$, if the support families $\mathcal{S}_1, \ldots, \mathcal{S}_n$ for all DL-atoms in Π are known to be complete.

Proof. We first show that $\mathcal{M}|_\Pi \subseteq RAS_{weak}(\Pi)$. Suppose towards a contradiction that there is some $I \in \mathcal{M}$, such that $I|_\Pi \notin RAS_{weak}(\Pi)$. Then for all ABoxes $\mathcal{A}'$, such that $\mathcal{A}' \subseteq \mathcal{A}$, we have that $I|_\Pi \notin AS_{weak}(\Pi')$ with $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. Consider an ABox $\mathcal{A}'' = \mathcal{A} \backslash \{P(\vec{c}) \mid \bar{p}_P(\vec{c}) \in I\}$. By our assumption $I|_\Pi \notin AS(\Pi'')$ with $\Pi'' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. There are several possibilities: (i) no extension of $I|_\Pi$ with guessed values of replacement atoms is a model of $\hat{\Pi}''$; (ii) no extension of $I|_\Pi$ with guessed values of replacement atoms is a compatible set; (iii) there is an interpretation $I' \subset I|_\Pi$ which is a model of $\mathcal{P}_{weak}^{I|_\Pi, \mathcal{O}''}$. The case (i) can not hold, as $\hat{\Pi} = \hat{\Pi}''$ and $I|_{\hat{\Pi}} \models \hat{\Pi}$.

Assume that (ii) is true. Consider the interpretation $I|_{\hat{\Pi}}$. Towards a contradiction, assume that it is not compatible for Π''. Then either (1) $I|_\Pi \models^{\mathcal{O}''} a$, but $ne_a \in I|_{\hat{\Pi}}$, or (2) $I \not\models^{\mathcal{O}''} a$, but $e_a \in I|_{\hat{\Pi}}$. Let us first look at (1). As $I|_\Pi \models^{\mathcal{O}''} a$, there must exist a support set S for it, which is coherent with $I|_{\hat{\Pi}}$. There are two possibilities: either $S \in \mathcal{S}_a$ or $S \notin \mathcal{S}_a$. In the former case S must contain ABox assertions $S_a^{\mathcal{A}}$, as otherwise some constraint of the form (r_5^*) is violated. Due to the rule (r_6^*) at least one assertion P_{ia} in $S_a^{\mathcal{A}}$ must be marked for deletion. Note that then P_{ia} is not present in $\mathcal{A}'$, and S is not

a relevant support set for a. If $\mathcal{S}_a$ is known to be complete, then we immediately arrive at a contradiction. Otherwise, the rule of the form (r_8^*) is applied, and as the evaluation postcheck for a succeeded by our assumption, we get a contradiction. If $S \notin \mathcal{S}_a$ then the support family $\mathcal{S}_a$ is not known to be complete, and again the rule of the form r_8^* is satisfied, and due to the successful evaluation postcheck a contradiction is obtained. Now suppose that (2) is true. As $I|_\Pi \not\models^{\mathcal{O}'} a$, we know that there are no support sets for a coherent with $I|_\Pi$. If $\mathcal{S}_a$ is known to be complete, then the constraint r_9^* is violated, which can not happen as $I \models \hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP}$. Otherwise, the body of the rule (r_7^*) must be satisfied, and as the evaluation postcheck succeeded for the atom a, we get a contradiction.

Finally, assume that (iii) holds, i.e. there is an interpretation $I' \subset I|_\Pi$, which is a model of $\mathcal{P}_{weak}^{I|_\Pi,\mathcal{O}'}$. $M = I|_\Pi \backslash I'$ contains only atoms over the signature of Π. Let us look at $I'' = I \backslash M$. We know that $I \in AS(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})$. Therefore, there must exist some rule r_{gl}^I in $(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})_{gl}^I$, such that $I'' \models B(r_{gl}^I)$, but $I'' \not\models H(r_{gl}^I)$. Observe that r_{gl}^I can not be from $\hat{\Pi}$ or $facts(\mathcal{A})$ or $\mathcal{COMP}$, as then $I' \not\models \mathcal{P}_{weak}^{I|_\Pi,\mathcal{O}'}$ by construction of the GL and weak reducts. Therefore, r_{gl}^I must be in $\mathcal{R}_{gl}^I$. However, the latter can not happen either, as there are no rules in $\mathcal{R}_{gl}^I$ which contain atoms over the signature of Π in their heads. Therefore, $I|_\Pi \in AS(\Pi'')$ holds, and we have a global contradiction, i.e. $I|_\Pi \in RAS_{weak}(\Pi)$ follows.

We now consider the case when all support families are known to be comlete, and prove that given this knowledge it holds that $\mathcal{M}|_\Pi = RAS_{weak}(\Pi)$. The inclusion $\mathcal{M}|_\Pi \subseteq RAS_{weak}(\Pi)$ has already been shown in a more general case. It is left to check that $\mathcal{M}|_\Pi \supseteq RAS_{weak}(\Pi)$. Towards a contradiction, assume that there is an interpretation I, such that $I \in RAS_{weak}(\Pi)$, but there is no extension I' of I which is in $\mathcal{M}$.

We have that there is an ABox $\mathcal{A}'$, such that $I \in AS(\Pi')$, where $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. Let us construct an extension I' of I as follows:

$$\begin{aligned} I' \;=\; & I \cup \{e_a \mid I \models^{\mathcal{O}'} a\} \cup \{ne_a \mid I \not\models^{\mathcal{O}'} a\} \cup \\ & \{\bar{p}_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A} \backslash \mathcal{A}'\} \cup facts(\mathcal{A}) \cup \mathcal{COMP} \cup \\ & \{Sup_a \mid a \text{ has a support set coherent with } I\}. \end{aligned}$$

Since by our assumption $I' \notin \mathcal{M}$, one of the following must hold:

(i) $I' \not\models (\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})_{gl}^{I'}$ or

(ii) there exists $I'' \subset I'$, such that $I'' \models (\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})_{gl}^{I'}$.

First assume that (i) is true. By construction of I', it satisfies $\hat{\Pi}$ and all rules of the forms (r_1^*)-(r_4^*). Moreover, constraints of the form (r_5^*) can not be violated, as none of DL-atoms a, such that $I \not\models^{\mathcal{O}'} a$ could have a support set consisting only of input assertions. The rule (r_7^*) and (r_8^*) are not present in the reduct, because all support families are complete by our assumption. Thus the rule r such that $I' \not\models r_{gl}^{I'}$ could only be of the forms (r_6^*) or (r_9^*). If the rule r was of the form (r_6^*), then there would be an a such that $I \not\models^{\mathcal{O}'} a$, but there is a support set for a coherent with I and containing an

ABox part that is present in $\mathcal{A}'$. The latter can not happen by Proposition 5.28. Hence, r must be of the form (r_9^*), which can not be true either as the support family $\mathcal{S}_a$ for a, such that $I \models^{\mathcal{O}'} a$ is complete, and it must contain some support set for a in $\mathcal{S}_a$.

Now, let (ii) hold, i.e. some $I'' \subset I'$ exists, s.t. $I'' \models (\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})^{I'}_{gl}$. Consider the set $I' \setminus I''$. Suppose first that $I' \setminus I''$ contains some replacement atoms. Observe that for every DL-atom a the interpretation I' contains exactly one out of e_a and ne_a. Therefore, there must exist some DL-atom a, such that I'' does not contain either e_a or ne_a. However, then the guessing rule $e_a \vee ne_a$ in $\hat{\Pi}^{I'}_{gl}$ is not satisfied by I'', leading to a contradiction.

Suppose that $I' \setminus I''$ contains some atoms from the language of Π. Then it holds that $I''|_\Pi$ is not a model of $\mathcal{P}^{I,\mathcal{O}'}_{weak}$, that is there is a rule $r^{I,\mathcal{O}'}_{weak}$ in $\mathcal{P}^{I,\mathcal{O}'}_{weak}$, such that $I''|_\Pi \models B(r^{I,\mathcal{O}'}_{weak})$, but $I''|_\Pi \not\models H(r^{I,\mathcal{O}'}_{weak})$. Consider the respective rule $r^{I'}_{gl}$ in $\hat{\Pi}^{I'}_{gl}$. We have that $I'' \not\models H(r^{I'}_{gl})$, therefore it must hold that $I'' \not\models B(r^{I'}_{gl})$. By construction of weak and GL reducts the sets of positive normal atoms in $B(r^{I'}_{gl})$ and $B(r^{I',\mathcal{O}'}_{weak})$ coincide. Hence, replacement atom e_a must occur positively in $B(r^{I'}_{gl})$, such that $e_a \in I' \setminus I''$. As we have already argued, the latter can not happen, leading to a contradiction.

Therefore, $I' \setminus I''$ must contain only atoms from the language of $\mathcal{R}$. If $I' \setminus I''$ contains some Sup_a, then there is a rule of the form (r_1^*) or (r_2^*) that is not satisfied. If $I \setminus I''$ contains some atom of the form $S^{\mathcal{P}}_a$ then the rule (r_3^*) is not satisfied. Finally, if there is an element of the form $S^{\mathcal{A},\mathcal{P}}_a$ in $I' \setminus I''$ then the rule (r_4^*) is not satisfied. Hence I' must be a minimal model of $(\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})^{I'}_{gl}$, i.e. $I' \in \mathcal{M}$. Therefore, $\mathcal{M}|_\Pi \supseteq RAS_{weak}(\Pi)$, which proves the statement.

□

7.1.3 System Installation and Usage

The dlliteplugin was tested on the Linux system Ubuntu 12.04 and 14.04, and in principle it should also work on OSx as the latter is Linux based. The plugin provides two external atoms for querying ontology concepts and roles respectively.

As a prerequisite for dlliteplugin installation one needs to have the dlvhex system correctly installed and configured on the machine. For installing the plugin one needs to run the following commands from the directory of the plugin in the specified order:

- `$./bootstrap.sh`

  This command will prepare the configure file and download all necessary libraries for owlcpp installation (libxml, fact++ and raptor);

- `$ ./configure -with-boost=[/path/to/boost-prefix] -with-owlcpp=[auto or path/to/owlcpp/installation]`

  If one has already a version of boost library installed on the machine, then the full path to boost needs to be specified. For automatic owlcpp installation and configuration (recommended) one needs to use the option `-with-owlcpp=auto`, but it is

also possible to build owlcpp manually and run `./configure` command with the option `-with-owlcpp=/path/to/owlcpp`, for that the Internet connection is required. Here `path/to/owlcpp` points to the main directory of an owlcpp build. In this case, owlcpp and the dlvhex must be built using the same version of boost. Moreover, make sure that owlcpp exports all symbols such that they can be used from shared libraries; this can be achieved by replacing the the line `fvisibility=hidden` of `jamroot.jam` file in the owlcpp directory with `fvisibility=default`, and removal of the line `fvisibility-inlines-hidden`.

- `$ make`

After following the described steps, the installation of the dlliteplugin should be successfully completed.

7.1.4 Command-line Options

There are several command line options available which allow the user to choose an appropriate mode of reasoning. Apart from the command line options related to the core of the system, the dlliteplugin provides the following additional options:

- `-repair=[ontology relative or full path]` ensures that the repair of the DL-program is computed with respect to the ontology specified;
- `-el` states that ontology is in $\mathcal{EL}$ and makes sure that the appropriate algorithm based on incomplete support families is used for repair computation;
- `-supnum=[int]` specifies the maximal number of rewritings to be computed, from which the support sets are then extracted and analyzed (available only with the enabled `-el` option)[11];
- `-supsize=[int]` specifies the maximal size of support sets that are constructed and exploited for the repair computation (available only with the enabled `-el` option);
- `-replimfact=[int]` specifies the maximal number of assertions allowed for deletion;
- `-repdelpred=[comma-separated set of predicates over` $\Sigma(\mathcal{O})$`]` specifies the concrete set of ontology predicates allowed for deletion;
- `-repleavepred=[comma-separated set of predicates over` $\Sigma(\mathcal{O})$`]` specifies the set of ontology predicates that are protected from deletion (must not be removed);

[11] Multiple support sets obtained from the same rewriting are counted as a single support set when it comes to the specified restriction. The name of the option is a bit misleading in that sense, but it is introduced for user convenience.

- `-replimpred=[int]` specifies the maximal number of predicates that can participate in the facts allowed for removal;
- `-repdelconst=[comma-separated set of constants over` $\Sigma(\mathcal{O})$`]` specifies the set of constants by which the removed ABox assertions can be grounded;
- `-repleaveconst=[comma-separated set of constants over` $\Sigma(\mathcal{O})$`]` specifies the set of constants that are protected from deletion (must not occur in the ABox assertions to be deleted);
- `-replimconst=[int]` specifies the number of constants that can occur in the removed assertions.

The options `-repdelpred` and `-repleavepred` (resp. `-repdelconst` and `repleaveconst`) are dual, and they are introduced for the user's convenience. In some cases there are just few predicates (resp. constants) that must be left untouched, while in more restrictive settings, on the contrary, there are just few predicates (resp. constants) that can be removed.

7.2 Evaluation

For evaluating the developed deletion repair answer set computation algorithms based on complete and partial support families, we have built two sets of benchmark suites consisting of DL-programs over ontologies in *DL-Lite*$_{\mathcal{A}}$ and $\mathcal{EL}$ respectively. The assessment of our algorithms concerned the following aspects:

- *Performance.* We evaluated the performance of deletion repair answer set computation in comparison to the standard answer set computation on various benchmarks including Network, Taxi, LUBM, OpenStreetMap and Policy. For the Family DL-program we additionally varied the following parameters:
  - size of the DL-program data part;
  - size of the ontology TBox;
  - number of rules in the DL-program;
- *Exploiting DL-programs expressive power.* We analyzed how various advanced expressive features allowed in the DL-programs like defaults, guesses, recursiveness, influence the repair answer set computation running time (Network, LUBM-diamond, LUBM-extended benchmarks, Taxi benchmark with time constraints).
- *Repair quality.* The selection functions introduced in Section 4.1 allow us to restrict the repair search space to application-relevant repair candidates, thus ensuring a certain level of quality of the results. We evaluated how the independent σ-selection functions, like bound on the number/type of assertions eligible for deletion influence the overall algorithm runtime (Taxi benchmark in all of the introduced settings).

- *Effects of support family completeness.* We assess the impact of the support family completeness on the repair answer set computation running time by comparing various settings, in which the number/size of computed support sets is unbounded/bounded by an integer upon the availability of information on the support family completeness (Policy, Open Street Map, LUBM-$\mathcal{EL}$ benchmarks).

- *Real world data.* For demonstrating the applicability of the developed algorithms to the real world scenarios, we conducted experiments on the DL-programs built over the Open Street Map data (Taxi with district information and Open Street Map benchmark).

7.2.1 Platform Description

The repair answer set computation approach was evaluated on a Linux server with two 12-core AMD 6176 SE CPUs with 128GB RAM running the HTCondor load distribution system[12], which is a specialized workload management system for compute-intensive tasks. We used the version 2.3.0 of the dlvhex system. For each run the system usage was limited to two cores and 8GB RAM. The timeout was set to 300 seconds for each instance. The experimental data is available at `http://www.kr.tuwien.ac.at/staff/dasha/thesis/experimental_data.zip`.

Clearly, it would be natural to evaluate our algorithms by comparing their performance with other algorithms, serving similar needs, which are implemented within the existing systems. However, to the best of our knowledge, no comparable system designed for repairing inconsistent DL-programs exists, therefore we had to proceed with comparison to the systems for standard answer set computation.

The list of systems for DL-programs evaluation includes the following:

- The DReW system [13] [Xia14] is designed for evaluating DL-programs by means of a rewriting to datalog. A straightforward implementation of the repair computation was realized within the DReW system with the naive guess of the repair ABox candidate, followed by a check of its suitability. However, such implementation turned out to be ineffective even on small instances, since in general the search space of the repairs is too big for its full exploitation, and guided search is vital to ensure scalability. We have not performed a full comparison of our implementation with the DReW system, since in its current version negative queries and the negative updates (operator $\uplus$) are not supported. Note, moreover, that

- The dlplugin of the dlvhex[14], which uses the RacerPro reasoner as a back-end for evaluation of the calls to the ontology, is another candidate for comparison. However, since the dlliteplugin used for standard answer set computation for DL-programs over lightweight ontologies scales better then the former [EFRS14], we use the latter for comparison in our experiments.

[12] `http://research.cs.wisc.edu/htcondor`
[13] `http://www.kr.tuwien.ac.at/research/systems/drew/`
[14] `https://github.com/hexhex/dlplugin`

7.2.2 Evaluation Workflow

We now describe the general workflow of the experimental evaluation.

In the first step of the evaluation process we **constructed benchmarks**. This was nontrivial, since first very few benchmarks already exist [Xia14] and second it is difficult to synthesize random test instances whose conflict space would effectively reflect realistic scenarios. We exploited the existing ontologies and aimed at building rules and constraints on top of them in such a way that for some data parts the constructed programs become inconsistent.

When the scenario was defined, we created shell scripts for **instance generation** with certain varying parameters (e.g. data size, rules size, TBox size), specific for each benchmark. The scripts are available at `https://github.com/hexhex/dlplugin/benchmarks`. We then **run the benchmarks** using the HTCondor system and finally **extract** the results from the log files of the runs.

For each benchmark we **present** our experimental results in tables. The first column p in the tables specifies the size of the instance (varied according to certain parameters specific for each benchmark), and the number of generated instances in round brackets. For example, the value 10(20) in the first column states that a set of 20 instances of with the parameter 10 were tested. The rest of the columns represent configurations, in which the system was tested, e.g. AS (RAS) stands for normal (repair) answer set computation. The test configuration vary from benchmark to benchmark, thus their meaning is separately clarified where tables are presented. The cells contain combinations of numbers of the form $t(m)[n]$, where t is the total average running time, m is the number of timeouts and n is the number of repair answer sets computed.

7.2.3 Benchmarks

For the evaluation of the developed algorithms we considered the following benchmarks.

(1) DL-programs over *DL-Lite*$_{\mathcal{A}}$ ontologies:

- (1.1) The family benchmark describes a scenario, that is built from an extended version of Example 4.1 with ABoxes $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$ containing 50 and 1000 children and information about their families;
- (1.2) The Network benchmark comprises rules with recursiveness and guessing features over an ontology containing data about availability of nodes and edges of a network. We considered networks with the topology of the Vienna metro transport system having 206 nodes and its fragments with 70 nodes;
- (1.3) The taxi-driver benchmark represents a driver-customer assignment problem over an ontology with ABoxes $\mathcal{A}_{50}$ and $\mathcal{A}_{500}$ containing information about 50 and 500 customers respectively. Based on certain conditions about the drivers, customers, their positions and intentions, the customers are assigned to drivers, who are supposed to serve their needs;

(1.4) The LUBM benchmark is a set of rules with various expressiveness features built over the famous LUBM ontology[15] in its *DL-Lite*$_\mathcal{A}$ form with information about one university. The original version of LUBM is in $\mathcal{ALEHI}(D)$ form. For creating the *DL-Lite*$_\mathcal{A}$ version of LUBM we rewrote if possible and removed otherwise the TBox axioms that do not fall into the *DL-Lite*$_\mathcal{A}$ DL. For ABox generation we used the dedicated Combo tool[16]. There are several variations of this benchmark that we considered, including a basic inconsistent program, version of the Nixon diamond and an extended assignment scenario;

(2) DL-programs over $\mathcal{EL}$ ontologies:

(2.1) The policy benchmark is built from an extended version of Example 4.1, we considered ABoxes $\mathcal{A}_{40}$, $\mathcal{A}_{100}$ and 1000 with 50, 100 and 1000 staff members respectively;

(2.2) The OpenStreetMap benchmark contains a set of rules over a MyITS ontology, which is an enhanced personalized route planning with semantic information with an ABox containing data from the OpenStreetMap project[17];

(2.3) The LUBM benchmark is a variant of (1.4) with an ontology in $\mathcal{EL}$. Similarly as for *DL-Lite*$_\mathcal{A}$ we got rid of those axioms from the original LUBM ontology that do not fall into the $\mathcal{EL}$ DL. Therefore, the ontology used for the benchmark (1.4) differs from the one in this benchmark in the form of some axioms that appear in it.

The ontology statistics for each of the built benchmarks is presented in the Table 7.2.3. Here the columns store information about the size of the TBox, number of columns, roles, ABox size and number of individuals.

7.2.4 DL-programs over *DL-Lite*$_\mathcal{A}$ Ontologies

Family Benchmark

The first benchmark is derived from Example 4.1. For our evaluation we have constructed different scenarios, varying the size of the TBox, the data part as well as the rule part of the DL-program.

1. Size of the data part. In the first experimental setting, we fixed two ABoxes $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$, of different size, where $\mathcal{A}_{50}$ contains 50 children (7 adopted), 20 female and 32 male adults; and twenty times that many for $\mathcal{A}_{1000}$. Every child has at most two parents of different sex and the number of children per parent varies from 1 to 3. Rules (11) and (12), not involved in conflicts, have been dropped from $\mathcal{P}$. Instances are

[15] `http://swat.cse.lehigh.edu/projects/lubm/`
[16] `http://code.google.com/p/combo-obda/`
[17] `http://www.openstreetmap.org/`

Benchmark		Ontology expressivity	TBox size	Concepts	Roles	ABox Size		Individuals
Family		$DL\text{-}Lite_{\mathcal{A}}$	3	5	1	$\mathcal{A}_{50}$	312	102
						$\mathcal{A}_{1000}$	6183	2021
Network		$DL\text{-}Lite_{\mathcal{A}}$	3	4	2	$\mathcal{A}_{67}$	204	67
			3	5	2	$\mathcal{A}_{161}$	672	161
Taxi	Basic	$DL\text{-}Lite_{\mathcal{A}}$	3	4	2	$\mathcal{A}_{50}$	259	75
						$\mathcal{A}_{500}$	4370	714
	Time		4	6	2		274	75
	Districts		389	339	41	$\mathcal{A}_{50}$	418	93
						$\mathcal{A}_{500}$	6744	723
LUBM	Basic	$DL\text{-}Lite_{\mathcal{A}}$	95	44	31		7293	1555
	Diamond							
	Extended		101	48	31		7412	1605
Policy		$\mathcal{EL}$	5	8	3	$\mathcal{A}_{40}$	199	64
						$\mathcal{A}_{100}$	475	148
						$\mathcal{A}_{1000}$	4615	1408
OSM		$\mathcal{EL}$	405	356	36		4195	1537
LUBM-basic		$\mathcal{EL}$	94	47	28		2285	832

Table 7.1: Ontology statistics for evaluated benchmarks

varied in terms of facts over $\mathbf{I}$ included in $\mathcal{P}$. The parameter reflecting the instance size is p, which ranges from 10 to 100. A benchmark instance has size p if for every child c, additional facts $boy(c)$ and $isChildOf(c, d)$ appear in $\mathcal{P}$ with a probability $p/100$, where d is a random male adult non-parent. In this benchmark we have the number of facts in $\mathcal{P}$ as a varying parameter, since the latter allows us to control the size of the actual conflict part in the program.

The results for this benchmark are provided in Table 7.2. For each probability p we generated 20 random instances with the fixed ABoxes of $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$, and evaluated the running time for the standard answer set computation (column AS) and the repair answer set computations (column RAS) without any restrictions on the repair (column no_restr) and limiting the number of allowed assertions for deletion to 10 for $\mathcal{A}_{50}$ and 20 for $\mathcal{A}_{1000}$. The numbers in square brackets refer to the number of instances (out of 20 per p) for which the requested repairs were identified.

As the numbers in the second column show, all of the considered instances are inconsistent, which is identified by the solver within 2 milliseconds. In most cases the repairs are found for both $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$ except for $lim = 10$ of $\mathcal{A}_{50}$, where the repairs are computed only for some of the instances up to $p = 40$.

2. Ontology TBox size. In our second setting, we built instances based on the size of the TBox, leaving the ontology ABox fixed to $\mathcal{A}_{50}$. The TBox axioms from Example 4.1 are extended by the inclusions $P \sqsubseteq Person$ for all concepts P, informally stating that

p	$\mathcal{A}_{50}$			$\mathcal{A}_{1000}$		
	AS	RAS		AS	RAS	
		no_restr	$lim = 10$		no_restr	$lim = 10$
10 (20)	0.14 (0)[0]	0.22 (0)[20]	1.73 (0)[20]	63.12 (0)[0]	37.03 (0)[20]	60.21 (0)[20]
20 (20)	0.14 (0)[0]	0.23 (0)[20]	2.10 (0)[19]	62.56 (0)[0]	38.56 (0)[20]	62.19 (0)[20]
30 (20)	0.14 (0)[0]	0.24 (0)[20]	2.33 (0)[18]	62.83 (0)[0]	40.03 (0)[20]	64.27 (0)[20]
40 (20)	0.14 (0)[0]	0.25 (0)[20]	2.88 (0)[11]	63.23 (0)[0]	41.81 (0)[20]	66.81 (0)[20]
50 (20)	0.14 (0)[0]	0.25 (0)[20]	3.93 (0) [1]	63.42 (0)[0]	43.86 (0)[20]	68.87 (0)[20]
60 (20)	0.15 (0)[0]	0.26 (0)[20]	3.93 (0) [2]	63.42 (0)[0]	45.87 (0)[20]	71.63 (0)[20]
70 (20)	0.14 (0)[0]	0.27 (0)[20]	4.00 (0) [0]	63.18 (0)[0]	47.83 (0)[20]	74.14 (0)[20]
80 (20)	0.15 (0)[0]	0.28 (0)[20]	4.08 (0) [0]	63.38 (0)[0]	49.71 (0)[20]	76.35 (0)[20]
90 (20)	0.15 (0)[0]	0.29 (0)[20]	4.48 (0) [0]	63.59 (0)[0]	52.18 (0)[20]	79.14 (0)[20]
100 (20)	0.14 (0)[0]	0.30 (0)[20]	4.42 (0) [0]	63.08 (0)[0]	54.14 (0)[20]	81.81 (0)[20]

Table 7.2: Family benchmark: data size variations, fixed $\mathcal{P}$ and $\mathcal{T}$

every individual known to be either child, adopted, male or female is a person. Moreover, for each concept P from the signature and $1 \leq i \leq \mathcal{T}_{max}$ with the probability $p/100$ (p ranges from 10 to 100) we added the following inclusions:

$$(1)\ PMemberOfSocGroup_i \sqsubseteq P \quad (2)\ \exists hasIDOfSocGroup_i \sqsubseteq Person.$$

Intuitively, (1) reflects that a P-member of a social group i is in the class P, while (2) states that each individual having ID of a certain social group i is a person.

The evaluation results for this setting are presented in Table 7.3. The column $\mathcal{T}_{max} = n$ means that for the given ontology instance each out of n TBox axioms was added with the probability p to $\mathcal{T}$. One can see that the repair computation is slower then the standard answer set computation, which becomes more obvious for the setting, where $\mathcal{T}_{max} = 5000$. This is due to the construction of support sets and their exploitation in our declarative approach for repair answer set computation. In the standard setting, we do not exploit the TBox extensively, and therefore its growing size does not have much impact on the running time. As expected, restricting the elimination of a number of ABox facts to k-bounded slows down the repair computation process.

3. Size of the rule part. In the third setting we evaluated the influence of the rule part size on the algorithm performance. Apart from the rules (11) and (12) from Example 4.1 that were excluded in the previous settings, we also added for $1 \leq i \leq R_{max}$ and for $1 \leq j \leq i$ with probability $p/100$ ($10 \leq p \leq 70$) the following rules:

$$(1)\ contact_i(X, Y) \leftarrow contact(X, Y), not\ omit(X, Y) \quad (2)\ omit_i(X, Y) \leftarrow omit(X, Y)$$
$$(3)\ contact_j(X, Y) \leftarrow contact_i(X, Y), not\ omit_j(X, Y) \quad (4)\ omit_j(X, Y) \leftarrow omit_i(X, Y).$$

The newly introduced predicate $contact_i(c, p)$ informally means that p is a contact representative for a child c within a social group i. The rules (1)-(4) state that if a contact for a child was identified, then this contact can be propagated to randomly chosen social groups i and j.

The results for repair case are presented in Table 7.4. Standard answer set computation times out even for smaller instances, which is due to a large number of rules present

p	$\mathcal{T}_{500}$			$\mathcal{T}_{5000}$		
	AS	RAS		AS	RAS	
		no_restr	$lim = 10$		no_restr	$lim = 10$
10 (20)	0.15 (0)[0]	0.32 (0)[20]	1.95 (0)[20]	0.28 (0)[0]	3.58 (0)[20]	6.03 (0)[20]
20 (20)	0.16 (0)[0]	0.47 (0)[20]	2.17 (0)[20]	0.48 (0)[0]	12.89 (0)[20]	15.96 (0)[20]
30 (20)	0.17 (0)[0]	0.68 (0)[20]	2.47 (0)[20]	0.75 (0)[0]	27.76 (0)[20]	31.42 (0)[20]
40 (20)	0.19 (0)[0]	0.93 (0)[20]	2.78 (0)[20]	1.10 (0)[0]	48.46 (0)[20]	53.24 (0)[20]
50 (20)	0.20 (0)[0]	1.25 (0)[20]	3.19 (0)[20]	1.51 (0)[0]	76.39 (0)[20]	81.54 (0)[20]
60 (20)	0.21 (0)[0]	1.58 (0)[20]	3.56 (0)[20]	1.99 (0)[0]	108.33 (0)[20]	114.71 (0)[20]
70 (20)	0.23 (0)[0]	2.09 (0)[20]	4.18 (0)[20]	2.56 (0)[0]	146.62 (0)[20]	152.91 (0)[20]
80 (20)	0.24 (0)[0]	2.54 (0)[20]	4.68 (0)[20]	3.17 (0)[0]	191.37 (0)[20]	198.72 (0)[20]
90 (20)	0.26 (0)[0]	3.06 (0)[20]	5.28 (0)[20]	3.91 (0)[0]	241.51 (0)[20]	248.19 (0)[20]

Table 7.3: Family benchmark: TBox size variations, fixed $\mathcal{P}$ and $\mathcal{A}_{50}$

in the programs. For normal answer set computation we did not enable optimization techniques which split a DL-program into independent components and evaluate each component separately, i.e. instead of module-based a monolithic evaluation heuristics was used. The monolithic heuristics was chosen to make the evaluation comparison fair. The repair model generator does not support any other heuristics at the moment and the extensions are nontrivial, which was already mentioned in the previous section.

Here the maximal number of rules that were added is specified in the column "names". Each out of the maximal number of rules is present in the test instance with the probability p. We can see that the growing number of rules makes an impact on the running time of the algorithm, which is non surprising, as the added rules introduce conflicts due to a cycle through negation. Restriction to elimination of 10 facts for $Rule_{max} = 50$ slows down the computation in comparison to the non-restricted scenario. For larger instance size, $Rules_{max} = 500$ the same restriction is too strict, thus no repairs are actually found. Weakening the restriction for larger instance size ($Rules_{max} = 5000$) produces again some repair answer sets, though only for smaller p. For larger p timeouts are obtained, which is natural as bearing thousands of rules even for a standard ASP solver computing answer sets for consistent programs is time-consuming.

p	$Rules_{max} = 50$		$Rules_{max} = 500$		$Rules_{max} = 5000$	
	RAS	$RAS_{lim=10}$	RAS	$RAS_{lim=10}$	RAS	$RAS_{lim=20}$
10 (20)	0.55 (0)[20]	2.09 (0)[20]	2.56 (0)[20]	23.23 (0)[0]	64.65 (0)[20]	110.92 (0)[20]
20 (20)	0.69 (0)[20]	2.35 (0)[20]	5.22 (0)[20]	77.30 (0)[0]	257.35 (11)[9]	300.00 (20)[0]
30 (20)	0.90 (0)[20]	2.67 (0)[20]	8.50 (0)[20]	158.23 (0)[0]	300.00 (20)[0]	300.00 (20)[0]
40 (20)	0.97 (0)[20]	2.86 (0)[20]	11.86 (0)[20]	128.87 (1)[0]	300.00 (20)[0]	300.00 (20)[0]
50 (20)	1.18 (0)[20]	3.11 (0)[20]	14.91 (0)[20]	144.71 (0)[0]	300.00 (20)[0]	300.00 (20)[0]
60 (20)	1.29 (0)[20]	3.28 (0)[20]	17.68 (0)[20]	164.70 (0)[0]	300.00 (20)[0]	300.00 (20)[0]
70 (20)	1.42 (0)[20]	3.19 (0)[20]	20.11 (0)[20]	186.38 (3)[0]	300.00 (20)[0]	300.00 (20)[0]

Table 7.4: Family benchmark: rule size variations, fixed $\mathcal{T}$ and $\mathcal{A}_{50}$

Network Benchmark

In our next scenario, the properties of the nodes and edges in a network are described by a fixed ontology $\mathcal{O}$ using predicates *Blocked*, *Broken*, *Avail* for nodes and $forbid$ for edges. The TBox encodes that if an edge is forbidden, then its endpoint must be blocked, and if a node is known to be broken, then it is automatically blocked, moreover blocked nodes are not available:

$\mathcal{O} = \{\exists forbid \sqsubseteq Block, \quad Broken \sqsubseteq Block, \quad Block \sqsubseteq \neg Avail\}$.

We considered two networks of 67 and 161 nodes with respectively 117 and 335 edges. The networks represent fragments of the Vienna public transportation system. In each network we randomly made 30% of the nodes broken and 20% of the edges forbidden; the larger network has in addition 47 blocked nodes. This information is stored in the data part of $\mathcal{O}$.

The experiments were run on two DL-programs $\mathcal{P}_{conn}$ and $\mathcal{P}_{guess}$ over $\mathcal{O}$. Both programs contain as facts edges and nodes of the graph, as well as randomly generated facts determining the portion of the nodes on which a condition expressed by the rules of the program is checked. For creating the data part of the $\mathcal{P}_{conn}$ program, we partitioned the set of nodes randomly into two sets, i.e. the set of *in* nodes and the set of *out* nodes. For each node n from the *in* set, the fact $in(n)$ is added with probability $p/100$. For each node n' from the set of *out* nodes, the fact $out(n')$ is added with probability p' computed in the following way: if $0 \leq p \leq 20$, then $p' = p * 4$, if $20 \leq p \leq 30$, then $p' = p * 3$. $\mathcal{P}_{conn}$ contains, moreover, the following rules:

$$\mathcal{P}_{conn} = \left\{ \begin{array}{ll} (1) & go(X,Y) \leftarrow open(X), open(Y), edge(X,Y); \\ (2) & route(X,Z) \leftarrow route(X,Y), route(Y,Z); \\ (3) & route(X,Y) \leftarrow go(X,Y), not\ \mathrm{DL}[; forbid](X,Y); \\ (4) & open(X) \leftarrow node(X), not\ \mathrm{DL}[; \neg Avail](X); \\ (5) & ok(X) \leftarrow in(X), out(X), route(X,Y); \\ (6) & fail \leftarrow in(X), not\ ok(X); \\ (7) & \bot \leftarrow fail. \end{array} \right\}$$

Intuitively, (1)-(4) recursively determine routes over non-blocked (*open*) nodes; where (3) expresses that by default a route is recommended unless it is known to be forbidden. Rules (5)-(7) encode the requirement that each node from the *in* set must be connected to at least one node from the *out* set via a route, which amounts to a variation of a generalized connectivity problem.

The running times and repair results for this benchmark with the network containing 67 nodes is given in Table 7.5. The same number of repairs is computed for all of the *RAS* settings, but the running times slightly vary as expected. The last column, where only broken nodes and forbidden edges are allowed for removal, has similar running times as the unrestricted setting. This is also the case for a larger network with 161 nodes (Table 7.6); however in contrast to the 67-node network, this restriction does not yield repairs. Probably one also needs to remove blocked/unavailable nodes from the ontology in order to obtain repairs.

p	AS	RAS			
		no_restr	$lim = 3$	$lim = 20$	$Broken, forbid$
2 (20)	0.10 (0)[12]	0.46 (0)[20]	0.84 (0)[20]	0.66 (0)[20]	0.46 (0)[20]
6 (20)	0.10 (0) [5]	0.45 (0)[16]	0.79 (0)[16]	0.61 (0)[16]	0.44 (0)[16]
10 (20)	0.09 (0) [3]	0.43 (0)[14]	0.76 (0)[14]	0.59 (0)[14]	0.43 (0)[14]
14 (20)	0.09 (0) [2]	0.41 (0)[10]	0.71 (0)[10]	0.54 (0)[10]	0.41 (0)[10]
18 (20)	0.09 (0) [0]	0.40 (0) [7]	0.67 (0) [7]	0.51 (0) [7]	0.40 (0) [7]
22 (20)	0.09 (0) [0]	0.41 (0) [9]	0.70 (0) [9]	0.54 (0) [9]	0.41 (0) [9]
26 (20)	0.09 (0) [0]	0.38 (0) [3]	0.63 (0) [3]	0.47 (0) [3]	0.38 (0) [3]
30 (20)	0.09 (0) [0]	0.37 (0) [2]	0.62 (0) [2]	0.46 (0) [2]	0.37 (0) [2]

Table 7.5: Network-connectivity benchmark results: $\mathcal{A}_{67}$

p	RAS				
	no_restr	$lim = 3$	$lim = 20$	$lim = 100$	$Broken, forbid$
2 (20)	179.49 (1)[19]	280.73 (16)[0]	288.64 (17)[3]	176.06 (1)[19]	125.47 (0)[0]
4 (20)	218.80 (8)[12]	291.80 (18)[0]	295.48 (19)[1]	226.25 (8)[12]	127.68 (0)[0]
8 (20)	230.79 (9)[11]	298.39 (19)[0]	300.00 (20)[0]	232.65 (9)[11]	126.97 (0)[0]
10 (20)	258.08 (14)[5]	300.00 (20)[0]	300.00 (17)[0]	259.69 (14)[6]	125.63 (0)[0]

Table 7.6: Network-connectivity benchmark results: $\mathcal{A}_{161}$

Another setting that we considered is a benchmark over the program $\mathcal{P}_{guess}$, which has the same rules (1) and (2) as $\mathcal{P}_{conn}$, while the rest of the rules are as follows:

(3*) $route(X, Y) \leftarrow go(X, Y), not\ \mathrm{DL}[Block \uplus block; forbid](X, Y).$
(4*) $open(X) \vee block(X) \leftarrow domain(X), not\ \mathrm{DL}[; \neg Avail](X).$
(5*) $open(X) \leftarrow node(X), not\ \mathrm{DL}[; Broken](X), not\ block(X).$
(6*) $negis(X) \leftarrow domain(X), route(X, Y), X \neq Y.$
(7*) $\bot \leftarrow domain(X), not\ negis(X).$

The rule (3*) has an update in the DL-atom, the rule (4*) amounts to guessing for all selected nodes (*domain* predicate) not known to be unavailable, whether they are blocked or not, i.e. it contains nondeterminism, which makes rule processing challenging. Other nodes are open by default, unless they are known to be broken, which is encoded in the rule (5*). Rules (6*) and (7*) check whether none of the *domain* nodes is isolated, i.e. does not have a connection to any other node via a route.

The results for $\mathcal{P}_{guess}$ with networks of 67 and 161 nodes are in Table 7.7 and Table 7.8 respectively. The facts $domain(n)$ are added for each node n with probability $p/10$. One could observe a strict increase in the running time for the network with 67 nodes for $p = 2$, $p = 10$ and standard answer set computation. The reason for such a behavior is that some of the considered instances are in fact consistent, but because of the guessing rules answer sets are not found within the time frame of 300 seconds, and therefore timeouts happen. For bigger p the instances are inconsistent and the conflict is quickly determined by the solver. The results for 161 nodes given in Table 7.8 show that the guided search (last column) increases the number of found repairs quit a bit, and less timeouts are hit for $p = 4$ and $p = 8$.

p	AS	RAS				
		no_restr	$lim = 3$	$lim = 10$	$limc = 10$	$Broken$
2 (20)	180.06 (12)[0]	0.51 (0)[20]	0.91 (0)[19]	0.90 (0)[20]	0.69 (0)[20]	0.50 (0)[20]
10 (20)	15.17 (1) [0]	1.33 (0)[16]	0.89 (0) [2]	1.61 (0)[16]	0.85 (0)[16]	1.31 (0)[16]
18 (20)	0.18 (0) [0]	1.68 (0) [8]	0.90 (0) [0]	1.40 (0) [8]	0.81 (0) [8]	1.68 (0) [8]
26 (20)	0.19 (0) [0]	0.62 (0) [1]	0.97 (0) [0]	0.95 (0) [1]	0.60 (0) [1]	0.62 (0) [1]
34 (20)	0.20 (0) [0]	0.79 (0) [1]	1.04 (0) [0]	1.02 (0) [1]	0.62 (0) [1]	0.78 (0) [1]

Table 7.7: Network-guessing benchmark results: $\mathcal{A}_{67}$

p	RAS			
	no_restr	$lim = 10$	$limc = 100$	$Broken$
2 (20)	178.52 (3)[15]	187.65 (2)[16]	175.64 (2)[16]	179.57 (3)[15]
4 (20)	201.89 (6)[10]	211.10 (7) [9]	213.66 (9) [7]	178.55 (3)[13]
8 (20)	212.18 (10) [2]	215.44 (10) [2]	205.77 (9) [3]	191.97 (7) [5]
10 (20)	190.58 (9) [0]	184.80 (8) [1]	191.54 (9) [0]	191.06 (9) [0]

Table 7.8: Network-guessing benchmark results: $\mathcal{A}_{161}$

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Driver} \sqsubseteq \neg \mathit{Cust} & (3)\ \exists \mathit{worksIn} \sqsubseteq \mathit{Driver} \\ (2)\ \mathit{EDriver} \sqsubseteq \mathit{Driver} & (4) \mathit{worksIn} \sqsubseteq \neg \mathit{notworksIn} \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (5)\ \mathit{cust}(X) \leftarrow \mathit{isIn}(X, Y), \mathit{not}\ \mathrm{DL}[; \neg \mathit{Cust}](X); \\ (6)\ \mathit{driver}(X) \leftarrow \mathit{not}\ \mathit{cust}(X), \mathit{isIn}(X, Y); \\ (7)\ \mathit{drives}(X,Y) \leftarrow \mathit{cust}(Y), \mathit{isIn}(Y,Z), \mathit{isIn}(X,Z), \\ \qquad\qquad \mathit{driver}(X), \mathit{not}\ \mathit{omit}(X, Y); \\ (8)\ \mathit{omit}(X,Y) \leftarrow \mathit{needsTo}(Y, Z),\ \mathrm{DL}[; \mathit{notworksIn}](X, Z), \\ \qquad\qquad \mathrm{DL}[\mathit{Driver} \uplus \mathit{driver};\ \mathit{EDriver}](X); \\ (9)\ \mathit{ok}(Y) \leftarrow \mathit{customer}(Y), \mathit{drives}(X, Y); \\ (10)\ \mathit{fail} \leftarrow \mathit{customer}(Y), \mathit{not}\ \mathit{ok}(Y); \\ (11)\ \bot \leftarrow \mathit{fail}. \end{array} \right\}$$

Figure 7.6: DL-program from Taxi-basic benchmark

Taxi-Driver Assignment Benchmark

The third experimental setting represents the taxi-driver assignment problem [EFRS14].

Imagine a system that assigns customers to taxi drivers under constraints, using (in a simplistic form) the DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ presented in Figure 7.6. The (external) ontology $\mathcal{O}$ has a taxonomy $\mathcal{T}$ in (1)-(3). The logic program $\mathcal{P}$ has the following rules: (5) and (6) single out customers resp. taxi drivers; (7) assigns taxi drivers to customers in the same region; and (8) forbids drivers of electro-cars to serve needs going outside their working region. Finally, the rules (9), (10) and a constraint (11) make sure that each customer is assigned to at least one driver.

1. Repair Quality Assessment. One might argue that in case of inconsistency there are not many possibilities for repairing the given system. Indeed, for instance, removing information about the drivers seems absurd at the first glance, as it will introduce new

individuals who are not anymore known to be drivers, and thus assumed to be customers due to the default (5). Observe that a complete removal of driver information will not make the system consistent, but on the contrary will introduce even more customers, who will then possibly need to be assigned to the drivers. Therefore, it is obvious that the guided repair search is often crucial and it should not only improve the repair quality but also reduce the computation runtime.

In this setting we evaluated the quality of the repair computation by considering the evaluation time of the repair computation under various independent selection functions. The functions include restriction to a certain set of predicates for deletion (in our case the facts with the predicate $EDriver$ only are allowed for deletion) and limitation of the number of removed facts, predicates and constants. Since the selection functions filter out irrelevant repair candidates, they ensure a certain quality of the computed repairs. While limiting the number of assertions for removal is natural and can be easily justified, one might wonder in which cases the removal of the drivers of electro-cars is of practical use. Here we can imagine that electro-cars have also possibility to use petrol, but for environmental reasons this is undesired, and the government wants to reduce the petrol usage. However, in case it is vital and some customers are left without drivers, they still can switch back to the petrol energy supply.

For the DL-program Π, we fixed two ABoxes: $\mathcal{A}_{50}$ and $\mathcal{A}_{500}$ of the ontology $\mathcal{O}$. The ABox $\mathcal{A}_{50}$ contains 50 customers, 20 drivers (among them 19 driving electro-cars), and 5 regions; every driver works in 2-4 regions, and the ABox $\mathcal{A}_{500}$ has 10 times as many customers and drivers as $\mathcal{A}_{50}$. In the program $\mathcal{P}$ from above, facts $isIn(c, r)$, $needsTo(c, r), goTo(d, r)$ for appropriate constants c, d, r from $\mathcal{A}$ are randomly added with probability $p/100$ under the following constraints: persons are in at most one region; customers need to go to at most one region, and their position is known in that case. Furthermore, driver positions are added as facts $isIn(d, r)$ with fixed probabilities of 0.3, 0.7 and 1 growing discretely in accordance with p.

For $\mathcal{A}_{50}$ and $\mathcal{A}_{500}$ the results are given in Table 7.9 and Table 7.10 respectively, where the first column shows in parentheses the number of instances generated per value p. The second and third column state results for standard and repair answer set computation, respectively, while the rest of the columns present the running times for repair computation under various selection functions, i.e. in the fourth and fifth column we restricted repairs by allowing removal of only a limited number of assertions (3 and 10) and in the sixth and seventh column we computed repairs where only facts containing 2 predicates and 10 constants are eliminated. Finally in the last column the results for removing only $EDriver$ facts are shown.

One can see that the limitation of the number of removed assertions makes the computation slower in comparison to the normal repair computation. For $repdel = EDriver$, the guided repair computation effectively reduces the search space of the candidates, and it helps the solver to find repairs quicker. In fact analysis of the program reveals that most of the valid repairs exclude certain $EDriver$ concept membership, since they often cause the omission of driver-customer assignments and thus violation of constraint (11). Results for a larger ABox (500 customers and 200 drivers, including

p	AS	RAS					
		no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
10 (20)	0.69 (0)[0]	0.14 (0)[13]	0.75 (0)[11]	0.75 (0)[13]	0.31 (0)[13]	0.26 (0)[13]	0.14 (0)[13]
20 (20)	0.37 (0)[0]	0.15 (0) [8]	0.89 (0) [4]	0.87 (0) [8]	0.32 (0) [8]	0.25 (0) [8]	0.15 (0) [8]
30 (20)	0.22 (0)[0]	0.16 (0) [7]	0.92 (0) [2]	0.89 (0) [7]	0.32 (0) [7]	0.26 (0) [7]	0.16 (0) [7]
40 (20)	0.58 (0)[0]	0.18 (0) [8]	1.06 (0) [1]	1.04 (0) [8]	0.36 (0) [8]	0.28 (0) [8]	0.18 (0) [8]
50 (20)	0.46 (0)[0]	0.18 (0) [7]	1.01 (0) [2]	0.98 (0) [7]	0.36 (0) [7]	0.29 (0) [7]	0.18 (0) [7]
60 (20)	0.22 (0)[0]	0.19 (0)[11]	1.02 (0) [1]	0.99 (0)[11]	0.38 (0)[11]	0.31 (0)[11]	0.19 (0)[11]
70 (20)	0.22 (0)[0]	0.21 (0) [4]	1.00 (0) [0]	0.99 (0) [4]	0.37 (0) [4]	0.29 (0) [4]	0.20 (0) [4]
80 (20)	1.02 (0)[0]	0.22 (0) [9]	1.10 (0) [1]	1.10 (0) [9]	0.40 (0) [9]	0.33 (0) [9]	0.22 (0) [9]
90 (20)	1.30 (0)[0]	0.23 (0)[12]	1.26 (0) [0]	1.20 (0)[12]	0.44 (0)[12]	0.36 (0)[12]	0.24 (0)[12]
100 (20)	1.47 (0)[0]	0.24 (0)[13]	1.20 (0) [0]	1.15 (0)[13]	0.45 (0)[13]	0.37 (0)[13]	0.26 (0)[13]

Table 7.9: Taxi-basic benchmark results: $\mathcal{A}_{50}$

p	RAS					
	no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
10 (20)	4.36 (0)[20]	44.06 (1)[19]	37.13 (0)[20]	6.65 (0) [20]	5.13 (0) [20]	6.50 (0) [20]
20 (20)	6.34 (0)[20]	24.05 (0)[20]	24.35 (0)[20]	8.80 (0) [20]	7.35 (0) [20]	8.94 (0) [20]
30 (20)	8.46 (0)[20]	40.88 (1)[19]	38.93 (0)[20]	11.08 (0)[20]	9.72 (0) [20]	11.58 (0)[20]
40 (20)	10.26(0)[20]	28.69 (0)[20]	28.72 (0)[20]	12.84 (0)[20]	11.60 (0)[20]	13.56 (0)[20]
50 (20)	12.64(0)[19]	31.58 (0)[19]	31.80 (0)[19]	15.47 (0)[19]	14.59 (0)[19]	16.66 (0)[19]
60 (20)	15.43(0)[20]	48.19 (1)[19]	48.28 (1)[19]	18.20 (0)[20]	18.11 (0)[20]	19.95 (0)[20]
70 (20)	18.33(0)[20]	37.61 (0)[20]	37.74 (0)[20]	20.86 (0)[20]	20.61 (0)[20]	23.49 (0)[20]
80 (20)	20.63(0)[20]	40.09 (0)[20]	40.10 (0)[20]	23.21 (0)[20]	23.16 (0)[20]	26.19 (0)[20]
90 (20)	21.81(0)[20]	54.53 (1)[19]	54.35 (1)[19]	24.34 (0)[20]	26.41 (0)[20]	27.48 (0)[20]
100 (20)	26.22(0)[20]	46.67 (0)[20]	46.73 (0)[20]	28.99 (0)[20]	29.63 (0)[20]	33.21 (0)[20]

Table 7.10: Taxi-basic benchmark results: $\mathcal{A}_{500}$

170 driving electro-cars), shown in Table 7.10, are similarly satisfactory.

2. Setting with rules involving time constraints. We modified the DL-program from above to take into account also time slots when the drivers are on duty (see Figure 7.7). Here the TBox and the rule part are extended. The additional rules (11) and (12) do not allow drivers to work at time slots which they are not assigned to.

The results for this setting are given in Table 7.11. The standard repair computation results in timeout for all instances due to the inequalities in rules (8)-(10), as most of the instances are inconsistent it takes the reasoner long to identify the conflict. For the repair computation we can get results within a second. We considered here restrictions on the number facts, predicates and constants that are removed. The running times naturally increase when the restriction is too strict, and then again slightly drop for less restrictive cases. When limiting repairs to removal of $EDriver$ facts only, a certain speed up in comparison to the non-restricted case is observed. The latter is non-surprising, as the guided search as in previous setting makes the repair computation quicker.

3. Real world data.

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Driver} \sqsubseteq \neg \mathit{Cust} & (3)\ \mathit{EDriver} \sqsubseteq \mathit{Driver} \\ (2)\ \exists \mathit{worksIn} \sqsubseteq \mathit{Driver} & (4)\ \mathit{DayDriver} \sqsubseteq \neg \mathit{NightDriver} \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (4)\ \mathit{cust}(X) \leftarrow \mathit{isIn}(X, Y), \mathit{not}\ \mathrm{DL}[\mathit{worksIn} \uplus \mathit{goTo}; \neg \mathit{Cust}](X); \\ (5)\ \mathit{driver}(X) \leftarrow \mathit{not}\ \mathit{cust}(X), \mathit{isIn}(X, Y); \\ (6)\ \mathit{drives}(X,Y) \leftarrow \mathit{cust}(Y), \mathit{isIn}(Y,Z), \mathit{isIn}(X,Z), \\ \qquad \mathit{driver}(X), \mathit{not}\ \mathit{omit}(X, Y); \\ (7)\ \mathit{omit}(X,Y) \leftarrow \mathit{needsTo}(Y, Z, T), \mathit{not}\ \mathrm{DL}[; \mathit{worksIn}](X, Z), \\ \qquad \mathrm{DL}[\mathit{Driver} \uplus \mathit{driver}; \mathit{EDriver}](X); \\ (8)\ \mathit{omit}(X,Y) \leftarrow \mathit{drives}(X,Y), \mathit{needsTo}(X,Y,T), \mathit{day}(T), \\ \qquad \mathrm{DL}[\mathit{Driver} \uplus \mathit{driver}; \neg \mathit{DayDriver}](X); \\ (9)\ \mathit{omit}(X,Y) \leftarrow \mathit{drives}(X,Y), \mathit{needsTo}(X,Y,T), \mathit{night}(T), \\ \qquad \mathrm{DL}[\mathit{Driver} \uplus \mathit{driver}; \neg \mathit{NightDriver}](X); \\ (10)\ \mathit{night}(T) \leftarrow \mathit{needsTo}(X,Y,T), T < 9; \\ (11)\ \mathit{night}(T) \leftarrow \mathit{needsTo}(X,Y,T), T > 21; \\ (12)\ \mathit{day}(T) \leftarrow \mathit{needsTo}(X,Y,T), T \leq 21, T \geq 9; \\ (13)\ \mathit{ok}(Y) \leftarrow \mathit{customer}(Y), \mathit{drives}(X, Y); \\ (14)\ \mathit{fail} \leftarrow \mathit{customer}(Y), \mathit{not}\ \mathit{ok}(Y); \\ (15)\ \bot \leftarrow \mathit{fail}. \end{array} \right\}$$

Figure 7.7: DL-program from Taxi-time benchmark

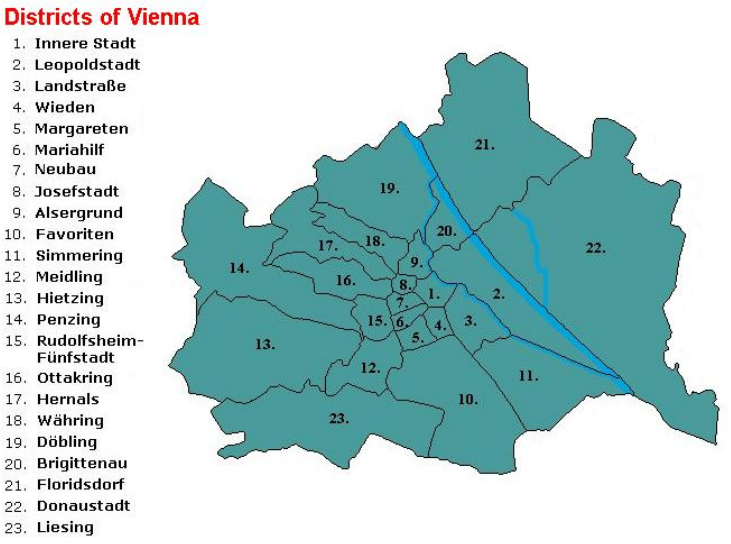


Figure 7.8: Mutual relations among Vienna districts

p	RAS					
	no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
10 (20)	0.30 (0)[20]	2.19 (0)[20]	2.12 (0)[20]	0.62 (0)[20]	0.46 (0)[20]	0.22 (0)[20]
20 (20)	0.36 (0)[20]	2.19 (0)[20]	2.15 (0)[20]	0.68 (0)[20]	0.52 (0)[20]	0.27 (0)[20]
30 (20)	0.40 (0)[20]	2.13 (0)[20]	2.15 (0)[20]	0.72 (0)[20]	0.56 (0)[20]	0.30 (0)[20]
40 (20)	0.46 (0)[20]	2.27 (0)[20]	2.26 (0)[20]	0.79 (0)[20]	0.62 (0)[20]	0.35 (0)[20]
50 (20)	0.56 (0)[20]	2.32 (0)[20]	2.31 (0)[20]	0.88 (0)[20]	0.71 (0)[20]	0.42 (0)[20]
60 (20)	0.62 (0)[20]	2.38 (0)[20]	2.37 (0)[20]	0.94 (0)[20]	0.78 (0)[20]	0.48 (0)[20]
70 (20)	0.71 (0)[20]	2.45 (0)[20]	2.45 (0)[20]	1.03 (0)[20]	0.87 (0)[20]	0.56 (0)[20]
80 (20)	0.80 (0)[20]	2.55 (0)[20]	2.55 (0)[20]	1.13 (0)[20]	0.97 (0)[20]	0.64 (0)[20]
90 (20)	0.93 (0)[20]	2.71 (0)[20]	2.68 (0)[20]	1.26 (0)[20]	1.10 (0)[20]	0.75 (0)[20]
100 (20)	0.98 (0)[20]	2.72 (0)[20]	2.72 (0)[20]	1.30 (0)[20]	1.15 (0)[20]	0.82 (0)[20]

Table 7.11: Taxi-time benchmark results: $\mathcal{A}_{50}$

Another scenario that we considered uses the Open Street Map[18] data over the modified version of the MyITS ontology [19]. This setting is modeled for demonstrating the applicability of the repair answer set computation approach for the real world data as well as the ability of coping with large TBoxes. The TBox of the MyITS ontology is extended with the axioms from the previous experimental setting, while its ABox apart from customer and driver information from above also contains the data about mutual spatial relations among the districts of Vienna (see Figure 7.8). These relations are stored using the predicates *adjoint* and $disjoint$. The respective fragment of the TBox and the rules of the program are as follows:

$$\mathcal{T} = \left\{ \begin{array}{ll} (1)\ Driver \sqsubseteq \neg Cust & (4)\ adjoint \sqsubseteq \neg disjoint \\ (2)\ \exists worksIn \sqsubseteq Driver & (5)\ EDriver \sqsubseteq Driver \\ (3)\ worksIn \sqsubseteq notworksIn & \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (5)\ cust(X) \leftarrow isIn(X, Y), not\ \mathrm{DL}[; \neg Cust](X); \\ (6)\ driver(X) \leftarrow not\ cust(X), isIn(X, Y); \\ (7)\ drives(X, Y) \leftarrow driver(X), cust(Y), needsTo(X, Z1), goTo(X, Z2), \\ \qquad\qquad \mathrm{DL}[; adjoint](Z1, Z2), not\ omit(X, Y); \\ (8)\ omit(X, Y) \leftarrow \mathrm{DL}[; EDriver](X), needsTo(Y, Z), \\ \qquad\qquad \mathrm{DL}[; notworksIn](X, Z); \\ (9)\ ok(Y) \leftarrow customer(Y), drives(X, Y); \\ (10)\ fail \leftarrow customer(Y), not\ ok(Y); \\ (11)\ \bot \leftarrow fail \end{array} \right\}$$

Intuitively, the rule (7) states that a driver can be assigned to a customer, only if the driver is going to a region adjoint to the one to which the customer needs to get. Similar as in previous scenarios, some of the assignments are dropped if they involve drivers of electro-cars aiming at the regions they are not assigned to.

The benchmark results for this setting and $\mathcal{A}_{50}$ are presented in Table 7.12 and for $\mathcal{A}_{500}$ in Table 7.13. Unsurprisingly, the restriction on the number of assertions allowed

[18] `http://www.openstreetmap.org/`

[19] `http://www.kr.tuwien.ac.at/research/projects/myits`

p	AS	RAS					
		no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
2 (20)	0.25 (0) [5]	4.12 (0) [5]	5.27 (0) [5]	5.32 (0) [5]	5.01 (0) [5]	4.98 (0) [5]	4.10 (0) [5]
10 (20)	0.25 (0) [0]	4.18 (0)[11]	6.19 (0) [7]	6.18 (0)[11]	5.22 (0)[11]	5.15 (0)[11]	4.13 (0) [3]
18 (20)	0.25 (0) [1]	4.24 (0)[14]	6.71 (0)[10]	6.74 (0)[14]	5.34 (0)[14]	5.19 (0)[14]	4.15 (0) [3]
26 (20)	0.25 (0) [1]	4.28 (0)[14]	7.26 (0) [9]	7.42 (0)[14]	5.50 (0)[14]	5.24 (0)[14]	4.22 (0) [5]
34 (20)	0.26 (0) [3]	4.39 (0)[19]	8.54 (0)[16]	8.52 (0)[19]	5.74 (0)[19]	5.40 (0)[19]	4.35 (0) [9]
42 (20)	0.27 (0) [5]	4.42 (0)[18]	9.35 (0)[18]	9.31 (0)[18]	5.86 (0)[18]	5.49 (0)[18]	4.51 (0)[16]
50 (20)	0.29 (0)[10]	4.49 (0)[19]	10.42 (0)[19]	10.29 (0)[19]	6.05 (0)[19]	5.54 (0)[19]	4.63 (0)[19]
58 (20)	0.32 (0)[14]	4.62 (0)[20]	11.48 (0)[20]	11.50 (0)[20]	6.33 (0)[20]	5.63 (0)[20]	4.76 (0)[20]
66 (20)	0.31 (0)[11]	4.61 (0)[20]	11.59 (0)[20]	13.42 (0)[20]	6.27 (0)[20]	5.71 (0)[20]	4.76 (0)[18]

Table 7.12: Taxi-districts benchmark results: $\mathcal{A}_{50}$

p	AS	RAS					
		no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
2 (20)	2.11 (0) [0]	9.22 (0) [7]	25.05 (0) [6]	24.91 (0) [7]	12.32 (0) [7]	10.24 (0) [6]	7.56 (0) [0]
10 (20)	2.23 (0) [0]	14.17 (0)[20]	46.37 (0)[20]	46.52 (0)[20]	20.54 (0)[20]	15.75 (0)[15]	12.16 (0) [4]
18 (20)	5.58 (0) [5]	15.96 (0)[20]	51.89 (0)[20]	52.44 (0)[20]	23.11 (0)[20]	17.93 (0)[20]	28.00 (0)[20]
26 (20)	17.95 (0)[12]	18.28 (0)[20]	55.30 (0)[20]	55.84 (0)[20]	25.57 (0)[20]	20.27 (0)[20]	31.76 (0)[20]
34 (20)	37.87 (0)[17]	20.81 (0)[20]	58.71 (0)[20]	58.51 (0)[20]	28.35 (0)[20]	22.93 (0)[20]	36.00 (0)[20]

Table 7.13: Taxi-districts benchmark results: $\mathcal{A}_{500}$

for deletion slows down the repair computation again. With the increase of this limit the running time slightly improves. As in the previous setting the restriction of the set of predicates allowed for deletion to $EDriver$ does not yield much of the computation overhead; however, in contrast to the previous setting the number of found repairs decreases. Here apart from the information about drivers working on electra-cars, one needs to expand the working area of the drivers too, thus removal of $notworksIn$ facts should again increase the number of obtained repairs.

LUBM-*DL-Lite*$_\mathcal{A}$

Our approach has been also evaluated on DL-programs over the famous LUBM ontology[20] in its *DL-Lite*$_\mathcal{A}$ form. For ABox generation we used the dedicated Combo tool[21]. We considered variations of rule settings including inconsistent rules, contradicting defaults (a version of the Nixon diamond setting) and an extended assignment problem in combination with multiple mutually related defaults.

1. LUBM-inconsistent .

For an inconsistent setting we have built a DL-program over a *DL-Lite*$_\mathcal{A}$ version together with a fixed, generated ABox (1 university with over 532 students, 61 courses and 29 teaching assistants).

[20] `http://swat.cse.lehigh.edu/projects/lubm/`

[21] `http://code.google.com/p/combo-obda/`

p	AS	RAS				
		no_restr	$lim = 3$	$lim = 20$	$limp = 2$	$limc = 20$
2 (20)	83.30 (0)[11]	3.50 (0)[20]	4.91 (0)[20]	5.07 (0)[20]	4.26 (0)[20]	4.20 (0)[20]
4 (20)	70.16 (0) [6]	3.48 (0)[20]	4.94 (0)[20]	5.08 (0)[20]	4.25 (0)[20]	4.20 (0)[20]
6 (20)	53.73 (0) [0]	3.48 (0)[20]	4.86 (0)[18]	5.17 (0)[20]	4.29 (0)[20]	4.20 (0)[20]
8 (20)	56.63 (0) [1]	3.49 (0)[20]	4.78 (0)[15]	5.01 (0)[20]	4.27 (0)[20]	4.21 (0)[20]
10 (20)	53.56 (0) [0]	3.48 (0)[20]	4.82 (0)[17]	5.08 (0)[20]	4.25 (0)[20]	4.19 (0)[20]
20 (20)	54.07 (0) [0]	3.49 (0)[20]	4.43 (0) [2]	4.98 (0)[20]	4.28 (0)[20]	4.20 (0)[20]
30 (20)	53.79 (0) [0]	3.49 (0)[20]	4.38 (0) [0]	4.93 (0)[20]	4.27 (0)[20]	4.02 (0)[13]
40 (20)	54.47 (0) [0]	3.50 (0)[20]	4.40 (0) [0]	4.95 (0)[20]	4.28 (0)[20]	3.76 (0) [3]
50 (20)	54.88 (0) [0]	3.51 (0)[20]	4.42 (0) [0]	4.98 (0)[20]	4.30 (0)[20]	3.72 (0) [1]
60 (20)	53.97 (0) [0]	3.51 (0)[20]	4.40 (0) [0]	4.83 (0)[15]	4.28 (0)[20]	3.71 (0) [1]
70 (20)	54.22 (0) [0]	3.53 (0)[20]	4.46 (0) [0]	4.69 (0)[10]	4.32 (0)[20]	3.69 (0) [0]
80 (20)	54.55 (0) [0]	3.52 (0)[20]	4.47 (0) [0]	4.48 (0) [2]	4.29 (0)[20]	3.68 (0) [0]
90 (20)	53.72 (0) [0]	3.50 (0)[20]	4.41 (0) [0]	4.43 (0) [0]	4.30 (0)[20]	3.74 (0) [0]
100 (20)	53.95 (0) [0]	3.52 (0)[20]	4.42 (0) [0]	4.43 (0) [0]	4.30 (0)[20]	3.71 (0) [0]

Table 7.14: LUBM-inconsistent benchmark results

$$\mathcal{P}=\left\{\begin{array}{l}(1)\ stud(X) \leftarrow not\ \mathrm{DL}[;\neg Student](X), \mathrm{DL}[;\ TA](X);\\ (2)\ \bot \leftarrow \mathrm{DL}[Student \uplus stud;\ TAof](X,\ Y), takesexam(X,\ Y)\end{array}\right\}$$

Rules encode default reasoning under constraints: teaching assistants (TAs) are normally students, and TAs must not take the exam of a course they are involved with. Facts $takesExam(c_1, c_2)$ have been randomly added to the program with a probability between 0.1 and 1.

The results for this benchmark are presented in Table 7.14. Apart from standard answer set computation we considered RAS computation under restricted settings, where only 3 and 20 facts (facts containing 20 constants) were allowed for deletion in $lim = 3$ and $lim = 20$ (resp. $limc = 20$), and 2 predicates ($limp = 2$). A first repair answer set for the non-restricted case is found for all instances within 4 seconds, while detecting inconsistency with ordinary answer set computation takes about 1 minute. One can see that some of the smaller instances are consistent (p ranging from 2 to 6 and $p = 10$). The computation of the answer sets in general in many scenarios is more time-consuming then identification of the conflicts for inconsistent instances. Therefore, we can observe a drop of the running time in the standard answer set case with the growth of the number of inconsistent instances. In fact, for the same there is a decrease in running times for $lim = 3$, $lim = 20$ and $limc = 20$. One can observe that with the growing p there are less and less repair answer sets found in these columns. The absence of repairs in our implementation amounts to the absence of answer sets in the declarative program extended by the support set information and ontology data part as described earlier in this chapter. This explains the decrease in running times for bigger p.

2. LUBM-diamond. We moreover considered default reasoning over the LUBM ontology. The defaults expressed in the DL-program below state that research assistants

$$\mathcal{O} = \{\ (1)\ RA \sqsubseteq Student\ \}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (2)\ emp(X) \leftarrow not\ \mathrm{DL}[Employee \uplus emp, Employee \uplus negemp;\ \neg Employee](X), \\ \qquad\qquad \mathrm{DL}[;\ RA](X); \\ (3)\ negemp(X) \leftarrow not\ \mathrm{DL}[Employee \uplus emp, Employee \uplus negemp;\ Employee](X), \\ \qquad\qquad \mathrm{DL}[;\ Student](X); \\ (4)\ \bot \leftarrow emp(X), negemp(X) \end{array} \right\}$$

Figure 7.9: DL-program from LUBM-diamond benchmark

p	RAS					
	no_restr	$lim = 3$	$lim = 20$	$limp = 2$	$limc = 20$	RA
2 (20)	3.36 (0)[20]	4.23 (0)[18]	4.30 (0)[20]	4.19 (0)[20]	4.19 (0)[20]	3.32 (0)[20]
6 (20)	3.82 (0)[20]	4.97 (0)[7]	5.32 (0)[20]	4.74 (0)[20]	4.67 (0)[20]	3.84 (0)[20]
10 (20)	4.92 (0)[20]	6.34 (0)[0]	6.83 (0)[20]	5.88 (0)[20]	5.83 (0)[20]	4.95 (0)[20]
14 (20)	7.54 (0)[20]	9.34 (0)[0]	9.76 (0)[20]	8.52 (0)[20]	8.48 (0)[20]	7.54 (0)[20]
18 (20)	11.48 (0)[20]	13.76 (0)[0]	14.35 (0)[20]	12.64 (0)[20]	12.49 (0)[19]	11.62 (0)[20]
22 (20)	17.89 (0)[20]	20.96 (0)[0]	21.67 (0)[19]	19.12 (0)[20]	18.95 (0)[19]	18.24 (0)[20]
26 (20)	27.83 (0)[20]	32.05 (0)[0]	32.71 (0)[18]	28.45 (0)[20]	28.34 (0)[16]	28.55 (0)[20]
30 (20)	234.05 (15)[5]	248.37 (16)[0]	248.12 (16)[4]	247.33 (16)[4]	247.54 (16)[2]	234.71 (15)[5]

Table 7.15: LUBM-diamond benchmark results

(RAs) are normally employees, while students are normally not employees; as the ontology entails that research assistants are students, the well-known Nixon diamond is instantiated (see Figure 7.9).

Table 7.15 shows the results of the experiments for an automatically generated ABox, which contains 1 university consisting of 39 RAs and 532 students. Here p is the percentile of the relevant domain (RAs, students, employees entailed by $\mathcal{O}$), which is generated as a set of facts in the program.

Unsurprisingly, the combinatorial nature of the defaults makes the standard answer set computation difficult and the timeouts are obtained even for relatively small instances. Due to the constraint (14) many instances are inconsistent, and it takes the solver a vast amount of time to identify the conflict, and there is no hope to obtain a result within the timeout. Therefore, we did not include the standard AS running times on these instances. On the other hand, due to the declarative implementation the repair ABox is found relatively quickly for smaller p, and the results for this setting are promising.

3. LUBM-Extended. Finally, we looked at an extended setting of the rules over the LUBM ontology, in which an assignment problem is encoded using a number of mutually related defaults (see Figure 7.10).

- The rules (7) - (9) encode a default, stating that all students are research assistants unless the contrary is derived.

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Student} \sqsubseteq \neg \mathit{NonStudent} & (4)\ \mathit{VisitPostDoc} \sqsubseteq \mathit{PostDoc} \\ (2)\ \mathit{VisitPostDoc} \sqsubseteq \mathit{ResearchAssistant} & (5)\ \mathit{Student} \sqsubseteq \mathit{OrgHelp} \\ (3)\ \mathit{VisitPostDoc} \sqsubseteq \mathit{NonStudent} & \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (6)\ \mathit{resas}(X) \leftarrow \mathrm{DL}[;\ \mathit{ResearchAssistant}](X); \\ (7)\ \mathit{student}(X) \leftarrow \mathit{resas}(X), \mathit{not}\ \mathit{negStudent}(X); \\ (8)\ \mathit{negStudent}(X) \leftarrow\ \mathit{resas}(X), \mathrm{DL}[\mathit{Student} \uplus \mathit{student};\ \neg \mathit{Student}](X); \\ (9)\ \mathit{helps}(Y, X) \leftarrow \mathit{resas}(X), \mathrm{DL}[;\ \mathit{PostDoc}](Y), \mathit{not}\ \mathit{omit}(Y, X); \\ (10)\ \mathit{omit}(Y, X) \leftarrow \mathit{helps}(Y, X), \mathrm{DL}[;\ \mathit{PostDoc}](X); \\ (11)\ \mathit{visitPostDoc}(Z) \leftarrow \mathrm{DL}[;\ \mathit{VisitPostDoc}](Z); \\ (12)\ \mathit{orghelp}(Z1) \leftarrow \mathrm{DL}[;\ \mathit{OrgHelp}](Z1); \\ (13)\ \mathit{supports}(Z1, Z) \leftarrow \mathit{orghelp}(Z1), \mathit{visitPostDoc}(Z), \mathit{not}\ \mathit{drop}(Z1, Z); \\ (14)\ \mathit{negStudent}(Z1) \leftarrow \mathit{orghelp}(Z1), \mathit{not}\ \mathit{student}(Z1); \\ (15)\ \mathit{student}(Z1) \leftarrow \mathit{orghelp}(Z1), \mathrm{DL}[\mathit{NonStudent} \uplus \mathit{negStudent};\ \neg \mathit{NonStudent}](Z1); \\ (16)\ \mathit{drop}(Z1, Z) \leftarrow \mathit{supports}(Z1, Z), \mathrm{DL}[;\ \mathit{InternationalStudent}](Z1). \end{array} \right\}$$

Figure 7.10: DL-program from LUBM-extended benchmark

p	AS	RAS				
		RAS	$lim = 20$	$limp = 2$	$limc = 20$	IS
2 (20)	3.97 (0)[0]	13.98 (0)[20]	38.90 (0)[20]	16.01 (0)[20]	15.24 (0)[20]	15.20 (0)[6]
6 (20)	4.25 (0)[0]	16.16 (0)[20]	115.62 (0)[19]	18.08 (0)[20]	18.63 (0)[19]	11.16 (0)[2]
10 (20)	4.64 (0)[0]	18.95 (0)[20]	245.40 (0)[7]	20.85 (0)[20]	20.79 (0)[4]	9.12 (0)[0]
14 (20)	4.86 (0)[0]	21.50 (0)[20]	236.40 (1)[3]	23.73 (0)[20]	23.50 (0)[1]	9.53 (0)[0]
18 (20)	5.33 (0)[0]	24.86 (0)[20]	230.21 (0)[1]	27.11 (0)[20]	26.86 (0)[0]	10.15 (0)[0]
22 (20)	5.54 (0)[0]	28.21 (0)[20]	228.12 (0)[0]	30.19 (0)[20]	29.93 (0)[0]	10.36 (0)[0]
26 (20)	5.71 (0)[0]	31.50 (0)[20]	222.78 (0)[0]	33.84 (0)[20]	33.26 (0)[0]	10.75 (0)[0]
30 (20)	6.07 (0)[0]	36.88 (0)[20]	225.18 (0)[0]	38.82 (0)[20]	38.47 (0)[0]	11.45 (0)[0]
34 (20)	6.36 (0)[0]	42.18 (0)[20]	241.30 (0)[0]	44.29 (0)[20]	44.01 (0)[0]	12.22 (0)[0]
38 (20)	6.55 (0)[0]	46.07 (0)[20]	245.77 (0)[0]	47.87 (0)[20]	47.64 (0)[0]	12.41 (0)[0]
42 (20)	6.93 (0)[0]	52.50 (0)[20]	255.74 (0)[0]	54.17 (0)[20]	56.91 (0)[0]	12.94 (0)[0]
46 (20)	7.15 (0)[0]	56.98 (0)[20]	276.52 (5)[0]	58.96 (0)[20]	58.47 (0)[0]	13.35 (0)[0]
50 (20)	7.53 (0)[0]	63.96 (0)[20]	276.07 (5)[0]	65.79 (0)[20]	65.50 (0)[0]	14.18 (0)[0]

Table 7.16: LUBM-extended benchmark results

- The rule (10) assigns a postdoc to every research assistant (who is a student by default). In case the "supposed" student has problems, there is always a person to contact, namely assigned postdoc. These assignments are collected into a *helps* predicate. It might, however, happen that a research assistant is a visiting postdoc and thus a postdoc (axiom (2), (4) in $\mathcal{O}$). In this case no help from another postdoc is needed (rule (11)).

- Visiting postdocs do not need help with their work-related problems, but they need support from the linguistic perspective, since they are foreigners, and often do not know the local language. Thus a person who can provide organizational help needs to be found for each postdoc. The rule (12) collects all visiting postdocs

into a respective predicate. The rule (13) collects persons capable of providing organizational help into another corresponding predicate. The rule (14) assigns such a person to a visiting postdoc using the *supports* predicate.

- One needs to keep in mind that not all people who can provide organizational help are equally good for assignments in rule (14). The latter can be international students, even though they are believed to be not students (stated in rules (15) - (16)).
- In (17) we eliminate all pairs, where a student, being involved in organizational help is assigned to a visiting postdoc, and a student turns out himself to be international.

The results of the experiments are given in Table 7.16. In contrast to the Nixon Diamond setting, the standard answer set computation algorithm outperforms the repair answer set computation. The inconsistency in this setting is identified quicker then the first repair is found. There are many DL-atoms without input predicates, so called *outer* DL-atoms. In the standard answer set setting, for these atoms all relevant constants are retrieved at an early stage, which speeds up the computation process. The restricted repairs are computed in this setting too, and the results are non-surprising neither with respect to the number of found repair answer sets nor with regard to the running times. The stricter the limit, the less repairs are found and the more time is needed. The last column of Table 7.16 presents the restriction of removal of the $InternationalStudents$ only. As one can see this guided search speeds up the computation but due to being too restrictive it significantly reduces the number of found repairs. Note that limiting the number of removed facts to 20 is slower then the rest of the posed restrictions, which limit the number of predicates and constants allowed for removal. The reason for this behavior stems from the structure of repairs, they do not involve many different predicates, the number of facts in all repairs exceeds 20.

7.2.5 DL-programs over $\mathcal{EL}$ Ontologies

We now describe the benchmarks based on DL-programs built over acyclic OWL 2 EL ontologies, which were used to analyze the algorithms based on incomplete support families.

Access Policy Control

The first benchmark is a slight modification of Example 5.7 with an additional TBox axiom $Blacklisted \sqsubseteq Unauthorized$. We have run experiments in two settings: (a) with complete support families and (b) with support families obtained under different restrictions, viz. bounded size and cardinality. We considered three ABoxes with 40, 100 and 1000 staff members, respectively, and generated facts of the form $hasowner(p_i, s_i)$, and such that $Staff(s_i), Project(p_i) \in \mathcal{A}$. For the setting where complete support families were computed, we used ABoxes with 100 and 1000 staff members, respectively.

p	Complete supp. family		p	Sup. set size restricted			Sup. set number restricted		
	$\mathcal{A}_{100}$	$\mathcal{A}_{1000}$		sizelim=1	sizelim=3	sizelim=∞	numlim=1	numlim=3	numlim=∞
10 (20)	2.24 (0)[20]	10.42 (0)[20]		$\mathcal{A} = 40$					
20 (20)	2.22 (0)[20]	10.25 (0)[20]	5 (20)	1.84 (0)[20]	1.83 (0)[20]	1.84 (0)[20]	61.25 (0)[0]	1.83 (0)[20]	1.84 (0)[20]
30 (20)	2.23 (0)[20]	10.23 (0)[20]	10 (20)	1.83 (0)[20]	1.83 (0)[20]	1.83 (0)[20]	61.98 (0)[0]	1.83 (0)[20]	1.83 (0)[20]
40 (20)	2.23 (0)[20]	10.25 (0)[20]	15 (20)	1.82 (0)[20]	1.82 (0)[20]	1.83 (0)[20]	61.71 (0)[0]	1.83 (0)[20]	1.82 (0)[20]
50 (20)	2.23 (0)[20]	10.27 (0)[20]	20 (20)	1.82 (0)[20]	1.82 (0)[20]	1.83 (0)[20]	62.60 (0)[0]	1.82 (0)[20]	1.83 (0)[20]
60 (20)	2.23 (0)[20]	10.27 (0)[20]	25 (20)	1.82 (0)[20]	1.81 (0)[20]	1.82 (0)[20]	63.12 (0)[0]	1.82 (0)[20]	1.82 (0)[20]
70 (20)	2.22 (0)[20]	10.26 (0)[20]	30 (20)	1.82 (0)[20]	1.82 (0)[20]	1.82 (0)[20]	63.19 (0)[0]	1.82 (0)[20]	1.83 (0)[20]
80 (20)	2.21 (0)[20]	10.29 (0)[20]	35 (20)	1.81 (0)[20]	1.82 (0)[20]	1.82 (0)[20]	63.68 (0)[0]	1.82 (0)[20]	1.82 (0)[20]
90 (20)	2.19 (0)[20]	10.45 (0)[20]							

Table 7.17: Policy benchmark results

p	Compl. s/f	Support set size restricted			Support set number restricted		
		sizelim=1	sizelim=2	sizelim=∞	numlim=1	numlim=2	numlim=∞
10 (20)	9.70 (0)[20]	10.18 (0)[11]	9.78 (0)[20]	9.64 (0)[20]	9.95 (0)[20]	9.70 (0)[20]	9.76 (0)[20]
20 (20)	9.43 (0)[20]	10.17 (0)[5]	9.40 (0)[20]	9.37 (0)[20]	9.72 (0)[20]	9.51 (0)[20]	9.45 (0)[20]
30 (20)	9.37 (0)[20]	10.78 (0)[0]	9.39 (0)[20]	9.39 (0)[20]	9.81 (0)[20]	9.43 (0)[20]	9.33 (0)[20]
40 (20)	9.34 (0)[20]	11.37 (0)[1]	9.33 (0)[20]	9.38 (0)[20]	10.27 (0)[20]	9.46 (0)[20]	9.35 (0)[20]
50 (20)	9.39 (0)[20]	14.01 (0)[0]	9.37 (0)[20]	9.37 (0)[20]	11.78 (0)[20]	9.59 (0)[20]	9.35 (0)[20]

Table 7.18: OpenStreetMap benchmark results

For the incomplete scenario, we used an ABox with 40 staff members. In each data set, 32% of staff members are unauthorized and 40% are blacklisted. Instances vary on facts $\mathit{hasowner}(p_i, s_i)$. For each s_i, p_i such that $\mathit{Staff}(s_i), \mathit{Project}(p_i) \in \mathcal{A}$, a fact $\mathit{hasowner}(p_i, s_i)$ is added to the program with probability $p/100$, where p ranges from 20 to 90 for the complete setting and from 5 to 35 for the incomplete one.

The total average running times (including support set computation and timeouts) for computing the first repair answer set for these settings are shown in Table 7.17. The columns for the incomplete case show the restriction on support sets we used in their generation, viz. size (resp. number) of support sets bounded by 1, 3 resp. unlimited; the latter means that in fact all support sets were computed, but the system is not aware of the completeness.

We exploit partial completeness for the number and size restriction case, i.e. if no more support sets for an atom are computed and the number/size limit were not yet reached, then the support family for the considered atom is complete. One can observe that the running times are almost constant for this setting, except for the column where the number of support sets is limited to 1. In the latter case there are no answer sets found and the solver needs about one minute to figure this out. The reason for such output is the number and size of support sets coherent for an interpretation at hand. Most of the support sets fit the limits specified. The structure of repairs on the nonground level for all instances is the same. Therefore the repair on the nonground level is also identified quickly, the solver realizes that the facts with a certain predicate need to be removed, and then the size of the instance does not have much impact on the running time.

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{BuildingFeature} \sqcap \exists \mathit{isLocatedInside}.\mathit{Private} \sqsubseteq \mathit{NoPublicAccess} \\ (2)\ \mathit{BusStop} \sqcap \mathit{Roofed} \sqsubseteq \mathit{CoveredBusStop} \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (9)\ \mathit{publicstation}(X) \leftarrow \mathrm{DL}[\mathit{BusStop} \uplus \mathit{busstop};\ \mathit{CoveredBusStop}](X), \\ \qquad\qquad\qquad\qquad\qquad \mathit{not}\ \mathrm{DL}[;\ \mathit{Private}](X); \\ (10)\ \bot \leftarrow \mathrm{DL}[\mathit{BuildingFeature} \uplus \mathit{publicstation};\ \mathit{NoPublicAccess}](X), \\ \qquad\qquad\quad \mathit{publicstation}(X). \end{array} \right\}$$

Figure 7.11: DL-program over OpenStreetMap ontology

Open Street Map

For the second benchmark, we added rules on top of the ontology developed in the MyITS project,[22] which enhanced personalized route planning with semantic information. The ontology contains 4601 axioms, where 406 axioms are in the TBox and 4195 are in the ABox. The fragment $\mathcal{O}$ relevant for our scenario and the rules P are shown in Figure 7.11. Intuitively, $\mathcal{O}$ states that building features located inside private areas are not publicly accessible and a covered bus stop is a bus stop with a roof. The rules P check that public stations do not lack public access, using CWA on private areas.

We used the method in [ESSXar] to extract data from the OpenStreetMap,[23] and we constructed an ABox $\mathcal{A}$ by extracting the sets of all bus stops (285) and leisure areas (682) of the Irish city Cork, as well as *isLocatedInside* relations between them (9) (i.e., bus stops located in leisure areas). As the data has been gathered by many volunteers, chances of inaccuracies may be high (e.g. imprecise gps data). As the data about roofed bus stops and private areas is unavailable yet, we randomly made 80% of the bus stops roofed and 60% of leisure areas private. Finally, we added for each bs_i such that $\mathit{isLocatedInside}(bs_i, la_j) \in \mathcal{A}$ the fact $\mathit{busstop}(bs_i)$ to $\mathcal{P}$ with probability $p/100$. Some instances are inconsistent since in our data set there are roofed bus stops, located inside private areas.

The results for both complete and incomplete support families are shown in Table 7.18. Similar as in the previous benchmark, the running times do not vary much in most of the settings. The exceptional column is column 3, which presents running times for the case when the support set size is restricted to 1. The results witness a slight running times increase, which is probably due to the evaluation postchecks that need to be performed for some DL-atoms, having support sets of size larger then 1. Note that here the support families are also not known to be complete, hence the computation is not guaranteed to be complete either. For that reason one can see that the number of repairs specified in column 3 differs from the number of repairs in other less restrictive configurations.

[22] http://www.kr.tuwien.ac.at/research/projects/myits/
[23] http://www.openstreetmap.org/

p	Compl. s/f	Support set size restricted			Support set number restricted		
		sizelim=1	sizelim=2	sizelim=∞	numlim=1	numlim=2	numlim=∞
5 (20)	7.50 (0)[20]	7.79 (0)[20]	7.58 (0)[20]	7.72 (0)[20]	8.55 (0)[0]	8.56 (0)[0]	7.67 (0)[20]
15 (20)	7.34 (0)[20]	7.55 (0)[20]	7.46 (0)[20]	7.54 (0)[20]	16.74 (0)[0]	16.58 (0)[0]	7.42 (0)[20]
25 (20)	7.23 (0)[20]	7.22 (0)[20]	7.27 (0)[20]	7.25 (0)[20]	82.52 (1)[0]	82.57 (1)[0]	7.13 (0)[20]
35 (20)	7.11 (0)[20]	7.12 (0)[20]	7.11 (0)[20]	7.11 (0)[20]	184.96 (8)[0]	185.14 (8)[0]	7.08 (0)[20]
45 (20)	7.07 (0)[20]	7.07 (0)[20]	7.07 (0)[20]	7.08 (0)[20]	282.24 (18)[0]	282.24 (18)[0]	7.08 (0)[20]
55 (20)	7.08 (0)[20]	7.09 (0)[20]	7.08 (0)[20]	7.08 (0)[20]	291.10 (19)[0]	290.92 (19)[0]	7.07 (0)[20]
65 (20)	7.06 (0)[20]	7.07 (0)[20]	7.07 (0)[20]	7.06 (0)[20]	300.00 (20)[0]	300.00 (20)[0]	7.08 (0)[20]
75 (20)	7.09 (0)[20]	7.09 (0)[20]	7.08 (0)[20]	7.08 (0)[20]	300.00 (20)[0]	300.00 (20)[0]	7.08 (0)[20]
85 (20)	7.07 (0)[20]	7.09 (0)[20]	7.07 (0)[20]	7.08 (0)[20]	300.00 (20)[0]	300.00 (20)[0]	7.08 (0)[20]
95 (20)	7.08 (0)[20]	7.08 (0)[20]	7.06 (0)[20]	7.08 (0)[20]	300.00 (20)[0]	300.00 (20)[0]	7.07 (0)[20]

Table 7.19: LUBM-$\mathcal{EL}$ benchmark results

LUBM-$\mathcal{EL}$

We have also tested our approach on a slightly extended $\mathcal{EL}$ version of the LUBM-inconsistent benchmark, in which we have introduced the following new TBox axioms:

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{PhDStudent} \sqcap \exists \mathit{assists}.\mathit{Lecturer} \sqsubseteq \mathit{TA} \\ (2)\ \mathit{GraduateStudent} \sqcap \exists \mathit{teaches}.\mathit{UndergraduateStudent} \sqsubseteq \mathit{TA} \end{array} \right\}$$

The running times and computed repairs for this benchmark are provided in Table 7.19. One can see that the repairs are identified for all instances in the support set size limiting cases, but the restriction of the number of support sets to 1 and 2 turned out to be too strong. As expected for the complete setting the repairs for all instances are found quickly. One might wonder, why there is a slight drop in the running time observed for the instances where the repairs were in the end identified.

In fact the reason for such an unexpected behavior is hidden in the instance generation properties for this benchmark. With the growing instance size, the programs become more and more inconsistent as the conflicting data grows. On the other hand, the growing data part introduces more and more possibilities how the instance can be repaired in the end, which explains the decrease of the running times.

7.2.6 General Results Discussion

We have presented the results of thorough evaluation of our novel algorithms for repair answer set computation of DL-programs over *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ ontologies. We have compared these results to standard answer set computation, as no comparable system for inconsistency handling in DL-programs exists. The DReW system can evaluate DL-programs over $\mathcal{EL}$ ontologies after transforming the input to datalog, where DL-atoms are replaced by datalog rewritings; the latter amount to succinct representations of support sets. However, DReW can not handle inconsistencies and how to inject repairs efficiently is non-obvious (naive attempts fail).

Based on our experimental results the following major conclusions can be drawn:

Scalability of algorithms. Our repair computation algorithms demonstrated the capability of repairing DL-programs of various size within reasonable time limits. The DL-programs with different expressive features including non-determinism, recursiveness and defaults can be effectively handled by our repair approach. In some settings even detecting inconsistency with a standard solver takes longer then the repair computation (e.g. Table 7.14). In other settings (e.g. Table 7.10), despite a certain overhead for computation of deletion repair answer sets compared to ordinary answer sets, the developed algorithms still scale well in general.

Repair quality. The improvement of the repair quality numerically by bounding the size of the ABox portion allowed for removal, naturally increases the computation time. However, exploitation of additional domain knowledge about the possible conflict structure by forcing the solver to consider only certain types of facts, leads in many settings to performance improvements compared to unrestricted repair search. We believe that for larger programs with conflicts of specific kinds, the user guidance of repair computation can drastically improve running times even further.

Partial vs complete support families. For the DL-programs over *DL-Lite*$_{\mathcal{A}}$ ontologies the complete support family exploitation shows promising results. For the DL-programs over $\mathcal{EL}$ ontologies, as expected using complete support families works for all settings well in practice. Despite possible post evaluation of support sets in case of incomplete support families the repairs are still found in many scenarios. As practice shows the available ontologies do not contain complicated constraints on the conceptual level, and either support sets size or number of support sets for DL-atoms often turn out to be limited.

Applicability. The novel algorithms for deletion repair answer set computation demonstrated their applicability for DL-programs over some real world data (Open Street Map benchmark results in Table 7.18). While most of the other benchmarks that we have run are synthetic, they vary widely w.r.t. the size of their TBox, ABox and the number of rules. The capability of our algorithms of handling such diverse DL-programs confirms the potential of our approach for real world scenarios.

Overall the obtained results are promising, and they distinctively show that the repair approaches that we have proposed make contribution to the applicability of DL-programs for practical settings.

CHAPTER 8

Conclusion

This chapter concludes the thesis with a summary of the results that we obtained (Section 8.1), and reviews directions for future research in the area (Section 8.2).

8.1 Summary

The main goal of this thesis was the development of a methodology for handling inconsistencies in hybrid KBs, that combine nonmonotonic rules and DL ontologies. A prominent example of loosely-coupled hybrid KBs are DL-programs, in which an ontology and rules are separate, but a sophisticated information flow between them is arranged through so-called DL-atoms that are allowed to occur in the bodies of the rules. On the one hand the interaction between components of DL-programs makes them highly powerful systems. On the other hand such data exchange can easily give rise to conflicts, making the overall system inconsistent, which is undesired. In this thesis we tackled the inconsistency issue in DL-programs. From a theoretical perspective we have proposed a repair methodology and analyzed the complexity of repairing DL-programs. From a practical perspective we have realized the developed algorithms for repairing DL-programs and experimentally confirmed their effectiveness.

At the beginning of this work there was no clear definition of repair for DL-programs. Therefore, first of all we have analyzed possible sources and reasons for inconsistencies, and formalized the problem of repairing DL-programs by introducing the notions of *repair* and *repair answer set*. We assumed that the rule part of the DL-program and the ontology TBox are well-developed (as it indeed often happens), and the reasons for inconsistencies lie in the ontology ABox. The novel notions of repair and repair answer set are thus based on changes of the ontology data part that enable answer sets. We have performed a careful complexity analysis, which revealed that repair answer sets do not have higher complexity than ordinary ones (more specifically, weak and FLP answer sets) in case if queries in DL-atoms can be evaluated in polynomial time. To ensure this

property, we concentrated on the Description Logics *DL-Lite*$_\mathcal{A}$ (which is of particular use for ontology based data access (OBDA)) and $\mathcal{EL}$. In principle, however, our general methodology for restoring consistency can be applied to DL-programs over ontologies in any DL, though for expressive ones its effectiveness is not any longer guaranteed.

Our refined notions incorporate criteria to select preferred repairs, with a focus on properties that allow to extend the existing evaluation method for DL-programs. We call such benign functions that select preferred repairs *independent*. At the heart of our method there is the generalized *Ontology Repair Problem (ORP)*. Roughly speaking, it asks for an ABox repair for which simultaneous entailment and non-entailment of sets of queries under possible temporary updates are ensured. While being intractable in general, this problem can be polynomially solved in relevant and non-trivial settings. To ensure practicality, we focused on deletion repairs, i.e. the set of possible solutions to ORP was restricted to subsets of the original ABox. Subsequently for repairing a given DL-program only removal of data assertions from its ontology part was allowed.

Since the naive implementation of the deletion repair answer set computation turned out to be ineffective, we proposed its optimized variants for DL-programs accessing *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}$ ontologies. The algorithms are based on the novel notion of *support sets*. Intuitively, a support set for a ground DL-atom is a portion of its input whose presence settles the DL-atom's concrete value. For DL-programs over *DL-Lite*$_\mathcal{A}$ ontologies, so-called *complete support families* were exploited; these allow to completely avoid ontology access during the repair computation. For DL-programs over $\mathcal{EL}$ DL complete support families are too expensive to compute, since their number and size could be possibly exponential. We thus exploited only selected support sets (e.g. of bounded size), forming *partial support families*. The latters form the basis for the repair algorithm in the $\mathcal{EL}$ DL case.

Besides optimizations of the actual repair computation, we have briefly studied the problem of simplifying the rules of the DL-program prior to repairing them. In particular, we investigated the forms of *tautologic* and *contradictory* ground DL-atoms. These have the same value regardless of the interpretation and the ontology at hand. Such *independent* DL-atoms resp. rules involving them can be eliminated prior to the repair or evaluation of DL-programs. We showed that contradictory DL-atoms have a simple form, and we presented a sound and complete calculus for determining tautologic DL-atoms. Based on it, we have determined the complexity of identifying independent DL-atoms; it turned out that this problem can be solved very efficiently in general, as well as relative to the predicate constraints.

Finally, we have implemented the developed algorithms using declarative means in the dlvhex system, which is a tool for evaluating answer set programs with arbitrary external computations (so-called HEX-programs). In our implementation the user can conveniently specify the repair search space by allowing for deletion of only certain types/number of facts from the ABox. We have, furthermore, built a set of benchmarks for experimental assessment of our repair approach. The evaluation of the developed algorithms on these benchmarks showed promising results. In particular, for inconsistent DL-programs we have observed that the repair answer set computation is often quicker

then the standard answer set computation. Moreover, user-guided restrictions put on the repairs search do not introduce much of the time overhead.

Overall, our empirical evaluation has revealed a great potential of the novel repair methodology for practical applications.

8.2 Outlook

The directions for future work in the considered area are manifold. They cover both theoretical and practical aspects of our inconsistency handling approaches.

Real-world settings. The repair framework for DL-programs that we have proposed can be effectively used to improve robustness of hybrid KBs. However, the developed methods have not yet been tested on the real world applications. Therefore, the search for suitable practical applications that go beyond syntactic benchmarks is an obvious future research direction. Such instances can hint on possible optimizations of our novel algorithms as well as their appropriate refinement towards practical settings.

Optimizations. In our algorithms we based the repair computation on the answer sets of the replacement program, where DL-atoms were replaced by normal atoms and additional guessing rules on their values were added. We tried to turn the answer set of the replacement program into a repair answer set by ABox changes. To optimize this approach one can aim at building repair answer sets incrementally e.g., by exploiting debugging based on stepping techniques [OPT12].

In a user-interactive manner one can go through the rules of a DL-program until the point when a conflict is identified. If the conflict occurred due to a value of a DL-atom, one can repair the ontology ABox, and continue the stepping process further. Intelligent backtracking will be needed here. Incremental repair answer set construction is not guaranteed for general program classes, but for example, for stratified DL-programs it might be of interest. Advanced learning techniques can be used for repair computation, e.g. cashing intermediate repairs/repair answer sets, considering correlation patters between them, and identifying mutual dependence of values of DL-atoms might be worthwhile. Localization and decomposition methods from databases can be exploited [EFGL07].

Module analysis. We considered the DL-programs as monolithic structures when applying our repair techniques. It is an interesting and a relevant quest to extend the approach for dealing with modular DL-programs. Splitting a program into separate components that can be individually evaluated is a well-known programming technique, which has fortunately been studied in the context of DL-programs [EFI$^+$11]. It is not clear, however, to which extent and for which program classes the repair methods can be adapted for the modular setting. This challenging issue forms another possible future research direction.

Other DLs and types of queries. In this thesis we focused on lightweight DLs. A relevant future work is the extension of our repair algorithms for other more expressive

DLs, e.g. $\mathcal{SHIQ}$, $\mathcal{SHOIN}$ and other DL-queries, e.g. conjunctive queries. This is a non-trivial task, since the complexity of such algorithms will clearly increase and advanced techniques are needed to ensure practicability.

Repairing rules and interfaces. So far we have concentrated only on repairing ontology data part, but it is also natural to allow changes in rules and interfaces. For repairing rules the works on ASP debugging can be used as a starting point, but the problem is complicated as the search space of possible changes is large. Priorities on the rules and atoms involving them might be applied to ensure high quality of rule repairs. The interfaces similarly admit numerous modifications, which makes this type of repair as difficult; user interaction is most probably required.

Going beyond deletion repairs. Developing and implementing algorithms for other types of repairs like bounded addition, etc. are of practical need. One can consider not only selection function with an independent property, but also look at arbitrary selection functions. Computation of repairs that are globally minimal is nontrivial and the complexity increase in comparison to deletion repairs is obvious. Intelligent techniques can still be exploited for determining repairs of these types.

Paraconsistent semantics. In the settings where all pieces of information in a DL-program are equally important, the repair approach might be unwanted. In these scenarios one has to seek for ways of reasoning in an inconsistent system. This is where the paracosistent/paracoherent semantics comes into play. Despite its obvious importance for applications, the problem of paraconsistent reasoning in the context of loosely-coupled hybrid KBs has been barely studied to the best of our knowledge and its exploration still remains.

The existing paracoherent semantics for ASP [EFM10] is based on the notions of here-and-there (HT) interpretations and the equilibrium models. In particular, the semi-equilibrium semantics which characterizes the programs in terms of bi-models has been introduced. The bi-models are three-valued interpretations, where atoms can be true, believed true or false. This paracoherent semantics for normal ASP programs has been recently implemented within the dlvhex system in the scope of the bachelor's final work co-supervised by the author of this thesis [H14].

For the DL-programs, the current semantics can not be straightly extended. Indeed, since normally DL-atoms are logically two-valued, the meaning of a DL-atom being believed to be true is unclear. The possible flow of information between the ontology and the logic program is also an obstacle for a smooth extension of the existing semantics. It is not obvious how to treat the information that is believed to be true and that is sent to the ontology, especially if the latter does not support the paraconsistent mode of reasoning. All these important open issues are left for future work.

Other repair related problems. One can consider brave and cautious reasoning over DL-programs, i.e. the problem of identifying whether a certain atom is entailed

from a single of multiple repair answer sets of a DL-program. Similar semantics as introduced for ontologies in [BBG14] can be considered for DL-programs; however, the rules interaction in the DL-program setting poses an additional challenge.

Furthermore, instead of looking at inconsistent DL-programs, one might investigate repair of consistent DL-programs, having answer sets that are not expected by the user. For instance, if a user is confident that a certain atom a has to be present in any answer set, but this is not observed in reality, he might want to fix this unwanted behavior by some program modifications. These problems are left for future work.

Domain-specific repair. Incorporation of domain specific information into the repair process is an important problem. It might be desired to leave some rules, interfaces or ontology parts of a DL-program untouched. The semantically specific information about irrefutable DL-atoms or ontology pieces can effectively control the repair process. This calls for methods for convenient representation and effective exploitation of such additional knowledge.

Policy for managing inconsistencies. Since there is a wide range of options on how a DL-program could be repaired, a policy language for managing inconsistencies in DL-programs that would guide the repair process in the user-interactive way is beneficial. Development of such a policy is an open research issue.

Other hybrid formalisms. Last but not least one could develop methods for repairing other hybrid formalisms including tight-coupling hybrid KBs or even more general representations like HEX-programs, where instead of ontology arbitrary sources of computation can be accessed from a logic program. Heterogeneity of external sources in HEX-programs makes both repair and paraconsistent reasoning a very challenging but interesting task.

Bibliography

[ABC99] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In Victor Vianu and Christos Papadimitriou *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, USA, pages 68–79. ACM Press, 1999.

[ABC00] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Specifying and querying database repairs using logic programs with exceptions. In Henrik Legind Larsen, Troels Andreasen, Henning Christiansen, Janusz Kacprzyk, and Slawomir Zadrozny, editors, *Proceedings of the Fourth International Conference on Flexible Query Answering Systems (FQAS 2000)*, Poland, pages 27–41. Springer, 2000.

[ABC01] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Scalar aggregation in fd-inconsistent databases. In Jan Van den Bussche and Victor Vianu, editors, *Proceedings of the Eighth International Conference on Database Theory (ICDT 2001)*, UK, pages 39–53. Springer, 2001.

[ABC03] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming*, 3(4-5):393–424, 2003.

[ACKZ09] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36:1–69, 2009.

[ADS83] Giorgio Ausiello, Alessandro D'Atri, and Domenico Saccà. Graph algorithms for functional dependency manipulation. *Journal of the ACM*, 30(4):752–766, 1983.

[AGM85] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[AKWS95] Shailesh Agarwal, Arthur M. Keller, Gio Wiederhold, and Krishna Saraswat. Flexible relation: An approach for integrating data from multiple, possibly inconsistent databases. In Philip S. Yu and Arbee L. P. Chen,

editors, *Proceedings of the Eleventh International Conference on Data Engineering (ICDE 1995)*, Taiwan, pages 495–504. IEEE Computer Society, 1995.

[BB02] Pablo Barceló and Leopoldo E. Bertossi. Repairing databases with annotated predicate logic. In Salem Benferhat and Enrico Giunchiglia, editors, *Proceedings of the Nineth International Workshop on Non-Monotonic Reasoning (NMR 2002)*, France, pages 160–170, 2002

[BB03] Pablo Barceló and Leopoldo E. Bertossi. Logic programs for querying inconsistent databases. In Verónica Dahl and Philip Wadler, editors, *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, volume 2562 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2003.

[BBG14] Camille Bourgaux, Meghyn Bienvenu, and François Goasdoué. Querying inconsistent description logic knowledge bases under preferred repair semantics. In Meghyn Bienvenu, Magdalena Ortiz, Riccardo Rosati, and Mantas Simkus, editors, *Proceedings of the Twenty-seventh International Workshop on Description Logics (DL 2014)*, Austria, volume 1193 of *CEUR Workshop Proceedings*, pages 96–99. CEUR-WS.org, 2014.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Kaelbling and Saffiotti, editors, *Proceedings of the nineteenth Conference Joint Conference on artificial Intelligence (IJCAI 2005)*, UK, pages 364–369, Morgan Kaufmann, 2005.

[BBL08] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope further. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *Proceedings of the Fifth International Workshop OWL: Experiences and Directions, (OWLED 2008)*, Germany, 2008.

[BBMR89] Alexander Borgida, and Ronald J. Brachman, and Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A structural data model for objects. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, USA, pages 58–67, 1989.

[BCM+07] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, second edition, 2007.

[BCRM08] Alexander Borgida, Diego Calvanese, and Mariano Rodriguez-Muro. Explanation in *DL-Lite*. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proceedings of the Twenty-first International Workshop on Description*

Logics (DL 2008), Germany, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[BE07] Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the Twenty-second Conference on Artificial Intelligence (AAAI 2007)*, Canada, pages 385–390. AAAI press, 2007.

[Bec04] Dave Beckett. New syntaxes for RDF. Technical report, Institute For Learning And Research Technology, Bristol, 2004.

[BEFS10] Markus Bögl, Thomas Eiter, Michael Fink, and Peter Schüller. The mcs-ie system for explaining inconsistency in multi-context systems. In Tomi Janhunen and Ilkka Niemelä, editors, *Proceedings of the Twelfth European Conference on Logics in Artificial Intelligence (JELIA 2010)*, Finland, *Lecture Notes in Computer Science*, pages 356–359. Springer, 2010.

[Bel77] Nuel Belnap. A useful four-valued logic. In J. Michael Dunn and George Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 7–37. Reidel Publishing Company, USA, 1977.

[Ber11] Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, Canada, 2011.

[BF00] Tim Berners-Lee and Mark Fischetti. *Weaving the web - the original design and ultimate destiny of the World Wide Web by its inventor*. HarperBusiness, 2000.

[BFS10] Piero A. Bonatti, Marco Faella, and Luigi Sauro. $\mathcal{EL}$ with default attributes and overriding. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *Proceedings of the Nineth International Semantic Web Conference (ISWC 2010)*, China, volume 6496 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2010.

[BG03] Marcello Balduccini and Michael Gelfond. Logic programs with consistency-restoring rules. In Patrick Doherty, John McCarthy, and Mary-Anne Williams, editors, *Proceedings of the Tenth International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, USA, pages 9–18. AAAI Press, 2003.

[BGL85] Ronald J. Brachman, Victoria P. Gilbert, and Hector J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts of KRYPTON. In Aravind K. Joshi, editor, *Proceedings of the Nineth Joint Conference on Artificial Intelligence (IJCAI 1985)*, USA, pages 532–539. Morgan Kaufmann, 1985.

[BH91] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithms. In Harold Boley and Michael M. Richter, editors, *Proceedings of the International Workshop on Processing Declarative Knowledge (PDK 1991)*, Germany, volume 567 of *Lecture Notes in Computer Science*, pages 67–86. Springer, 1991.

[BHS05] Leopoldo E. Bertossi, Anthony Hunter, and Torsten Schaub. Introduction to inconsistency tolerance. In Leopoldo E. Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Proceedings of the International Workshop on Inconsistency Tolerance*, pages 1–14, 2005.

[Bie08] Meghyn Bienvenu. Complexity of abduction in the $\mathcal{EL}$ family of lightweight description logics. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation (KR 2008)*, Australia, pages 220–230. AAAI Press, 2008.

[Bie12] Meghyn Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-sixth Conference on Artificial Intelligence (AAAI 2012)*, Canada, pages 705–711. American Association for Artificial Intelligence, 2012.

[BMPS+91] Ronald Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks— Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann, 1991.

[BR13] Meghyn Bienvenu and Riccardo Rosati. New inconsistency-tolerant semantics for robust ontology-based data access. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Proceedings of the Twenty-sixth International Workshop on Description Logics (DL 2013)*, Germany, volume 1014 of *CEUR Workshop Proceedings*, pages 53–64. CEUR-WS.org, 2013.

[Bre89] Gerhard Brewka. Preferred subtheories: An extended logical framework for default reasoning. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI 1989)*, USA, pages 1043–1048. Morgan Kaufmann, 1989.

[Bry97] François Bry. Query answering in information systems with integrity constraints. In Janice I. DeGross and Kuldeep Kumar, editors, *Proceedings of the First Working Conference on Integrity and Internal Control in Information Systems: Increasing the Confidence in Information Systems (IICIS 1997)*, Switzerland, pages 113–130. Chapman and Hall Ltd, 1997.

[CB00] Alexander Celle and Leopoldo E. Bertossi. Querying inconsistent databases: Algorithms and implementation. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv and Peter J. Stuckey, editors, *Procedings of the First International Conference on Computational Logic (CL 2000)*, UK, volume 1861 of *Lecture Notes in Computer Science*, pages 942–956. Springer, 2000.

[CDR98] Diego Calvanese, Giuseppe De Giacomo, and Riccardo Rosati. A note on encoding inverse roles and functional restrictions in $\mathcal{ALC}$ knowledge bases. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt and Christopher A. Welty, editors, *Proceedings of the 1998 International Workshop on Description Logics (DL 1998)*, Italy, volume 11 of *CEUR Workshop Proceedings*, pages 69–71. CEUR-WS.org, 1998.

[CFR+14] David Carral, Cristina Feier, Ana Armas Romero, Bernardo Cuenca Grau, Pascal Hitzler, and Ian Horrocks. Is your ontology as hard as you think? rewriting ontologies into simpler DLs. In Meghyn Bienvenu, Magdalena Ortiz, Riccardo Rosati, and Mantas Simkus, editors, *Proceedings of the Twenty-seventh International Workshop on Description Logics (DL 2014)*, Austria, volume 1193 of *CEUR Workshop Proceedings*, pages 128–140. CEUR-WS.org, 2014.

[CGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth Symposium on Principles of Database Systems (SPODS 1998)*, USA, pages 149–158. ACM press, 1998.

[CGP12] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Journal of Artificial Intelligence*, 193:87–128, 2012.

[CHPS09] Jeremy Carroll, Ivan Herman, and Peter F. Patel-Schneider. OWL 2 web ontology language rdf-based semantics, 2009. `http://www.w3.org/TR/2009/WD-owl2-rdf-based-semantics-20090421/`.

[CKC81] Alain Colmerauer, Henry Kanoui, and Michel Van Caneghem. Last steps towards an ultimate PROLOG. In Patrick J. Hayes, editors, *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI 1981)*, Canada, pages 947–948. William Kaufmann, 1981.

[CLLR07] Diego Calvanese, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[CM05] Jan Chomicki and Jerzy Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In Leopoldo E. Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Proceedings of the Inconsistency Tolerance*, pages 119–150. Springer, 2005.

[CMR+14] Marco Console, José Mora, Riccardo Rosati, Valerio Santarelli, and Domenico Fabio Savo. Effective computation of maximal sound approximations of description logic ontologies. In *Proceedings of the thirteenth International Semantic Web Conference - ISWC 2014*, Italy, 2014. Proceedings, Part II, volume 8797 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2014.

[COSS12a] Diego Calvanese, Magdalena Ortiz, Mantas Simkus, and Giorgio Stefanoni. The complexity of explaining negative query answers in DL-Lite. In *Proceedings Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, Italy, pages 583–587, 2012.

[dBBC+09] Jos de Bruijn, Philippe Bonnard, Hugues Citeau, Sylvain Dehors, Stijn Heymans, Jörg Pührer, and Thomas Eiter. Combinations of rules and ontologies: State-of-the-art survey of issues. Technical Report Ontorule D3.1, Ontorule Project Consortium, 2009. `http://ontorule-project.eu/`.

[DBV12] Marc Denecker, Maurice Bruynooghe, Joost Vennekens Approximation fixpoint theory and the semantics of logic and answers set programs. In *Proceedings of Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, pages 178–194.

[dCH15] Newton C. A. da Costa and Federico Holik. A formal framework for the study of the notion of undefined particle number in quantum mechanics. *Synthese Journal*, 192(2):505–523, 2015.

[DEFK09] Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner. Modular nonmonotonic logic programming revisited. In *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP 2009)*, volume 5649 in Lecture Notes in Computer Science, pages 145–159. Springer, 2009.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *Journal of ACM Computing Surveys*, 33(3):374–425, 2001.

[DLLR04] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tackling inconsistencies in data integration through source preferences. In *Proceedings of the International Workshop on Information Quality in Information Systems (IQIS 2004)*, France, pages 27–34. ACM, 2004.

[DTEK09] Minh Dao-Tran, Thomas Eiter, and Thomas Krennwallner. Realizing default logic over description logic knowledge bases. In *Proceedings of the Tenth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009)*, volume 5590 of *Lecture Notes in Computer Science*, pages 602–613. Springer, 2009.

[EEFS05] Thomas Eiter, Esra Erdem, Michael Fink, and Ján Senko. Updating action domain descriptions. *Journal of Artificial Intelligence*, 174(15):1172–1221, 2010.

[EFFW07] Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran. Complexity Results for Answer Set Programming with Bounded Predicate Arities. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):123–165, 2007.

[EFGL07] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Repair Localization for Query Answering from Inconsistent Databases. *ACM Transactions on Database Systems*, 33(2), 2007.

[EFI+11] Thomas Eiter, Michael Fink, Giovambattista Ianni, Thomas Krennwallner, and Peter Schüller. Pushing Efficient Evaluation of HEX Programs by Modular Decomposition. In *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, volume 6645 of *LNCS*, pages 93–106. Springer, 2011.

[EFK+12] Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl, and Peter Schüller. Exploiting unfounded sets for HEX-program evaluation. In *Proceedings of the Thirteenth European Conference on Logics in Artificial Intelligence (JELIA 2012)*, France, pages 160–175, Springer, 2012.

[EFK+14] Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl, and Peter Schüller. Efficient HEX-program evaluation based on unfounded sets. *Journal of Artificial Intelligence Research*, 49:269–321, 2014.

[EFM10] Thomas Eiter, Michael Fink, and João Moura. Paracoherent answer set programming. In *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR 2010)*, Canada, pages 486–496, 2010.

[EFRS14] Thomas Eiter, Michael Fink, Christoph Redl, and Daria Stepanova. Exploiting support sets for answer set programs with external evaluations. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-eighth Conference on Artificial Intelligence (AAAI 2014)*, Canada, pages 1041–1048. AAAI Press, 2014.

[EFS12a] Thomas Eiter, Michael Fink, and Daria Stepanova. Semantic independence in DL-programs. In *Proceedings of the Sixth International Conference on*

Web Reasoning and Rule Systems (RR 2012), volume 7497 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2012.

[EFS12b] Thomas Eiter, Michael Fink, and Daria Stepanova. Semantic independence in DL-programs. Technical Report INFSYS RR-1843-12-07, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, 2012.

[EFS13a] Thomas Eiter, Michael Fink, and Daria Stepanova. Data repair of inconsistent DL-programs. In Francesca Rossi, editor, *Proceedings of the twenty-third International Joint Conference on artificial Intelligence (IJCAI 2013)*, 2013.

[EFS13b] Thomas Eiter, Michael Fink, and Daria Stepanova. Inconsistency management for description logic programs and beyond. In Wolfgang Faber and Domenico Lembo, editors, *Proceedings of the seventh International Conference on Web Reasoning and Rule Systems, (RR 2013)*, Germany, volume 7994 of *Lecture Notes in Computer Science*, pages 1–3. Springer, 2013.

[EFS14a] Thomas Eiter, Michael Fink, and Daria Stepanova. Computing repairs for inconsistent DL-programs over $\mathcal{EL}$ ontologies. In Eduardo Fermé and João Leite, editors, *Proceedings of the fourteenth Joint European Conference on Logics in Artificial Intelligence (JELIA 2014)*, volume 8761 of *Lecture Notes in Computer Science*, pages 426–441. Springer, 2014.

[EFS14b] Thomas Eiter, Michael Fink, and Daria Stepanova. Towards practical deletion repair of inconsistent DL-programs. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI 2014)*, Czech Republic, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 285–290, IOS Press, 2014.

[EFS14c] Thomas Eiter, Michael Fink, and Daria Stepanova. Towards practical deletion repair of inconsistent dl-programs. In Meghyn Bienvenu, Magdalena Ortiz, Riccardo Rosati, and Mantas Simkus, editors, *Proceedings of the Twenty-seventh International Workshop on Description Logics (DL 2014)*, Austria, volume 1193 of *CEUR Workshop Proceedings*, pages 169–180. CEUR-WS.org, 2014.

[EFSW10] Thomas Eiter, Michael Fink, Peter Schüller, and Antonius Weinzierl. Finding explanations of inconsistency in multi-context systems. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Proceedings of the Twelfth International Conference on Knowledge Representation and Reasoning (KR 2010)*, Canada, pages 329–339. AAAI press 2010.

[EFW14] Thomas Eiter, Michael Fink, and Antonius Weinzierl. Preference-based diagnosis selection in multi-context systems. In Thomas Eiter, Hannes Strass, Miroslaw Truszczynski, and Stefan Woltran, editors, *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of his 60th Birthday*, volume 9060 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2014.

[EG93] Thomas Eiter and Georg Gottlob. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In Dale Miller, editor, *Proceedings of the 1993 International Symposium on Logic Programming (ILPS 1993)*, Canada, pages 266–278. MIT Press, 1993.

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer Set Programming: A Primer. In *Proceedings of the fifth International Reasoning Web Summer School (RW 2009)*, Italy, volume 5689 of *LNCS*, pages 40–110. Springer, 2009.

[EIKS08] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, and Roman Schindlauer. Exploiting conjunctive queries in description logic programs. *Annals of Mathematics and Artificial Intelligence*, 53(1–4):115–152, 2008.

[EIL+08] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the Semantic Web. *Journal of Artificial Intelligence*, 172(12-13):1495–1539, 2008.

[EIST04] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. Nonmonotonic description logic programs: implementation and experiments. In Franz Baader and Andrei Voronkov, editors, *Proceedings of the eleventh International Workshop on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2004)*, volume 3452 of *Lecture Notes in Computer Science*, pages 511–527. Springer, 2004.

[EIST05] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In Kaelbling and Saffiotti *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, Scotland. pages 90–96, 2005.

[EIST06] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *Proceedings of the Third European Conference on Semantic Web: Research and Applications (ESWC 2006)*, Montenegro, volume 4011 of *LNCS*, pages 273–287. Springer, 2006.

[ELLP12] Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors. *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*. Springer, 2012.

[ELS97] Thomas Eiter, Nicole Leone, and D. Saccà. On the partial semantics for disjunctive deductive databases. *Annals of Mathematics and Artificial Intelligence*, 19(1-2):59–96, 1997.

[ELW13] Andreas Ecke, Michel Ludwig, and Dirk Walther. The concept difference for $\mathcal{EL}$-terminologies using hypergraphs. In Gioele Barabucci, Uwe M. Borghoff, Angelo Di Iorio, and Sonja Maier, editors, *Proceedings of the International workshop on Document Changes: Modeling, Detection, Storage and Visualization*, Italy, volume 1008 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[ESSXar] Thomas Eiter, Patrik Schneider, Mantas Simkus, and Gouhui Xiao. Using *openstreetmap* data to create benchmarks for ontology-based query answering systems. In *OWL Reasoner evaluation workshop*, 2014.

[FEGC+11] Michael Fink, Adil El Ghali, Amina Chniti, Roman Korf, Antonia Schwichtenberg, François Lévy, Jörg Pührer, and Thomas Eiter. D2.6 Consistency maintenance. Final report. Technical Report 2.6, ONTORULE ICT-2009-231875 Project, 2011.

[FFP05] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Consistent query answers on numerical databases under aggregate constraints. In Gavin M. Bierman and Christoph Koch, editors, *Proceedings of the Tenth International Symposium on Database Programming Languages (DBPL 2005)*, volume 3774 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2005.

[FGC+11] Michael Fink, Adil El Ghali, Amina Chniti, Roman Korf, Antonia Schwichtenberg, François Lévy, Jörg Pührer, Thomas Eiter. D2.6 Consistency maintenance. Final report. Technical Report 2.6, ONTORULE ICT-2009-231875 Project, 2011.

[Fin12a] Michael Fink. Paraconsistent hybrid theories. In Manuela M. Veloso, editor, *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, Italy, pages 391–401. AAAI Press, 2012.

[Fin12b] Michael Fink. Paraconsistent hybrid theories. In *Proceedings of the Tthirteenth Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, Italy, pages 141–151. AAAI press, 2012.

[FLL07] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, India, pages 372–379, 2007.

[FLP11] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

[FM07] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *Journal of Computer and System Sciences*, 73(4):610–635, 2007.

[FMC+09] Peter Fox, Deborah L. McGuinness, Luca Cinquini, Patrick West, Jose Garcia, James L. Benedict, and Don Middleton. Ontology-supported scientific data frameworks: The virtual solar-terrestrial observatory experience. *Journal of Computers & Geosciences*, 35(4):724–738, 2009.

[GCS10] Sergio Alejandro Gómez, Carlos Iván Chesñevar, and Guillermo Ricardo Simari. Reasoning with inconsistent ontologies through argumentation. *Journal of Applied Artificial Intelligence*, 24(1&2):102–148, 2010.

[GGZ03] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Transactions Knowledge and Data Engineering*, 15(6):1389–1408, 2003.

[GHKS07] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Just the right amount: extracting modules from ontologies. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the Sixteenth International Conference on World Wide Web (WWW 2007)*, Canada, pages 717–726. ACM, 2007.

[Gin87] Matthew L. Ginsberg. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann, USA, 1987.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GL91a] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–385, 1991.

[Goo05] John Goodwin. Experiences of using OWL at the ordnance survey. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter F. Patel-Schneider, editors, *Proceedings of the Workshop on OWL: Experiences and Directions (OWLED 2005)*, Ireland, volume 188 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[GPST08] M. Gebser, J. Pührer, T. Schaub, and H. Tompits. A meta-programming technique for debugging answer-set programs. In *Proceedings of the Twenty-third Conference on Artificial Intelligence (AAAI 2008)*, USA, pages 448–453. AAAI Press, 2008.

[GPW07] Georg Gottlob, Reinhard Pichler, and Fang Wei. Efficient datalog abduction through bounded treewidth. In *Proceedings of the Twenty-second Conference on Artificial Intelligence (AAAI 2007)*, Canada, pages 1626–1631. AAAI Press, 2007.

[GR95] Peter Gärdenfors and Hans Rott. Belief revision. In Dov Gabbay, Christopher J. Hogger, and John Alan Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 35–132. Oxford University Press, 1995. 1995.

[GS14] Fabien Gandon and Guus Schreiber. Rdf 1.1 xml syntax specification (2014), 2014. `http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225`.

[GTH06] Tom Gardiner, Dmitry Tsarkov, and Ian Horrocks. Framework for an automated comparison of description logic reasoners. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 654–667. Springer, 2006.

[Hay80] Patrick J. Hayes. The logic of frames. In *Frame Conceptions and Text Understanding, DeGruyter*, 1980.

[H14] Andreas Humenberger. Implementing Paracoherent Answer Set Semantics. Bachelor thesis, Vienna University of Technology, Austria, 2014.

[HDG+06] Matthew Horridge, Nick Drummond, John Goodwin, Alan L. Rector, Robert Stevens, and Hai Wang. The manchester OWL syntax. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *Proceedings of the Workshop on OWL: Experiences and Directions (OWLED 2006)*, USA. volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[Hey06] Stijn Heymans. *Decidable Open Answer Set Programming.* PhD thesis, Theoretical Computer Science Lab (TINF), CS Dept, Vrije Universiteit Brussel, 2006.

[HH00] Jeff Heflin and James A. Hendler. Dynamic ontologies on the web. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, USA, pages 443–449. AAAI Press / The MIT Press, 2000.

[HKE+10] Stijn Heymans, Roman Korf, Michael Erdmann, Jörg Pührer, and Thomas Eiter. F-logic#: Loosely coupling f-logic rules and ontologies. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2010)*, pages 248–255. IEEE Computer Society, 2010.

[HLH13] Shasha Huang, Qingguo Li, and Pascal Hitzler. Reasoning with inconsistencies in hybrid MKNF knowledge bases. *Logic Journal of the IGPL*, 21(2):263–290, 2013.

[HLSWar] Peter Hansen, Carsten Lutz, Inanç Seylan, and Frank Wolter. Query rewriting under *el*-TBoxes:efficient algorithms. In Meghyn Bienvenu, Magdalena Ortiz, Riccardo Rosati, and Mantas Simkus, editors, *Proceedings of the twenty-seventh International Workshop on Description Logics (DL 2014)*, Austria, volume 1193 of *CEUR Workshop Proceedings*, pages 197–208. CEUR-WS.org, 2014.

[HNV07] Stijn Heymans, Davy Van Nieuwenborgh, and Dirk Vermeir. Open answer set programming for the semantic web. *Journal of Applied Logic*, 5(1):144–169, 2007.

[HOSG08] Petr Hlinený, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Computer Journal*, 51(3):326–362, 2008.

[HP08] Miki Hermann and Reinhard Pichler. Counting complexity of minimal cardinality and minimal weight abduction. In Steffen Hölldobler, Carsten Lutz, and Heinrich Wansing, editors, *Proceedings of the Eleventh Joint European Conference on Logics in Artificial Intelligence*, volume 5293 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2008.

[HP10] Miki Hermann and Reinhard Pichler. Counting complexity of propositional abduction. *Journal Computer Systems and Science*, 76(7):634–649, 2010.

[HPF+06] Stijn Heymans, Jos de Bruijn, Livia Predoiu, Cristina Feier, Davy Van Nieuwenborgh. G-hybrid knowledge bases. In *Proceedings the Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006)*, 2006.

[HPS12] Ian Horrocks, Bijan Parsia, and Uli Sattler. OWL 2 web ontology language rdf-based semantics, 2012. `http://www.w3.org/TR/owl2-direct-semantics/`.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium, 2004.

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003

[HRDA11] Sajjad Hussain, Jos De Roo, Ali Daniyal, and Syed Sibte Raza Abidi. Detecting and resolving inconsistencies in ontologies using contradiction derivations. In *Proceedings of the Thirty-fifth Annual IEEE International Conference on Computer Software and Applications (COMPSAC 2011)*, pages 556–561. IEEE Computer Society, 2011.

[HvHH+05] Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A framework for handling inconsistency in changing ontologies. In *Proceedings of the International Semantic Web Conference (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 353–367. Springer, 2005.

[HvHtT05] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the Nineteenth Conference Joint Conference on Artificial Intelligence (IJCAI 2005)*, Scotland, pages 349–350, 2005.

[IJ84] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[JHQ+09] Qiu Ji, Peter Haase, Guilin Qi, Pascal Hitzler, and Steffen Stadtmüller. RaDON - repair and diagnosis in ontology networks. In *Proceedings of the Sixth European Semantic Web Conference (ESWC 2009)*, volume 5554 of *Lecture Notes in Computer Science*, Greece, pages 863–867. Springer, 2009.

[JS08] Andrew G. James and Kent A. Spackman. Representation of disorders of the newborn infant by SNOMED. In Stig Kjær Andersen, Gunnar O. Klein, Stefan Schulz, and Jos Aarts, editors, *Proceedings of the Twenty-first International Congress of the European Federation for Medical Informatics (EFMI 2008)*, Sweden, volume 136 of *Studies in Health Technology and Informatics*, pages 833–838. IOS Press, 2008.

[KAH08] Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. A coherent well-founded model for hybrid MKNF knowledge bases. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI 2008)*, Greece, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 99–103. IOS Press, 2008.

[KBR86] Thomas Kaczmarek, Raymond Bates, and Gabriel Robins. Recent developments in nikl. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI 1986)*, Philadelphia, pages 978–985, 1986.

[KKS13] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. The incredible ELK: from Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies. *Journal of Automated Reasoning*, 53(1):1–61. Springer, 2013.

[KL92] Michael Kifer and Eliezer L. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 9(2):179–215, 1992.

[KLT+10] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to query answering in *DL-Lite*. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR 2010)*, Canada, pages 247–257. AAAI Press, 2010.

[KLWW12] Boris Konev, Michel Ludwig, Dirk Walther, and Frank Wolter. The logical difference for the lightweight description logic EL. *Journal of Artificial Intelligence Research (JAIR)*, 44:633–708, 2012.

[Kow88] Robert A. Kowalski. The early years of logic programming. *Journal Communication of the ACM*, 31(1):38–43, 1988.

[KPSG06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in owl ontologies. In *Proceedings of the Third European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science*, pages 170–184. Springer, 2006.

[Kre07] Thomas Krennwallner. Integration of conjunctive queries over description logics into hex-programs. Master thesis, Vienna University of Technology, Austria, 2007.

[KRH08] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. ELP: Tractable rules for OWL 2. In *Proceedings of the Seventh International Semantic Web Conference (ISWC 2008)*, pages 649–664. Springer, 2008.

[Krö12] Markus Krötzsch. OWL 2 profiles: An introduction to lightweight ontology languages. In Thomas Eiter and Thomas Krennwallner, editors, *Proceedings of the Eigth International Summer School on Reasoning Web*, Austria, volume 7487 of *Lecture Notes in Computer Science*, pages 112–183. Springer, 2012.

[Lee05] Joohyung Lee. A model-theoretic counterpart of loop formulas. In Kaelbling and Saffiotti, editors, *Proceedings of the nineteenth Conference Joint Conference on artificial Intelligence (IJCAI 2005)*, Scotland, pages 503–508.

[Lem04] Domenico Lembo. *Dealing with Inconsistency and Incompleteness in Data Integration*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma La Sapienza, Italy, 2004.

[Lev66] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Journal Soviet Physics Doklady*, 10:707, 1966.

[Lev84] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Journal of Artificial Intelligence*, 23(2):155–212, 1984.

[Lif85] Vladimir Lifschitz. Computing circumscription. In Aravind K. Joshi, editor, *Proceedings of the Nineth International Joint Conference on Artificial Intelligence (IJCAI 1985)*, pages 121–127. Morgan Kaufmann, 1985.

[Llo87] John W. Lloyd. *Foundations of Logic Programming, Second Edition.* Springer, 1987.

[LLR+10] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant semantics for description logic ontologies. In Sonia Bergamaschi, Stefano Lodi, Riccardo Martoglia, and Claudio Sartori, editors, *Proceedings of the Nineteenth Italian Symposium on Advanced Database Systems (SEBD 2010)*, Italy, pages 103–117. Springer, 2010.

[LLR+11] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Query rewriting for inconsistent *DL-Lite* ontologies. In Sebastian Rudolph and Claudio Gutierrez, editors, *Proceedings of the fifth International Conference on Web Reasoning and Rule Systems (RR 2011)*, Ireland, volume 6902 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2011.

[LLR++11] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi and Domenico Fabio Savo. Inconsistency-Tolerant Semantics for Description Logic Ontologies. In Giansalvatore Mecca and Sergio Greco, editors, *Proceedings of the 19th Italian Symposium on Advanced Database Systems*, Italy, pages 103–117. Springer, 2010.

[LNS96] James J. Lu, Anil Nerode, and V. S. Subrahmanian. Hybrid knowledge bases. *Journal IEEE Transactions on Knowledge and Data Engineering*, 8(5):773–785, 1996.

[LRMC11] Mikhail K. Levin, Alan Ruttenberg, Anna Maria Masci, and Lindsay G. Cowell. *owl_cpp*, a C++ library for working with OWL ontologies. In Olivier Bodenreider, Maryann E. Martone, and Alan Ruttenberg, editors, *Proceedings of the Second International Conference on Biomedical Ontology*, USA, volume 833 of *CEUR Workshop Proceedings*, pages 255–257. CEUR-WS.org, 2011.

[LSS13a] Domenico Lembo, Valerio Santarelli, and Domenico Fabio Savo. A graph-based approach for classifying OWL 2 QL ontologies. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Proceedings of*

the Twenty-sixth International Workshop on Description Logics (DL 2013), Germany, volume 1014 of *CEUR Workshop Proceedings*, pages 747–759. CEUR-WS.org, 2013.

[LTW08] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in el using a database system. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *Proceedings of the Fifth International Workshop OWL: Experiences and Directions, (OWLED 2008)*, Germany, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[LW10] Carsten Lutz and Frank Wolter. Deciding inseparability and conservative extensions in the description logic $\mathcal{EL}$. *Journal of Symbolic Computation*, 45(2):194–228, 2010.

[LWW07] Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative extensions in expressive description logics. In Manuela M. Veloso *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, India, pages 453–458.

[LZ04] Fangzhen Lin and Xishun Zhao. On odd and even cycles in normal logic programs. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, USA, pages 80–85. AAAI Press / The MIT Press, 2004.

[MB87] Robert MacGregor and Raymond Bates. The LOOM knowledge representation language. Technical Report ISI/RS-87-188, Information Science Institute, University of Southern California, Marina del Rey (CA), USA, 1987.

[McC80] John McCarthy. Circumscription - A form of non-monotonic reasoning. *Journal of Artificial Intelligence*, 13(1-2):27–39, 1980.

[McC84] John McCarthy. Applications of circumscription to formalizing common sense knowledge. In *Proceedings of the First International Workshop on Nonmonotonic Reasoning (NMR 1984)*, USA, pages 295–324. AAAI press, 1984.

[McD82] Drew V. McDermott. Nonmonotonic logic II: nonmonotonic modal theories. *Journal of the ACM*, 29(1):33–57, 1982.

[MD80] Drew V. McDermott and Jon Doyle. Non-monotonic logic I. *Journal of Artificial Intelligence*, 13(1-2):41–72, 1980.

[MH09] Yue Ma and Pascal Hitzler. Paraconsistent reasoning for owl 2. In Axel Polleres and Terrance Swift, editors, *Proceedings of The Third International Conference on Web Reasoning and Rule Systems (RR 2009)*, USA, volume 5837 of *Lecture Notes in Computer Science*, pages 197–211. Springer, 2009.

[MHL08] Yue Ma, Pascal Hitzler, and Zuoquan Lin. Paraconsistent reasoning for expressive and tractable description logics. In Franz Baader, Carsten Lutz and Boris Motik, editors *Proceedings of the Twenty-first International Workshop on Description Logics (DL2008)*, Germany, 2008.

[Mid11] Cornelis A. Middelburg. A survey of paraconsistent logics. *Computing Research Repository*, abs/1103.4324, 2011.

[Min85] Marvin Minsky. A framework for representing knowledge. In Ronald J. Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*, USA, pages 245–262. Morgan Kaufmann, 1985.

[MMSA13] Maria Vanina Martinez, Cristian Molinaro, V. S. Subrahmanian, and Leila Amgoud. *A General Framework for Reasoning On Inconsistency*. Springer Briefs in Computer Science. Springer, 2013.

[Moo85] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Journal of Artificial Intelligence*, 25(1):75–94, 1985.

[MPP+14] Maria Vanina Martinez, Francesco Parisi, Andrea Pugliese, Gerardo I. Simari, and V. S. Subrahmanian. Policy-based inconsistency management in relational databases. *International Journal of Approximate Reasoning*, 55(2):501–528, 2014.

[MPPS12a] Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 2 web ontology language structural specification and functional-style syntax (second edition), 2012. `http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`.

[MPPS12b] Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 2 web ontology language XML serialization (second edition), 2012. `http://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/`.

[MR10] Boris Motik and Riccardo Rosati. Reconciling Description Logics and Rules. *Journal of the ACM*, 57(5):1–62, 2010.

[MRR11] Giulia Masotti, Riccardo Rosati, and Marco Ruzzi. Practical abox cleaning in *DL-Lite* (progress report). In Riccardo Rosati, Sebastian Rudolph, and Michael Zakharyaschev, editors, *Description Logics*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[MS06] Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the Thirteenth International Conference on Logic Programming, Artificial Intelligence and Reasoning (LPAR 2006)*, Cambodia, volume 4246 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2006.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In Frank Pfenning, editor, *Proceedings of the Twenty-first Conference on Automated Deduction (CADE 2007)*, Germany, volume 4603 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2007.

[MSH14] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165-228, 2009.

[NBM13] Riku Nortje, Arina Britz, and Thomas Meyer. Module-theoretic properties of reachability modules for $\mathcal{SRIQ}$. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Proceedings of the Twenty-sixth International Workshop on Description Logics (DL 2013)*, Germany, volume 1014 of *CEUR Workshop Proceedings*, pages 868–884. CEUR-WS.org, 2013.

[Ngu08] Ngoc Thanh Nguyen. *Advanced Methods for Inconsistent Knowledge Management*. Advanced Information and Knowledge Processing. Springer, 2008.

[NMSN14] Andreas Nolle, Christian Meilicke, Heiner Stuckenschmidt, and German Nemirovski. Efficient federated debugging of lightweight ontologies. In Roman Kontchakov and Marie-Laure Mugnier, editors, *Proceedings of the Eighth International Conference on Web Reasoning and Rule Systems (RR 2014)*, Greece, volume 8741 of *Lecture Notes in Computer Science*, pages 206–215. Springer, 2014.

[OPT12] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Stepwise debugging of description-logic programs. In Esra Erdem, Joohyung Lee, Yuliya Lierler and David Pearce, editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, pages 492–508, 2012.

[PÏ14] Jörg Pührer. *Stepwise Debugging in Answer-Set Programming: Theoretical Foundations and Practical Realisation. Dissertation.* PhD thesis, Vienna University of Technology, Austria, 2014.

[Pap94] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[Pea96] David Pearce. A new logical characterisation of stable models and answer sets. In Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski, editors, *Proceedings of the Second International Workshop on Non-Monotonic Extensions of Logic Programming, NMELP 1996*, Germany, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 1996.

[Pel91] Christof Peltason. The BACK system - an overview. *SIGART Bulletin*, 2(3):114–119, 1991.

[PHE10] Jörg Pührer, Stijn Heymans, and Thomas Eiter. Dealing with inconsistency when combining ontologies and rules using DL-programs. In *Proceedings of the Seventh Extended Semantic Web Conference, part I (ESWC 2010)*, Greece, pages 183–197. Springer, 2010.

[PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal of Data Semantics*, 10:133–173, 2008.

[Prz91] Teodor C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9(3/4):401–424, 1991.

[PSEK09] Enrico Pontelli, Tran Cao Son, and Omar El-Khatib. Justifications for logic programs under answer set semantics. *Journal of Theory and Practice of Logic Programming*, 9(1):1–56, 2009.

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the Fourteenth International Conference on World Wide Web (WWW 2005)*, Japan, pages 633–640. ACM, 2005.

[PT07] Jeff Z. Pan and Edward Thomas. Approximating OWL-DL ontologies. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI 2007)*, Canada, pages 1434–1439. AAAI Press, 2007.

[PUMH10] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.

[Qui67] M. Ross Quillan. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, pages 410–430, 1967.

[Red14] Christoph Redl. *Answer Set Programming with External Sources: Algorithms and Efficient Evaluation*. PhD thesis, Vienna University of Technology, Austria, 2014.

[Rei80] Raymond Reiter. A logic for default reasoning. *Journal of Artificial Intelligence*, 13(1-2):81–132, 1980.

[Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Journal of Artificial Intelligence*, 32(1):57–95, 1987.

[RGG+94] Alan L. Rector, Aldo Gangemi, E. Galeazzi, Andrzej J. Glowinski, and Angelo Rossi-Mori. The GALEN CORE model schemata for anatomy: Towards a re-usable application-independent model of medical concepts. In Pedro Barahona, Manuela Veloso, and Joanne Bryant, editors, *Twelfth International Congress of the European Federation for Medical Informatics (MIE 1994)*, Portugal, pages 229–233, 1994.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, 3e*. Pearson Education, 2010.

[Ros05] Ricardo Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

[Ros07] Riccardo Rosati. On conjunctive query answering in el. In Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris, editors, *Description Logics*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[RP05] Robert G. Raskin and Michael J. Pan. Knowledge representation in the semantic web for earth and environmental terminology (SWEET). *Computers & Geosciences*, 31(9):1119–1125, 2005.

[RRGM12] Riccardo Rosati, Marco Ruzzi, Mirko Graziosi, and Giulia Masotti. Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *Proceedings of the Eleventh International Semantic Web Conference (ISWC 2012)*, USA, volume 7650 of *Lecture Notes in Computer Science*, pages 337–349. Springer, 2012.

[SAR+07] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J. Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J. Mungall, The OBI Consortium, Neocles Leontis, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Richard H Scheuermann, Nigam Shah, Patricia L Whetzel, and Suzanna Lewis. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25(11):1251–1255, Nov 2007.

[SCC97] Kent A. Spackman, Keith E. Campbell, and Roger A. Côté. SNOMED RT: a reference terminology for health care. In *Proceedings of the American Medical Informatics Association Annual Symposium (AMIA 1997)*, USA. AMIA, 1997.

[Sch05] Stefan Schlobach. Diagnosing terminologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, USA, pages 670–675. AAAI Press / The MIT Press, 2005.

[Sch06] Roman Schindlauer. *Answer-Set Programming for the Semantic Web*. PhD thesis, Vienna University of Technology, Austria, 2006.

[SCM12] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Annals of Mathematics and Artificial Intelligence*, 64(2-3):209–246, 2012.

[SGB00] Robert Stevens, Carole A. Goble, and Sean Bechhofer. Ontology-based knowledge representation for bioinformatics. *Briefings in Bioinformatics*, 1(4):398–414, 2000.

[SGM95] Geri Steve, Aldo Gangemi, and Angelo Rossi Mori. Modelling a sharable medical concept system: Ontological foundation in GALEN. In Pedro Barahona, Mario Stefanelli, and Jeremy C. Wyatt, editors, *Proceedings of the Fifth Conference on Artificial Intelligence in Medicine in Europe (AIME 1995)*, Italy, volume 934 of *Lecture Notes in Computer Science*, pages 411–412. Springer, 1995.

[SGWB10] Thomas Scharrenbach, Rolf Grütter, Bettina Waldvogel, and Abraham Bernstein. Structure preserving TBox repair using defaults. In Volker Haarslev, David Toman, and Grant E. Weddell, editors, *Proceedings of the Twenty-third International Workshop on Description Logics (DL 2010)*, Spain, volume 573 of *CEUR Workshop Proceedings*, 2010.

[SI03] Chiaki Sakama and Katsumi Inoue. An abductive framework for computing knowledge base updates. *Journal on Theory and Practice of Logic Programming*, 3(6):671–713, 2003.

[SMH12] Giorgio Stefanoni, Boris Motik, and Ian Horrocks. Small datalog query rewritings for el. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Description Logics*, Italy, volume 846 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[SMK+14] Stefan Schulz, Catalina Martínez-Costa, Daniel Karlsson, Ronald Cornet, Mathias Brochhausen, and Alan L. Rector. An ontological analysis of reference in health record statements. In Pawel Garbacz and Oliver Kutz, editors, *Proceedings of the Eighth International Conference on Formal Ontology in Information Systems (FOIS 2014)*, Brazil, pages 289–302. IOS Press, 2014.

[SPS09] Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*. Springer, 2009.

[Ste14] Daria Stepanova. Inconsistencies in hybrid knowledge bases. In *Proceedings of the Fourteenth Doctoral Consortium on Knowledge Represenation (DC of KR 2014)*, 2014.

[Syr06] Tommi Syrjänen. Debugging Inconsistent Answer-Set Programs. In Ürgen Dix and Anthony Hunter, editors, *Proceedings of the Eleventh International Workshop on Nonmonotonic Reasoning, (NMR 2006)*, UK, pages 77–83, Department of Informatics, Technical Report, IfI-06-04, 2006.

[The00] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Journal Nature Genetics*, 25:25–9, 2000.

[TRKH08] Tuvshintur Tserendorj, Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Approximate OWL-reasoning with screech. In Diego Calvanese and Georg Lausen, editors, *Proceedings of the Second International Conference on Web Reasoning and Rule Systems*, Germany, volume 5341 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2008.

[UD07] Mathias Uslar and Nikolai Dahlem. Semantic web technologies for power grid management. In Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, and Marc Ronthaler, editors, *Proceedings of INFORMATIK 2007: Informatik trifft Logistik. Band 1. Beiträge der 37. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Germany, volume 109 of *LNI*, pages 242–250, 2007.

[War77] David H D Warren. Prolog: The language and its implementation compared with lisp. In *ACM SIGPLAN Notices*, pages 109–115, 1977.

[WEY+13] Yisong Wang, Thomas Eiter, Jia-Huai You, Li-Yan Yuan, and Yi-Dong Shen. Eliminating nonmonotonic DL-atoms in description logic programs. In Wolfgang Faber and Domenico Lembo, editors, *Proceedings of the Seventh International Conference on Web reasoning and Web systems (RR 2013)*, Germany, pages 168–182. Springer, 2013.

[WGS05] Holger Wache, Perry Groot, and Heiner Stuckenschmidt. Scalable instance retrieval for the semantic web by approximation. In Mike Dean, Yuanbo Guo, Woochun Jun, Roland Kaschek, Shonali Krishnaswamy, Zhengxiang Pan, and Quan Z. Sheng, editors, *Proceedings of the Sixth International Conference on Web Information Systems Engineering (WISE 2005)*, USA, volume 3807 of *Lecture Notes in Computer Science*, pages 245–254. Springer, 2005.

[Wij09] Jef Wijsen. Consistent query answering under primary keys: a characterization of tractable queries. In Ronald Fagin, editor, *Proceedings of the Twelfth International Conference on Database Theory (ICDT 2009)*, Russia, volume 361 of *ACM International Conference Proceeding Series*, pages 42–52. ACM, 2009.

[WWT10] Zhe Wang, Kewen Wang, and Rodney W. Topor. A new approach to knowledge base revision in *DL-Lite*. In Maria Fox and David Poole, editors, *Pro-*

ceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI 2010), USA, pages 369–374. AAAI Press, 2010.

[Xia14] Guohui Xiao. *Inline Evaluation of Hybrid Knowledge Bases*. PhD thesis, Vienna University of Technology, Austria, 2014.

[YY94] Jia-Huai You and Li-Yan Yuan. A three-valued semantics for deductive databases and logic programs. *Journal of Computer and System and Sciences*, 49(2):334–361, 1994.

[ZPR09] Yuting Zhao, Jeff Z. Pan, and Yuan Ren. Implementing and evaluating a rule-based approach to querying regular $\mathcal{EL}+$ ontologies. In Ge Yu, Mario Köppen, Shyi-Ming Chen, and Xiamu Niu, editors, *Proceedings of the Nineth International Conference on Hybrid Intelligent Systems (HIS 2009)*, China, pages 493–498. IEEE Computer Society, 2009.

APPENDIX A

Curriculum Vitae

Daria Stepanova

Curriculum Vitae

Favoritenstraße 9-11
Vienna, Austria
+43 680 5040619
+43 (1) 58801184801
daria.stepanova@kr.tuwien.ac.at
http://www.kr.tuwien.ac.at/staff/dasha/

Personal Information

Born May 7, 1988 in St.Petersburg, Russia
Nationality Russian
Languages Russian (native speaker), English (proficient user), German (intermediate)

Education

2011–present **PhD student at Vienna University of Technology**.
Faculty Informatics
Program Mathematical Logic in Computer Science
Thesis Inconsistencies in Hybrid Knowledge Bases
Supervisor O. Univ. Prof. Dr. techn. Thomas Eiter
Co-supervisor Proj. Ass. Dr. Michael Fink
ETCS passed 42.0, average grade point: 1.08

2005–2010 **Diploma student at St. Petersburg State University**.
equiv. to BSc + MSc
Faculty Mathematics and Mechanics
Specialization Applied Informatics
Thesis Computer-aided Resolution of Unsatisfiabilities in Ontologies
Supervisor Prof. Dr. techn. Boris Novikov
graduation with distinction

Continuing Education

2006–2007 **Student at St. Petersburg State University**.
Department of Linguistics
English for professionals
Level completed: advanced

2011–2012 **Student at Vienna University of Technology**.
German as a foreign language
Levels completed: A 2.1, B 2.2

Work Experience

2011–present **Research assistant at Vienna University of Technology**, *Vienna*, Austria.
Involved in projects:
Reasoning in Hybrid Knowledge Bases (FWF P20840)(2011–2012)
`http://www.kr.tuwien.ac.at/research/projects/hybridkb/index.html`
Evaluation of ASP Programs with External Source Access (FWF P24090)(2014)
`http://www.kr.tuwien.ac.at/research/projects/hexhex/`

2009–2011 **Specialist on system integration at "eKassir"**, *St.Petersburg*, Russia.
Software for payment systems
Technical support and remote assistance, system installation and adjustment on the client's side
Business trips: Podgorica (Montenegro), Akkra (Ghana).

2008–2009 **Research intern at Newcastle University**, *Newcastle-upon-Tyne*, UK.
Involved in project: Trust Economics
`https://wiki.cs.ncl.ac.uk/trusteconomics/TrustEconomics`
Development of a knowledge-based system prototype for information security decision-making

2007–2008, 2009–2010 **Part-time tutor of English for corporate clients at American Language Masters (ALM)**, *St.Petersburg*, Russia.

Supervising Bachelor Theses

- Andreas Humenberger (graduated April, 2014): *Implementing Paracoherent Answer Set Semantics* (co-supervised with Dr. Michael Fink)

Talks given at Conferences and Workshops

Towards Practical Deletion Repair of Inconsistent DL-programs
21st European Conference on Artificial Intelligence
Prague, Czeck Republic, August, 2014
Towards Practical Deletion Repair of Inconsistent DL-programs
27th International Workshop on Description Logics

Vienna, Austria, July, 2014

Data Repair of Inconsistent DL-programs

23rd Joint International Conference on Artificial Intelligence

Beijing, China, August, 2013

Semantic Independence in DL-programs

6th International Conference on Web Reasoning and Rule Systems

Vienna, Austria, September, 2012

A Knowledge Base for Justified Information Security Decision-making

4th International Conference on Software and Data Technologies

Sofia, Bulgaria, July, 2009

A Knowledge Base for Justified Information Security Decision-making (progress report)

Trust Economics Workshop

Newcastle-upon-Tyne, United Kingdom, December, 2008

Demos and Posters

Inconsistenies in Hybrid Knowledge Bases (PhD Description)

Poster presentation

14th International Conference on Principles of Knowledge Representation and Reasoning (Doctoral Consortium)

Vienna, Austria, July, 2014

Exploiting Support Sets for Answer Set Programs with External Evaluations

Poster presentation

28th Conference on Artificial Intelligence

Quebec, Canada, July, 2014

AngryHEX: An Angry Birds-playing Agent based on HEX-Programs

Poster presentation as a member of AngryHex team

Angry Birds AI competition (`http://www.aibirds.org`)

Prague, Czeck Republic, August, 2014

Data Repair of Inconsistent DL-programs

Poster presentation

23rd Joint International Conference on Artificial Intelligence

Beijing, China, August, 2013

Reviewing for Journals

- Journal of Web Semantics (JWS): 2014

Reviewing for Conferences and Workshops

- International Workshop on Nonmonotonic Reasoning (NMR): 2014

- International Workshop on Reactive Concepts in Knowledge Representation (REACTKNOW): 2014
- International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR): 2013
- AAAI Conference on Artificial Intelligence (AAAI): 2013
- International Semantic Web Conference (ISWC): 2013
- International Conference on Scalable Uncertainty Management (SUM): 2013
- International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR): 2013
- Conference on Knowledge Engineering and Semantic Web (KESW): 2013

Program Committee Member

- Conference on Knowledge Engineering and Semantic Web (KESW): 2013, 2014

Volunteer Work in Organizing Conferences and Public Events

- Vienna Summer of Logic (VSL 2014), Vienna, Austria, July, 9-24, 2014
- 25th International Conference on Computer-aided verification (CAV 2013), St. Petersburg, Russia, July, 13-19, 2013
- 6th International Conference on Web Reasoning and Rule Systems, Vienna, Austria, September, 10-12, 2012

Publications

Thomas Eiter, Michael Fink, and Daria Stepanova. Computing repairs for inconsistent DL-programs over $\mathcal{EL}$ ontologies. In *Proceedings of the 14th European Conference on Logics in Artificial Intelligence, (JELIA 2014), 426-441*, 2014.

Thomas Eiter, Michael Fink, and Daria Stepanova. Towards practical deletion repair of inconsistent DL-programs. In *Proceedings of 21st European Conference on Artificial Intelligence (ECAI 2014), 285-290*, 2014.

Thomas Eiter, Michael Fink, , and Daria Stepanova. Towards practical deletion repair of inconsistent DL-programs. In *Proceedings of the 27th International Workshop on Description Logics (DL workshop 2014), 169-180*, 2014.

Thomas Eiter, Michael Fink, Christoph Redl, and Daria Stepanova. Exploiting support sets for answer set programs with external computations. In *Proceedings of 28th Conference on Artificial Intelligence (AAAI 2014), 1041-1048*, 2014.

Daria Stepanova. Inconsistencies in hybrid knowledge bases. *Proceedings of 14th Doctoral Consortium on Knowledge Represenation (DC of KR 2014)*, 2014.

Thomas Eiter, Michael Fink, and Daria Stepanova. Data repair of inconsistent DL-programs. *Proceedings of 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2013.

Thomas Eiter, Michael Fink, and Daria Stepanova. Inconsistency management

for Description Logic Programs and beyond. *Proceedings of 6th International Conference on Web Reasoning and Rule Systems (RR 2013), 1-3*, 2013.

Thomas Eiter, Michael Fink, and Daria Stepanova. Semantic independence in DL-programs. *Proceedings of 6th International Conference on Web Reasoning and Rule Systems (RR 2012), 58-74*, 2012.

Daria Stepanova, Simon E. Parkin, and Aad van Moorsel. A knowledge base for justified information security decision-making. *Proceedings of 4th International Conference on Software and Data Technologies (ICSOFT 2009), 326-331*, 2009.