Towards Practical Ontology-based Data Access for Existential Rules

Peng Xiao

BEng, MEng

Institute for Integrated and Intelligent Systems

School of Information and Communication Technology

Griffith University

Submitted in fulfilment of the requirements of the degree of Doctor of Philosophy

October, 2020

Principal Supervisor
Prof. Kewen Wang

Principal Supervisor
Dr. Zhe Wang

Statement of Originality

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Signed: 

Date: 10/02/2020

Acknowledgements

This thesis would not have been completed without the support and assistance from a number of people.

First, I would give my sincere gratefulness to my supervisors Prof. Kewen Wang and Dr. Zhe Wang for their exceptional support and guidance throughout my PhD study. I thank Kewen for teaching me being a qualified researcher, I am greatly affected by his academic attitude. I thank him for providing me valuable advice on both research and life. I thank Zhe for his immense help and feedback to my research, I have gained so much in the discussion with him, without his suggestions and experience I would not be able to tackle the difficulties encountered in my research problems. I will not forget their encouragement when I was finding my foot and suffering from pressure. I could not have asked for more supportive supervisors and I sincerely appreciate their supervision.

I thank Griffith University for giving me the opportunity to do PhD and providing scholarship support. I also would like to thank the academic members of the Institute for Integrated and Intelligent Systems and ICT school. I thank Natalie Dunstan and Mel Gilbert for assisting me with my student affair. I thank Prof. Adual Sattar for holding seminars for my important milestones and supporting my requests in the PhD program.

I extend my thanks to my colleagues in our research group. I have received a lot in our weekly group meetings during the early stage of my PhD. I thank YiFan Jin for providing me helps when I was new to everything in Australia. I thank ZhiQiang Zhuang and Pouya Ghiasnezhad for all of the interesting research discussions. I thank my friend ZongJie Ma and his partner Summer, traveling with them was really relaxing and brounght me a lot of joy.

I also want to thank my master supervisor Hai Wan, who brounght me into the field of KR and encourage me doing PhD.

Finally, I want to thank my parents for their loves and understandings, they have always been my backing across the sea. I thank my sister and brother for their caring and life support. I wish to give my special thanks to my girl friend, Chi-Yu Chiu, I could not have gone through the toughness without her company during my whole PhD study.

Publication Arising From This Thesis

This thesis contains work that is published and/or prepared for publication. The details of these publications and their locations in this thesis are outlined below.

1. P. Xiao , K. Wang, and Z. Wang. Incosistency-Tolerant Forgetting in DL-lite. In International Semantic Web Conference 2017 (Posters, Demos & Industry Tracks). **(Chapter 5)**
2. P. Xiao , Z. Wang, K. Wang. Practical Datalog Rewriting for Existential Rules, 32nd International Workshop on Description Logics, 2019. **(Chapter 3)**
3. Z. Wang, P. Xiao, K. Wang, Z. Zhuang and H. Wan. Query Answering for Existential Rules via Efficient Datalog Rewriting. Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, 1933–1939. **(Chapter 3)**

The following work is prepared for publication:

1. P. Xiao , K. Wang, and Z. Wang. Practical Query Abduction for Existential Rules. **(Chapter 4)**

Abstract

Nowadays, knowledge reasoning is gaining more attentions in Artificial Intelligence, which stimulates the development of modern information systems. As the key ingredient of new generation of information systems, ontology-based data access (OBDA), which employs domain-specific knowledge provided by an ontology to reason over data, has received considerable attention in recent years. However, current research on ontological reasoning is not sufficient to establish practical OBDA with regards to scalability, feasibility, and usability. The primary aim of this thesis is to promote the applications of OBDA, by addressing the typical limitations of several research problems that are important to the practicality of OBDA systems. In this thesis, we focus on studying these problems in existential rules, a prominent family of ontological languages that proves to be both expressive and tractable.

Observing that current query rewriting techniques are not scalable over expressive ontologies, we propose a novel datalog rewriting approach for existential rules based on the notion of unfolding. While datalog rewritability cannot be guaranteed in general existential rules, we propose a novel abstract class called weakly-separable rules for datalog rewriting and show that it can generalize several combinations of existing well-accepted classes. We develop a prototype of query answering system called Drewer based on our proposed datalog rewriting method and evaluations show that our system has superior performance to state-of-the-art systems.

While query answering is the essential reasoning task for OBDA, it is necessary to provide appropriate explanations to query answers. We study the problem of query abduction, which is the underlying problem of explaining negative query answers. To make the abduction process more user-oriented, we present a novel abduction framework that discriminates between predicates expressing high-level and low-level concepts. We also develop an efficient algorithm for its computation based on first-order rewriting in existential rules, which shows scalability over large databases in experiments.

Forgetting is a well-known mechanism that can have a variety of potential applications to the manipulation of ontologies. However, current studies about forgetting cannot handle inconsistent ontologies, which hinders its applications to OBDA scenarios where errors of knowledge might occur. We present the first study of inconsistency-tolerant forgetting, that is, forgetting with the presence of inconsistencies. Three different definitions based on inconsistency-tolerant query answering are proposed and their rationality is illustrated by comparing to other possible solutions. We explored their properties and computation methods in a light-weight Description Logic language, DL-lite$_{core}$.

Contents

Abstract **iv**

1 Introduction **1**
1.1 Backgrounds . . . 1
1.2 Challenges in Ontology-based Data Access . . . 4
1.3 Contributions . . . 8
1.4 Thesis Organization . . . 9

2 Ontology-based Data Access **11**
2.1 Preliminaries . . . 11
2.1.1 Mathematical Basics . . . 11
2.1.2 Computational Theory . . . 12
2.1.3 First-order Logic . . . 15
2.2 Ontology Languages . . . 17
2.2.1 Description Logics . . . 18
2.2.2 Existential Rules . . . 22
2.3 Ontology Reasoning and Manipulation . . . 24
2.3.1 Query Answering . . . 25
2.3.2 Query Rewriting . . . 27
2.3.3 Decidable classes for Existential Rules . . . 30
2.3.4 Query Abduction . . . 32
2.3.5 Inconsistency-Tolerant Query Answering . . . 35
2.3.6 Forgetting . . . 39

3 Efficient Datalog Rewriting for Existential Rules **44**
3.1 Compact Datalog Rewriting . . . 46
3.2 Datalog Rewritable Classes . . . 53
3.3 Efficient Rewriting Algorithms . . . 62
3.4 Systems and Benchmarks . . . 65
3.5 Evaluation . . . 69

4 Practical Abduction for Existential Rules **75**
4.1 Selective Explanations . . . 77
4.2 Computing Explanations . . . 84
4.3 Compact Explanations . . . 89
4.4 Evaluation . . . 95

5 Inconsistency-tolerant Forgetting for Ontologies **97**
5.1 Inconsistency-tolerant Forgetting Definitions 98
5.2 Inconsistency-tolerant forgetting for DL-lite 105
5.3 Graph-based Implementation and Evaluation 116
5.3.1 Graph-based Implementation 116
5.3.2 Evaluation 117

6 Conclusion and Future Work **119**
6.1 Conclusion 119
6.2 Future Work 120

References **122**

Chapter 1

Introduction

1.1 Backgrounds

In recent years, the growing importance of Knowledge Representation and Reasoning (KR&R) in Artificial Intelligence have stimulated activities exploiting large volumes of Web data with semantic approaches.

To integrate data with semantics, *ontologies*, which are typically used to formalize domain knowledge, have received significant attention in KR studies. An ontology is a formal representation of the knowledge about a domain in terms of types, properties and relationships of entities, which allows users to have an abstract overview of data from the domain, and to infer implicit knowledge from incomplete information provide by the data.

Introducing ontologies to data leads to a new paradigm of data access, named *ontology-based data access* (OBDA) [Calvanese et al., 2007c], which is regarded as the key ingredient of the next generation of information system. In a nutshell, the general idea of OBDA is to place an ontological layer between user queries and the actual databases so that users are separated from the actual structure of data and the data is accessed after being enriched by domain-specific knowledge provided by the ontology.

To explain OBDA in more details, let us consider the following query,

$$q(x, y) = \mathsf{flight}(x), \mathsf{has\_flight}(y, x), \mathsf{fly\_to}(x, \text{London}), \mathsf{city}(\text{London}).$$

The query asks for information on flights that fly to London city and the airlines they belong to. Assume we have information about flights, airlines, and airports, which is represented as the following database $\mathcal{D}$:

$\mathsf{flight}(\text{AE806}),\ \mathsf{has\_flight}(\text{ExpressAir}, \text{AE806}),$
$\mathsf{arrive\_at}(\text{AE806}, \text{LCAirport}),\ \mathsf{airport}(\text{LCAirport}),\ \mathsf{locate\_in}(\text{LCAirport}, \text{London}),$
$\mathsf{city}(\text{London}).$

Since $\mathcal{D}$ does not have a record on $\mathsf{fly\_to}$, when the query is posed to $\mathcal{D}$, there is no result for the query. However, if we adopt a simple ontology Π that contains the following rule to describe a straightforward idea in this scenario,

$$\forall x, y, z.(\mathsf{fly\_to}(x, z) \leftarrow \mathsf{arrive\_at}(x, y), \mathsf{airport}(y), \mathsf{locate\_in}(y, z)).$$

which says that if a flight x arrives at an airport located in city y, it is a flight that flies to city y, then our knowledge can be enriched with the information of $\mathsf{fly\_to}$ and it can be entailed that $(\text{AE806}, \text{ExpressAir})$ is an answer to the query.

It shall be noted that the information in $\mathcal{D}$ is not necessarily the same as what is actually stored in databases. In general, these data are gathered from heterogeneous data sources, thus have different vocabularies and could be raw data that are obscure to users. In OBDA, such issue is usually addressed by *data mapping* techniques, which unify data using a predefined declarative mapping so that they are available for the ontology. Thus, users can focus on the design of queries, which are formulated solely according to the ontology. Data mapping is one essential component in the research of OBDA, but it's not the focus of this thesis. In the setting of OBDA in this thesis, we assume all data have been retrieved as facts that can directly apply to ontologies. In fact, these mappings can also be regarded as a special type of ontologies that unify data with additional vocabulary.

The above scenario is typically a kind of knowledge reasoning process, where ontologies are logical theories employed to represent knowledge and the querying results are what we can obtain from data through reasoning. Such a reasoning process is actually defined as ontological query answering, which is the main reasoning task in OBDA.

To satisfy the requirements of knowledge reasoning nowadays, OBDA systems should scale well over large volumes of data and if possible, be as efficient as traditional relational database management systems (RDBMS). Moreover, they should be able to handle data with complex forms. All these capabilities are highly related to the computational properties and expressiveness of the language used to represent ontologies. This makes the choice of formal language for ontologies rather important to the application of OBDA.

A major family of formalisms for representing and reasoning over ontologies is the family of Description Logics (DLs) [Baader et al., 2003], which have been extensively studied. DLs are decidable fragments of *first-order logic*, having a clear-cut semantics and convenient syntax. As DLs are very suitable to model relationships in taxonomic knowledge, they are widely used in the *Semantic Web* and underpin the *Web Ontology Language* (OWL) [McGuinness et al., 2004]. Various families of DLs have been proposed and studied [Calvanese et al., 2007a, Baader et al., 2005, Trivela et al., 2015, Kontchakov et al., 2011]. In particular, light-weight DLs such as DL-lite and $\mathcal{EL}$ were designed to achieve tractability in query answering. Research into query answering over these languages has also produced a series of practical prototype systems [Trivela et al., 2015, Calvanese et al., 2011, Rodriguez-Muro et al., 2013, Venetis et al., 2014].

Though DLs are successful in ontological reasoning, they still have critical limitations. Because of the convenient syntax of DLs, many real-world problems containing complex relations between objects cannot be expressed by DLs. Furthermore, in DLs only unary and binary relations can be described, which could not satisfy the need of modeling diverse web data [Bellomarini et al., 2017]. Recently, a new more expressive formalism called *existential rules* (a.k.a *Datalog*±) is proposed. Existential rules are rule formulas having the form $H \leftarrow B$, whose head and body are conjunctions of positive atoms. In particular, existentially quantified variables are allowed in the heads of existential rules. For instance, the following formula is an existential rule,

$$\forall x.(\exists y.(\mathsf{fly\_to}(x, y), \mathsf{city}(y)) \leftarrow \mathsf{flight}(x)).$$

The introduction of existential variables gives existential rules the ability to describe individuals that may not be presented in the current ontology, which has been recognized as a crucial ability to model domain knowledge. General existential rules are undecidable for query answering, therefore, appropriate restrictions to the syntax of rules need to

be made, which give rise to a variety of classes of existential rules [Calì et al., 2012, Gottlob et al., 2014d, Baget et al., 2011, Grau et al., 2013]. These classes are not only expressive enough to cover popular DLs such as DL-lite and $\mathcal{EL}$, but also tractable in the problem of query answering, which makes existential rules a promising formalism for OBDA. Moreover, in recent years, it has become a trend to extend existing studies of ontological reasoning problems to cover existential rules [Gottlob et al., 2014a, Gottlob et al., 2015, König et al., 2013, Wan et al., 2016, Wang et al., 2018].

1.2 Challenges in Ontology-based Data Access

In this section, we will talk about the challenges faced in the application of OBDA in four ways, corresponding to the different research problems in OBDA. We will introduce current studies of these research problems and demonstrate their limitations.

(1) **Efficient query answering in expressive ontologies.** Due to the limitations of DLs, expressive languages for ontologies need to be considered to establish flexible and applicable OBDA for modern applications. Also, as the most important reasoning problem in OBDA, query answering in the prominent expressive language, existential rules is not as well studied as in DLs. Though several tractable classes of existential rules has been proposed, where query answering in polynomial time is possible, it is a challenging task to develop practical algorithms and system prototypes for these classes in real-world applications where the scales of ontologies and data are significantly large.

One prominent technique to achieve efficient query answering accepted by many researchers is *query rewriting*. Briefly, in query rewriting approaches, queries and ontologies are reformulated into formulas expressed in formalisms for which mature data management and retrieval systems are already available. In the above scenario, one can rewrite the query into the following one,

$$q'(x, y) = \mathsf{flight}(x), \mathsf{has\_flight}(y, x), \mathsf{arrive\_at}(x, y), \mathsf{locate\_in}(y, z), \mathsf{airport}(y).$$

which is obtained by replacing $\mathsf{fly\_to}$ with its causes as shown in the ontology. As we have embedded necessary information from the ontology into q' through rewriting, even without the ontology, q' can still produce the same answer as q does, which means it becomes a classical query answering problem in the field of database [Abiteboul et al.,

1995]. Hence, the task of query answering can be accomplished using RDBMS rather than inefficient logical reasoners.

While rewriting-based query answering has been studied in various DL fragments [Hansen et al., 2015, Trivela et al., 2015, Eiter et al., 2012, Pérez-Urbina et al., 2010], very few practical approaches to rewriting for existential rules are proposed. Existing query rewriting systems for existential rules are typically based on first-order rewritings, i.e., the queries are rewritten into first-order formulas [Gottlob et al., 2014b, König et al., 2013, König et al., 2015a]. Unfortunately, evaluations have revealed that first-order rewriting systems cannot provide sufficient scalability even for ontologies of medium sizes (with a few hundred of rules) [König et al., 2015b, Bienvenu et al., 2017]. In most known techniques, the sizes of first-order rewritings are worst-case exponential w.r.t the length of the query [Kontchakov et al., 2011]. Recently, because of the advance of datalog solvers, such as RDFox [Nenov et al., 2015], VLog [Urbani et al., 2016], rewriting queries into datalog programs becomes a competitive alternative for query rewriting. Compared to first-order rewriting, the results of datalog rewriting are usually more compact, and more datalog rewritable classes of existential rules are known. However, current studies about datalog rewriting for existential rules are mostly theoretical [Gottlob and Schwentick, 2012, Bienvenu et al., 2014c], and very few practical datalog rewriting systems for existential rules have been developed. Filling this gap would be an important step on the road to practical OBDA.

(2) **Explanations to query answers.** Explainability of intelligent systems is one important research topic in Artificial Intelligence. In the setting of OBDA, query answering is fully interpretable and the explanations to query answers can potentially benefit the applications of ontologies in *data exchange* [Fagin et al., 2005], *medical diagnosis* [Bertaud-Gounot et al., 2012] and *life sciences* [Bard and Rhee, 2004]. In the literature, there are two main types of explanations for query answering, one of them is used to explain why a query answer is obtained, i.e., give the facts that support its entailment. Such explanations are also called *explanations to positive answers*, which are generally defined as a minimal subset of the database that are sufficient to derive the query answers, which have been well studied in the DL literature [Kalyanpur et al., 2007, Suntisrivaraporn et al., 2008, Baader and Suntisrivaraporn, 2008] and also explored in existential rules [Ceylan et al., 2019].

While there are also scenarios where some query answers are expected to occur in the result, but they are not entailed with current knowledge, then users are interested in why they cannot be entailed and what kind of information is missing in the database. We call such query answers *negative answers* and the problem of explaining negative answers can be formalized as *query abduction*, a new research topic in abductive reasoning [Calvanese et al., 2013]. In the example, say we observe that a flight CU125 also flies to London, which cannot be entailed by Π and $\mathcal{D}$. An explanation of this surprising observation is that the database $\mathcal{D}$ misses the fact of $\mathsf{arrive\_at}(\text{CU125}, \text{LCAirport})$.

In query abduction, because no hypothesis is presumed, sometimes we need to introduce new (abstract) objects to represent the explanations, which means the search space of explanations can be infinite. Moreover, without restrictions, there can be a large number of explanations for a query abduction problem, which is not favorable to users in practice. To reduce the number of possible explanations, one typical approach is to set *abducibles*, which are the only predicate symbols allowed to occur in an explanation [Du et al., 2011]. The most recent effort from [Du et al., 2014] tries to detect similar explanations and represent them compactly. However these approaches are far from feasible for actual scenarios, as they still cannot scale well over large database. Therefore, it is necessary to propose more refined definitions for query abduction so that the problem can be appropriately handled in practice. Besides, current works on query abduction [Calvanese et al., 2013, Wang et al., 2015, Du et al., 2014, Du et al., 2011] are mostly based on light-weight ontology languages; given the complex nature of the problem, it's a big challenge to solve it in existential rules.

(3) **Modularization of ontologies.** To satisfy the requirement of web reasoning, ontologies are growing much larger nowadays, due to the high complexity of ontological reasoning, reasoning tasks such as query answering over these ontologies can be rather inefficient, even with sophisticated algorithms and optimizations. However, in many cases, it is unnecessary to consider a whole ontology for reasoning. For example, when the content of a query is related to only a small portion of knowledge from the ontology, a possible strategy is to extract a module containing all needed information from the ontology, then query answering can be performed over a much smaller theory, thus efficiency is achieved. For this purpose, a well known concept, *forgetting*, which has been thoroughly studied in classical logic and logic programming, is introduced for ontologies [Wang et al., 2008, Konev et al., 2009, Koopmann and Schmidt, 2015, Arenas et al.,

2016]. Informally, forgetting is a mechanism that eliminates or hides symbols from a set of logical formulas, while the result still retain a certain kind of equivalence with the original formulas.

Again in our example, if a user is only interested in flights and their destination cities, we can forget all other symbols from the ontology and database, as a result, the rule in the ontology and facts about arrive_at, airport, locate_in in the database are removed. In order to have the same query answers as before, an extra operation that need to be performed is to add fly_to(AE806, London) to $\mathcal{D}$.

Adapting forgetting to ontologies gives rise to a variety of applications, such as ontology reuse, ontology versioning, privacy protection [Kontchakov et al., 2008]. While forgetting is well-studied in DLs [Kontchakov et al., 2008, Wang et al., 2008, Koopmann and Schmidt, 2015, Wang et al., 2010] and explored in existential rules [Wang et al., 2018], it still suffers from several limitations. Particularly, current studies of forgetting are not able to handle inconsistent ontologies; this is an important issue that need to be addressed, as we shall demonstrate next.

(4) **Inconsistency-Tolerance.** Dealing with inconsistent information is one of the essential issues in logical reasoning. As inconsistencies can easily arise in the development of ontologies due to the errors of knowledge engineers, this problem is gaining increasing importance in real-world scenarios. Consider the following sentence:

$$\bot \leftarrow \mathsf{fly\_to}(x, y), \mathsf{fly\_to}(x, z), y \neq z.$$

which describes a natural constraint that a flight cannot fly to two different places. With this constraint, an inconsistency would occur if we add the fact fly_to(AE806, Paris) to the database $\mathcal{D}$. To handle the inconsistencies, one straightforward approach is to resolve inconsistencies by cleaning the database; however, on many occasions it may not be practical because cleaning procedures are usually non-deterministic and removal of any data may result in loss of critical information. As in the above case, although the information about flights is in conflict, we can still come up with a conclusion that the flight AE806 flies to a capital city. The idea of extracting meaningful information with the presence of inconsistencies leads to the notion of *inconsistency-tolerant query answering*, which was first explored in the field of database [Arenas et al., 1999], and was then adapted for ontologies [Calvanese et al., 2007b]. In recent years, various

semantics for inconsistency-tolerant query answering were proposed [Calvanese et al., 2007b, Bienvenu and Bourgaux, 2016, Baget et al., 2016], and were studied in different formalisms [Zhang et al., 2014, Bienvenu and Bourgaux, 2016, Bienvenu et al., 2014a], including existential rules [Lukasiewicz et al., 2012, Lukasiewicz et al., 2013]. Apart from query answering, inconsistency-tolerant approaches are also introduced for other reasoning problems, such as query abduction [Du et al., 2015], however, to the best of our knowledge, no inconsistency-tolerant approach for forgetting is studied.

1.3 Contributions

The prime aim of this thesis is to provide practical solutions for reasoning problems related to OBDA, then to facilitate the development of modern OBDA systems that are scalable, feasible and efficient in real-world applications. Specifically, we studied the problems of query rewriting, query abduction, and inconsistency-tolerant forgetting. Our studies mainly focus on the expressive language existential rules, except for inconsistency-tolerant forgetting, the main results of which are based on light-weight DLs.

The contributions of this thesis are summarized as follows,

- We present both a practical approach and a prototype system for datalog rewriting and query answering over a wide range of ontologies expressed in existential rules. The rewriting process is based on the notion of unfolding [Wang et al., 2018]. While such a rewriting process may not terminate, we move on to identify classes of ontologies where the rewriting process terminates, introducing a class by combining existing well-accepted classes. And we also introduce a concrete efficient algorithm for the computation of datalog rewritings. A prototype system *Drewer* based on our proposed approach is implemented. Experiments show that it is able to handle a wide range of benchmarks in the literature and shows superior or comparable performance to state-of-the-art systems on both the compactness of rewriting and the efficiency of query answering.

- We propose a novel framework for query abduction, where a new type of explanation called selective explanations is introduced. Selective explanations generalize

the existing representative explanations and allows users to distinguish high-level pattern explanations from low-level concrete explanations. We present an algorithm for computing selective explanations based on an efficient query rewriting method for existential rules obtained recently. We also introduce a compact representation for selective explanations which can significantly reduce the number of generated explanations and can be computed efficiently. Evaluations show that our approach can effectively generate meaningful explanations and scale well over large databases.

- We present the first study about forgetting over inconsistent ontologies. When traditional forgetting definitions are not available for inconsistent ontologies, we propose three different definitions for inconsistency-tolerant forgetting, called strong forgetting, semantics-independent forgetting and γ-forgetting respectively. These definitions are based on inconsistency-tolerant query answering semantics and available for most ontology languages, including existential rules. We study their properties and propose algorithms for their computation in the language of DL-lite$_{core}$. Experiments show that our forgetting can be effectively computed for large inconsistent ontologies.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

1. In chapter 2, we first briefly introduce preliminary knowledge to the topics of this thesis. Then we introduce ontology languages and reasoning problems that we studied for practical ontology-based data access.

2. In chapter 3, we present our work about datalog rewriting for existential rules. We first demonstrate how to produce a datalog rewriting for existential rules using unfolding. Then we discuss about conditions of terminations and datalog rewritable classes of existential rules. Finally, we present our prototype system *Drewer* and experimental results.

3. In chapter 4, we present our novel work on query abduction. We define selective explanations and study their computation. Besides, a compact representation is

introduced for the explanations. An evaluation is performed to show the effectiveness of our query abduction approach.

4. In chapter 5, we present our work of inconsistency-tolerant forgetting. It begins with discussions about desiderata of forgetting for inconsistent ontologies and exploration of possible definitions. Then we define three different forgettings based on inconsistency-tolerant semantics. Their properties are studied and algorithms are designed for their computation.

5. Finally, in chapter 6, the works in this thesis are summarized and future directions are discussed.

Chapter 2

Ontology-based Data Access

This chapter will firstly gives an overview of necessary basic knowledge for this thesis, which includes fundamentals from discrete mathematics, computational theory, and first-order logic. Then we will introduce ontological languages concerned in this thesis, including light-weight DLs and existential rules. The third part of this chapter discusses about important ontology reasoning problems related to OBDA, which include query answering, query abduction and forgetting. These problems are the main concerns of this thesis, and some existing techniques developed for them will also be introduced.

2.1 Preliminaries

2.1.1 Mathematical Basics

An *n-ary* relation over a set $\mathcal{S}$ is subset $\mathcal{R}$ of $\mathcal{S}^n$ (n times cross-product of $\mathcal{S}$ with itself), we write $\mathcal{R}(s_1, \ldots, s_n)$ for each tuple $(s_1, \ldots, s_n) \in \mathcal{R}$. A binary relation is a 2-ary relation, particularly, we write $x\mathcal{R}y$ to denote $(x, y) \in \mathcal{R}$.

A *binary relation* $\mathcal{R}$ over $\mathcal{S}$ is *reflexive* if $(x, x) \in \mathcal{R}$ for all $x \in \mathcal{S}$; it is *symmetric* if $(x, y) \in \mathcal{R}$ implies $(y, x) \in \mathcal{R}$; and it is *transitive* if $(x, y) \in \mathcal{R}$ and $(y, z) \in \mathcal{R}$ implies $(x, z) \in \mathcal{R}$.

A *preorder* of a set $\mathcal{S}$ is binary relation over $\mathcal{S}$ that is reflexive and transitive. A binary relation $\mathcal{R}$ is *anti-symmetric* if $(x, y) \in \mathcal{R}$ implies $(y, x) \notin \mathcal{R}$. A partial order of $\mathcal{S}$ is

a preorder of $\mathcal{S}$ that is anti-symmetric. A total order of $\mathcal{S}$ is a partial order $\mathcal{R}$ of $\mathcal{S}$ satisfying that for any $x, y \in \mathcal{S}$, either $(x, y) \in \mathcal{R}$ or $(y, x) \in \mathcal{R}$.

A *partition* of a nonempty set $\mathcal{S}$ is a set $\mathcal{P}$ of subsets of $\mathcal{S}$ such that (1) $\bigcup_{\mathcal{S}_i \in \mathcal{P}} \mathcal{S}_i = \mathcal{S}$ (2) $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for $i \neq j$. A partition $\mathcal{P}$ is a bipartition if $|\mathcal{P}| = 2$.

A *graph* is a pair $G = (V, E)$, where V is a finite set of *vertexes* and E is a finite set of *edges* such that $E \subseteq V \times V$. A *directed graph* is a graph whose edges are ordered pairs.

Let $G = (V, E)$ be a directed graph. Given a vertex v in G, the *in-edges* of v are edges that end at v and the *out-edges* of v are edges that start from v. A directed path in G is a nonempty sequence of vertexes $p = (v_0, \ldots, v_n)$ where n is the length of p and for $0 \leq i < n$, $(v_i, v_{i+1}) \in E$. An undirected path in G is a nonempty sequence of vertexes $p = (v_0, \ldots, v_n)$ and for $0 \leq i < n$, $(v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E$. A *circle* (resp. *undirected circle*) in G is a directed path (resp. undirected path) $p = (v_0, \ldots, v_n)$ in G such that $v_0 = v_n$. Two vertexes v, v' in G are *connected* if there exists a directed path in G between v and v'. A *strongly connected component* (SCC) of G is a subset C of V such that for any $v, v' \in C$, there exists a path from v to v'.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs, a *homomorphism* from G to G' is a function $h : V \to V'$ such that for any $(v, v') \in E$, $(h(v), h(v)') \in E'$. G' is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. G' is a *core* of G if G' is a subgraph of G and there exists a homomorphism from G' to G. It can be proved that each graph has a unique core.

A *tree* is a directed graph that has exactly one vertex with no in-edges, which is called the *root* of the tree, and contains no undirected circles.

2.1.2 Computational Theory

In this section, we introduce basic notions and results from complexity theory that are used in this thesis. The content of this section is based on a book about introduction to computational theory [Sipser, 1996].

Turing machine is a powerful and widely accepted computational model for a general purpose computer. A Turing machine constitutes a head and a tape with infinite length, which simulates unlimited and unrestricted memory. And the core of a Turing machine

is a transition function that describes how the machine behaves (head moving right or left) when reading different symbols at different states. Formally,

Definition 2.1. A deterministic Turing machine (DTM) is a tuple $M = (\mathcal{Q}, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $\mathcal{Q}, \Sigma, \Gamma$ are all finite sets and

1. $\mathcal{Q}$ is the set of states of the machine,
2. Γ is the alphabet for tapes, containing the *blank symbol* $\sqcup$,
3. δ is the transition function of the machine, $\delta : \mathcal{Q} \times \Gamma \to \mathcal{Q} \times \Gamma \times \{L, R\}$, where L, R indicate the moving directions of the head.
4. $q_0 \in \mathcal{Q}$ is the starting state, $q_{\text{accept}} \in \mathcal{Q}$ is the accept state and $q_{\text{reject}} \in \mathcal{Q}$ is the reject state.

The computing process of a DTM $M = (\mathcal{Q}, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ works as follows. M receives a string $w = w_1 w_2 \dots w_n$ as input on the leftmost n cells of the tape, where w_n are symbols from Γ except the blank symbol $\sqcup$, and the rest of tape is filled with $\sqcup$. Initially, M is at state q_0 and the head starts on the leftmost cell of the tape. Then machine proceed according to the rules described by δ: let $\delta(q, a) = (r, b, L)$ and M is at q, when the head is over a cell containing symbol $a \in \Gamma$, the machine writes symbol $b \in \Gamma$ replacing a and goes to state r, and after writing, the head is moved to the left of the cell. The machine *halts* when it enters either the accept state q_{accept} or the reject state q_{reject}. Note that a machine can continue the process forever as long as it never enters the halting states. Given a Turing machine M, we say M *accepts* string w if it halts at accept state, and M *rejects* w if it halts at reject state.

A *language* is a collection of strings. Given a decision problem X, the language of X is

$$\mathcal{L} = \{w \mid w \text{ is the encoding of input } y \text{ of } X, \text{the answer of } X \text{ for } y \text{ is true.}\}$$

Definition 2.2. Let $\mathcal{L} \subseteq \Gamma^*$ be a language, M be a Turing machine with Γ as its alphabet. We say M *decides* $\mathcal{L}$, if for any $x \in \Gamma^*$, it holds that if $x \in \mathcal{L}$, then M accepts x, otherwise M rejects x. A language L is *decidable* if it can be decided by a Turing machine, otherwise it is *undecidable*.

An important variant of Turing machine is *nondeterministic Turing machine* (NTM). The transition function of a NTM has the form

$$\delta : \mathcal{Q} \times \Gamma \to 2^{\mathcal{Q} \times \Gamma \times \{L,R\}}$$

It means that at every step of transition, the machine may proceed with several possibilities. So the computation of an NTM forms a tree whose branches correspond to different choices of the transition. An NTM accepts an input if there exists a branch that leads to the accept state, and it rejects an input if there is no branch can lead to the accept state.

Definition 2.3. A Turning machine M decides a language $\mathcal{L}$ in time $O(f(n))$ if taking any $w \in \Gamma^*$ as input, M can halt in k steps, where $k \leq f(n)$.

Definition 2.4. A *time complexity class* $\text{DTIME}(f(n))$ ($\text{NTIME}(f(n))$) is a collection of languages that can be decided by a DTM (NTM) in time $O(f(n))$.

The following are complexity classes that considered in this thesis.

$$\begin{aligned} \text{P} &= \bigcup_{k \in \mathbb{N}} (\text{DTIME}(O(n^k))) \\ \text{NP} &= \bigcup_{k \in \mathbb{N}} (\text{NTIME}(O(n^k))) \\ \text{EXPTIME} &= \bigcup_{k \in \mathbb{N}} (\text{DTIME}(O(2^{n^k}))) \end{aligned}$$

The above classes satisfies relation:

$$\text{P} \subseteq \text{NP} \subseteq \text{EXPTIME}$$

Generally, problems in P is accepted to be *tractable*, and problems in NP (or beyond NP) are considered *intractable*. Note that whether P = NP is still an important open problem in computer science, while it is widely believed that the two classes are unequal.

To obtain the decidability and complexity class of a problem, an important mathematical tool, *reduction* is needed.

Definition 2.5. A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w halts with $f(w)$ on its tape.

Definition 2.6. A language $\mathcal{L}_1$ is *reducible* to a language $\mathcal{L}_2$, written $\mathcal{L}_1 \leq_m \mathcal{L}_2$ if there is a computable function f satisfying, for every w,

$$w \in \mathcal{L}_1 \iff f(w) \in \mathcal{L}_2$$

$\mathcal{L}_1$ is *polynomially reducible* to $\mathcal{L}_2$, written $\mathcal{L}_1 \leq \mathcal{L}_2$ if f can be done in polynomial time. f is called the *reduction* of $\mathcal{L}_1$ to $\mathcal{L}_2$.

If a language $\mathcal{L}$ is undecidable and we have $\mathcal{L}' \leq_m \mathcal{L}$, we can safely say $\mathcal{L}'$ is also undecidable. We say a language $\mathcal{L}$ is *at least difficult* as a language $\mathcal{L}'$ if $\mathcal{L}' \leq \mathcal{L}$.

Definition 2.7. Let $\mathcal{C}$ be a complexity class, a language $\mathcal{L}$ is called $\mathcal{C}$-hard if for any $\mathcal{L}' \in \mathcal{C}$, $\mathcal{L}' \leq \mathcal{L}$. $\mathcal{L}$ is called $\mathcal{C}$-complete if $\mathcal{L}$ is also in $\mathcal{C}$.

Definition 2.8. Let $\mathcal{L}$ be a language in complexity class $\mathcal{C}$, the *complement* of $\mathcal{L}$, denoted $\overline{\mathcal{L}}$, is $\Sigma^* \setminus \mathcal{L}$. A language $\mathcal{L}'$ is in class co-$\mathcal{C}$ if $\mathcal{L}' \in \overline{\mathcal{L}}$.

2.1.3 First-order Logic

First-order logic is the foundation of logics discussed in this thesis. In this section we introduce first-order logic without functions, which is sufficient to support the topics we studied in this thesis.

In the formulation of first-order logic, three mutually disjoint infinite sets of symbols N_P, N_I, N_V are considered, where N_P is a set of n-ary *predicate* symbols ($n \geq 0$), N_I is a set of *constant* symbols and N_V is set of variable symbols.. In particular, N_P is usually assumed to contain two special 0-ary predicates, $\top$ and $\bot$, called top and bottom respectively.

Definition 2.9. A first-order formula is recursively defined as follows:

(1) A term is an element in $\mathsf{N}_I \cup \mathsf{N}_V$.

(2) An atom is a formula with expression $p(t_0, \ldots, t_n)$, where p is an n-ary predicate symbol and t_i $(0 \leq i \leq n)$ are terms.

(3) If ϕ is a formula, then $\neg\phi$ is a formula.

(4) if ϕ and ψ are formulas, then $\phi \oplus \psi$ is a formula, where $\oplus \in \{\vee, \wedge, \rightarrow\}$.

(5) If ϕ is a formula and x is a variable, then $\forall x.\phi$ and $\exists x.\phi$ are formulas.

A variable in a first-formula is *quantified* if it is in the scope of a quantifier, otherwise, it is a *free* variable. A *sentence* is a first-formula without free variables. In addition, we call a first-order formula which is a disjunction of atoms of form $\alpha_0 \vee \neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \neg\alpha_n$, a *horn clause*. A horn clause is usually written as $\alpha_0 \leftarrow \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$, in this form, we call α_0 the *head* of the clause and the set $\{\alpha_1, \ldots, \alpha_n\}$ the *body* the clause.

The semantics of a first-order logic is given by the notion of *interpretation*.

Definition 2.10. Let $\mathcal{L}$ be a first-order logic. An interpretation for $\mathcal{L}$ is a triple $\mathcal{I} = (U, I, V)$, where

(1) U is an infinite set called the universe of $\mathcal{I}$.

(2) I is a mapping function that satisfies: (i) for any constant $c \in \mathsf{N}_I$, $I(c) \in U$; (ii) for any $P \in \mathcal{P}$, $I(P) \subseteq U^n$, where n is the arity of P.

(3) V is a mapping function for variables, called *variable assignment*. For each variable $x \in \mathsf{N}_V$, $V(x) \in U$.

Let t be a term and $\mathcal{I}$ be an interpretation, for shorthand, we use $t^{\mathcal{I}}$ to denote the element associated with t by $\mathcal{I}$. It's extended to predicate symbols and atoms in a natural way. Let V be a variable assignment, $V[x \mapsto u]$ is a variable assignment identical to V except that it maps x to u. And by $\mathcal{I}[x \mapsto u]$, we denote that the variable assignment V in $\mathcal{I}$ is replaced by $V[x \mapsto u]$.

Definition 2.11. Let $\mathcal{I}$ be an interpretation with universe U, and ϕ be a first-order formula, we write $\mathcal{I} \models \phi$ to indicate $\mathcal{I}$ satisfies ϕ, which is recursively defined as follows:

1. $\mathcal{I} \models \top$; $\mathcal{I} \not\models \bot$.
2. $\mathcal{I} \models p(t_0, \ldots, t_n)$, iff $(t_0^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) \in p^{\mathcal{I}}$.
3. $\mathcal{I} \models \neg\phi$, if $\mathcal{I} \not\models \phi$.

4. $\mathcal{I} \models \phi_1 \vee \phi_2$, if $\mathcal{I} \models \phi_1$ or $\mathcal{I} \models \phi_2$.

5. $\mathcal{I} \models \phi_1 \wedge \phi_2$, if $\mathcal{I} \models \phi_1$ and $\mathcal{I} \models \phi_2$.

6. $\mathcal{I} \models \exists x.\phi$, if there exists $u \in U$ such that $\mathcal{I}[x \mapsto u] \models \phi$.

7. $\mathcal{I} \models \forall x.\phi$, if for all $u \in U$, it satisfies $\mathcal{I}[x \mapsto u] \models \phi$.

A *first-order theory* is a finite set of first-order formulas. An interpretation $\mathcal{I}$ is a *model* of a first-order theory Π if $\mathcal{I}$ satisfies every formula in Π. Π is *satisfiable* or *consistent* if Π has at least one model. Given two theories Π and Π', we say Π *entails* Π', denoted $\Pi \models \Pi'$, if every model of Π is also a model of Π'. And we say Π and Π' are *equivalent* if $\Pi \models \Pi'$ and $\Pi' \models \Pi$.

A *substitution* is a mapping function $\sigma : \mathsf{N}_V \to \mathsf{N}_V \cup \mathsf{N}_I$, it can be expressed as a (possibly empty) set $\{t_1 \mapsto t'_1, \ldots, t_n \mapsto t'_n\}$ such that for $1 \leq i \leq n$, $t_i\sigma = t'_i$ and for other t not defined in σ, $t\sigma = t$. It extends to other first-order formulas in a natural way.

Given two sets of atoms $\mathcal{A}$, $\mathcal{A}'$, a *homomorphism* from $\mathcal{A}$ to $\mathcal{A}'$ is a substitution σ that satisfies $\mathcal{A}\sigma \subseteq \mathcal{A}'$. A *unifier* between $\mathcal{A}$ and $\mathcal{A}'$ is a substitution τ such that $\mathcal{A}\tau = \mathcal{A}'\tau$. A unifier τ between $\mathcal{A}$ and $\mathcal{A}'$ is *most general*, if for every unifier τ' between $\mathcal{A}$ and $\mathcal{A}$, there exists a substitution σ s.t $\tau\sigma = \tau'$.

A *signature* is a set of predicate symbols. A first-order formula is said to be defined over a signature Σ if all predicate symbols in the formula belong to Σ. Given a first-order theory Π, we use $\mathsf{sig}(\Pi)$ to denote the signature of Π, which is the set of predicate symbols occurring in Π, and we use $\mathsf{const}(\Pi)$ and $\mathsf{vars}(\Pi)$ to denote the set of constants and variable occurring in Π respectively. The *Herbrand Base* of Π, denoted $\mathsf{HB}(\Pi)$, is the set of all atoms that can be constructed from $\mathsf{sig}(\Pi)$ and $\mathsf{const}(\Pi)$.

2.2 Ontology Languages

In this section, we introduce Description Logics (DLs) and existential rules, which encompass the most widely used knowledge representation languages in the context of ontology-based data access.

2.2.1 Description Logics

Description Logics are decidable fragments of first-order logic. However, traditional DLs are mostly intractable: checking satisfiability of the 'basic' DL $\mathcal{ALC}$ is EXPTIME-complete, and that of $\mathcal{SROIQ}$, which underpins OWL2, is 2NEXPTIME-complete. Therefore, to achieve tractable reasoning and handle large scale of data, two light-weight families of DLs, the DL-lite family [Calvanese et al., 2007a] and the $\mathcal{EL}$ family [Baader et al., 2008] are proposed, which respectively associate to the simplified profiles of OWL2, OWL2 QL and OWL2 EL [Krötzsch, 2012]. Because of their nice properties over query answering, lightweight DLs receive significant attention in recent years. In this thesis, we focus on lightweight DLs.

We first introduce the core language of DL-lite family, DL-lite$_{core}$. Three mutually disjoint infinite sets $\mathsf{N}_C, \mathsf{N}_R, \mathsf{N}_I$ are considered in the construction of DL axioms. N_C and N_R are sets of *concept names* and *role names* respectively, and as in first-order logic N_I is a set of constant symbols, which are also called individual names in DLs. Concept names (resp. role names) are actually unary (resp. binary) predicate symbols.

The concepts and roles of DL-lite$_{core}$ are constructed in the following syntax, where $A \in \mathsf{N}_C$, $R \in \mathsf{N}_R$:

$$B \to A \mid \exists R, \quad R \to P \mid P^-, \quad C \to B \mid \neg B$$

Particularly, we say A is an *atomic concept*, R is an *atomic role*, and B is a *general concept*. A *concept inclusion axiom* is of form $B \sqsubseteq C$, if C occurs negatively, the axiom is also called *negative inclusion axiom*. A DL-lite$_{core}$ TBox is a set of concept inclusion axioms. A DL-lite ABox is a set of *membership assertions* of form $A(a)$, and $P(a, b)$, where a, b are individuals from N_I. A DL ontology is a pair $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox.

DL-lite$_{core}$ can be extended to other fragments in DL-lite by introducing more complex syntax and constructs. DL-lite$_{\mathcal{F}}$ allows *role functionality axioms* of form $(func\ R)$. DL-lite$_{\mathcal{R}}$ allows *role inclusion axioms* of form $R \sqsubseteq S$, where $S \to R \mid \neg R$. There are a few other expressive fragments in the DL-lite family, which are omitted here. The following is a set of DL-lite TBox axioms adapted from the Lehigh University

Benchmark (LUBM) [Guo et al., 2005], which is a university domain ontology frequently used in OBDA evaluations.

Example 2.1.

$$\begin{aligned}\mathcal{T}_U = \{&\mathsf{Faculty} \sqsubseteq \mathsf{Employee}, \quad \mathsf{Professor} \sqsubseteq \mathsf{Faculty}, \quad \mathsf{Student} \sqsubseteq \neg\mathsf{Professor},\\ &\mathsf{Employee} \sqsubseteq \exists\mathsf{worksFor}, \quad \mathsf{Student} \sqsubseteq \exists\mathsf{takeCourse}, \quad \mathsf{member} \equiv \mathsf{memberOf}^-\\ &\mathsf{ResearchAssistant} \sqsubseteq \exists\mathsf{member}^-, \quad \mathsf{worksFor} \sqsubseteq \mathsf{memberOf}\}\end{aligned}$$

The above axioms describe the hierarchy of a university. The first and the second axioms say a faculty is an employee and a professor is a faculty member. $\exists\mathsf{worksFor}$ *is a complex concept constructed with the role* worksFor *using existential restriction, and the fourth axiom says an employee is an individual that works for someone. The third axiom is a restriction, which states that a student cannot be a professor. Particularly, the sixth axiom is a simplified representation of two axioms* $\mathsf{member} \sqsubseteq \mathsf{memberOf}^-$ *and* $\mathsf{memberOf}^- \sqsubseteq \mathsf{member}$*, which states* member *is the inverse role of* memberOf*.*

The semantics of a DL ontology is also given by interpretations.

Definition 2.12. An interpretation $\mathcal{I}$ to a DL ontology over Σ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty domain of $\mathcal{I}$, $\cdot^{\mathcal{I}}$ is a mapping function over $\mathsf{N}_C \cup \mathsf{N}_R \cup \mathsf{N}_I$, which satisfies that,

1. $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, for $A \in \mathsf{N}_C \cap \Sigma$.
2. $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, for $P \in \mathsf{N}_R \cap \Sigma$.
3. $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, for $a \in \mathsf{N}_I$.

In the construction of DL-lite, $\cdot^{\mathcal{I}}$ is extended to satisfy,

1. $(\neg B)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$.
2. $(\exists R)^{\mathcal{I}} = \{o \mid \exists o' \in \Delta^{\mathcal{I}} \text{ st. } (o, o') \in R^{\mathcal{I}}\}$.
3. $(P^-)^{\mathcal{I}} = \{(o', o) \mid (o, o') \in P^{\mathcal{I}}\}$.

By default, the semantics of DL ontologies follow the *unique name assumption*(UNA), which requires that different individuals $a, b \in \mathsf{N}_I$ cannot be interpreted as the same object, i.e., $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is said to satisfy a concept inclusion axiom $B \sqsubseteq C$, if $B^{\mathcal{I}} \sqsubseteq C^{\mathcal{I}}$; $\mathcal{I}$ is said to satisfy a membership assertion $A(a)$ (resp. $R(a, b)$) if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp. $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$)). Analogous to concept inclusion axioms, $\mathcal{I}$ satisfies a role inclusion axiom $R \sqsubseteq S$, if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$. Besides, $\mathcal{I}$ satisfies $(func\ R)$, if $(o, o_1) \in R^{\mathcal{I}}$ and $(o, o_2) \in R^{\mathcal{I}}$, then $o_1 = o_2$.

$\mathcal{I}$ is said to satisfy an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, if $\mathcal{I}$ satisfies every axiom in $\mathcal{T}$ and every assertion in $\mathcal{A}$.

Example 2.2. *Let $\mathcal{O} = (\mathcal{T}_U, \mathcal{A})$, where $\mathcal{T}_U$ is the DL-lite TBox from Example 2.1, and*

$$\mathcal{A} = \{\mathsf{Student}(\mathsf{John}), \mathsf{Student}(\mathsf{Joy}), \mathsf{Faculty}(\mathsf{Frank}), \mathsf{member}(ICT, Frank)\}$$

Then a possible interpretation $\mathcal{I}^U = (\Delta, \cdot^{\mathcal{I}^U})$ that satisfies $\mathcal{O}$ can be constructed as follows:

the domain: $\Delta^U = \{John, Joy, Frank, ICT, Math\}$.

the interpretations of concepts:

$$\mathsf{Student}^{\mathcal{I}^U} = \{John, Joy\}, \quad \mathsf{Faculty}^{\mathcal{I}^U} = \{Frank\}, \quad \mathsf{Employee}^{\mathcal{I}^U} = \{\mathsf{Frank}\}$$

the interpretations of roles:

$$\mathsf{member}^{\mathcal{I}^U} = \{(ICT, Frank)\}, \quad \mathsf{memberOf}^{\mathcal{I}^U} = \{(Frank, ICT)\},$$
$$\mathsf{takeCourse}^{\mathcal{I}^U} = \{(Jonh, Math), (Joy, Math)\}, \quad \mathsf{worksFor}^{\mathcal{I}^U} = \{(Frank, ICT)\}$$

Apart from the classical interpretation, there is one important type of interpretation for ontologies called *canonical interpretation* (*canonical model*) [Kontchakov et al., 2011]. Intuitively, a canonical interpretation of an ontology is an interpretation that can be homomorphically embedded into any other interpretations of the ontology. In the following, we introduce the construction of canonical models of DL-lite ontologies.

Definition 2.13 (Canonical Interpretation)**.** Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a DL-lite ontology, the *Canonical interpretation* of $\mathcal{O}$, denote $\mathcal{I}_\mathcal{O}$, is defined as follows: the domain of $\mathcal{I}_\mathcal{O}$, denoted $\Delta^{\mathcal{I}_\mathcal{O}}$, consists of paths of the form $ac_{R_1} \dots c_{R_n}$, with $a \in \mathsf{const}(\mathcal{A})$, $n \geq 0$, $a \rightsquigarrow c_{R_1}$ and $c_{R_{i-1}} \rightsquigarrow c_{R_i}$, for $1 < i \leq n$, where the $\rightsquigarrow$ is defined by the following two conditions:

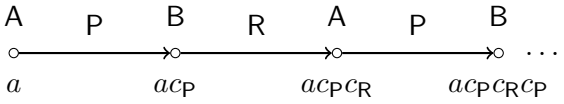
- $a \rightsquigarrow c_{R_1}$, if $\mathcal{O} \models \exists R_1(a)$ but $R_1(a, b) \notin \mathcal{A}$ for all $b \in \mathsf{const}(\mathcal{A})$;
- $c_{R_i} \rightsquigarrow c_{R_{i+1}}$, if for $i \leq n$, $\mathcal{T} \models \exists R_i^- \sqsubseteq \exists R_{i+1}$ and $R_i^- \neq R_{i+1}$.

The last element of a domain path σ is denoted as $tail(\sigma)$, and $\mathcal{I}_\mathcal{O}$ is defined as follows:

$$
\begin{aligned}
a^{\mathcal{I}_\mathcal{O}} &= a \text{ for } a \in \mathsf{const}(\mathcal{A}),\\
A^{\mathcal{I}_\mathcal{O}} &= \{a \in \mathsf{const}(\mathcal{A}) \mid \mathcal{O} \models A(a)\} \cup\\
&\quad \{\sigma c_R \in \Delta^{\mathcal{I}_\mathcal{O}} \mid \mathcal{T} \models \exists R \sqsubseteq A\},\\
P^{\mathcal{I}_\mathcal{O}} &= \{(a, b) \in \mathsf{const}(\mathcal{A}) \times \mathsf{const}(\mathcal{A}) \mid P(a, b) \in \mathcal{A}\} \cup\\
&\quad \{(\sigma, \sigma c_P) \in \Delta^{\mathcal{I}_\mathcal{O}} \times \Delta^{\mathcal{I}_\mathcal{O}} \mid tail(\sigma) \rightsquigarrow c_P\} \cup\\
&\quad \{(\sigma c_P, \sigma) \in \Delta^{\mathcal{I}_\mathcal{O}} \times \Delta^{\mathcal{I}_\mathcal{O}} \mid tail(\sigma) \rightsquigarrow c_{P^-}\}.
\end{aligned}
$$

Note that the canonical interpretation of a DL-lite ontology may still be infinite.

Example 2.3 ([Kontchakov et al., 2010])**.** *Consider ontology* $\mathcal{O} = (\mathcal{T}, \{\mathsf{A}(a)\})$*, where* $\mathcal{T} = \{\mathsf{A} \sqsubseteq \exists \mathsf{P}, \exists \mathsf{P}^- \sqsubseteq \mathsf{B}, \mathsf{B} \sqsubseteq \exists \mathsf{R}, \exists \mathsf{R}^- \sqsubseteq \mathsf{A}\}$*. The canonical model of* $\mathcal{O}$ *is depicted below,*



The concept and role constructs in $\mathcal{EL}$ are more complex than DL-lite, which make it expressive enough to model professional ontologies like SNOMED CT [Donnelly, 2006].

The syntax of concept constructs in $\mathcal{EL}$ is as follows:

$$C, D \to \top \mid \{a\} \mid C \sqcap D \mid \exists P.C$$

Notice that the major difference between DL-lite and $\mathcal{EL}$ is that in $\mathcal{EL}$, existential restrictions are also applied to concepts, but inverse of role is not allowed. A common

feature shared by DL-lite and $\mathcal{EL}$ is that there are no disjunctions and no universal restrictions, which guarantees their tractabilities. Similar to DL-lite, TBoxs in $\mathcal{EL}$ ontologies consist of inclusion axiom of form $C \sqsubseteq D$ and ABoxs is a set of membership assertions.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ interprets the constructs in $\mathcal{EL}$ as follows:

1. $(\top)^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $(\bot)^{\mathcal{I}} = \emptyset$.
2. $(\{a\})^{\mathcal{I}} = \{a^{\mathcal{I}}\}$.
3. $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$.
4. $(\exists P.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in C^{\mathcal{I}}, (x, y) \in P^{\mathcal{I}}\}$

In addition, we say a DL language is a horn DL if its axioms can be represented by first-order horn clauses. Apparently, DL-lite$_{core}$ is a horn DL.

2.2.2 Existential Rules

Though DLs are widely accepted and successful underlying languages for ontologies, there are still a lot of real world problems that cannot be expressed by DLs, for example, roles (which are actually binary relations) could not satisfy the need of modeling relational data with more than two attributes (expressing these data in binary relation is very clumsy), so a family of more expressive language, existential rules (a.k.a Datalog±), is proposed.

Datalog± is a family of languages that extend datalog with useful modelling features, such as adding existential quantifiers to the rule heads (+ in ±) [Calì et al., 2012]. Datalog rules extended in this way are also called *tuple generating dependencies* (TGDs).

A *datalog clause* (or *datalog rule*) is a horn clause without function terms, and particularly, it satisfies that variables occur in the head also occur in the body.

A TGD r is an extension of a datalog clause, which is a first-order formula of form,

$$\forall \vec{x}.\forall \vec{y}.(\exists \vec{z}.\psi(\vec{x}, \vec{z}) \leftarrow \phi(\vec{x}, \vec{y}))$$

where ϕ and ψ are conjunctions of atoms. The formula ψ is the *head* of r, and the formula ϕ is the *body* of the σ. For simplicity, the universal quantifiers in the formula are usually omitted. The sets of atoms in the head and the body of σ are denoted by $\mathsf{head}(r)$ and $\mathsf{body}(r)$ respectively. Variables occurring both in the body and the head are called *frontier variables*.

Besides TGDs, Datalog± is also extended with two types of constraints, *negative constraints* and *equality-generating dependencies* (EGDs), which are also important elements for representing ontologies. Negative constraint is a first-order formula of form $\bot \leftarrow \forall \vec{x} \phi(\vec{x})$, where $\phi(\vec{x})$ is a conjunction of atoms. It is easy to see that it is a generalization of negative inclusions axiom in DL-lite. An EGD is a first-order formula of form $\forall \vec{x}.(x_i = x_j \leftarrow \phi(\vec{x}))$, where $\phi(\vec{x})$ is a conjunction of atoms and x_i, x_j are variables in $\vec{x}$. Note that if UNA is not considered, according to the semantics of first-order logic, the formula actually performs an implication that x_i and x_j are interpreted to the same object.

A Datalog± program consists of TGDs, EGDs and negative constraints. In some research problems, such as query answering, where consistency is not the main concern (or consistency is guaranteed), EGDs and negative constraints are usually ignored because they do not contribute to the reasoning results. By default, when we refer to existential rules, it means only TGD rules.

A *ground atom* (*ground fact*) is an atom whose terms are all constants. A *database* is set of ground atoms. Similar to a DL ontology, we also represent an ontology in existential rules as a pair $\mathcal{O} = (\Pi, \mathcal{D})$, where Π is a set of existential rules (or a Datalog± program containing constraints), $\mathcal{D}$ is a database. In fact, such ontologies taking databases into account are also called *knowledge bases* (KBs). Note that Π and $\mathcal{D}$ are similar to the TBox and ABox components in a DL ontology; when we use these notations to represent an ontology, we assume they can also represent TBoxs and ABoxs.

Next we introduce an important notion used to check the implication of dependencies, called *chase*. There are two types of chase rules, *restricted chase* and *oblivious chase*. Intuitively, in restricted chase, TGDs are applied in the sense of repairing database, i.e., they are applied only when they are not satisfied, while in the case of oblivious chase, TGDs are always applied to generate new information. Given that restricted chase

cannot fully capture the structure of models of existential rules, we focus on oblivious chase.

To describe chase, an extra infinite set of symbols called *labeled nulls* is considered, we denote it with N_N. Labeled nulls are used to represent 'fresh' terms introduced in the implications of rules, in other words, they are placeholders for unknown values, thus can be treated as special variables. An *instance* is a (possibly infinite) set of atoms that contains only constants and labeled nulls. Now we are ready to define (oblivious) chase for existential rules.

Let r be a TGD of form $\exists\vec{z}.\psi(\vec{x},\vec{z}) \leftarrow \phi(\vec{x},\vec{y})$. Given an instance I, r is said to be applicable to I if there exists a homomorphism σ from $\mathsf{body}(r)$ to I. Let r be applicable to I with homomorphism σ, σ' is defined as follows: for each variable $x \in \vec{x}$, $x\sigma = x\sigma'$, and for each variable $z \in \vec{z}$, $z\sigma' = z'$, where z' is a fresh label null not occurring in I. The result of applying r to I with σ, denoted $r^\sigma(I)$, is $I \cup \mathsf{head}(r)\sigma'$; Let H be the set of all homomorphisms from $\mathsf{body}(r)$ to I, we define $r(I) = \bigcup_{\sigma\in H} r^\sigma(I)$. Given a database $\mathcal{D}$, a set of TGDs Π, the *chase of level up to* k of $\mathcal{D}$ w.r.t Π, denoted $\mathsf{chase}^k(\mathcal{D},\Pi)$, is defined as follows: (1) $\mathsf{chase}^0(\mathcal{D},\Pi) = \mathcal{D}$; (2) for $k \geq 1$, $\mathsf{chase}^k(\mathcal{D},\Pi) = \bigcup_{r\in\Pi}(r(\mathsf{chase}^{k-1}(\mathcal{D},\Pi)))$. In fact, the chase result above is the canonical model of ontology $(\Pi,\mathcal{D})$.

It has been proved that query answering (as well as other classical reasoning tasks) under general existential rules is undecidable [Beeri and Vardi, 1981], so the syntax of the rule must be restricted (the - in $\pm$), which leads to a variety of decidable classes of Datalog$\pm$. We will discuss these classes in the next section.

2.3 Ontology Reasoning and Manipulation

Ontology-based data access is regarded as the essential element of new generation of information system. Efficient approaches to solve classical reasoning problems over expressive ontologies are important to the application of ODBA. Given the complexity of expressive languages and the scale of real-world data, finding these approaches is very challenging. In this section, we will introduce reasoning problems studied in this thesis and some typical methods solving these problems will be discussed.

2.3.1 Query Answering

Query answering is the main reasoning task in ontology-based data access.

A *conjunctive query* (CQ) q is a first-order formula of form $\exists \vec{y}.\phi(\vec{x}, \vec{y})$, where $\phi(\vec{x}, \vec{y})$ is a conjunction of atoms. We called $\vec{x}$ the answer variables of q. If $\vec{x}$ is empty, q is a *boolean conjunctive query* (BCQ). A CQ can also be represented as a datalog clause of form $\mathsf{Q}(\vec{x}) \leftarrow \exists \vec{y}.\phi(\vec{x}, \vec{y})$ where Q is special predicate called *query predicate*, which does not occur in the body of the query. A *union of conjunctive queries* (UCQ) is a set of conjunctive queries that have the same query predicate. Given the specialty of BCQs, it is often convenient to consider a BCQ as the set of atoms in it.

The satisfaction in an interpretation and the entailment by an ontology of a BCQ q are the same as in first-order logic: we write $\mathcal{I} \models q$ to denote that q is satisfied in $\mathcal{I}$, and q is entailed by an ontology $\mathcal{O}$, denoted $\mathcal{O} \models q$, if for every interpretation $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models q$. While for a non-BCQ q, let $(x_1, \ldots, x_n)$ be the set of answer variables of q, then a tuple $(a_1, \ldots, a_n) \subseteq \mathsf{const}(\mathcal{O})$ is a *certain answer* to q over $\mathcal{O}$, if $\mathcal{O} \models q[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$, where $q[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$ is obtained from q by substituting every answer variable x_i with a_i. Also, $\mathsf{ans}(q, \mathcal{O})$ is used to denote the set of all certain answers to q over $\mathcal{O}$.

In the setting of OBDA, the data components are usually separated from ontologies: an ontology containing only rules (or TBox) serves as a middle layer between user queries and different databases, which produces the notion of *ontology-mediate query answering* [Bienvenu et al., 2014c]. An *ontology-mediate query* (OMQ) is a pair (q, Π), where q is a conjunctive query and Π is an ontology without data, i.e., a set of rules or a set of TBox axioms. Then ontology-mediate query answering is the task of evaluating an ontology-mediate query over different databases.

As noticed in the definition of answering CQs, the task of answering non-BCQs can be straightforwardly reduced to the task of answering BCQs: for every possible answer $(a_1, \ldots, a_n)$, check whether the BCQ $q[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$ can be entailed. Therefore, in the studies of query answering for ontology, it is usual to consider BCQs only.

There are two major approaches for query answering over ontologies expressed in existential rules or Horn Description Logics, the forward chaining approaches and the query

rewriting approaches. The forward chaining approaches are about computing the canonical model of the ontology. As we have discussed, the canonical model of an ontology $\mathcal{O}$ is a model $\mathcal{I}$ of $\mathcal{O}$ that can be homomorphically embedded into any other model of $\mathcal{O}$, that is, for every $\mathcal{I}' \in \mathsf{Mod}(\mathcal{O})$, there exists a homomorphism h from $\mathcal{I}$ to $\mathcal{I}'$. Particularly, deciding the entailment of a BCQ q by a database $\mathcal{D}$ is equivalent to checking whether there exists a homomorphism from q to $\mathcal{D}$. According to the definition of homomorphism, the homomorphical relation is apparently transitive, that means, if there is a homomorphism from q to the canonical model of $\mathcal{O}$, then there is a homomorphism from q to all other models of $\mathcal{O}$; thus we have the following well-known result for query answering.

Theorem 2.1 ([Kontchakov et al., 2011])**.** *For every consistent ontology $\mathcal{O}$ in existential rules or horn DLs, and every BCQ q, $\mathcal{O} \models q$ if and only if $\mathcal{I}_\mathcal{O} \models q$, where $\mathcal{I}_O$ is the canonical model of $\mathcal{O}$.*

The procedure of computing canonical models, such as the chase in existential rules, is called *forward chaining*. Forward chaining based approaches highly rely on the termination of the forward chaining procedure, as we know the canonical model of an ontology can be infinite. Moreover, it is shown that computing a full model can be rather inefficient when query answers depend on only a small portion of the model [Benedikt et al., 2018]. Sometimes, the results of forward chaining are reserved in the databases to avoid repeated computations when there are multiple queries to be handled, the enriched results are also called *materializations* of databases.

In contrast to forward chaining based approaches, the key idea of query rewriting approaches is to reformulate queries and ontologies (without data) into other formalisms so that the querying task can be accomplished by existing mature data management systems, and moreover, the results can be applied to any databases.

In particular, when we are talking about the complexity of reasoning problems for ontologies, there are two different settings: (1) *combined complexity*, which takes all the components (including rules and data) in the ontology as inputs, and (2) *data complexity*, which fixes the rules in the ontology and only considers data as inputs. Hence, if an ontology-mediate query answering problem can be translated into a traditional query answering problem, then we can consider the problem as tractable w.r.t data complexity.

In the next subsection, we will focus on introducing the query rewriting approaches.

2.3.2 Query Rewriting

Definition 2.14 (Rewriting)**.** Let q be a CQ with query predicate Q and Π be a set of existential rules (or a TBox), a *datalog rewriting* of q w.r.t Π is a set of datalog clauses $\mathcal{R}_\mathcal{D} \cup \mathcal{R}_q$, where $\mathcal{R}_\mathcal{D}$ does not contain Q and $\mathcal{R}_q$ is a UCQ whose query predicate is Q, and for any database $\mathcal{D}$, it holds that

$$\mathsf{ans}(q, \Pi \cup \mathcal{D}) = \mathsf{ans}(\mathcal{R}_q, \mathcal{R}_\mathcal{D} \cup \mathcal{D})$$

If $\mathcal{R}_\mathcal{D}$ is empty, then $\mathcal{R}_q$ is a first-order rewriting (or UCQ rewriting) of q w.r.t Π.

When q is a BCQ, we can use the following condition alternatively,

$$\Pi \cup \mathcal{D} \models q \text{ iff } \mathcal{R}_\mathcal{D} \cup \mathcal{D} \cup \mathcal{R}_q \models \mathsf{Q}$$

For a result of first-order rewriting of BCQs, we sometimes call it a rewriting set, because it is literally a set of BCQs. We say a set of existential rules Π is *first-order rewritable* (resp. *datalog rewritable*) if for any BCQ q, there exists a finite first-order rewriting (resp. datalog rewriting) of q w.r.t Π.

There are two main kinds of rewriting methods, one is *resolution-based rewriting*, the other is *backward chaining rewriting*. Resolution-based rewriting is usually applied in DLs, which is done by introducing a set resolution calculus according to the fixed syntax of DL axioms [Trivela et al., 2015]. While current first-rewriting algorithms for existential rules are mostly based on backward chaining, such as the XRewrite algorithm [Gottlob et al., 2014c]. Our discussion focuses on rewriting methods based on backward chaining for existential rules.

Intuitively, in backward chaining, a rewriting of a query is computed by exhaustively searching replacements for atoms in the query according to the rules in ontology. Such searching process is performed in a 'backward' manner, that is, when an atom in the head of a rule r is searched, the atoms in the body of r will be considered for the next round of searching.

The classical backward chaining rewriting process is based on a unification operation between the current query and a rule head. However, normal unification is problematic in the rewriting in existential rules because the existential variables in a rule head may 'glue' multiple atoms in the head together, then it is inappropriate to unify single atoms. Hence, a specific unification operation for rewriting in existential rules is necessary.

Definition 2.15 (Piece-Unification [Baget et al., 2011])**.** For a BCQ q and an existential rule r, a piece-unifier between q and r is a tuple $\mu = (B, H, \tau)$, where $\emptyset \subset B \subseteq q$, $H \subseteq \mathsf{head}(r)$, and τ is a substitution from $\mathsf{terms}(q') \cup \mathsf{terms}(H)$ to $\mathsf{terms}(q')$ that satisfies:

1. $B\tau = H\tau$, i.e., τ is a unifier between B and H.
2. If z is an existential variable of r, then z can only be mapped to a variable z' that occurs only in B, i.e., $z' \in \mathsf{vars}(B)$ and $z' \notin \mathsf{vars}(q \setminus B)$.

In the above definition, B is a *piece* of q, that is, a minimal subset of atoms in q that have to be replaced together. Note that condition 2 excludes the cases where z is unified with a constant, with a variable in $\mathsf{head}(r)$ other than z, or with a variable in q occurring both inside and outside B. Particularly, the variables occur both in B and $q \setminus B$ are called *separating variables*.

A piece-unifier $\mu = (B, H, \tau)$ between q and r is *most general* if there does not exist another piece-unifier $\mu = (B, H, \tau')$ such that τ' is more general than τ.

Definition 2.16. Let q be a BCQ, r be an existential rule, and μ be a piece-unifier between q and r, the *one-step rewriting* of q by r with μ, denoted $\mathsf{FORew}_1(q, r, \mu)$, is a BCQ

$$(q \setminus B)\tau \cup body(r)\tau$$

By $\mathsf{FORew}_1(q, r)$, we denote the union of $\mathsf{FORew}_1(q, r, \mu)$ for all the most general piece-unifiers μ between q and r. For a set of queries $\mathcal{Q}$ and a set of rules Π, $\mathsf{FORew}_1(\mathcal{Q}, \Pi) = \{\mathsf{FORew}_1(q, r) \mid r \in \Pi, q \in \mathcal{Q}\}$.

A result of first-order rewriting needs not to be a most compact one. Given two BCQs q, q', we say q is *more general* than q' (or q' is *contained* in q), denoted by $q \preceq_h q'$, if there is a homomorphism from q to q'. It can be easily checked that if q is more general than q' and q' can be entailed by an ontology $\mathcal{O}$, then q is also entailed by $\mathcal{O}$. That

means for any query q that is not most general in a first-order rewriting result, q can always be removed from the set without affecting the completeness of the first-order rewriting. Given a set of BCQs $\mathcal{Q}$, a *cover* of $\mathcal{Q}$, denoted $\mathsf{cover}(\mathcal{Q})$, is a minimal subset $\mathcal{Q}'$ of $\mathcal{Q}$ such that each BCQ q in $\mathcal{Q}$ is contained in some BCQ q' in $\mathcal{Q}'$. We say $\mathcal{Q}$ is minimal if there does not exists $q, q' \in \mathcal{Q}$ s.t $q \preceq_h q'$. Apparently, a cover of $\mathcal{Q}$ is a minimal set of BCQs.

Obtaining minimality is one major concern in first-order rewriting [Venetis et al., 2016], because it may greatly reduce the size of a rewriting, thus affecting the efficiency of query answering. There even exist cases where a rewriting result of infinite size has a finite cover. However, the cost of computing a cover for a set of BCQs can be very expensive when the set is large. Note that deciding the $\preceq_h$ relation for two BCQs is an NP-complete problem, and to compute a cover of $\mathcal{Q}$, such checking may need to be done up to quadratic times w.r.t the size of $\mathcal{Q}$.

With the above knowledge, a general first-order rewriting process for existential rules can be obtained, as described with Algorithm 1.

Algorithm 1: [König et al., 2015b]

Input : A set of existential rules Π, a BCQ q

Output: A minimal rewriting set of q with Π

```
1 begin
2     F ← {q}; // result set
3     E ← {q}; // explore set
4     while E ≠ ∅ do
5         R ← FORew_1(E, Π); // One-step rewriting
6         C ← cover(R ∪ F); // Compute the cover set
7         E ← C \ F; // next exploration set
8         F ← C;
9     return F
```

Theorem 2.2 ([König et al., 2015b]). *Given a set of existential rules* Π *and a BCQ* q*, Algorithm 1 returns a minimal UCQ rewriting of* q *w.r.t* Π *and it terminates if* Π *is first-order rewritable.*

Note that as query answering in general existential rules is undecidable, algorithm 1 may not terminate and generates an infinite rewriting.

2.3.3 Decidable classes for Existential Rules

As we have discussed above, query answering in general existential rules are undecidable; to achieve decidability, restrictions to the rules need to be made. In this section, we will introduce decidable classes of existential rules. Here we only cover several important classes that are discussed in this thesis. For a more complete picture of decidable classes proposed, we refer interested readers to [Baget et al., 2011].

For a set of existential rules, we assume every rule in it has a distinct set of variables. We first introduce a fundamental class called *guarded existential rules* [Calì et al., 2012].

Definition 2.17 (Guarded)**.** An existential rule r is *guarded* if there is a body atom α of r satisfies that any variable v occurs in the body of r also occurs in α. α is said to be the guard of r. A set of existential rules is guarded if all its rules are guarded.

The idea behind the class of guarded rules is that if there is a guard for every rule, then the structure of the chasing model will be restricted to have bounded *treewidth*, which is a notion that describes how close is a graph to a tree. Notably, guarded existential rules are datalog rewritable [Gottlob et al., 2014e].

Next we introduce two abstract decidable classes of existential rules, which capture decidability through the process of forward chaining and backward chaining respectively.

Definition 2.18 (Finite Expansion Set [Baget et al., 2011])**.** Given a set of existential rules Π, we say Π is a *finite expansion set* (*fes*), if for any database $\mathcal{D}$, there exists a certain k such that for any BCQ q, $\mathsf{chase}^k(\mathcal{D}, \Pi) \models q$ if and only if $\Pi \cup \mathcal{D} \models q$.

Definition 2.19 (Finite Unification Set [Baget et al., 2011])**.** Given a set of existential rules Π, we say Π is a *finite unification set* (*fus*), if for any BCQ q, there exists a finite first-order rewriting of q w.r.t Π.

Note that these two abstract classes of existential rules are not recognizable [Baget et al., 2015]. However, many concrete classes of existential rules belong to these abstract classes have been presented in the literature.

For a set of existential rules to have a finite expansion over any database, an essential requirement is that it can guarantee that fresh constants are not infinitely generated in the chasing process. Existential rules enjoying such a property can usually be captured by acyclicity. Next we present two typical classes defined through acyclicity conditions in two different graphs specially constructed from the rules.

A *predicate position* is a pair of form $\mathsf{P}[i]$, where P is an n-ary predicate and $1 \leq i \leq n$. We say a variable v occurs at position $\mathsf{P}[i]$ if there is an atom $\mathsf{P}(t_1, \ldots, t_n)$ with $t_i = v$.

Given a set of existential rules Π, the *predicate position graph* of Π, denoted $G_p(\Pi)$, is a labelled directed graph whose vertices are the set of predicate positions of Π. The edges of $G_p(\Pi)$ are built as follows: for each rule $r \in \Pi$ and each frontier variable v of r, there is an edge from position $\mathsf{P}[i]$ in $\mathsf{body}(r)$ where v occurs at position $\mathsf{P}'[j]$ in $\mathsf{head}(r)$ where v occurs; and there is a special edge from position $\mathsf{P}[i]$ in $\mathsf{body}(r)$ where v occurs to positions in $\mathsf{head}(r)$ where existential variables occurs.

Definition 2.20 (Weakly-acyclic)**.** A set of existential rules is *weakly-acyclic* if its predicate position graph satisfies any special edge of the graph is not in a cycle.

If a set of existential rules is weakly-acyclic, it might be recursive, but it guarantees that the generation of any fresh constant is not depended on a previously obtained fresh constant, thus the chasing process will eventually stop. While predicate position graphs are defined based on predicates, the following graph tries to precisely represent the dependency relation between rules.

Definition 2.21. Given two existential rules r, r', we say r' *depends* on r, if there exists a database $\mathcal{D}$ satisfying,

1. there is a homomorphism σ from $\mathsf{body}(r)$ to $\mathcal{D}$,
2. there is a homomorphism σ' from $\mathsf{body}(r')$ to $r^{\sigma}(\mathcal{D})$,
3. $\mathsf{body}(r)\sigma' \not\subseteq \mathcal{D}$ and $\mathsf{head}(r)\sigma' \not\subseteq r^{\sigma}(\mathcal{D})$.

Given a set of existential rules Π, the *graph of rule dependency* for Π, is a directed graph whose nodes are the rules in Π, and where there is an edge from r to r' if r' depends on r. With this graph, we can immediately define a class where a rule will not be repeatedly applied in chasing.

Definition 2.22 (Acyclic Graph of Rule Dependency (aGRD))**.** A set of existential rules is aGRD if its graph of rule dependency is acyclic.

Existential rules in the following two classes are *fus*, i.e., they are first-order rewritable.

Definition 2.23 (Linear)**.** An existential rule is *linear* if its body contains only one atom. A set of existential rules is linear if all its rules are linear.

Though linear existential rules seem quite simple, they are sufficient to generalize inclusion axioms in DL-lite, and are more expressive. The next class is defined based on a variable marking process.

Definition 2.24. Given a set of existential rules Π, the *backward marking* of Π, is constructed as follows:

1. For every rule $r \in \Pi$, let v be a variable in $\mathsf{body}(r)$, if there exists an atom α in $\mathsf{head}(r)$ such that $v \notin \mathsf{vars}(\alpha)$, then each occurrence of v in $\mathsf{body}(r)$ is marked.
2. Apply exhaustively: for every rule $r \in \Pi$, if a variable in $\mathsf{body}(r)$ occurs at position π is marked, then for every rule $r' \in \Pi$ (r' possibly equals to r), for every variable v in $\mathsf{head}(r')$ occurs at π, the occurrences of v in $\mathsf{body}(r')$ are marked.

Definition 2.25 (Sticky)**.** For a set of existential rules Π, Π is *sticky* if in the backward marking, there is no rule $r \in \Pi$ such that a marked variable occurs in $\mathsf{body}(r)$ more than once.

Linear and sticky classes can be generalized by an abstract property called *backward shyness* [Thomazo, 2013], which roughly says that in the backward chaining rewriting of every rule, every generated variable doesn't occur in two atoms of a rewriting.

All concrete classes of existential rules we introduce above are tractable, that is, query answering in these classes is in polynomial time w.r.t data complexity.

2.3.4 Query Abduction

It has been argued that to meet the usability requirements set by users, an intelligent system should be equipped with the ability to explain its reasoning services [McGuinness

and Patel-Schneider, 1998]. In the scenario of OBDA, while *explaining positive query answers*, that is, finding the minimal causes for querying results, have been studied in DLs and existential rules [Kalyanpur et al., 2007, Ceylan et al., 2019], it is suggested that OBDA systems should also explain *negative query answers* [Borgida et al., 2008]. Negative answers are answers that cannot be entailed but are expected to occur in the querying result. The idea of explaining negative query answers is formalized by adopting abductive reasoning.

Definition 2.26 (Query Abduction Problem [Calvanese et al., 2013])**.** A *query abduction problem* (*QAP*) is of the form $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$, where Π is a set of existential rules, $\mathcal{D}$ is a set of ground facts (i.e., database), q is a boolean conjunctive query (the observation), and Σ is a set of predicates, called *abducibles*. An explanation to Λ is a set of facts (whose terms are constants or labeled nulls) $\mathcal{E}$ such that $\mathsf{sig}(\mathcal{E}) \subseteq \Sigma$ and $\Pi \cup \mathcal{D} \cup \mathcal{E} \models q$.

Unlike in the classical abduction problem, where a finite set of *hypotheses* (expressed in ground facts) is provided to search for possible explanations [Gottlob et al., 2007], the explanations in query abduction are not restricted by single atoms, but by the predicate symbols. In the above definition, fresh constants (labeled nulls) are allowed in the atoms of an explanation, which means there may exist infinite number of explanations to a query abduction problem.

Example 2.4. *Consider a QAP* $\Lambda = (\Pi, \emptyset, q, \Sigma)$*, where* Π *consists of a single rule,*

$$\mathsf{hasAncester}(x, z) \leftarrow \mathsf{hasParent}(x, y), \mathsf{hasAncestor}(y, z).$$

The above rule says the ancestors of a person's parent are also his ancestors. Let our observation q *be* $\mathsf{hasAncestor}(John, Ben)$ *and abducibles* Σ *be* $\{\mathsf{hasAncester}, \mathsf{hasParent}\}$*, then according to the definition, we have an infinite set of explanations of the form* $\{\mathsf{hasParent}(John, u_1), \ldots, \mathsf{hasParent}(u_{n-1}, u_n), \mathsf{hasAncestor}(u_n, Ben)\}$*, where* u_i *are fresh constants and* $n > 1$*.*

Though definition 2.26 presents us with the basic requirements for explanations, some explanations it admits may be unintuitive or meaningless. Specifically, explanations satisfying the following restrictions are desired [Elsenbroich et al., 2006].

Definition 2.27. Given a query abduction problem $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$, and an explanation $\mathcal{E}$ to Λ, we say:

(1) $\mathcal{E}$ is *consistent* if $\Pi \cup \mathcal{D} \cup \mathcal{E} \not\models \bot$.

(2) $\mathcal{E}$ is *relevant* if $\mathcal{E} \not\models q$.

(3) $\mathcal{E}$ is *explanatory* if $\Pi \cup \mathcal{D} \not\models q$.

Intuition regarding these restrictions on explanations is obvious: an explanation should be consistent with the current knowledge, because an inconsistent theory logically entails everything. For any observation, a trivial explanation is the observation itself; nevertheless such an explanation does not help to explain the situation. If the observation is already entailed by the current knowledge, any set of facts can be an explanation to the abduction problem.

Apart from the above considerations, one of the most concerning issues in computing explanations is compactness. We hope an explanation will be as simple as possible, as we do not want excess hypothesizing. Several studies have discussed about how explanations should be evaluated and compared [Du et al., 2011, Du et al., 2014, Soler-Toscano, 2019].

Because an explanation is allowed to have non-ground facts, the minimality of explanations should not be defined by the simple subset inclusion relation. Consider two explanations $\mathcal{E}_1 = \{\mathsf{A}(a, u)\}$ and $\mathcal{E}_2 = \{\mathsf{A}(a, u'), \mathsf{B}(a)\}$, where u, u' are fresh constants, $\mathcal{E}_1$ is not a subset of $\mathcal{E}_2$, however, $\mathcal{E}_1$ is intuitively smaller than $\mathcal{E}_2$, as u and u' are essentially equivalent.

When labeled nulls are considered in atoms, we extend substitutions to include mappings of labeled nulls, that is, a substitution is a mapping function $\sigma : \mathsf{N}_V \cup \mathsf{N}_N \to \mathsf{N}_V \cup \mathsf{N}_I \cup \mathsf{N}_N$. . A *renaming* is a substitution ρ that substitutes a variable (resp. labeled null) to another variable (resp. labeled null) such that for each pair t, t', if $t \neq t'$, then $t\rho \neq t'\rho$.

For two sets of facts $\mathcal{A}, \mathcal{A}'$, we denote $\mathcal{A} \preceq_m \mathcal{A}'$ if there exists a renaming ρ such that $\mathcal{A}'\rho \subseteq \mathcal{A}$. And $\mathcal{A} \prec_m \mathcal{A}'$ if $\mathcal{A}'\rho \subset \mathcal{A}$. An explanation $\mathcal{E}$ to a QAP Λ is *minimal* if $\mathcal{E}' \preceq_m \mathcal{E}$ implies $\mathcal{E} \preceq_m \mathcal{E}'$ for all explanations $\mathcal{E}'$ to Λ. The set of all minimal and consistent explanations to Λ up to renaming is denoted by $\mathsf{expl}(\Lambda)$.

Only requiring explanations to be minimal is not sufficient to have a compact result for query abduction. As in the following examples, minimal explanations can be redundant.

Example 2.5. *Consider a QAP* $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$*, where* Π *consists of rules,*

$$\exists y.\mathsf{memberOf}(x, y), \mathsf{Department}(y) \leftarrow \mathsf{Employee}(x).$$

$$\mathsf{Employee}(x) \leftarrow \mathsf{worksFor}(x, y).$$

Let $\mathcal{D} = \{\mathsf{Department}(dev), \mathsf{Person}(John), \mathsf{Person}(Tom)\}$*,* $\Sigma = \{\mathsf{worksFor}\}$ *and* $q = \{\exists x.\mathsf{memberOf}(John, x)\}$*. Then there are four minimal explanations to* Λ*,* $\mathcal{E}_1 = \{\mathsf{worksFor}(John, John)\}$*,* $\mathcal{E}_2 = \{\mathsf{worksFor}(John, Tom)\}$*,* $\mathcal{E}_3 = \{\mathsf{worksFor}(John, dev)\}$*, and* $\mathcal{E}_4 = \{\mathsf{worksFor}(John, u)\}$*, where* u *is a fresh constant. Among the four explanations,* $\mathcal{E}_3$ *is abnormal, because John should work for a person other than a department. With* $\mathcal{E}_1$ *and* $\mathcal{E}_2$*, we are uncertain about who John is actually working for. In fact, the uncertainty reflected by* $\mathcal{E}_1$ *and* $\mathcal{E}_2$ *can be expressed by* $\mathcal{E}_4$*.*

If we assume a large database for the above example, a great number of explanations will be computed by combination of constants occurring in the database, while most such explanations can be covered by a general explanation. Therefore, explanations should be compared semantically. To address the issue, the author in [Du et al., 2014] proposed to compute a particular set of explanations, called *representative explanation*. Similar to the containment relation between two BCQs, given two explanations $\mathcal{E}$ and $\mathcal{E}'$, $\mathcal{E}'$ is said to be *subsumed* by $\mathcal{E}$ if $\mathcal{E} \preceq_h \mathcal{E}'$.

Definition 2.28 (Representative explanations[Du et al., 2014])**.** Given a QAP Λ, a representative explanation to Λ is a minimal explanation $\mathcal{E}$ to Λ such that for all minimal explanation $\mathcal{E}'$ to Λ , if $\mathcal{E}' \preceq_h \mathcal{E}$ then $\mathcal{E} \preceq_h \mathcal{E}'$.

In example 2.5, there is only one representative explanation to the abduction problem, which is $\mathcal{E}_4$, all other explanations are subsumed by $\mathcal{E}_4$.

It should be noted that the set of all representative explanations to a QAP may still be infinite. As we can see in Example 2.4, every explanation of the form $\{\mathsf{hasParent}(John, u_1), \ldots, \mathsf{hasParent}(u_{n-1}, u_n), \mathsf{hasAncestor}(u_n, Ben)\}$ is a representative explanation.

2.3.5 Inconsistency-Tolerant Query Answering

Previous discussion on query answering is based on one important assumption that the considered ontology is consistent. However, if an ontology is not consistent, query

answering over it is trivial, because according to the semantics of first-order logic, an unsatisfiable theory can entail everything. So novel approaches different from classical query answering need to be proposed.

In the database research field, a database is inconsistent if it violates certain integrity constraints predefined for the database, which is common in the applications that rely on multiple sources of data. To handle inconsistencies, one trivial approach is to fix the database so that constraints can be satisfied. However, such an approach requires modification of data, which may not be encouraged in some scenarios. Furthermore, it is usually difficult to decide the optimal way to fix the database. For these reasons, a concept called *consistent query answering* (CQA) [Chomicki, 2007] is proposed, that is, querying database with the existence of inconsistencies.

One fundamental notion in CQA is *repair*, which is a possible result that are close to the original inconsistent database after fixing inconsistencies. A query answer is a *consistent answer* to a query over an inconsistent database if the answer is a query answer to the query over all repairs of the database.

The idea of CQA is exploited in the query answering of inconsistent ontologies [Lembo et al., 2010], which gives rise to so called *inconsistency-tolerant query answering*. Repairs of ontologies are more complicated than repairs of databases as ontologies contains not only data (ABox assertions), but also rules (TBox axioms) representing background knowledge. To simplify the problem, data (ABox) repairs are adopted for inconsistency-tolerant query answering, that is, repairs are obtained by changing only the data (ABox) component of the ontology. Such repairs make sense, because background knowledge is usually considered reliable, while data is collected from various source with different means, thus are more likely to contain errors.

We focus on data (ABox) repairs for ontologies, the formal definition of which is given below. $\mathcal{D}$ is said to be Π-consistent if $\Pi \cup \mathcal{D}$ is consistent.

Definition 2.29. Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, a subset $\mathcal{D}'$ of $\mathcal{D}$ is a repair of $\mathcal{O}$ if $(\Pi, \mathcal{D}')$ is consistent and there does not exist another $\mathcal{D}''$ such that $\mathcal{D}' \subset \mathcal{D}'' \subseteq \mathcal{D}$ and $\mathcal{D}''$ is Π-consistent.

Let $\mathcal{O} = (\Pi, \mathcal{D})$, we use $\mathsf{repair}_\Pi(\mathcal{D})$ to denote the set of all repairs of $\mathcal{O}$. With the definition of repair, we are now able to introduce inconsistent-tolerant semantics for ontologies.

Definition 2.30 (AR semantics)**.** Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, a BCQ q, q is entailed by $\mathcal{O}$ under the AR semantics if for every repair $\mathcal{B}$ of $\mathcal{O}$, we have $\Pi \cup \mathcal{B} \models q$.

As we can see, AR semantics is analogous to CQA, a query is considered entailed only when it can be obtained from each repair. The following semantics follows a more cautious principle: only assertions that occur in every repair are considered for query answering.

Definition 2.31 (IAR semantics)**.** Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, a BCQ q, q is entailed by $\mathcal{O}$ under the IAR semantics if $\Pi \cup \mathcal{B}_I \models q$, where $\mathcal{B}_I$ is the intersection of all repairs of $\mathcal{O}$.

In AR and IAR semantics, some entailed assertions are ignored in query answering because their causes do not occur in all repairs, however, these consequences may not violate the constraints in the ontology and could be useful information in some scenarios. Given this consideration, it is necessary to introduce more relaxed semantics. Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, the *closed logical consequence* of $\mathcal{O}$ is defined as the set $\mathsf{clc}_\Pi(\mathcal{D}) = \{\alpha \in \mathsf{HB}(\mathcal{O}) \mid \mathcal{D}' \text{ is } \Pi\text{-consistent}, \mathcal{D}' \subseteq \mathcal{D}, \Pi \cup \mathcal{D}' \models \alpha\}$.

Definition 2.32 (CAR semantics)**.** Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, a BCQ q, q is entailed by $\mathcal{O}$ under the CAR semantics if q is entailed by $\mathcal{O}' = (\Pi, \mathsf{clc}_\Pi(\mathcal{D}))$ under the AR semantics.

Analogously, we can obtain a variant from IAR semantics.

Definition 2.33 (ICAR semantics)**.** Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, a BCQ q, q is entailed by $\mathcal{O}$ under the ICAR semantics if q is entailed by $\mathcal{O}' = (\Pi, \mathsf{clc}_\Pi(\mathcal{D}))$ under the IAR semantics.

For notation, given an inconsistency-tolerant semantics γ, we use $\mathcal{O} \models_\gamma q$ to denote that q is entailed by $\mathcal{O}$ under γ.

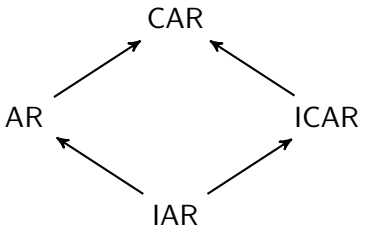


FIGURE 2.1: Partical order over inconsistency-tolerant semantics

Example 2.6. *Consider the DL ontology* $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ *where*

$$\mathcal{T} = \{\mathsf{A} \sqsubseteq \mathsf{D}, \exists \mathsf{R} \sqsubseteq \mathsf{B}, \mathsf{A} \sqsubseteq \mathsf{C}, \mathsf{B} \sqsubseteq \mathsf{C}, \mathsf{A} \sqsubseteq \neg \mathsf{B}\}$$
$$\mathcal{A} = \{\mathsf{A}(a), \mathsf{R}(a, c), \mathsf{A}(b)\}$$

Let $q = \{\mathsf{C}(a)\}$ *be a BCQ. There are two repairs of* $\mathcal{O}$, $\mathcal{B}_1 = \{\mathsf{A}(a), \mathsf{A}(b)\}$ *and* $\mathcal{B}_2 = \{\mathsf{R}(a, c), \mathsf{A}(b)\}$. *It can be checked that both repairs entail* q_1, *thus we have* $\mathcal{O} \models_{\mathsf{AR}} q_1$, *while* $\mathcal{O} \not\models_{\mathsf{IAR}} q_1$ *because* $\mathsf{A}(b)$ *is the only assertion occurs in both repairs. Let* $q' = \{D(a)\}$, *as* $\mathsf{clc}_{\mathcal{T}}(\mathcal{A}) = \{\mathsf{A}(a), \mathsf{R}(a, c), \mathsf{B}(a), \mathsf{D}(a), \mathsf{A}(b), \mathsf{D}(b), \mathsf{C}(a), \mathsf{C}(b)\}$, *according to the definitions, we have* $\mathcal{O} \not\models_{\mathsf{AR}} q'$ *but* $\mathcal{O} \models_{\mathsf{CAR}} q'$.

According to the definitions, it is not hard to see that these four semantics are comparable (not pairwise), they actually form a 'lattice shape' partial order, where IAR is the lower bound and CAR is the upper bound.

Theorem 2.3 ([Lembo et al., 2010])**.** *Given an ontology* $\mathcal{O}$, *for any BCQ* q, *we have*

- $\mathcal{O} \models_{\mathsf{IAR}}$ *implies* $\mathcal{O} \models_{\mathsf{AR}}$ *and* $\mathcal{O} \models_{\mathsf{ICAR}}$.
- $\mathcal{O} \models_{\mathsf{AR}}$ *implies* $\mathcal{O} \models_{\mathsf{CAR}}$.
- $\mathcal{O} \models_{\mathsf{ICAR}}$ *implies* $\mathcal{O} \models_{\mathsf{CAR}}$.

Besides the above four semantics, a number of novel semantics have been proposed. For example, [Bienvenu et al., 2014a] introduces preferences over repairs so that only preferred repairs are considered in query answering. For more details about these variants, interested readers are referred to the survey in [Bienvenu and Bourgaux, 2016].

Particularly, a general modifier-based framework is proposed to generalize all these semantics [Baget et al., 2016], in which every inconsistency-tolerant semantics can be constructed by different modifiers and inference strategies, corresponding to the processes of selecting repairs and answering queries over repairs with different cautiousness.

Regarding the computation of query answers under inconsistency-tolerant semantics, several approaches have been proposed for simple DLs, such as rewriting-based approaches [Lembo et al., 2015, Tsalapati et al., 2016]. Intuitively, rewriting-based approaches further rewrite the result of basic rewriting so that wrong answers violating the constraints are filtered. These approaches can be naturally adapted into the framework of ODBA, thus making it efficient and feasible; however they are only applicable to IAR or ICAR. Due to the high complexity of AR semantic (even for DL-lite$_{core}$, query answering under AR is CoNP-complete with respect to data complexity), there is still no efficient algorithm for querying answering under AR semantics; [Bienvenu et al., 2014b] presented an approach that encodes the problem into an SAT program (propositional formulas), which is then solved by SAT solvers.

2.3.6 Forgetting

Forgetting is a particular form of reasoning that eliminates or hides certain symbols from a set of logical formulas, while the result still reserves a certain kind of equivalence with the original formulas. Forgetting is firstly introduced in classical logic [Lin and Reiter, 1994, Lang et al., 2003a] as a technique to eliminate variables from a set of propositional formulas. In recent years, a variety of potential applications of forgetting in ontology maintenance were found, which lead to the extensive study of forgetting in different DLs [Wang et al., 2008, Konev et al., 2009, Koopmann and Schmidt, 2015] and existential rules [Wang et al., 2018]. In the following, we will introduce two basic definitions of forgetting for ontologies.

The first one is *model-based forgetting*, it requires the strongest equivalence between forgetting results and the original ontology. Informally, model-based forgetting produces a weaker ontology that the projections of its models and models of the original ontology on certain symbols are the same.

Given two interpretations $\mathcal{I}$ and $\mathcal{I}'$, we say $\mathcal{I}$ and $\mathcal{I}'$ agree on all predicate symbols except those in Σ, denotes $\mathcal{I} \sim_\Sigma \mathcal{I}'$, if it satisfies that

(1) $\mathcal{I}$ and $\mathcal{I}'$ have the same domain universe,

(2) for every predicate $P \notin \Sigma$, $P^{\mathcal{I}} = P^{\mathcal{I}'}$.

For two set of interpretations $\mathbf{I}$ and $\mathbf{I}'$, we say $\mathbf{I}$ and $\mathbf{I}'$ are equivalent over Σ, denoted $\mathbf{I} \equiv_\Sigma \mathbf{I}'$, if it satisfies that there is an interpretation $\mathcal{I}$ in $\mathbf{I}$ if and only if there is an interpretation $\mathcal{I}'$ in $\mathbf{I}'$ such that $\mathcal{I} \sim_\Sigma \mathcal{I}'$.

Definition 2.34. Given a consistent ontology $\mathcal{O}$, a signature Σ, an ontology $\mathcal{O}'$ is a result of model-based forgetting of $\mathcal{O}$ about Σ, if it satisfies that

(1) $\mathcal{O} \models \mathcal{O}'$,

(2) $\mathsf{sig}(\mathcal{O}') \subseteq \mathsf{sig}(\mathcal{O}) \setminus \Sigma$,

(3) $\mathsf{Mod}(\mathcal{O}) \equiv_\Sigma \mathsf{Mod}(\mathcal{O}')$.

The second condition in the definition is the purpose of forgetting and the third states the equivalence result based on models. Note that the first condition is necessary, as it forces the forgetting result to be strictly weaker than the original ontology, otherwise, any information outside $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$ can be added to the result.

Example 2.7. *Consider* $\mathcal{O} = \{\Pi, \emptyset\}$*, where* Π *consists of rules*

$$\exists y.\mathsf{B}(x,y) \wedge \mathsf{C}(x,y) \leftarrow \mathsf{A}(x), \quad \exists z.\mathsf{E}(y,z) \leftarrow \mathsf{C}(x,y) \wedge \mathsf{D}(x),$$
$$\exists z.\mathsf{E}(y,z) \leftarrow \mathsf{E}(x,y), \quad \mathsf{A}(x) \leftarrow \mathsf{E}(x,y), \quad \mathsf{D}(x) \leftarrow \mathsf{F}(x).$$

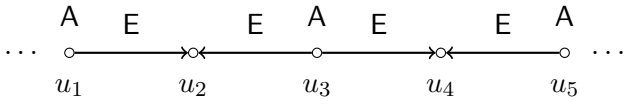
Then a result of model-based forgetting of $\mathcal{O}$ *about* $\{\mathsf{B}, \mathsf{D}\}$ *is* $(\Pi', \emptyset)$*, where* Π' *has rules,*

$$\exists y.\mathsf{C}(x,y) \leftarrow \mathsf{A}(x), \quad \exists z.\mathsf{E}(y,z) \leftarrow \mathsf{C}(x,y) \wedge \mathsf{F}(x).$$
$$\exists z.\mathsf{E}(y,z) \leftarrow \mathsf{E}(x,y), \quad \mathsf{A}(x) \leftarrow \mathsf{E}(x,y).$$

Because of the strong conditions, it is not hard to see that the model-based forgetting results of an ontology are all equivalent; in other words, a result of model-based forgetting is always unique up to equivalence.

Though preserving model equivalence guarantees the close relation between the ontology and the forgetting results, a result of model-based forgetting is often inexpressible in first-order logic [Wang et al., 2018].

Example 2.8. *Consider* $\mathcal{O} = (\Pi, \mathcal{D})$ *where* Π *is from the above example, and* $\mathcal{A} = \{\mathsf{A}(a), \mathsf{C}(a, b)\}$*, then it is impossible to express a result of model-based forgetting of* $\mathcal{O}$ *about* $\{\mathsf{E}\}$ *in first-order logic. The rule* $\exists z.\mathsf{E}(y, z) \leftarrow \mathsf{E}(x, y)$ *induces an infinite chain for the interpretations of* $\mathcal{O}$*. Because* A *inherits this structure from* E *with the rule* $\mathsf{A}(x) \leftarrow \mathsf{E}(x, y)$*, if* E *is forgotten from the ontology, this chain cannot be reproduced for* A *using only first-order formulas.*



While model-based forgetting is unnecessarily strong for OBDA, another definition of forgetting based on query answering is more suitable for scenarios in OBDA. The definition of *query-based forgetting* can be obtained by changing the third requirement of definition 2.34.

Definition 2.35. Given a consistent ontology $\mathcal{O}$, a signature Σ, an ontology $\mathcal{O}'$ is a result of query-based forgetting of $\mathcal{O}$ about Σ, if it satisfies that

1. $\mathsf{sig}(\mathcal{O}') \subseteq \mathsf{sig}(\mathcal{O}) \setminus \Sigma$,
2. for any BCQ q not containing predicates in Σ, $\mathcal{O} \models q$ if and only if $\mathcal{O}' \models q$.

It should be noted that the above definitions consider forgetting for a whole ontology containing a database (i.e., a knowledge base), while in the cases where forgetting is defined for only rules (or TBox), the query-based definition is a bit different. Specifically, for a set of existential rules Π, Π' over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$ is a forgetting of Π about Σ, if for any q and $\mathcal{D}$ over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$, $\Pi \cup \mathcal{D} \models q$ iff $\Pi' \cup \mathcal{D} \models q$.

According to the definition 2.35, there exists a result of query-based forgetting for the case in example 2.8, because there is no fact related E in the database $\mathcal{D}$ and no fact about E can be entailed from the ontology, which means the infinite chain will not occur in the canonical model of $\mathcal{O}$, then to obtain a result of query-based forgetting, it is would be sufficient to just eliminate all the rules where E occurs.

Next we briefly discuss about the computation of forgetting. When considering ontologies with databases, to express the forgetting results, the syntax of database need to be extended to allow label nulls. For example, let $\mathcal{O} = (\{\mathsf{A} \sqsubseteq \exists \mathsf{R}\}, \{\mathsf{A}(a)\})$, say we want to forget A from $\mathcal{O}$, then to capture the entailment $\mathcal{O} \models \exists x.\mathsf{R}(a, x)$, an extended ABox where labeled nulls are allowed must be introduced, in this example, an suitable forgetting result would be $(\emptyset, \{\mathsf{R}(a, u)\})$.

The general idea to compute forgetting results for an ontology is to first unfold the formulas in the ontology, that is, exhaustively compute all the logical consequence of the ontology, then delete those formulas that are relevant to the forgotten symbols. In DLs, the unfolding process is usually achieved by *resolution* [Bachmair et al., 2001], a well-known technique developed in theorem proving, used to saturate a set of logical formulas by a set of predefined resolution calculus. While for existential rules, the authors in [Wang et al., 2018] proposed an unfolding process based on piece-unification. However, these unfolding processes do not necessarily terminate if forgetting results of the ontology are not guaranteed to exist. It is shown that for an ontology in DL-lite$_{core}$, a result of query-based forgetting always exists if only concepts are forgotten from the ontology, which can be computed by the following syntax-based procedure.

Theorem 2.4 ([Wang et al., 2008])**.** *Let $\mathcal{O}' = (\mathcal{T}', \mathcal{A}')$ be the ontology returned by Algorithm 2, then $\mathcal{O}'$ is a result of query-based forgetting of $\mathcal{O}$ about Σ, furthermore, $\mathcal{T}'$ is a result of model-based forgetting of $\mathcal{T}$ about Σ.*

Algorithm 2: [Wang et al., 2008]

Input : A consistent ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ in DL-lite$_{core}$, a set of concept names Σ

Output: A query-based forgetting of $\mathcal{O}$ about Σ

begin
  for A *in* Σ **do**
    If $A \sqsubseteq A \in \mathcal{T}$, then delete $\{A \sqsubseteq A\}$ from $\mathcal{T}$;
    If $A \sqsubseteq \neg A \in \mathcal{T}$, then delete axioms of form $A \sqsubseteq C$ and $B \subseteq \neg A$ from $\mathcal{T}$, and replace each axiom of $B \sqsubseteq A$ in $\mathcal{T}$ by $B \sqsubseteq \neg B$;
    Replace each axiom $B \sqsubseteq \neg A$ in $\mathcal{T}$ with $A \sqsubseteq \neg B$;
    For each axiom $B_i \sqsubseteq A (1 \leq i \leq m)$ in $\mathcal{T}$ and each axiom $A \sqsubseteq C_j (1 \leq j \leq n)$ in $\mathcal{T}$, where B_i is a concept name and C_i is a general concept, if $B_i \sqsubseteq C_i$ is not in $\mathcal{T}$, then add it to $\mathcal{T}$;
    for $A(a) \in \mathcal{A}$ **do**
      If $A \sqsubseteq B \in \mathcal{T}$, where B is a concept name, then add $B(a)$ to $\mathcal{A}$;
      If $A \sqsubseteq \exists P^- \in \mathcal{T}$, then add $P(u_{P^-}, a)$ to $\mathcal{A}$;
      If $A \sqsubseteq \exists P \in \mathcal{T}$, then add $P(a, u_P)$ to $\mathcal{A}$;
    Delete all axioms where A occurs from $\mathcal{O}$;
  return $\mathcal{O}$

Chapter 3

Efficient Datalog Rewriting for Existential Rules

Rewriting approaches are particularly promising for ontology-mediated query answering as they allow the task to be implemented on top of existing highly-optimised database querying engines. While many algorithms and systems have been developed for various description logics [Pérez-Urbina et al., 2010, Eiter et al., 2012, Zhou et al., 2015, Venetis et al., 2016], particularly for DL-Lite and $\mathcal{EL}$ [Kontchakov et al., 2010, Stefanoni et al., 2013, Trivela et al., 2015, Hansen et al., 2015, Bienvenu et al., 2017], it is challenging to extend them to more general existential rules, which allow predicates of arbitrary arities (instead of only unary and binary predicates) and variable permutations in the rules.

Existing query rewriting systems for existential rules are typically based on first-order rewriting. A limitation of such an approach is that it can only handle ontologies and queries that are first-order rewritable. Well-accepted first-order rewritable classes in existential rules are the linear and sticky classes. Yet many practical ontologies do not necessarily fall into these classes, such as some ontologies formulated in $\mathcal{EL}$.

Even for ontologies and queries that are first-order rewritable, the results of rewriting can suffer from a significant blow up due to the scale and complexity of ontologies and the lengths of queries. It has been shown by experiments that handling large UCQs (what first-order rewritings are) is very difficult and inefficient in DBMSs [Rosati and Almatelli, 2010, Bienvenu et al., 2017].

Let us consider the following example where the result of first-order rewriting is significantly large.

Example 3.1. *Consider the following existential rules* Π*, where* $n \geq 0$*,*

$$r_{a0} : \exists z.\mathsf{A}_0(x,z) \leftarrow \mathsf{A}_1(x,y),\ \ldots,\ r_{an} : \exists z.\mathsf{A}_n(x,z) \leftarrow \mathsf{A}_{n+1}(x,y),$$
$$r_{b0} : \exists z.\mathsf{B}_0(x,z) \leftarrow \mathsf{B}_1(x,y),\ \ldots,\ r_{bn} : \exists z.\mathsf{B}_n(x,z) \leftarrow \mathsf{B}_{n+1}(x,y).$$

Let $q = \{\mathsf{A}_0(x,y), \mathsf{B}_0(x,z)\}$ *be a BCQ, then a first-order rewriting* $\mathcal{Q}$ *of* q *w.r.t* Π *is a set of BCQs (UCQ) of form* $\{\mathsf{A}_i(x,y), \mathsf{B}_j(x,z)\}$*, where* $0 \leq i,j \leq n+1$*.* $\mathcal{Q}$ *is minimal and the size of* $\mathcal{Q}$ *is* $(n+1)^2$*.*

It is often the case that in first-order rewriting, to achieve completeness, one has to list all combinations of possible replacements of atoms in the query, which unavoidably leads to rewritings of huge sizes. However, if we consider taking datalog as the target query language, the result of rewriting will be more compact. In the above example, a possible datalog rewriting of q w.r.t Π contains only $4n+3$ rules,

$$\mathsf{Q} \leftarrow \mathsf{A}_0(x,y) \wedge \mathsf{B}_0(x,z),$$
$$\mathsf{Q} \leftarrow \mathsf{P}_{r_{a0}}(x) \wedge \mathsf{B}_0(x,z), \quad \mathsf{Q} \leftarrow \mathsf{A}_0(x,y) \wedge \mathsf{P}_{r_{b_0}}(x),$$
$$\mathsf{P}_{r_{ai}}(x) \leftarrow \mathsf{A}_{i+1}(x,y), \quad \mathsf{P}_{r_{ai}}(x) \leftarrow \mathsf{P}_{r_{a(i+1)}}(x,y), \qquad (0 \leq i \leq n)$$
$$\mathsf{P}_{r_{bi}}(x) \leftarrow \mathsf{B}_{i+1}(x,y), \quad \mathsf{P}_{r_{bi}}(x) \leftarrow \mathsf{P}_{r_{b(i+1)}}(x,y), \qquad (0 \leq i \leq n)$$

It is shown for description logics that executing (non-recursive) datalog rewritings is much more feasible for DBMSs than equivalent first-order rewritings [Hansen et al., 2015]. All ontologies and queries that are first-order rewritable are trivially datalog rewritable, and more datalog rewritable classes are known, such as the guarded existential rules [Gottlob et al., 2014d]. However, existing research on datalog rewriting of existential rules are mostly theoretical [Gottlob and Schwentick, 2012, Bienvenu et al., 2014c] (refer to [Ahmetaj et al., 2018] for a detailed discussion).

Though several algorithms and systems have been developed for datalog rewriting for various description logics [Eiter et al., 2012, Trivela et al., 2015, Hansen et al., 2015], very few systems have been developed for datalog rewriting over more general existential

rules. A notable exception is ChaseGoal [Benedikt et al., 2018], which however, relies on the termination of the chase procedure.

In this chapter, we fill the gap by presenting both a practical approach and a prototype system for datalog rewriting and query answering over a wide range of ontologies expressed in existential rules. Our algorithm is based on the notion of unfolding [Wang et al., 2018] and to achieve compactness of rewriting, we separate the results of unfolding into short rules by introducing the so-called separating predicates and reusing such predicates when possible. While such a rewriting process may not terminate, we move on to identify classes of ontologies where the rewriting process terminates, introducing a class by combining existing well-accepted classes. And we introduce an efficient algorithm for computing the datalog rewritings. Finally, we implemented a prototype system, *Drewer*, and experiments show that it is able to handle a wide range of benchmarks in the literature. Moreover, Drewer shows superior or comparable performance over state-of-the-art systems on both the compactness of rewriting and the efficiency of query answering.

3.1 Compact Datalog Rewriting

In this section, we introduce a compact datalog rewriting approach. Before we present our approach, we redefine some notions for datalog rewriting in existential rules.

It is known that a set of existential rules Π can be transformed into a set of rules Π' whose heads are singletons in a way that preserves query answering [Gottlob et al., 2014c]. Hence, in the following discussions, we assume all existential rules have atomic heads.

For two set of rules Π, Π', we say Π and Π' are *fact-equivalent* over a set of predicates Σ, denoted $\Pi \equiv^F_\Sigma \Pi'$, if for any fact α over Σ, $\Pi \models \alpha$ iff $\Pi' \models \alpha$,

Definition 3.1 (Fact-preserving datalog rewriting)**.** A *fact-preserving datalog rewriting* (FPDR) of a set of existential rules Π is a datalog program Π_d that preserves fact derivation, that is, $\Pi_d \equiv^F_{\mathsf{sig}(\Pi)} \Pi$; and it is a *strong* FPDR if additionally, $\Pi \models \Pi_d$.

Recall that a CQ can be represented by a datalog rule $\mathsf{Q}(\vec{x}) \leftarrow \phi(\vec{x}, \vec{y})$. Because existential rules can fully cover datalog rules, we consider the query and the ontology in an OMQ as a whole, that is, an OMQ is of the form $Q = \Pi \cup \{q(\vec{x})\}$, where Π is set of

existential rules, and $q(\vec{x})$ is a CQ. Datalog rewriting for an OMQ is relaxed to preserve only the query answers.

Definition 3.2 (Query-preserving datalog rewriting)**.** A *query-preserving datalog rewriting* (QPDR) of an OMQ $Q = \Pi \cup \{q(\vec{x})\}$ is a datalog program Π_{Q} such that $\Pi_{\mathsf{Q}} \equiv^{F}_{\{\mathsf{Q}\}} \Pi$.

Clearly, an FPDR of Π is also a QPDR of $\Pi \cup \{q(\vec{x})\}$ for any $q(\vec{x})$, but the converse does not necessarily hold.

Now we are ready to present the approach, which is based on the notion of unfolding in existential rules [Wang et al., 2018].

Each rule r is assumed to have form $\exists \vec{z}.\psi(\vec{x}, \vec{z}) \leftarrow \phi(\vec{x}, \vec{y})$, we use $\vec{x}_r$, $\vec{y}_r$ and $\vec{z}_r$ to denote $\vec{x}$, $\vec{y}$, $\vec{z}$ in r respectively. $\vec{x}_r$ and $\vec{z}_r$ are exactly frontier variables and existential variables of r. A rule r can be *unfolded* by a rule r' if there exists a piece unifier $\mu = (B, \tau)$ of $\mathsf{body}(r)$ and $\mathsf{head}(r')$, and the result is $\mathsf{unf}^{\mu}(r, r')$:

$$\exists \vec{z}.[\mathsf{head}(r)\tau' \wedge \mathsf{head}(r')\tau] \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \bigwedge \mathsf{body}(r')\tau'$$

where $\vec{z}$ consists of all the variables in the head but not in the body, and τ' is a *safe extension* of τ by substituting variables $\vec{z}_r \cup \vec{y}_{r'}$ with fresh variables.

Example 3.2. *Let* $\Pi_{ex1} = \{r_1 : \exists y.\mathsf{A}(x, y) \leftarrow \mathsf{B}(x, z),\ r_2 : \exists z.\mathsf{B}(y, z) \leftarrow \mathsf{A}(x, y)\}$ *and* $q_{ex} = \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{A}(v, w)$.

Then, q_{ex} *can be unfolded by* r_1 *with a piece unifier* $\mu = (\{\mathsf{A}(v, w)\}, \{x \mapsto v, y \mapsto w\})$, *and* $\mathsf{unf}^{\mu}(q_{ex}, r_1) = \exists w.[\mathsf{Q} \wedge \mathsf{A}(v, w)] \leftarrow \mathsf{A}(u, v) \wedge \mathsf{B}(v, z)$. *On the other hand,* $\mu = (\{\mathsf{A}(u, v)\}, \{x \mapsto u, y \mapsto v\})$ *is not a piece unifier, as it does not correctly unify the existential variable* y.

Note that the result of unfolding can be simplified when the unified rule heads of r and r' do not share existential variables, i.e., $\mathsf{vars}(\mathsf{head}(r)\tau') \cap \mathsf{vars}(\mathsf{head}(r')\tau) \cap \vec{z} = \emptyset$. In this case, the two heads can be separated and result in

$$\exists \vec{z_1}.\mathsf{head}(r)\tau' \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \bigwedge \mathsf{body}(r')\tau' \qquad (*)$$

where $\vec{z_1}$ consists of all the variables in the head but not in the body. Note that $\exists \vec{z_2}.\mathsf{head}(r')\tau \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \bigwedge \mathsf{body}(r')\tau'$ is implied by r' and thus is redundant.

For a rule set Π, $\mathsf{unfold}(\Pi)$ is the smallest rule set containing Π such that $\mathsf{unf}^{\mu}(r, r') \in \mathsf{unfold}(\Pi)$ for each $r, r' \in \mathsf{unfold}(\Pi)$ and each μ (disregarding variable renaming).

Towards a datalog rewriting method, we observe that when a strong datalog rewriting exists for a rule set, it can be obtained via unfolding.

An *unfolding sequence* of a set of rules Π is inductively defined as follows: each rule $r \in \Pi$ is an unfolding sequence corresponding to itself; and if θ is an unfolding sequence (whose corresponding rule, denoted θ as well) is unfoldable by a rule $r \in \Pi$ with piece unification μ, then $\theta\mu r$ is an unfolding sequence, and the result of the sequence is $\mathsf{unf}^{\mu}(\theta, r)$. From Proposition 5 in [Wang et al., 2018], each rule $r \in \mathsf{unfold}(\Pi)$ is a result of some unfolding sequence of Π. The *unfold chaining* on a set of rules Π is a sequence of rule sets Π^i_{unf} ($i \geq 0$), where $\Pi^0_{\mathsf{unf}} = \Pi$ and $\Pi^{i+1}_{\mathsf{unf}} = \Pi^i_{\mathsf{unf}} \cup \{\, \mathsf{unf}^{\mu}(r, r') \mid r \in \Pi^i_{\mathsf{unf}} \text{ and } r' \in \Pi \,\}$. Then, $\mathsf{unfold}(\Pi) = \bigcup_{i=0}^{\infty} \Pi^i_{\mathsf{unf}}$.

Lemma 3.1. *For a set of rules Π, the following conditions hold:*

(1) $\Pi \models r$ *for each* $r \in \mathsf{unfold}(\Pi)$.

(2) for each dataset $\mathcal{D}$ and each fact α, $\Pi \cup \mathcal{D} \models \alpha$ iff there exist $n \geq 0$ and a rule $r \in \Pi^n_{\mathsf{unf}}$ s.t. $\{r\} \cup \mathcal{D} \models \alpha$.

(3) for each datalog rule r, $\Pi \models r$ iff there exist $n \geq 0$ and a rule $r' \in \Pi^n_{\mathsf{unf}}$ s.t. $r' \models r$.

The above lemma can be directly obtained from Proposition 2 of [Wang et al., 2018]. Note that the requirement of rule aggregation in the Proposition is unnecessary when we consider the entailment of facts (instead of CQs), or that of datalog rules (with single head atoms).

Proposition 3.1. *For a set of rules Π, a strong FPDR of Π exists iff a finite subset of $\mathsf{unfold}(\Pi)$ is an FPDR of Π.*

Proof. The "if" direction is clear and we only need to show the "only if" direction. Suppose a strong FPDR of Π exists, denoted Π_d. Then, for each rule $r \in \Pi_d$, by the definition of strong FPDR, $\Pi_d \models r$. From Lemma 3.1 (3), there is a rule $r' \in \mathsf{unfold}(\Pi)$ such that $r' \models r$. That is, there exist a homomorphism σ from $\mathsf{body}(r')$ to $\mathsf{body}(r)$ and its safe extension σ' satisfying $\mathsf{head}(r) \subseteq \mathsf{head}(r')\sigma'$. From the fact that r is a datalog rule and σ' is a safe extension, $\mathsf{head}(r)$ does not share any existential variable with

$\mathsf{head}(r')\sigma' \setminus \mathsf{head}(r)$ in $r'\sigma'$. That is, r' must be a datalog rule, as otherwise the head atoms in r' could be separated. Let Π' be the collection of all such r' for all $r \in \Pi$. We want to show that Π' is a strong FPDR. From Lemma 3.1 (1), $\Pi \models \Pi'$. Also, for each dataset $\mathcal{D}$ and each fact α, $\Pi_d \cup \mathcal{D} \models \alpha$ implies that $\Pi \cup \mathcal{D} \models \alpha$, which in turn implies that $\Pi' \cup \mathcal{D} \models \alpha$. □

Clearly, a naive method to compute a datalog rewriting using the above unfolding is impractical, as the datalog rules obtained from unfolding can be very large (indeed, are often of unbounded sizes). In what follows, we introduce a practical approach for datalog rewriting by splitting long datalog rules generated via unfolding into compact ones. As a first step, we present an alternative operator, which we simply call rewriting, between two existential rules.

Definition 3.3. For two rules r, r' and a piece unifier $\mu = (B, \tau)$ of $\mathsf{body}(r)$ and $\mathsf{head}(r')$, the result of rewriting r by r' with μ, denoted $\mathsf{rew}^{\mu}(r, r')$, consists of rule (*) and the following two rules

$$\mathsf{P}(\vec{x}) \leftarrow \bigwedge \mathsf{body}(r')\tau', \tag{3.1}$$

$$\exists \vec{z_1}.\mathsf{head}(r)\tau' \leftarrow \bigwedge(\mathsf{body}(r) \setminus B)\tau \wedge \mathsf{P}(\vec{x}), \tag{3.2}$$

where $\vec{x} = \vec{x}_r\tau \cup \mathsf{vars}(\mathsf{body}(r) \setminus B)\tau \cap \vec{x}_{r'}\tau$, P is a fresh predicate with arity $|\vec{x}|$, called a *separating predicate*, and $\vec{z_1}$, τ and τ' are as in *(*)*.

Intuitively, P separates the body resulted from unfolding for the compactness of individual rules, and the head is always separated for datalog rewriting. Note that rule (*) can be obtained by unfolding (3.2) by (3.1), yet it is generated for the correctness of rewriting as we show later. We call rules of the form (*) *auxiliary rules* which will be deleted after the whole rewriting process is completed.

Example 3.3. *For* Π_{ex1} *and* q_{ex} *in Example 3.2,* $\mathsf{rew}^{\mu}(q_{ex}, r_1)$ *consists of the following rules:*

$$r_3 : \mathsf{P}(v) \leftarrow \mathsf{B}(v, z), \qquad r_4 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{P}(v),$$

$$r_5 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{B}(v, z).$$

Replacing $\mathsf{unf}^{\mu}(r, r')$ with $\mathsf{rew}^{\mu}(r, r')$ for unfolding leads to a set of rules that are equivalent to $\mathsf{unfold}(\Pi)$ w.r.t. fact derivation (over original predicates) and query answering. Yet allowing the unfolding of separating predicates (i.e., including them in piece unifiers) clearly forfeits their purpose, as they were introduced to split long rules and their unfolding simply reverses the split. Hence, the unfolding of separating predicates must not be allowed.

Furthermore, it is possible to reuse separating predicates. This is achieved through a labelling function $\lambda(\cdot)$ such that $\lambda(\mathsf{P}) = \mathsf{head}(r')\tau$. Intuitively, the label records how P is introduced (e.g., the head atom involved in the piece unification). When introducing a new separating predicate P' with the same arity and if $\lambda(\mathsf{P})$ is equivalent to $\lambda(\mathsf{P}')$ up to variable renaming, we *reuse* P to replace P'.

We are ready to define our datalog rewriting.

Definition 3.4. The *rewrite chaining* on a rule set Π is a sequence of rule sets Π^{i}_{rew} ($i \geq 0$), where $\Pi^{0}_{\mathsf{rew}} = \Pi$, and $\Pi^{i+1}_{\mathsf{rew}} = \Pi^{i}_{\mathsf{rew}} \cup \{\, \mathsf{rew}^{\mu}(r, r') \mid r \in \Pi^{i}_{\mathsf{rew}}, r' \in \Pi \,\}$ for $i \geq 0$ satisfying the following two conditions: (i) separating predicates are reused whenever possible, and (ii) any rule that is implied by another rule is eliminated.

The *rewriting* of Π, $\mathsf{rewrite}(\Pi)$, is obtained from $\Pi^{\infty}_{\mathsf{rew}}$ by deleting all auxiliary and non-datalog rules.

Example 3.4. *For Π_{ex1} and q_{ex} in Examples 3.2 and 3.3, the rewriting of q_{ex} by r_1 and r_2 include additionally the following (non-exhaustive list of) rules:*

$$r_6 : \mathsf{P}'(v) \leftarrow \mathsf{A}(x, v), \qquad r_7 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{P}'(v),$$
$$r_8 : \mathsf{Q} \leftarrow \mathsf{A}(u, v) \wedge \mathsf{A}(x, v), \qquad r_9 : \mathsf{P}'' \leftarrow \mathsf{B}(u, z),$$
$$r_{10} : \mathsf{Q} \leftarrow \mathsf{P}'', \qquad r_{11} : \mathsf{Q} \leftarrow \mathsf{B}(u, z).$$

Note that rules r_9 and r_{10} cannot be obtained without keeping auxiliary rules r_5 and r_8 during the rewriting.

We establish the correctness of our rewriting approach. We use $\mathsf{rewrite}^{\mathsf{q}}$ to denote the variant of $\mathsf{rewrite}$ where only rules with query or separating predicates in their heads are rewritten (i.e., in Definition 3.3, $\mathsf{head}(r) = \mathsf{Q}$ or $\mathsf{head}(r) = \mathsf{P}'(\vec{x})$ for some separating predicate P').

The following corollary can be obtained from Lemma 3.1 (2) and the connection between unfolding and rewriting, i.e., unfolded rules of the form (*) are included in the rewrite chaining.

Corollary 3.1. *For a set of rules* Π *and a pair of dataset* $\mathcal{D}$ *and fact* α *over* $\mathsf{sig}(\Pi)$, $\Pi \cup \mathcal{D} \models \alpha$ *iff there exist* $n \geq 0$ *and a datalog rule* $r \in \Pi^n_{\mathsf{rew}}$ *s.t.* $\{r\} \cup \mathcal{D} \models \alpha$.

Recall that a rule r is *applicable* to a dataset $\mathcal{D}$ if there is a homomorphism σ from $\mathsf{body}(r)$ to $\mathcal{D}$, and the result of *applying* r to $\mathcal{D}$ with σ is $\mathcal{D} \cup \mathsf{head}(r)\sigma'$, where σ' is an extension of σ by substituting each existential variable with a distinct fresh labelled-null. We call an atom (or a fact) with a separating predicate a *separating atom* (or *separating fact*).

Proposition 3.2. *For a rule set* Π, *a BCQ* q, *and the OMQ* $Q = (\Pi, q)$, $\mathsf{rewrite}(\Pi)$ *(or* $\mathsf{rewrite}^{\mathsf{q}}(Q)$*) is an FPDR (resp., QPDR) of* Π *(resp.,* Q*) whenever* $\mathsf{rewrite}(\Pi)$ *(resp.,* $\mathsf{rewrite}^{\mathsf{q}}(Q)$*) is equivalent to a finite set of rules.*

To prove Proposition 3.2, we first show a lemma.

Lemma 3.2. *For each* $i \geq 0$, *each pair of dataset* $\mathcal{D}$ *and fact* α *over* $\mathsf{sig}(\Pi)$, *if* $\Pi^i_{\mathsf{rew}} \cup \mathcal{D} \models \alpha$ *then* $\Pi \cup \mathcal{D} \models \alpha$.

Proof. We show the statement in the lemma by an induction on i. For $i = 0$, the statement trivially holds. Assume the statement holds for $i \geq 0$, and we want to show it holds for $i + 1$. That is, suppose $\Pi^{i+1}_{\mathsf{rew}} \cup \mathcal{D} \models \alpha$, we want to show that $\Pi \cup \mathcal{D} \models \alpha$. Consider w.l.o.g. the three rules r_0, r_1, r_2 in $\Pi^{i+1}_{\mathsf{rew}} \setminus \Pi^i_{\mathsf{rew}}$ (while in some cases there are fewer rules due to condition (iii) of Definition 3.4) of the forms respectively (*), (3.1), and (3.2) as in Definition 3.3 that are the result of rewriting r by r' for some $r \in \Pi^i_{\mathsf{rew}}$ and $r' \in \Pi$. In what follows, we consider three cases to show that each non-separating fact derivable from r_0, r_1, r_2 is derivable from Π^i_{rew}, which based on the inductive assumption implies $\Pi \cup \mathcal{D} \models \alpha$.

Suppose r_0 contributes to the derivation of α. From Lemma 3.1 (1), $\Pi^i_{\mathsf{rew}} \models r_0$ and thus α is derivable from Π^i_{rew}.

Suppose r_1 contributes to the derivation of α. If the predicate P in $\mathsf{head}(r_1)$ is a fresh separating predicate, then r_1 can only be applied together with r_2 and it is captured by

r_0 as above. Otherwise, P is reused to replace some separating predicate P' introduced during rewriting. From the conditions of reuse, there is a variable renaming π_1 between $\lambda(\mathsf{P}')$ and $\lambda(\mathsf{P})$, where $\lambda(\mathsf{P}') = \mathsf{head}(r')\tau$ and $\lambda(\mathsf{P}) = \mathsf{A}(\vec{u})$ for some atom $\mathsf{A}(\vec{u})$. That is, $\mathsf{head}(r')\tau\pi_1 = \mathsf{A}(\vec{u})$. Suppose r_1 is applied with substitution σ, resulting a fact about P, and there is another rule $r_2' : H \leftarrow B \wedge \mathsf{P}(\vec{w})$ of the form (3.2) in Π^i_{rew} that can be applied with substitution σ' to derive $H\sigma'$. We want to show that $\Pi^i_{\mathsf{rew}} \cup \mathcal{D} \models H\sigma'$. Note that P in $\mathsf{body}(r_2')$ may also be a reused predicate by replacing some P'', and in this case, there is a variable renaming π_2 between $\lambda(\mathsf{P}'')$ and $\lambda(\mathsf{P})$. By Definition 3.4 and the condition of reuse, r_2' must be obtained by the rewriting of a rule $r'' : H \leftarrow B \wedge \mathsf{A}(\vec{w})$ (up to variable renaming) in Π^i_{rew}, and $\lambda(\mathsf{P}'') = \mathsf{A}(\vec{w}) = \lambda(\mathsf{P})\pi_2^-$. Since, $\lambda(\mathsf{P}'') = \lambda(\mathsf{P})\pi_2^- = \lambda(\mathsf{P}')\pi_1\pi_2^-$, that is, $\mathsf{A}(\vec{w}) = \mathsf{head}(r')\tau\pi_1\pi_2^-$, by applying r' with substitution $\tau\sigma$ and then applying r'' with substitution σ', fact $H\sigma'$ can be derived. That is, α is derivable from Π^i_{rew}.

Suppose r_2 contributes to the derivation of α. If the predicate P in $\mathsf{body}(r_2)$ is a fresh separating predicate, then r_2 can only be applied together with r_1, and it is captured by r_0; otherwise, r_2 can only be applied together with a rule r_1' of the form (3.1) in Π^i_{rew}. In the latter case, with a similar argument for r_1 and r_2' above, we can show that α is derivable from Π^i_{rew}. □

Proof of Proposition 3.2. Let $\Pi_d = \mathsf{rewrite}(\Pi)$. From the definition of rewriting, it is clear that Π_d is a datalog program; we want to show for each dataset $\mathcal{D}$ and each fact α involving only predicates in Π, $\Pi \cup \mathcal{D} \models \alpha$ iff $\Pi_d \cup \mathcal{D} \models \alpha$. The "if" direction follows from Lemma 3.2, since Π_d being finite implies $\Pi_d \subseteq \Pi^i_{\mathsf{rew}}$ for some $i \geq 0$. For the "only if" direction, from Corollary 3.1, we only need to show that for each $i \geq 0$ and each datalog rule $r \in \Pi^i_{\mathsf{rew}}$, $\Pi_d \models r$. W.l.o.g., assume r is not implied by any other rule in Π^i_{rew} as otherwise, we can replace r with the rule containing it for our discussion. If r is a non-auxiliary rule, by the definition of rewriting (Definition 3.4) $r \in \Pi_d$. Otherwise if $r = r_0$ is an auxiliary rule of the form (*) generated from rewriting r by r' with piece unifier μ as in Definition 3.3, there are two rules r_1 and r_2 of the forms (3.1) and (3.2) respectively, and clearly $\{r_1, r_2\} \models r$. r_1 is a datalog rule and $r_1 \in \Pi_d$. To show r_2 is also a datalog rule, towards a contradiction, suppose there is $z \in \mathsf{vars}(\mathsf{head}(r))\tau'$, $z \in \mathsf{body}(r')\tau'$, but $z \notin \vec{x}$. Then, $z \in \vec{x}_r\tau$ and hence $z \in \vec{x}_r\tau \cap \vec{x}_{r'}\tau$, which by the definition of $\vec{x}$ implies $z \in \vec{x}$ and contradicts $z \notin \vec{x}$. Hence, $r_2 \in \Pi_d$ and $\Pi_d \models r$. □

3.2 Datalog Rewritable Classes

The rewriting method in the previous section does not necessarily terminate; for example, it does not terminate on a simple rule set:

$$\Pi_{ex2} = \{\mathsf{C}(x, y) \leftarrow \mathsf{C}(x, z) \wedge \mathsf{C}(z, y)\}$$

Yet it terminates on the class of finite unification sets (fus), for which several concrete classes have been identified, such as the class of linear rules (lin), the class of sticky rules (sticky), and the class of aGRD rules (agrd). For details of these classes, readers can go back to section 2.3.3.

Proposition 3.3. *For a set of rules* Π *in* fus, $\mathsf{rewrite}(\Pi)$ *is finite; and for any BCQ* q, $\mathsf{rewrite}(\Pi \cup \{q\})$ *is finite.*

Proof. We first show that the sizes of generated auxiliary rules of the form (*) are bounded. For each $r \in \Pi$, $\Pi \cup \{\mathsf{head}(r)\}$ is UCQ rewritable. Let m be maximum size of such a UCQ rewriting for all $r \in \Pi$. Then, each auxiliary rule generated during rewriting has at most m body atoms. Suppose an auxiliary rule r_0 with body size greater than m is generated, then from the definition of unfolding, $\mathsf{body}(r_0)$ can be obtained during UCQ rewriting. For it to be excluded as a UCQ rewriting, there must be another UCQ rewriting B that can be homomorphically mapped to $\mathsf{body}(r_0)$. Then, $r_0' = \mathsf{head}(r) \leftarrow B$ can be generated during our rewriting and clearly $r_0' \models r_0$. Hence, r_0 is eliminated. □

However, Π_{ex2} is not fus. It is not hard to see that the termination issue is caused by the generation of infinitely many auxiliary rules. We use $\mathsf{rewrite_a}()$ to denote the variant of $\mathsf{rewrite}()$ where auxiliary rules (of the form (*)) are not generated during rewriting.

Lemma 3.3. *For a set of rules* Π, $\mathsf{rewrite_a}(\Pi)$ *is finite.*

Proof of Lemma 3.3. Suppose each rule in Π contains at most m body atoms and the maximum arity of predicates is l. We show that there are a bounded number of rules of the forms (3.1) and (3.2) in $\mathsf{rewrite_a}(\Pi)$. First, each rule has a single head atom and its body has at most m atoms. In particular, for rules of the form (3.2), note that B contains at least one atom, and thus the rule body is at most the size of $\mathsf{body}(r)$. That is, rewriting does not increase the number of body atoms. Also, only a bounded number of

separating predicates are generated due to predicate reuse and their arities are bounded by l. For each separating predicate P, $\lambda(\mathsf{P})$ always consists of a single atom and the arity of P is bound by l. □

A rule set Π is *separable* if $\mathsf{rewrite_a}(\Pi) \equiv^{F}_{\mathsf{sig}(\Pi)} \Pi$. Intuitively, the condition requires the rule bodies (in particular, the bodies of auxiliary rules) can be separated during rewriting. The class of separable rule sets is denoted sep. Clearly, a separable rule set always admits a (finite) datalog rewriting, yet the definition does not suggest how to effectively identify such a rule set. Thus, we first show that the existing shy class [Leone et al., 2019] is a subclass of sep and then extend it to cover more practical rule sets.

A *position* is of the form $\mathsf{A}[i]$ with A being an n-ary predicate and $1 \leq i \leq n$, and a variable v occurs at position $\mathsf{A}[i]$ if there is an atom $\mathsf{A}(t_1, \ldots, t_n)$ with $t_i = v$. For a set of rules Π and an existential variable z in Π, a position $\mathsf{A}[i]$ is *invaded* by z if there is a rule $r \in \Pi$ such that $\mathsf{head}(r) = \mathsf{A}(t_1, \ldots, t_n)$ and either $t_i = z$ or t_i is a frontier variable that occurs in $\mathsf{body}(r)$ only at positions that are invaded by z. Recall that we assume each rule has a distinct set of variables. Then, a variable x in Π is *attacked* by z if x only occurs in positions invaded by z. Two atoms in the same rule body are *chained* if (1) they share a variable that is attacked, or (2) they each contains a frontier variable and these two variables are both attacked by the same variable. Finally, Π is *shy* if it does not contain two chained atoms, and we denote the class of shy rule sets as shy. It can be seen that Π_{ex2} is shy, and every shy rule set is also separable.

Theorem 3.1. $\mathsf{shy} \subset \mathsf{sep}$.

To prove Theorem 3.1, we need some preparation. Similar as unfolding sequences, we define a *rewriting sequence* of a set of rules Π inductively as follows: each rule $r \in \Pi$ is a rewriting sequence; and if θ is an rewriting sequence which can be rewritten by a rule $r \in \Pi$ with piece unification μ, then $\theta\mu r^{(*)}$, $\theta\mu r^{(1)}$, $\theta\mu r^{(2)}$ are three rewriting sequences corresponding to the three rules in $\mathsf{rew}^{\mu}(\theta, r)$ of the forms respectively (*), (3.1), and (3.2) as in Definition 3.3. We have the following observation: For a rewriting sequence θ, two rules r_1, r_2, two piece unifiers μ_1, μ_2, and some $\epsilon_1, \epsilon_2 \in \{(*), (1), (2)\}$, if $\theta\mu_1 r_1^{\epsilon_1}\mu_2 r_2^{\epsilon_2}$ is a rewriting sequence, then $\theta\mu_1 r_1^{(*)}\mu_2 r_2^{\epsilon_2}$ is also a rewriting sequence. It can be seen from the fact that $\mathsf{body}(\theta\mu_1 r_1^{(*)})$ contain all the non-separating atoms in $\mathsf{body}(\theta\mu_1 r_1^{\epsilon_1})$. For a rewriting sequence $\theta = r_0\mu_1 r_1^{\epsilon_1} \ldots \mu_n r_n^{\epsilon_n}$ for some $n \geq 0$, θ

corresponds to an unfolding sequence if $\epsilon_1, \ldots, \epsilon_n$ are all $(*)$, and we will call it an unfolding sequence for simplicity. θ^* is obtained from θ by replacing $\epsilon_n, \ldots, \epsilon_n$ all with $(*)$, and let $seg(\theta)$ consist of all rewriting sequences of the form

$$r_0\tau_0\tau_1 \cdots \tau_{i_1-1}\mu_{i_1} r_{i_1}^{\epsilon'_1}\tau_{i_1+1} \cdots \tau_{i_2-1}\mu_{i_2} r_{i_2}^{\epsilon'_2} \ldots \mu_{i_m} r_{i_m}^{\epsilon'_m}$$

where τ_i is the substitution in μ_i for $1 \leq i \leq n$, $m \leq n$, $1 \leq i_j < i_k \leq n$ for $1 \leq j < k \leq m$, and $\epsilon'_j \in \{(1), (2)\}$ for $1 \leq j \leq m$.

Lemma 3.4. *For a rewriting sequence θ, if θ^* is datalog then θ is datalog and all rewriting sequence in $seg(\theta^*)$ are datalog.*

Proof. To show the first half of the lemma, suppose $\theta = r_0\mu_1 r_1^{\epsilon_1} \ldots \mu_n r_n^{\epsilon_n}$ with $mu_i = (B_1, \tau_i)$ for all $1 \leq k \leq n$ and let $\theta_k = r_0\mu_0 r_1^{\epsilon_1} \ldots \mu_k r_k^{\epsilon_k}$ for $1 \leq k \leq n$. Let $ext(\theta_k)$ be the set of existential variables in θ_k. We want to show by induction on $k \geq 1$ that (i) $ext(\theta_k^*) = \mathsf{vars}(\mathsf{head}(r_0))\tau_1 \cdots \tau_k \cap (ext(r_1)\tau_1 \cdots \tau_k \cup ext(r_2)\tau_2 \cdots \tau_k \cup \ldots \cup ext(r_k)\tau_k)$, and (ii) $ext(\theta_k) \subseteq ext(\theta_k^*)$.

When $k = 1$, (i) is clear from Definition 3.3, and for (ii), $r_0\mu_1 r_1^{(1)}$ is datalog, and $ext(r_0\mu_1 r_1^{(2)}) = ext(r_0\mu_1 r_1^{(*)})$ as shown in the proof of Proposition 3.2.

Suppose (i) and (ii) hold for $k-1$ with $k \geq 2$, we show the case for k. Again, (i) is not hard to see from Definition 3.3, noting that $extr_k\tau_k$ cannot occur in $\mathsf{body}(\theta_{k-1}^*) \setminus B_{k-1}$ due to the definition of piece unifier. For (ii), again $\theta_{k-1}\mu_k r_k^{(1)}$ is datalog. Consider $\theta_{k-1}\mu_k r_k^{(2)}$ and take l to be the largest number satisfying $1 \leq l < k$ and $\epsilon_l = (1)$. Then, with a similar argument as for (i), $ext(\theta_{k-1}\mu_k r_k^{(2)}) = \mathsf{vars}(\mathsf{head}(\theta_l))\tau_l \cdots \tau_k \cap (ext(r_{l+1})\tau_{l+1} \cdots \tau_k \cup ext(r_{l+2})\tau_{l+2} \cdots \tau_k \cup \ldots \cup ext(r_k)\tau_k)$. Now we only need to show that $\mathsf{vars}(\mathsf{head}(\theta_l))\tau_l \cdots \tau_k \cap (ext(r_{l+1})\tau_{l+1} \cdots \tau_k \cup ext(r_{l+2})\tau_{l+2} \cdots \tau_k \cup \ldots \cup ext(r_k)\tau_k) \subseteq \mathsf{vars}(\mathsf{head}(r_0))\tau_1 \cdots \tau_k \cap (ext(r_1)\tau_1 \cdots \tau_k \cup ext(r_2)\tau_2 \cdots \tau_k \cup \ldots \cup ext(r_k)\tau_k)$. Towards a contradiction, suppose there is $x \in \mathsf{vars}(\mathsf{head}(\theta_l))\tau_l \cdots \tau_k$ and $x \in ext(r_j)\tau_j \cdots \tau_k$ for some $l \leq j \leq k$ but $x \notin \mathsf{vars}(\mathsf{head}(r_0))\tau_1 \cdots \tau_k$. As $x \notin \mathsf{vars}(\mathsf{head}(r_0))\tau_1 \cdots \tau_k$, $x \notin \mathsf{vars}(\mathsf{head}(\theta_{l-1}))\tau_{l-1} \cdots \tau_k$, also since $x \in ext(r_j)\tau_j \cdots \tau_k$, x unifies with an existential variable and $x \notin \mathsf{vars}(\mathsf{body}(\theta_{l-1}) \setminus B_l)\tau_{l-1} \cdots \tau_k$ due to the requirement of piece unifiers. From the definition of $\vec{x}$ in Definition 3.3, $x \notin \mathsf{vars}(\mathsf{head}(\theta_l))\tau_l \cdots \tau_k$, which contradicts the assumption $x \in \mathsf{vars}(\mathsf{head}(\theta_l))\tau_l \cdots \tau_k$. We have shown the first half of the lemma.

For the second half of the lemma, note that for each $\theta_i \in seg(\theta^*)$, θ_i^* also is a rewriting sequence. The fact that both θ_i^* and θ^* are rewriting sequences indicates that the datalog rule corresponding to θ^* can be obtained by further unfolding θ_i^*, that is, θ_i^* is the prefix of some unfolding sequence θ' that is datalog. Since unfolding cannot eliminate existential variables, θ_i^* is datalog, and by the first half of the lemma, θ_i is also datalog. □

Proof of Theorem 3.1. Clearly, $\mathsf{rewrite_a}(\Pi) \subseteq \mathsf{rewrite}(\Pi)$. From Corollary 3.1, we want to show that for each $i \geq 0$ and each datalog rule $r \in \Pi^i_{\mathsf{rew}}$ over $\mathsf{sig}(\Pi)$, $\mathsf{rewrite_a}(\Pi) \models r$, and we show this claim by an induction on rewriting sequences. Note that r must correspond to an unfolding sequence $\theta = r_0\mu_1 r_1^{(*)} \dots \mu_k r_k^{(*)}$, whereas $\mathsf{rewrite_a}(\Pi)$ consists of datalog rules with superscripts in $\{(1), (2)\}$. We want to show for all $k \geq 0$, r can be obtained by unfolding datalog rules in $seg(\theta)$. Since datalog rules in $seg(\theta)$ are in $\mathsf{rewrite_a}(\Pi)$, unless eliminated due to being implied by another datalog rule, it implies $\mathsf{rewrite_a}(\Pi) \models r$.

The case of $k = 0$ trivially holds. For $k = 1$, if $r_0\mu_1 r_1^{(*)}$ is a datalog rule, then both $r_0\mu_1 r_1^{(1)}$ and $r_0\mu_1 r_1^{(2)}$ are datalog rules, and $r_0\mu_1 r_1^{(*)}$ can be obtained by unfolding $r_0\mu_1 r_1^{(2)}$ by $r_0\mu_1 r_1^{(1)}$.

Assume the statement holds for $k \geq 1$, we want to show it holds for $k+1$. By the induction assumption, θ can be obtained by unfolding datalog rules corresponding to $\theta_1, \dots, \theta_n \in seg(\theta)$ for all $1 \leq i \leq n$. That is, θ corresponds to some unfolding sequence $\theta_1\mu_1' \dots \mu_{n-1}'\theta_n$. Suppose θ can be rewritten by $r \in \Pi$ with piece unifier $\mu = (B, \tau)$ of $\mathsf{body}(\theta)$ and $\mathsf{head}(r)$ with $B = \{\alpha_1, \dots, \alpha_m\}$. As θ corresponds to $\theta_1\mu_1' \dots \mu_n'\theta_n$, each α_j $(1 \leq j \leq m)$ must occur in some $\mathsf{body}(\theta_{i_j})\tau_{i_j}' \cdots \tau_n'$ with $1 \leq i_j \leq n$ (where each τ_i' is the substitution in the piece unifier μ_i'). Since Π is shy, the existential variables in r cannot be unified with a variable shared by two body atoms in θ. Hence, each θ_{i_j} $(1 \leq j \leq m)$ can be rewritten by r with a piece unifier $\mu_j'' = (\{\alpha_j\}, \tau_{i_j}' \cdots \tau_n'\tau)$.

We want to show that if $\theta\mu r^{(*)}$ is datalog then it can be obtained by unfolding datalog rules in $seg(\theta\mu r^{(*)})$. Note that $\theta\mu r^{(*)}$ is of the form $\exists \vec{z_1}.\mathsf{head}(\theta)\tau' \leftarrow \bigwedge(\mathsf{body}(\theta) \setminus B)\tau \wedge \bigwedge \mathsf{body}(r)\tau'$ and it corresponds to the unfolding sequence $\theta_1\mu_1' \dots \mu_{n-1}'\theta_n\mu r$. For each $1 \leq j \leq m$, there are two rewriting sequences $\theta_{i_j}\mu_j'' r^{(1)}$, which corresponds to a rule of the form $r_j^{(1)} = \mathsf{P}_j(\vec{x}_j) \leftarrow \bigwedge \mathsf{body}(r)\tau'$, and $\theta_{i_j}\mu_j'' r^{(2)}$, which corresponds to

$r_j^{(2)} = \mathsf{head}(\theta_{i_j})\tau \leftarrow \bigwedge(\mathsf{body}(\theta_{i_j}) \setminus \{\alpha_j\}\tau'_{i_j} \cdots \tau'_n\tau \wedge \mathsf{P}_j(\vec{x}_j)$. Clearly, $r_j^{(2)}$ can by unfolded by $r_j^{(1)}$ with $\mu_j^* = (\{\mathsf{P}_j(\vec{x}_j)\}, \emptyset)$ for all $1 \leq j \leq m$. Also, $\theta_{i_j}\mu''_j r^{(1)}$ and $\theta_{i_j}\mu''_j r^{(2)}$ are both in $seg(\theta\mu r^{(*)})$. By Lemma 3.4, $\theta\mu r^{(*)}$ being datalog implies $r_j^{(1)}$ and $r_j^{(2)}$ both to be datalog. Moreover, $\theta\mu r^{(*)}$ can be obtained from the following unfolding sequence

$$
\begin{aligned}
&\theta_1\ \mu'_1\tau'_2 \cdots \tau'_n\ \theta_2\ \mu'_2\tau'_3 \cdots \tau'_n \ldots \\
&\theta_{i_1-1}\ \mu'_{i_1-1}\tau'_{i_1} \cdots \tau'_n\ \ r_1^{(2)}\ \mu_1^*\ r_1^{(1)}\ \ \mu'_{i_1}\ \theta_{i_1+1}\ \mu'_{i_1+1}\tau'_{i_1+2} \cdots \tau'_n \ldots \\
&\theta_{i_2-1}\ \mu'_{i_2-1}\tau'_{i_2} \cdots \tau'_n\ \ r_2^{(2)}\ \mu_2^*\ r_2^{(1)}\ \ \mu'_{i_2}\ \theta_{i_2+1}\ \mu'_{i_2+1}\tau'_{i_2+2} \cdots \tau'_n \ldots \\
&\ldots \\
&\mu'_{n-1}\ \theta_n
\end{aligned}
$$

We have shown that if $\theta\mu r^{(*)}$ is datalog then it can be obtained by unfolding datalog rules in $seg(\theta\mu r^{(*)})$, and that is, the statement holds for $k+1$.

To show the strictness of the set containment, the rule set $\Pi = \{r_1 = \mathsf{A}(x,y) \leftarrow \mathsf{B}(x),\ r_2 = \mathsf{C}(x,y,z) \leftarrow \mathsf{A}(x,y) \wedge \mathsf{A}(x,z),\ r_3 = \mathsf{D}(x) \leftarrow \mathsf{C}(x,y,z)\}$ belongs to sep but not in shy. It is not shy because frontier variables y and z in r_2 are both attacked by the existential variable y in r_1, which makes the two body atoms in r_2 chained. Yet it is separable, as any datalog rule in $\mathsf{rewrite}(\Pi)$ over $\mathsf{sig}(\Pi)$, e.g., $\mathsf{D}(x) \leftarrow \mathsf{B}(x)$, is derivable from $\mathsf{rewrite}_\mathsf{a}(\Pi)$. □

An example of separable but not shy rule set is $\Pi_{ex3} = \{r_1 = \mathsf{A}(x,y) \leftarrow \mathsf{B}(x),\ r_2 = \mathsf{C}(x,y,z) \leftarrow \mathsf{A}(x,y) \wedge \mathsf{A}(x,z),\ r_3 = \mathsf{D}(x) \leftarrow \mathsf{C}(x,y,z)\}$. It is not shy because frontier variables y and z in r_2 are both attacked by the existential variable y in r_1, which makes the two body atoms in r_2 chained. Yet it is separable, as any datalog rule in $\mathsf{rewrite}(\Pi)$ over $\mathsf{sig}(\Pi)$, e.g., $\mathsf{D}(x) \leftarrow \mathsf{B}(x)$, is derivable from $\mathsf{rewrite}_\mathsf{a}(\Pi)$.

The class of separable rule set can be expanded to cover more datalog rewritable cases. Note that OMQ $\Pi_{ex1} \cup \{q_{ex}\}$ from Example 3.2 is not separable, yet a datalog rewriting does exist. We call a rule set Π *weakly-separable* if it can be transformed into a finite set of rules Π' such that $\mathsf{rewrite}_\mathsf{a}(\Pi') \equiv^F_{\mathsf{sig}(\Pi)} \Pi$, and the extended class is denoted as wsep. From the definition, a weakly-separable rule set always admits a (finite) datalog rewriting. Indeed, the wsep class consists of all datalog rewritable rule sets Π, as one can take Π' as the FPDR of Π, which is finite and separable. Next, we want to extend

the shy class (i) to allow blocks of, instead of individual, body atoms to be separable, and (ii) to combine it with concrete subclasses of fus.

A *block* $B \subseteq \mathsf{body}(r)$ for some rule r is a smallest non-empty set such that if $\alpha \in B$ then for each atom β chained to α, $\beta \in B$. We say a block B *depends* on a rule r if $\mathsf{head}(r)$ share the same predicate with an atom in B. For a set of rules Π, the *dependent rule set* of a block B, $\mathsf{dep}(B)$, is the smallest set of rules $r \in \Pi$ such that B depends on r or some rule in $\mathsf{dep}(B)$ contains a block that depends on r.

Definition 3.5. Let fus-shy consist of rule sets Π satisfying for each block B in Π with $|B| \geq 2$, $\mathsf{dep}(B) \in \mathsf{lin} \cup \mathsf{stky} \cup \mathsf{agrd}$.

The OMQ $\Pi_{ex1} \cup \Pi_{ex2} \cup \{q_{ex}\}$ belongs to fus-shy, as the only block with size greater than 1 is $B = \{\mathsf{A}(u, v), \mathsf{A}(v, w)\}$ in q_{ex} and $\mathsf{dep}(B) = \Pi_{ex1}$ which is fus; yet it is neither shy nor fus. We can show that each rule set that belongs to fus-shy is weakly separable, and thus is datalog rewritable. Intuitively, we can transform the rule set by fully unfolding the blocks B of sizes greater than 1 by $\mathsf{dep}(B)$, resulting an equivalent and separable rule set.

Theorem 3.2. $\mathsf{lin} \cup \mathsf{stky} \cup \mathsf{agrd} \cup \mathsf{shy} \subset \mathsf{fus\text{-}shy} \subset \mathsf{wsep}$.

We associate each block B in a set of rules with a fresh predicate R_B, called a *block predicate*. To avoid generating trivial block predicates, we require a block not to contain block or separating predicates. For a rule r, its *block separation*, denoted $\mathsf{bsep}(r)$, consists of the following rules

$$\mathsf{R}_B(\vec{x}_B) \leftarrow \bigwedge B \text{ for each block } B \text{ in } \mathsf{body}(r), \tag{3.3}$$

$$\exists \vec{z}_1.\mathsf{head}(r) \leftarrow \bigwedge_{B \text{ in } \mathsf{body}(r)} \mathsf{R}_B(\vec{x}_B), \tag{3.4}$$

where $\vec{x}_B = \vec{x}_r \cup \mathsf{vars}(\mathsf{body}(r) \setminus B) \cap \mathsf{vars}(B)$. Also, we reuse the block predicates in the same way as for separating predicates, by assigning $\lambda(\mathsf{R}_B) = B$. For a set of rules Π, let $\mathsf{bsep}(\Pi) = \bigcup_{r \in \Pi} \mathsf{bsep}(r)$. The *block-rewrite chaining* on a rule set Π is a sequence of rule sets Π^i_{b} $(i \geq 0)$, where $\Pi^0_{\mathsf{b}} = \Pi$, and $\Pi^{i+1}_{\mathsf{b}} = \Pi^i_{\mathsf{b}} \cup \{\, \mathsf{rew}^{\mu}(r, r') \mid r \in \mathsf{bsep}(\Pi^i_{\mathsf{b}}),\ r' \in \Pi \,\}$ for $i \geq 0$ satisfying the two conditions in Definition 3.4 plus (iii) block predicates are reused whenever possible. The *block-rewriting* of Π, $\mathsf{rewrite}^{\mathsf{b}}(\Pi)$, is obtained from $\Pi^{\infty}_{\mathsf{b}}$

by deleting all auxiliary and non-datalog rules, and $\mathsf{rewrite}^{\mathsf{b}}_{\mathsf{a}}(\Pi)$ is the variant where auxiliary rules are not generated.

To prove Theorem 3.2 and Proposition 3.4, we first establish the following lemmas.

Lemma 3.5. *For a set of rules* Π, $\mathsf{rewrite}^{\mathsf{b}}(\Pi) \equiv^{F}_{\mathsf{sig}(\Pi)} \Pi$ *and* $\mathsf{rewrite}^{\mathsf{b}}_{\mathsf{a}}(\Pi) \equiv^{F}_{\mathsf{sig}(\Pi)} \mathsf{rewrite}_{\mathsf{a}}(\Pi)$.

Proof. For each pair of dataset $\mathcal{D}$ and fact α over $\mathsf{sig}(\Pi)$, we first show if $\mathsf{rewrite}^{\mathsf{b}}(\Pi) \cup \mathcal{D} \models \alpha$ then $\Pi \cup \mathcal{D} \models \alpha$. And we can show this by an induction on block-rewrite chaining Π^{i}_{b} in a similar way as in the proof of Lemma 3.2. In particular, the reuse of block predicate does not cause the derivation of any new facts over $\mathsf{sig}(\Pi)$, as clearly if R_B is reused to replace R'_B then $\lambda(\mathsf{R}_B)$ is equivalent to $\lambda(\mathsf{R}'_B)$ up to variable renaming and the blocks they respectively represent would have the same instantiation. The same proof applies to show if $\mathsf{rewrite}^{\mathsf{b}}_{\mathsf{a}}(\Pi) \cup \mathcal{D} \models \alpha$ then $\mathsf{rewrite}_{\mathsf{a}}(\Pi) \cup \mathcal{D} \models \alpha$.

Next, we show if $\Pi \cup \mathcal{D} \models \alpha$ then $\mathsf{rewrite}^{\mathsf{b}}(\Pi) \cup \mathcal{D} \models \alpha$, and by Corollary 3.1, we want to show that for each $i \geq 0$ and each datalog rule $r \in \Pi^{i}_{\mathsf{rew}}$ over $\mathsf{sig}(\Pi)$, $\mathsf{rewrite}^{\mathsf{b}}(\Pi) \models r$. Similar as in the proof of Theorem 3.1, suppose r corresponds to an unfolding sequence θ, we show by an induction on the length of θ that r can be obtained by unfolding datalog rules $r_1, \ldots, r_n$ in $\mathsf{rewrite}^{\mathsf{b}}(\Pi)$, which implies $\mathsf{rewrite}_{\mathsf{a}}(\Pi) \models r$. In particular, from the definition of $\mathsf{bsep}(r)$, when θ is rewritten by a rule r' with piece unifier $\mu = (B, \tau)$, B can always be mapped to some body of r_i $(1 \leq i \leq n)$. Hence, $\theta\mu r'^{(*)}$ can be obtained from unfolding datalog rules in $\mathsf{rewrite}^{\mathsf{b}}(\Pi)$.

Finally, to show if $\mathsf{rewrite}_{\mathsf{a}}(\Pi) \cup \mathcal{D} \models \alpha$ then $\mathsf{rewrite}^{\mathsf{b}}_{\mathsf{a}}(\Pi) \cup \mathcal{D} \models \alpha$, we combine the argument above and the proof of Theorem 3.1, as from Lemma 3.1 (2), $\mathsf{rewrite}_{\mathsf{a}}(\Pi) \cup \mathcal{D} \models \alpha$ implies that there is a datalog rule r over $\mathsf{sig}(\Pi)$ corresponding to a unfolding sequence consisting of the rules $\mathsf{rewrite}_{\mathsf{a}}(\Pi)$ such that $\{r\} \cup \mathcal{D} \models \alpha$, and we show that for each of such r, it can be obtained from unfolding datalog rules in $\mathsf{rewrite}^{\mathsf{b}}_{\mathsf{a}}(\Pi)$. □

Lemma 3.6. *For a set of rules* Π, *if all blocks in* $\mathsf{unfold}(\Pi)$ *of sizes greater than 1 occur in* Π *(up to variable renaming), then* $\mathsf{rewrite}^{\mathsf{b}}_{\mathsf{a}}(\Pi) \equiv^{F}_{\mathsf{sig}(\Pi)} \mathsf{rewrite}^{\mathsf{b}}(\Pi)$.

Proof. The proof again resembles that of Theorem 3.1, and we want to show (from Lemma 3.1 (2)) for each pair of dataset $\mathcal{D}$ and fact α over $\mathsf{sig}(\Pi)$, and each datalog rule r over $\mathsf{sig}(\Pi)$ corresponding to a unfolding sequence consisting of the rules $\mathsf{rewrite}^{\mathsf{b}}(\Pi)$ such that $\{r\} \cup \mathcal{D} \models \alpha$, $\mathsf{rewrite}^{\mathsf{b}}_{\mathsf{a}}(\Pi) \models r$. Suppose r corresponds to an unfolding sequence

θ, we show by an induction on the length of θ that r can be obtained by unfolding datalog rules $r_1, \ldots, r_n$ in $\mathsf{rewrite}^{\mathsf{b}}(\Pi)$, which implies $\mathsf{rewrite}^{\mathsf{b}}(\Pi) \models r$. As all blocks in $\mathsf{unfold}(\Pi)$ of sizes greater than 1 occur in Π, after the first step of rewriting, each potential block occurs as the body of a distinct rule due to $\mathsf{bsep}(\Pi)$. Note that rewriting does not generate new blocks that do not occur in $\mathsf{unfold}(\Pi)$. Hence, when θ is rewritten by a rule r' with piece unifier $\mu = (B, \tau)$, B can always be mapped to some body of r_i $(1 \leq i \leq n)$. □

Proof of Theorem 3.2. For the first set containment, suppose Π is in $\mathsf{lin} \cup \mathsf{stky} \cup \mathsf{agrd}$, then clearly for each block B in Π, $\mathsf{dep}(B) \subseteq \Pi \in \mathsf{lin} \cup \mathsf{stky} \cup \mathsf{agrd}$, and hence $\Pi \in \mathsf{fus\text{-}shy}$. Suppose $\Pi \in \mathsf{shy}$, then clearly Π does not contain any block B with $|B| \geq 2$, and also $\Pi \in \mathsf{fus\text{-}shy}$. For the strictness of the set containment, consider the OMQ $\Pi_{ex_1} \cup \Pi_{ex2} \cup \{q_{ex}\}$, which belongs to $\mathsf{fus\text{-}shy}$ but is nether fus nor shy.

For the second set containment, suppose $\Pi \in \mathsf{fus\text{-}shy}$, and we want to show that $\Pi \in \mathsf{wsep}$. After the first step of the block-rewrite chaining of Π, namely Π^i_{b} for $i \geq 0$, each block B in Π with $|B| \geq 2$ is separated into the body of a distinct rule of the form $\mathsf{R}_B(\vec{x}_B) \leftarrow \bigwedge B$ and gets rewritten separately. As $\mathsf{dep}(B) \in \mathsf{lin} \cup \mathsf{stky} \cup \mathsf{agrd}$, B cannot be infinitely unfolded. Also, as a block does not contain block or separating predicates, it is not hard to see that Π^i_{b} cannot introduce new blocks that are not in $\mathsf{unfold}(\Pi)$. Thus, for block B in Π with $|B| \geq 2$, there is an $k_B \geq 0$ such that $\Pi^{k_B}_{\mathsf{b}}$ contains all the blocks (with sizes greater than 1) that can be obtained from unfolding B with the rules in $\mathsf{dep}(B)$. Take k be the maximum of all such k_B and $\Pi' = \Pi^k_{\mathsf{b}}$. Note that all blocks in $\mathsf{unfold}(\Pi')$ of sizes greater than 1 occur in Π' (up to variable renaming), and thus by Lemmas 3.5 and 3.6, $\mathsf{rewrite}_{\mathsf{a}}(\Pi') \equiv^F_{\mathsf{sig}(\Pi)} \mathsf{rewrite}(\Pi')$. Moreover, we can show $\Pi' \equiv^F_{\mathsf{sig}(\Pi)} \Pi$ by an induction on block-rewrite chaining Π^i_{b} in a similar way as in the proof of Lemma 3.5. By Proposition 3.2, $\mathsf{rewrite}(\Pi') \equiv^F_{\mathsf{sig}(\Pi)} \Pi$, and thus $\mathsf{rewrite}_{\mathsf{a}}(\Pi') \equiv^F_{\mathsf{sig}(\Pi)} \Pi$.

To show the strictness of the second set containment, the rule set $\Pi = \{r_1 = \mathsf{A}(x, y) \leftarrow \mathsf{B}(x),\ r_2 = \mathsf{C}(x, y, z) \leftarrow \mathsf{A}(x, y) \wedge \mathsf{A}(x, z),\ r_3 = \mathsf{D}(x) \leftarrow \mathsf{C}(x, y, z),\ r_4 = \mathsf{A}(x, y) \leftarrow \mathsf{A}(x, z) \wedge \mathsf{A}(z, y)\}$ belongs to wsep but not in $\mathsf{fus\text{-}shy}$. It is not in $\mathsf{fus\text{-}shy}$ because there is a block $B = \{\mathsf{A}(x, y) \wedge \mathsf{A}(x, z)\}$ in r_2 and $\mathsf{dep}(B) = \{r_1, r_4\}$ is not fus. Yet Π is datalog rewritable, and a datalog rewriting is $\{r_2, r_3, r_4\} \cup \{\mathsf{D}(x) \leftarrow \mathsf{B}(x)\}$. Hence, Π belongs to wsep. □

Moreover, a rule set in fus-shy coupled with any BCQ is also datalog rewritable.

Proposition 3.4. *For a rule set* Π *that belongs to* fus-shy *and any BCQ* q*, the OMQ* $\Pi \cup \{q\}$ *belongs to* wsep.

Proof. Let $Q = \Pi \cup \{q\}$ and we first show $\mathsf{unfold}(Q)$ contains a finite number of blocks up to variable renaming. Let $\Pi' = \Pi^k_{\mathsf{b}}$ with $k \geq 0$ that contains all blocks in $\mathsf{unfold}(\Pi)$ (of sizes greater 1) as in the proof of Theorem 3.2, where we establish the existence of such a k. Since only a finite number of blocks can be generated in Π^k_{b}, suppose set Ξ consists of all such blocks. Since $\Pi \subseteq \Pi'$, each block in $\mathsf{unfold}(Q)$ would occur in $\mathsf{unfold}(\Pi' \cup \{q\})$. We will examine the new blocks generated in $\mathsf{unfold}(\Pi' \cup \{q\})$.

First, each new block must be of the form $A\sigma \cup \bigcup_{i=1}^{n} B_i\sigma_i$ with $A \subseteq \mathsf{body}(q)$, $B_i \in \Xi$ for $1 \leq i \leq n$, and σ, σ_i being substitutions; that is, each new block must be a combination of atoms in the query body and existing blocks in Ξ. This can be shown by an induction on the unfolding sequences. The case of $q\mu r$ is straightforward, for some $r \in \Pi'$ and some piece unifier μ. Suppose θ is a unfolding sequence starting with q and the blocks in θ satisfies the above condition. Consider $\theta\mu r$ with $r \in \Pi'$ and $\mu = (A', \tau)$ a piece unifier, and suppose $A' = A_1 \cup A_2$ with A_1 consists of atoms from q (with variable substation) and A_2 consists of atoms from rules in Π' (with variable substation). Consider the unfolding sequence θ' obtained from θ by removing q from the front and any rules that unfold atoms only from q, then θ' is rewritable by r with a piece unifier (disregarding the substitution details) (A_2, τ). Hence, the above condition also holds for $\theta\mu r$.

Second, for each new block of the form $A\sigma \cup \bigcup_{i=1}^{n} B_i\sigma_i$, each B_i must come from some $r \in \Pi'$. By the definition of a block, for two blocks B_i and B_j to be chained, there needs to be a variable u in B_i and a variable v in B_j that are both attacked by the same existential variable in Π'. For B_i and B_j (and other atoms) to form a new block that does not exist in Π', the two blocks need to be chained through q; that is, there exists a variable in q that are unified with both u and v (through the sequence of unfolding). Suppose there are m variables in q, $x_1, \ldots, x_m$. We can also define *positions* in each block B, denoted $B[j, k]$ for the j-th position of the k-th predicate in B. Note that a block B is chained to another block through a x_i $(1 \leq i \leq m)$ in q if x_i and the variable at some position $B[j, k]$ are both attacked by the same existential variable in Π'. For each x_i, there are bounded number of blocks B from Ξ and positions of B that

can be chained via x_i. Hence, there is a finite number of new blocks can be formed in $\mathsf{unfold}(\Pi' \cup \{q\})$.

Finally, let $\Pi'' = \Pi' \cup \{q\}$; as there are finitely many blocks that can be formed in $\mathsf{unfold}(\Pi'')$, we can find a $l \geq 0$ such that Π''^{l}_{unf} contains all the blocks in $\mathsf{unfold}(\Pi'')$ (up to variable renaming) of sizes greater than 1. Take $Q' = \Pi''^{l}_{\mathsf{unf}}$. By Lemmas 3.5 and 3.6, $\mathsf{rewrite}_{\mathsf{a}}(Q') \equiv^{F}_{\mathsf{sig}(\Pi)} \mathsf{rewrite}(Q')$. Moreover, we can show $Q' \equiv_{\{\mathsf{Q}\}} Q$. By Proposition 3.2, $\mathsf{rewrite}(\Pi') \equiv^{F}_{\mathsf{sig}(\Pi)} \Pi$, and thus $\mathsf{rewrite}_{\mathsf{a}}(\Pi') \equiv^{F}_{\mathsf{sig}(\Pi)} \Pi$. □

3.3 Efficient Rewriting Algorithms

In this section, we introduce an efficient method for computing datalog rewritings of the form $\mathsf{rewrite}_{\mathsf{a}}(\Pi)$ whenever they exist, and discuss how it can be adapted to compute $\mathsf{rewrite}(\Pi)$. Inspired by [Hansen et al., 2015], we compute a decomposed representation of the heads and bodies of the resulting rules generated during the rewriting, such that the representation is compact due to structure sharing and the datalog rewriting can be conveniently extracted from such a representation. For the ease of presentation, we first present a variant of the representation and then further simplify it.

For a set of rules Π, its *rewriting forest* F_Π has nodes of the form $(\mathsf{P}(\vec{x}), H, B_1, B_2)$, where P is a fresh predicate not occurring in Π, $\vec{x}$ is a vector of variables, H is an atom, and B_1, B_2 are sets of atoms; and edges are labelled with piece unifiers. The roots of F_Π correspond to the rules in Π, i.e., are of the form $(\top, \mathsf{head}(r), \emptyset, \mathsf{body}(r))$ for all rules $r \in \Pi$, where $\top$ represents true. Intuitively, each node $((\mathsf{P}(\vec{x}), H, B_1, B_2)$ represents two rules as follows:

$$\mathsf{P}(\vec{x}) \leftarrow \bigwedge B_1, \tag{1*}$$

$$\exists \vec{z}.H \leftarrow \bigwedge B_2, \tag{2*}$$

where $\vec{z}$ consists of all the variables in the head but not in the body. Rules (1*) and (2*) correspond to the rules (3.1) and (3.2) generated during rewriting. Hence, P can be reused in the same way as separating predicates through the labelling function $\lambda(\cdot)$. Moreover, a node n is *blocked* by another node n' if its corresponds rules are both implied by those of n'.

Formally, F_Π has the smallest number of nodes and edges satisfying the following conditions: For each node $n = (\mathsf{P}(\vec{x}), H, B_1, B_2)$ and each root node $n' = (\top, H', \emptyset, B_2')$, let $\vec{x}' = \mathsf{vars}(H') \cap \mathsf{vars}(B_2')$; and

1. for each piece unifier $\mu = (B, \tau)$ of B_2 and H', n has a child $n'' = (\mathsf{P}''(\vec{x}''), H'', B_1'', B_2'')$ whenever n'' is not blocked s.t.
   - $\vec{x}'' = \mathsf{vars}(H)\tau \cup \mathsf{vars}(B_2 \setminus B)\tau \cap \vec{x}'\tau$, $\lambda(\mathsf{P}'') = H'\tau$,
   - $H'' = H\tau'$, where τ' is as in (*), $B_1'' = B_2'\tau'$ and $B_2'' = (B_2 \setminus B)\tau \cup \{\mathsf{P}''(\vec{x}'')\}$;
2. if n is not a root, for each piece unifier μ of B_1 and H', n has a child n'' whenever n'' is not blocked s.t.
   - $\vec{x}'' = \vec{x}\tau \cup \mathsf{vars}(B_1 \setminus B)\tau \cap \vec{x}'\tau$,
   - $H'' = \mathsf{P}(\vec{x})\tau'$ and $B_2'' = (B_1 \setminus B)\tau \cup \{\mathsf{P}''(\vec{x}'')\}$,
   - B_1'' and $\lambda(\mathsf{P}'')$ are as in Condition 1;

Our algorithm starts with the nodes corresponding to the rules in Π and expands the rewriting forest based on the above conditions. The expansion terminates due to the blocking condition, and the number of nodes are bounded based on the same argument as for Lemma 3.3. Let $\mathsf{datalog}(F_\Pi)$ be the set of datalog rules obtained as above from the nodes of F_Π.

Theorem 3.3. *For a set of rules* Π, F_Π *is always finite and* $\mathsf{datalog}(F_\Pi) \equiv^F_{\mathsf{sig}(\Pi)} \mathsf{rewrite_a}(\Pi)$.

Proof. We first show the finiteness of F_Π. Suppose each rule in Π contains at most m body atoms and the maximum arity of predicates is l. From the definition of F_Π, for each node $n = (\mathsf{P}(\vec{x}), H, B_1, B_2)$ in F_Π, B_1, B_2 each consists of at most m atoms. Also, for each fresh predicate P, $\lambda(\mathsf{P})$ always consists of a single atom and clearly $|\vec{x}| \leq l$. By the predicate reuse condition, only a bounded number of new predicates can be introduced.

Next, we want to show $\mathsf{datalog}(F_\Pi) \equiv^F_{\mathsf{sig}(\Pi)} \mathsf{rewrite_a}(\Pi)$. For each node $n = (\mathsf{P}(\vec{x}), H, B_1, B_2)$, we show by induction that it corresponds to rules in Π or rules of the form (3.1) and (3.2) generated during the rewriting (†). This is clear for root nodes from the definition of F_Π. Then, suppose node n'' is generated from n and n', and n, n' both satisfy (†).

Case (A): If n'' is generated under Condition 1, then by our assumption n corresponds to either (a) a rule $r \in \Pi$ or (b) two rules of the form (3.1) and (3.2) generated during the

rewriting. And n' corresponds to $r' \in \Pi$. In case (a), $H = \mathsf{head}(r)$ and $B_2 = \mathsf{body}(r)$. By the definition, rewriting results in the following rules

$$\mathsf{P}''(\vec{v})\tau \leftarrow \bigwedge \mathsf{body}(r')\tau',$$
$$\exists \vec{z}.\mathsf{head}(r)\tau' \leftarrow \bigwedge (\mathsf{body}(r) \setminus B)\tau \wedge \mathsf{P}''(\vec{v})\tau,$$

where $\vec{v} = \vec{x}_r\tau \cup \mathsf{vars}(\mathsf{body}(r) \setminus B)\tau \cap \vec{x}_{r'}\tau$, and P'' is a separating predicates with $\lambda(\mathsf{P}'') = \mathsf{head}(r')\tau$. Since variables in $\mathsf{head}(r)$ that are not in $\vec{x}_r$ are existential variables and cannot occur in $\vec{x}_{r'}\tau$, $\vec{v} = \mathsf{vars}(\mathsf{head}(r))\tau \cup \mathsf{vars}(\mathsf{body}(r) \setminus B)\tau \cap \vec{x}_{r'}\tau$. Comparing with rules (1*) and (2*), one can verify that n'' corresponds to the above rules obtained from rewriting.

In case (b), one of the rules n corresponds to is of the form $\exists \vec{z}.H \leftarrow \bigwedge B_2$, and by the definition, rewriting results in the following rules

$$\mathsf{P}''(\vec{v})\tau \leftarrow \bigwedge \mathsf{body}(r')\tau',$$
$$\exists \vec{z}.H \leftarrow \bigwedge (B_2 \setminus B)\tau \wedge \mathsf{P}''(\vec{v})\tau,$$

where $\vec{v} = (\mathsf{vars}(H) \cap \mathsf{vars}(B_2))\tau \cup \mathsf{vars}(B_2 \setminus B)\tau \cap \vec{x}_{r'}\tau$, and $\lambda(\mathsf{P}'') = \mathsf{head}(r')\tau$. Since variables in H that are not in B_2 are existential variables and cannot occur in $\vec{x}_{r'}\tau$, $\vec{v} = \mathsf{vars}(H)\tau \cup \mathsf{vars}(B_2 \setminus B)\tau \cap \vec{x}_{r'}\tau$. By comparing with rules (1*) and (2*), one can verify that n'' corresponds to the above rules obtained from rewriting.

Case (B): If n'' is generated under Condition 2, then one of the rules n corresponds to is of the form $\mathsf{P}(\vec{x}) \leftarrow \bigwedge B_1$, and by the definition, rewriting results in the following rules

$$\mathsf{P}''(\vec{v})\tau \leftarrow \bigwedge \mathsf{body}(r')\tau',$$
$$\exists \vec{z}.\mathsf{P}(\vec{x}) \leftarrow \bigwedge (B_1 \setminus B)\tau \wedge \mathsf{P}''(\vec{v})\tau,$$

where $\vec{v} = \vec{x}\tau \cup \mathsf{vars}(B_1 \setminus B)\tau \cap \vec{x}_{r'}\tau$ and $\lambda(\mathsf{P}'') = \mathsf{head}(r')\tau$. By comparing with rules (1*) and (2*), one can verify that n'' corresponds to the above rules obtained from rewriting.

For the other direction, we show for each rule r generated during the rewriting (i.e., one of the forms (3.1) and (3.2)), there is a node n in F_Π that corresponds to r (‡). This can be shown through induction in a similar way as above. In particular, suppose r is obtained from rewriting r_1 by r_2, and both r_1, r_2 satisfy (‡). Since a piece unifier cannot

contain a separating predicate, r_1 is either an original rule (i.e., $r_1 \in \Pi$) or of one of the forms (3.1) and (3.2).

If $r_1, r_2 \in \Pi$ then the proof is as in Case (A) (a), there is a child n of the root corresponding to r_1 that corresponds to the rules resulted from rewriting r_1 by r_2, including r. If r_1 is of the form (3.1) and $r_2 \in \Pi$ then the proof is as in Case (B). If r_1 is of the form (3.2) and $r_2 \in \Pi$ then the proof is as in Case (A) (b). The above cases cover all the possibilities, and in each of the case, n can be found as a child of the node corresponding to r_1. □

To further simplify the representation, note that H and B_2 in each non-root node can be computed on the fly. In particular, for the computation of H'' and B_2'' under Conditions 2, it only needs the current node n'' and its parent node n; whereas under Condition 1, B_2'' refers to B_2 in the parent node n, which can be computed through back-tracking till a root.

To adapt rewriting forests for the computation of $\mathsf{rewrite}(\Pi)$, we can expand each node with a fourth component B_3, which is a set of atoms. Intuitively, it is used to capture rules $\exists \vec{z}.H \leftarrow \bigwedge B_3$, that corresponds to the rules of the form (*) generated during the rewriting. In particular, $B_3'' = (B_2 \backslash B)\tau \cup B_2'\tau'$ in Conditions 1, and $B_3'' = (B_1 \backslash B)\tau \cup B_2'\tau'$ in Condition 2. Furthermore, there is a third condition: if n is not a root, for each piece unifier μ of B_3 and H', n has a child n'' whenever n'' is not blocked such that $\vec{x}'' = \mathsf{vars}(H)\tau \cup \mathsf{vars}(B_3 \setminus B)\tau \cap \vec{x}'\tau$, $B_3'' = (B_3 \setminus B)\tau \cup B_2'\tau'$, and everything else is defined as in Condition 1. Again, B_3 in each node can be computed on the fly via back-tracking. The difference from B_1 and B_2 is that the sizes of B_3 are not necessary bounded, unless $\mathsf{rewrite}(\Pi)$ is finite.

3.4 Systems and Benchmarks

In this section we introduce several state-of-the-art query rewriting and query answering systems, which will be evaluated in our experiments. Some of them are based on rewriting techniques, while some are not. Besides, a wide range of benchmarks covering light-weight DLs and existential rules will be considered in our evaluations, we provide a brief introduction to them as well.

Graal [1] is a query answering system dedicated for existential rules based on first-order rewriting [König et al., 2015a]. To avoid the blow up issue in first-order rewriting, instead of considering all rules in the ontology in the process of rewriting, Graal computes entailment relations over atoms, specifically called *compiled preorders*, to replace the reasoning of some simple rules in the ontology. For example, the datalog rule $\mathsf{B}(\vec{x}) \leftarrow \mathsf{A}(\vec{x})$ can be compiled into a preorder $\mathsf{A} \preceq \mathsf{B}$. Then when answering or rewriting $\mathsf{B}(\vec{x})$, one can ignore the rule and directly search facts with A in the database or unify the query with rules whose heads contain A. However, the approach has several limitations, only rules of particular form (linear datalog rules) can be replaced by preorders; the resulting rewriting is not a standard first-order rewriting, the evaluation of it needs to consider the preorders, which means it can only be handled by specially configured querying engines.

Rapid [2] is an efficient rewriting solver dedicated for DL-lite, using an optimized resolution-based rewriting approach [Trivela et al., 2015]. The general process of the approach is as follows: the ontology is first saturated using resolution, then redundant rules are cleaned from the saturation set, the result of which is a datalog rewriting; finally, the first-order rewriting is computed by unfolding the saturation set. Because of the feature of above approach, the solver can be configured to output either datalog rewriting or first-order rewriting.

Iqaros [3] is a query answering system for first-order rewritable DLs [Venetis et al., 2016]. It features on various optimization techniques to minimize the result of first-order rewriting, which can effectively speed up the evaluations. First, the system will try to simplify the query by detecting atoms that don't have any instances in the database. Because the system adopts a combined approach to perform query answering, it will saturate the database using datalog rules in the ontology. then queries in the rewriting result rewritten by datalog rules are unnecessary, thus removed from the result. Finally, similar to most rewriting systems, Iqaros will check query subsumption for the rewriting.

Pagoda [4] is a query answering system for OWL2 ontologies using a hybrid approach that combines a datalog engine with a fully-fledged OWL2 reasoner [Zhou et al.,

[1] http://graphik-team.github.io/graal/
[2] www.image.ece.ntua.gr/~gstam/
[3] code.google.com/p/iqaros/
[4] https://www.cs.ox.ac.uk/isg/tools/PAGOdA/

2015]. The system provides 'pay as you go' performance. For each ontology, it will compute two datalog approximations, which can guarantee the upper bound and lower bound query answers respectively. For the task of query answering, the system first compute query answers using a datalog engine for both approximations, if the upper bound answers exactly coincide with the lower bound answers, then these answers are certain answers for the query; otherwise, the task will be left to the OWL2 reasoner to solve.

DLV$^{\exists}$ [5] is an extension of the well known DLV system for answering (unrestricted) conjunctive queries in existential rules [Leone et al., 2019]. It handles a special class of existential rules called parsimonious programs. While parsimonious programs are unrecognizable, existential rules in the shy class are parsimonious. The system performs a special chase procedure, called parsimonious chase, which is more efficient than standard chase, and can terminate and produce sound and complete chase result for query answering over parsimonious programs.

ChaseGoal is a goal-driven chase system that can handle existential rules with equality dependencies [Benedikt et al., 2018]. Instead of performing chase directly, it takes the target query into account and produce a final datalog program for chasing. In fact, the final program can be regarded as a datalog rewriting, however, the transforming process can only work for ontologies where the chase terminates.

Grind [6] is a tool to check the first-order rewritability of particular concepts and conjunctive queries in $\mathcal{EL}$ ontologies [Hansen et al., 2015] and to produce non-recursive datalog rewritings, if which exist.

Next we introduce the benchmarks considered in our evaluations, a summary of which is presented in table 3.1

Adolena and LUBM are selected from a widely used benchmark, which is introduced with the rewriting system, requiem[7]. In particular, LUBM is a manually created ontology, describing domain knowledge about universities. Its datasets can be synthetically generated according to a parameter n (the number of universities) for

[5] https://www.mat.unical.it/dlve/
[6] http://www.informatik.uni-bremen.de/tdki/hansen/doku.php?id=grind
[7] www.cs.ox.ac.uk/projects/requiem

different sizes, thus it is often used in large scale evaluations. In many researches, LUBM is trimmed or modified into different versions for differnt purposes.

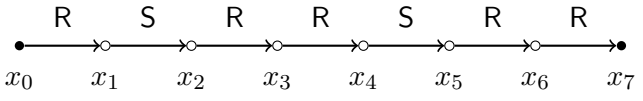
OpenGALEN2 and OBOprotein are ontologies providing clinical and biomedical information. They contain a large number of axioms (up to 35k). We use the DL-lite versions of them, which are from a test suite proposed with the Rapid system. Particularly, OBOprotein is recognized as a very challenging benchmark. Recent evaluations have shown it cannot be handled by most rewriting systems [König et al., 2013, König et al., 2015b].

Reactome and Uniprot are realistic ontologies publicly available through the European Bioinformatics Institute (EBI) linked data platform[8]. They are are in OWL2, rich of existentially quantified and disjunctive rules. We remove axioms that are not expressible in existential rules from the ontologies for evaluations. The resulting ontologies are still not first-order rewritable.

RS is a hand crafted benchmark from [Bienvenu et al., 2017]. It contains an ontology in DL-lite, which has only 6 rules,

$$\mathsf{S}(x,y) \leftarrow \mathsf{P}(x,y), \qquad \mathsf{R}(y,x) \leftarrow \mathsf{P}(x,y),$$
$$\exists y.\mathsf{P}(x,y) \leftarrow \mathsf{A}_{\mathsf{P}}(x), \qquad \mathsf{A}_{\mathsf{P}}(x) \leftarrow \mathsf{P}(x,y),$$
$$\exists y.\mathsf{P}(y,x) \leftrightarrow \mathsf{A}_{\mathsf{P}^-}(x), \qquad \mathsf{A}_{\mathsf{P}^-}(x) \leftarrow \mathsf{P}(y,x).$$

Along with the ontology, it provides specially constructed queries, which are chains of atoms with the predicates of R and S. An example query for them can be depicted as follows,



where x_0 and x_7 are answer variables. Though the ontology is simple, the rewritings of these queries turn out to be very complicated, especially when the lengths of queries are long. Following the chain structure, one can customize queries with any lengths by arranging R and S. And it is worth noting that the order of R and S in the chain can greatly affect the results of rewriting.

[8] http://www.ebi.ac.uk/rdf/platform

ChaseBench is a suite of benchmarks offering various scenarios for chase-based reasoning systems [Benedikt et al., 2017]. Ontologies in ChaseBench are all weakly-acyclic existential rules, which means termination of chase over these ontologies can be guaranteed. STB-128 and ONT-256 are generated by the tool iBench [Arocena et al., 2015], equipped with large datasets. While DEEP200/300 are designed for "pure stress" test, whose databases are small, for each extensional predicate (predicate used in facts and does not occur in rule heads), there is only one fact, however, the TGDs in these ontologies are so complex that they can produce up to 500M facts for the case of DEEP300. Above ontologies all contain existential rules with predicates of arities more than two.

Table 3.1: Summary of the datasets

Ontology	Exp	#R	#F
RS	DL-Lite	6	-
Adolena	DL-Lite	72	-
OpenGALEN2	DL-Lite	27K	-
OBOprotein	DL-Lite	35K	-
LUBM-100	DL-Lite	136	12M
Reactome	OWL2	292	2.5M
Uniprot	OWL2	390	1.2M
STB-128	Ex. Rules	199	1M
ONT-256	Ex. Rules	529	2M
DEEP200	Ex. Rules	1.2K	1K
DEEP300	Ex. Rules	1.3K	1K

3.5 Evaluation

We have implemented a prototype system, Drewer (Datalog REWriting for Existential Rules) [9], with our piece unification module adapted from the first-order rewriting system Graal, and we deployed VLog[10] as our datalog engine. VLog is a high-performance Datalog engine using column-based data structures [Urbani et al., 2016]. It is highly memory efficient and can process large programs with thousands of rules.

All our experiments were performed on a laptop with a processor at 2.2 GHz and 8GB of RAM. The system and experiments can be found at `https://www.ict.griffith.edu.au/aist/Drewer`.

[9] The system is currently a prototype and is completed by Peng Xiao with the help of his supervisors, no other team members are involved.

[10] `https://github.com/knowsys/vlog4j`

We evaluated ontologies including the DL-Lite versions of LUBM, Adolena, Open-GALEN2, OBOprotein and RS. We used the existential rule fragments of Reactome and Uniprot, which are more expressive than DL-Lite. And to evaluate our system's ability over 'true' existential rules (i.e., rules can not be expressed in DLs), we choose DEEP200/300, STB-128, and ONT-256 from ChaseBench. All the tested ontologies are found to be in fus-shy and thus can be handled by Drewer, whereas none of the compared rewriting system could successfully handle all of them.

We conducted two sets of experiments to evaluate the performance of our system. In the first set of experiments, we compared our system with state-of-the-art query rewriting systems regarding the compactness and efficiency of query rewriting. We used original queries that come with the datasets and evaluated 5 queries per ontology, except for RS which has only 3 long queries with length of 15.

Table 3.2 records the sizes and times for query rewriting, where sizes are measured by the numbers of atoms and the times are in milliseconds. We set a 5 minutes time limit per query, ran each query 3 times, and reported the average. Rapid(DD) and Rapid(DU) denote respectively the datalog and UCQ rewritings from Rapid, and TO denotes time out whereas a "-" means the system could not handle the OMQ. In particular, Grind failed to handle most of the queries for OBOprotein and all of the queries for Reatome and Uniprot.

It can be seen from Table 3.2, except for LUBM and Uniprot, the sizes of datalog rewritings are often comparable to or much smaller than those of UCQ rewritings. The reason for the two exceptional cases is that the tested queries can be entailed by their subsets with the ontologies, for example, in the LUBM benchmark, we have the following query and rules,

$$\text{q2} : \mathsf{Q}(x, y) \leftarrow \mathsf{Person}(x) \wedge \mathsf{teacherOf}(x, y) \wedge \mathsf{Course}(y).$$
$$r_1 : \mathsf{Person}(x) \leftarrow \mathsf{teacherOf}(x, y).$$
$$r_2 : \mathsf{Course}(y) \leftarrow \mathsf{teacherOf}(x, y).$$

Clearly, q2 can be entailed by an atomic query $\mathsf{Q}(x, y) \leftarrow \mathsf{teacherOf}(x, y)$ with r_1 and r_2. In such case, most of the generated queries in the rewriting will become redundant, which are removed by rewriting systems that perform rewriting minimization.

TABLE 3.2: Full comparison on query rewriting

Ontology	Query	Datalog Rewriting						UCQ Rewriting					
		Drewer		Rapid(DD)		Grind		Graal		Rapid(DU)		Iqaros	
		size	time	size	time	size	time	size(*)	time	size	time	size	time
Adolena	q1	43	9	43	18	27	386	2+29	29	27	20	27	28
	q2	32	2	31	10	28	120	2+29	30	50	28	50	33
	q3	34	2	32	10	27	145	1+31	38	104	34	104	94
	q4	40	20	39	10	33	116	2+34	47	224	50	224	97
	q5	38	203	37	12	32	87	1+36	34	624	75	624	565
LUBM	q1	3	2	3	5	2	302	1+1	19	2	8	2	17
	q2	51	3	46	15	38	123	1+0	12	1	9	1	17
	q3	21	0	20	12	20	73	1+47	60	4	8	4	40
	q4	70	2	63	21	55	72	1+1	25	2	15	2	32
	q5	56	1	52	15	42	64	1+6	44	10	16	10	65
OpenGALEN2	q1	3	0	3	5	2	99469	1+2	117	2	4	2	49
	q2	1296	56	1276	82	1152	99352	1+1275	175	1152	56	1152	7525
	q3	93	1372	92	39	26	109664	5+87	208	488	44	488	11963
	q4	162	1	155	15	147	113195	1+154	108	147	11	147	951
	q5	82	1773	81	25	37	111972	19+62	211	324	40	324	8698
OBOprotein	q1	30	48	29	19	29	12953	20+7	259	27	16	27	6732
	q2	1357	868	1356	720	-	-	1264+92	6866	1356	963	1356	25062
	q3	34580	99	33919	128341	-	-	1+33918	338	33887	699	33887	56202
	q4	34880	2906	34879	127786	-	-	682+34085	3239	34733	12832	34733	72424
	q5	1386	1144	27907	TO	-	-	TO	TO	36612	TO	36612	TO
RS	q1	14	153	TO	TO	TO	TO	14+4	904	TO	TO	TO	TO
	q2	22	849	TO	TO	TO	TO	100+4	13554	TO	TO	TO	TO
	q3	25	8976	TO	TO	TO	TO	143+4	40875	TO	TO	TO	TO
Reactome	q1	3	0	3	9	-	-	1+2	34	3	11	3	12
	q2	18	1	13	9	-	-	TO	TO	32	16	32	27
	q3	18	0	13	10	-	-	TO	TO	32	18	32	28
	q4	21	0	15	8	-	-	TO	TO	32	24	32	52
	q5	21	0	15	7	-	-	TO	TO	32	15	32	50
Uniprot	q1	1	0	1	5	-	-	1+0	15	1	9	1	8
	q2	1	0	1	6	-	-	1+0	16	1	4	1	8
	q3	15	0	12	6	-	-	1+0	19	1	5	1	12
	q4	15	0	12	6	-	-	1+0	34	1	6	1	13
	q5	15	0	11	7	-	-	1+0	34	1	7	1	16
DEEP300	q1	245	286	-	-	-	-	117+2	927	-	-	-	-
	q2	27	4	-	-	-	-	528+3	12417	-	-	-	-
	q3	81	16	-	-	-	-	624+2	17196	-	-	-	-
	q4	90	31	-	-	-	-	TO	TO	-	-	-	-
	q5	249	187	-	-	-	-	TO	TO	-	-	-	-
STB-128	q1	7	7	-	-	-	-	4+2	97	-	-	-	-
	q2	14	4	-	-	-	-	4+2	56	-	-	-	-
	q3	6	2	-	-	-	-	8+0	81	-	-	-	-
	q4	8	0	-	-	-	-	16+5	259	-	-	-	-
	q5	10	2	-	-	-	-	8+5	121	-	-	-	-
ONT-256	q1	4	11	-	-	-	-	8+2	94	-	-	-	-
	q2	15	6	-	-	-	-	4+6	127	-	-	-	-
	q3	7	1	-	-	-	-	8+6	139	-	-	-	-
	q4	11	4	-	-	-	-	12+5	176	-	-	-	-
	q5	13	3	-	-	-	-	48+1	2378	-	-	-	-

Note that to achieve efficiency as well as compactness in rewriting, Graal makes use of the so-called compiled preorders, hence, the UCQ rewritings produced by Graal need to be coupled with the compiled preorders for query answering. For a fair comparison, we report the rewriting sizes for Graal in the form of $x + y$, where x is the size of the UCQ rewriting and y is that of the datalog rules corresponding to the compiled pre-orders. Since not all preorders are used for specific queries, we tracked for each query those pre-orders actually used for query answering (following Graal's native query answering process) and y is the number of only those used preorders. Though for Graal less rules are considered in rewriting, which produces a much smaller rewriting as we have seen, we show later that the smaller rewriting sizes for Graal do not necessary mean faster query answering (compared to Drewer) due to the overhead of processing the complied pre-orders. On the other hand, this optimization only works well for less expressive ontologies like LUBM, it is not as effective as in expressive ontologies such as DEEP300 and ONT-256.

Regarding the time efficiency of rewriting, Drewer again is superior or comparable to other systems in almost all cases. Especially for OBOprotein, RS, the three ontologies from the ChaseBench, which are challenging ontologies for rewriting, our system showed outstanding performance. In particular, all other systems time out on q5 for OBOprotein, whereas Drewer took only around 1 second to complete the rewriting. And all other systems except for Graal failed to complete their rewriting on the three long queries for RS, due to the large sizes of generated UCQ rewritings. Besides, the reason for Graal's timeouts in Reactome could be that its algorithm does not terminate on those queries, as it is reported by Graal's checker that Reactome does not fall in the existing concrete classes in fus. And though Rapid and Iqaros can successfully handle Reactome, but they are dedicated to DL-lite, that means they may filter out non-linear axioms when parsing the ontologies.

To evaluate the overall performance of Drewer in query answering, we conducted a second set of experiments comparing it with other query answering systems. Note that for fair comparisons, all systems we evaluated (including Drewer) are in-memory systems, that is, when the systems perform query anwering, all data is loaded into the memory from the disk. Besides Graal and Iqaros, we compare Drewer with chased-based systems VLog and DLV$^{\exists}$ [Leone et al., 2019], both of which show state-of-the-art performance, and the hybrid query answering system for OWL2, Pagoda. To compare with Graal on

the quality of different rewritings, we used a datalog translation for both of its UCQ rewritings and the used pre-orders, and deployed the same datalog engine VLog for query answering.

TABLE 3.3: Comparison on query answering

Ontology	Query	Drewer		VLog		DLV$^{\exists}$	Graal(VLog)		Iqaros		Pagoda
		Total	QA	Total	QA	Total	Total	QA	Total	QA	Total
LUBM-100	q1	20288	14	21111	1313	TO	21320	65	21549	2504	129664
	q2	22570	2284	22394	2714	70631	21266	332	22372	2637	130523
	q3	20835	764	22179	2459	74358	21667	785	22332	2622	130642
	q4	21879	1595	21660	1958	69886	20932	164	23732	2524	130058
	q5	21122	913	21138	1315	TO	21037	157	23272	2732	130317
Reactome	q1	2865	46	3347	387	20049	2946	22	7000	1491	10194
	q2	2824	50	3287	360	20073	TO	TO	6256	1399	10188
	q3	2900	76	3300	390	20063	TO	TO	6787	1411	10183
	q4	2911	76	3312	380	20254	TO	TO	6496	1451	10184
	q5	2880	56	3252	360	20023	TO	TO	6606	1522	10184
Uniprot	q1	1862	110	TO	TO	10239	2030	99	3388	748	33048
	q2	2559	819	TO	TO	10658	2104	226	3079	809	33320
	q3	1777	16	TO	TO	-	1957	84	2999	743	33008
	q4	1763	16	TO	TO	-	1991	87	2772	734	33005
	q5	1760	16	TO	TO	-	1969	80	2719	742	33005
DEEP200	q1	789	193	3875	2452	454	962	287	-	-	-
	q2	647	175	3668	2460	448	5330	4715	-	-	-
	q3	675	185	3635	2456	457	5185	4565	-	-	-
	q4	698	197	3660	2460	461	TO	TO	-	-	-
	q5	696	193	3661	2451	443	TO	TO	-	-	-
DEEP300	q1	896	279	TO	TO	553	1489	752	-	-	-
	q2	849	322	TO	TO	567	11013	10371	-	-	-
	q3	792	292	TO	TO	575	14735	14089	-	-	-
	q4	933	388	TO	TO	568	TO	TO	-	-	-
	q5	730	238	TO	TO	548	TO	TO	-	-	-
STB-128	q1	8942	426	39947	9302	21856	10042	215	-	-	-
	q2	8460	23	39783	9118	21721	9827	83	-	-	-
	q3	8655	28	40012	9376	21458	10703	102	-	-	-
	q4	9052	47	40284	9632	21879	10819	269	-	-	-
	q5	8169	36	39790	9162	24279	10574	139	-	-	-
ONT-256	q1	25687	47	28356	2065	63558	31102	152	-	-	-
	q2	27535	86	28272	2068	TO	31223	168	-	-	-
	q3	28411	81	28194	2061	66547	29747	204	-	-	-
	q4	29030	71	28231	2066	TO	29626	211	-	-	-
	q5	28956	52	28323	2089	TO	31897	2151	-	-	-

Table 3.3 presents the average times (in milliseconds) for answering the 5 queries over each ontology, where we ran each query separately (which may involve running the chase procedure separately for each query). A TO (Timeout) shows the system failed to answer the query in 5 minutes. We separate the query answering times (QA, including

rewriting, chase computation, and query evaluation times whenever applicable) out of the total times, except for DLV$^{\exists}$ and Pagoda as it was difficult to make the separation.

Overall, our (datalog) rewritings demonstrate higher efficiency in query answering compared to UCQ rewritings (by Graal and Iqaros) on Reactome and ChaseBench ontologies. While Drewer could successfully handle all queries, Graal was time out on 4 queries for Reactome and 2 queries for DEEP200 and DEEP300, which is due to its inefficiency of rewriting over these expressive ontologies. Compared to chase-based systems, our system demonstrates superior time efficiency than VLog on all tested cases, and outperforms (sometimes by magnitudes) or is comparable to DLV$^{\exists}$ on most of the cases. In particular, VLog failed on Uniprot and DEEP300, and DLV$^{\exists}$ struggled (for some queries) on LUBM-100 and ONT-256. This is because VLog (or DLV$^{\exists}$) computes the standard full (resp., parsimonious) chase whereas the queries are often related to only a small portion of it. Similarly, the Pagoda system is inefficient in all the three DL scenarios, which is because the cost for materializing the two datalog approximations over large scale datasets is too high. However, chase-based systems still have the advantage that when answering multiple queries over the same dataset, they only need to compute the materialization once, while Drewer need to compute the chase on the datalog rewriting of each query. Yet in cases where computing the full chase is challenging, e.g., DEEP300 for VLog, our datalog rewriting becomes a critical factor for successful handling of the OMQs.

Chapter 4

Practical Abduction for Existential Rules

While query answering is the essential reasoning task in OBDA, in order to provide solid reasoning services, OBDA systems should be equipped with the ability to explain query answers. Particularly, explaining negative answers can be formalized by the the problem of query abduction: (1) a negative query answer is an observation in the problem; (2) the abduction results are exactly what the database needs to entail the negative answer. Compared to query answering, query abduction for expressive ontology languages is far from well-studied, which is due to the complex nature of the problem. There are several works on query abduction [Borgida et al., 2008, Calvanese et al., 2013, Du et al., 2014, Wang et al., 2015], but they are either based on a light-weight ontology language (and cannot be extended to existential rules) or not scalable over large datasets (even for representative explanations, which try to compactly represent explanations, we will discuss about it in details later).

Existing research efforts showed that query abduction is challenging, as the computational complexity of query abduction is indeed higher than query answering [Calvanese et al., 2013]. Also, different from traditional abduction, query abduction allows new constants in explanations, which makes it difficult (if not impossible) to compute all explanations even in some light-weight ontology languages such as $\mathcal{EL}$ and DL-Lite. Though representative explanations are proposed to compactly represent the (possibly

infinite many) new constants using substitutable null values, the approach does not distinguish the different roles of explanations with and without null values. We illustrate the problem using an example on disease tracking.

Example 4.1. *When we trace the spread of an infectious disease with hereditary vulnerability, the following ontology* Π *could be useful:*

$$\mathsf{high\_risk}(x) \leftarrow \mathsf{contact}(x, y), \mathsf{high\_risk}(y).$$
$$\mathsf{high\_risk}(x) \leftarrow \mathsf{has\_parent}(x, y), \mathsf{diagnosed}(y).$$

Here, $\mathsf{contact}(x, y)$ *means that* x *has close contact with* y *and* $\mathsf{diagnosed}(y)$ *means that* y *is diagnosed with the disease. Suppose we have the dataset* $\mathcal{D} = \{\mathsf{contact}(John, Alice), \mathsf{diagnosed}(Bob)\}$.

Now, if we have an observation that John is high-risk for the disease, that is, $q : \{\mathsf{high\_risk}(John)\}$*, yet we cannot derive this from existing knowledge* Π *and* $\mathcal{D}$*. Then this scenario can be captured by a query abduction problem* $\Lambda = (\Pi, q, \mathcal{D}, \Sigma)$*, where* $\Sigma = \{\mathsf{high\_risk}, \mathsf{contact}, \mathsf{has\_parent}\}$*. For the QAP, we have the following two obvious explanations among others:*

$$\mathcal{E}_1 = \{\mathsf{high\_risk}(Alice)\} \text{ and } \mathcal{E}_2 = \{\mathsf{has\_parent}(John, Bob)\}.$$

And by allowing anonymous constants in explanations, we also have the following explanation:

$$\mathcal{E}_3 = \{\mathsf{has\_parent}(John, n), \mathsf{diagnosed}(n)\},$$

where n *is a labelled null that does not appear in* Π *or* q.

As we have introduced before, allowing null values in the explanations introduces issues. First, the number of potential explanations may be infinite, even for representative explanations. Note that according to definition 2.28, it can be checked that explanations of the form $\mathcal{E}_4^{(k)} = \{\mathsf{contact}(John, n_1), \mathsf{contact}(n_1, n_2), \ldots, \mathsf{contact}(n_{k-1}, n_k), \mathsf{high\_risk}(n_k)\}$ are legitimately representative explanations for any $k \geq 1$. Also, we note that $\mathcal{E}_3$ is different from $\mathcal{E}_1$ and $\mathcal{E}_2$ in the sense the former essentially represents a pattern of (or abstract) explanations up to the instantiation of null values, while $\mathcal{E}_1$ and $\mathcal{E}_2$ are specific ones referring to concrete cases. It is hard to tell in general which kind of explanations

would be better. In some cases, a user may prefer specific explanations to abstract ones; for instance, to trace high risk individuals like in $\mathcal{E}_1$. In other cases, the user may have the opposite preference; for instance, to know John is high-risk due to hereditary reasons expressed as $\mathcal{E}_3$ rather than hypothesizing (for each diagnosed case) the parent of John like in $\mathcal{E}_2$. Thus, it would be interesting to introduce a new definition of query abduction so that preferences over types of explanations can be represented.

In this chapter, we propose a novel notion of selective explanations, which generalises the existing representative explanations and allows users to distinguish high-level pattern explanations from low-level concrete explanations. For the computation of selective explanations, we present an algorithm based on an efficient query rewriting method for existential rules. We also introduce a compact representation for selective explanations which can significantly reduce the number of generated explanations and can be computed efficiently. Finally, we implement a prototype system that computes selective explanations, and our experimental evaluation shows our system can scale over very large datasets.

4.1 Selective Explanations

In our previous discussions, a query abduction problem with an ontology allows the introduction of labeled null values that do not appear in the given ontology or the observation and thus, the set of all explanations for a query abduction can be very large or even infinite. As a result, various criteria are employed to select preferred explanations. A common approach is to pick only subset-minimal explanations (up to renaming). Yet such minimal explanations are still in large numbers, and often have repetative patterns with only some constants varying. For this reason, the notion of representative explanations is proposed to capture explanations of the same pattern using a single representative. However, the set of representative explanations can still be infinite for some ontologies. Also, the notion of representative explanations cannot distinguish between the representatives (demonstrating a high-level pattern) and the concrete cases. Thus, in this section, we introduce a flexible framework for the notion of selective explanations. In the discussion of this chapter, we consider existential rules without constraints for all query abduction problems, which means our obtained explanations are always consistent.

We first recall precise definitions of some concepts about explanations of query abduction. In the definition of the query abduction problem, an explanation is allowed to be a set of (not necessarily ground) facts, thus an explanation may contain labelled nulls that can be substituted by constants. Hence, the notion of minimal explanations need to take renaming into consideration. For two sets of facts $\mathcal{A}, \mathcal{A}'$, $\mathcal{A} \preceq_m \mathcal{A}'$ if there exists a renaming ρ such that $\mathcal{A}'\rho \subseteq \mathcal{A}$. And $\mathcal{A} \prec_m \mathcal{A}'$ if $\mathcal{A}'\rho \subset \mathcal{A}$. An explanation $\mathcal{E}$ to a QAP Λ is minimal if $\mathcal{E}' \preceq_m \mathcal{E}$ implies $\mathcal{E} \preceq_m \mathcal{E}'$ for all explanations $\mathcal{E}'$ to Λ.

And the notion of representative explanations is based on the homomorphism relations on the set of all minimal explanations, thus when a constant in a minimal explanation can be replaced by a labeled null, then the explanation is represented by a more general explanation. A representative explanation to a QAP Λ is a minimal explanation $\mathcal{E}$ to Λ such that for all minimal explanation $\mathcal{E}'$ to Λ, if $\mathcal{E}' \preceq_h \mathcal{E}$, then $\mathcal{E} \preceq_h \mathcal{E}'$. The set of all representative explanations is denoted as $\mathsf{rexpl}(\Lambda)$.

In Example 4.1, the explanations $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$ are all representative explanations. Note that the minimality condition in the definition of representative explanations is necessary, as otherwise an explanation $\mathcal{E} = \mathcal{E}_3 \cup \{\mathsf{has\_parent}(John, n'), \mathsf{diagnosed}(n')\}$ would be equally preferred as $\mathcal{E}_3$ w.r.t. $\preceq_h$.

As we have shown, there are an infinite number of representative explanations for the QAP in Example 4.1 with the above abducibles, due to the continuing introduction of new labelled nulls. Also, the set of representative explanations include both specific explanations referring to concrete constants, like $\mathcal{E}_1$ and $\mathcal{E}_2$, and abstract explanations showing patterns with substitutable null values, like $\mathcal{E}_3$. We argue that in actual application scenarios, these two types of explanations satisfy different requirements of users with different goals, and should be divided.

In what follows, we present a novel notion of selective explanations based on a fine-grained partition of abducibles and a preference on (concrete) constants or (substitutable) nulls in each of the parts.

For a QAP with abducibles Σ, a *selection criteria* is a tripartition $S = (\Sigma_C, \Sigma_P, \Sigma_N)$ of Σ, where Σ_C, Σ_P and Σ_N are called *case*, *pattern* and *normal* abducibles, respectively. Intuitively, S specifies that for (parts of) explanations on predicates in Σ_C, concrete constants are preferred than nulls in the explanations (showing specific cases); whereas

for those on predicates in Σ_P, it is the other way round, i.e., substitutable nulls are preferred than constants (showing patterns). A selection criteria is usually decided by a user in a particular scenario.

Given a set of formulas Φ and a signature Σ, we use $\Phi|_\Sigma$ to denote the set of formulas in Φ that are over Σ.

Definition 4.1 (Selective Explanation)**.** Given a QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ and a selection criteria $S = (\Sigma_C, \Sigma_P, \Sigma_N)$, for two explanations $\mathcal{E}$ and $\mathcal{E}'$ to Λ, $\mathcal{E} \preceq_S \mathcal{E}'$ if there exists a substitution σ such that (1) $\mathcal{E}|_{\Sigma_C} \subseteq \mathcal{E}'\sigma|_{\Sigma_C}$, (2) $\mathcal{E}\sigma|_{\Sigma_P} \subseteq \mathcal{E}'|_{\Sigma_P} \cup \mathcal{D}$, and (3) $\mathcal{E}\sigma|_{\Sigma_N} \subseteq \mathcal{E}'|_{\Sigma_N}$. And $\mathcal{E} \prec_S \mathcal{E}'$ if $\mathcal{E} \preceq_S \mathcal{E}'$ and $\mathcal{E}' \not\preceq_S \mathcal{E}$.

A *selective explanation* $\mathcal{E}$ to Λ w.r.t. S is a representative explanation to Λ such that for all representative explanations $\mathcal{E}'$ to Λ, if $\mathcal{E}' \preceq_S \mathcal{E}$, then $\mathcal{E} \preceq_S \mathcal{E}'$. The set of all selective explanations to Λ w.r.t. S (up to renaming) is denoted as $\mathsf{rexpl}_S(\Lambda)$.

Intuitively, condition (1) says that $\mathcal{E}$ is not less preferred over $\mathcal{E}'$ on Σ_C if $\mathcal{E}$ does not contain more atoms or nulls than $\mathcal{E}'$ has. Similarly, condition (2) says that it holds on Σ_P if $\mathcal{E}$ does not contain more atoms or constants than those in $\mathcal{E}'$ together with those in $\mathcal{D}$ closely associated. The reason why the condition requires a union with $\mathcal{D}$ can be see in the example below. Finally, condition (3) says that the preference coincides with $\preceq_h$ on Σ_N.

Example 4.2. *Consider the QAP Λ in Example 4.1. Let $S = (\{\mathsf{high\_risk}, \mathsf{contact}\}, \{\mathsf{has\_parent}, \mathsf{diagnosed}\}, \emptyset)$, which means (representative) explanations revealing more concrete cases on* high_risk, contact *and high-level patterns on* has_parent, diagnosed *are preferred.*

For explanations $\mathcal{E}_2 = \{\mathsf{has\_parent}(John, Bob)\}$ and $\mathcal{E}_3 = \{\mathsf{has\_parent}(John, n), \mathsf{diagnosed}(n)\}$. There exists a substitution $\sigma = \{n \mapsto Bob\}$ satisfying $\mathcal{E}_3\sigma|_{\Sigma_P} \subseteq \mathcal{E}_2|_{\Sigma_P} \cup \mathcal{D}$. That is, Conditions (1) – (3) in Definition 4.1 hold for $\mathcal{E}_3 \preceq_S \mathcal{E}_2$. The converse does not hold and hence $\mathcal{E}_3 \prec_S \mathcal{E}_2$. Intuitively, $\mathcal{E}_3$ is preferred as it shows a pattern, whereas $\mathcal{E}_2$ can be obtained from $\mathcal{E}_3$ by substituting n with Bob (and eliminating $\mathsf{diagnosed}(Bob)$ as it is in $\mathcal{D}$). Assume in the database we have a list of persons diagnosed with the disease, then each person will correspond to an explanation, while these explanations can be ignored as they are all less preferred than $\mathcal{E}_3$.

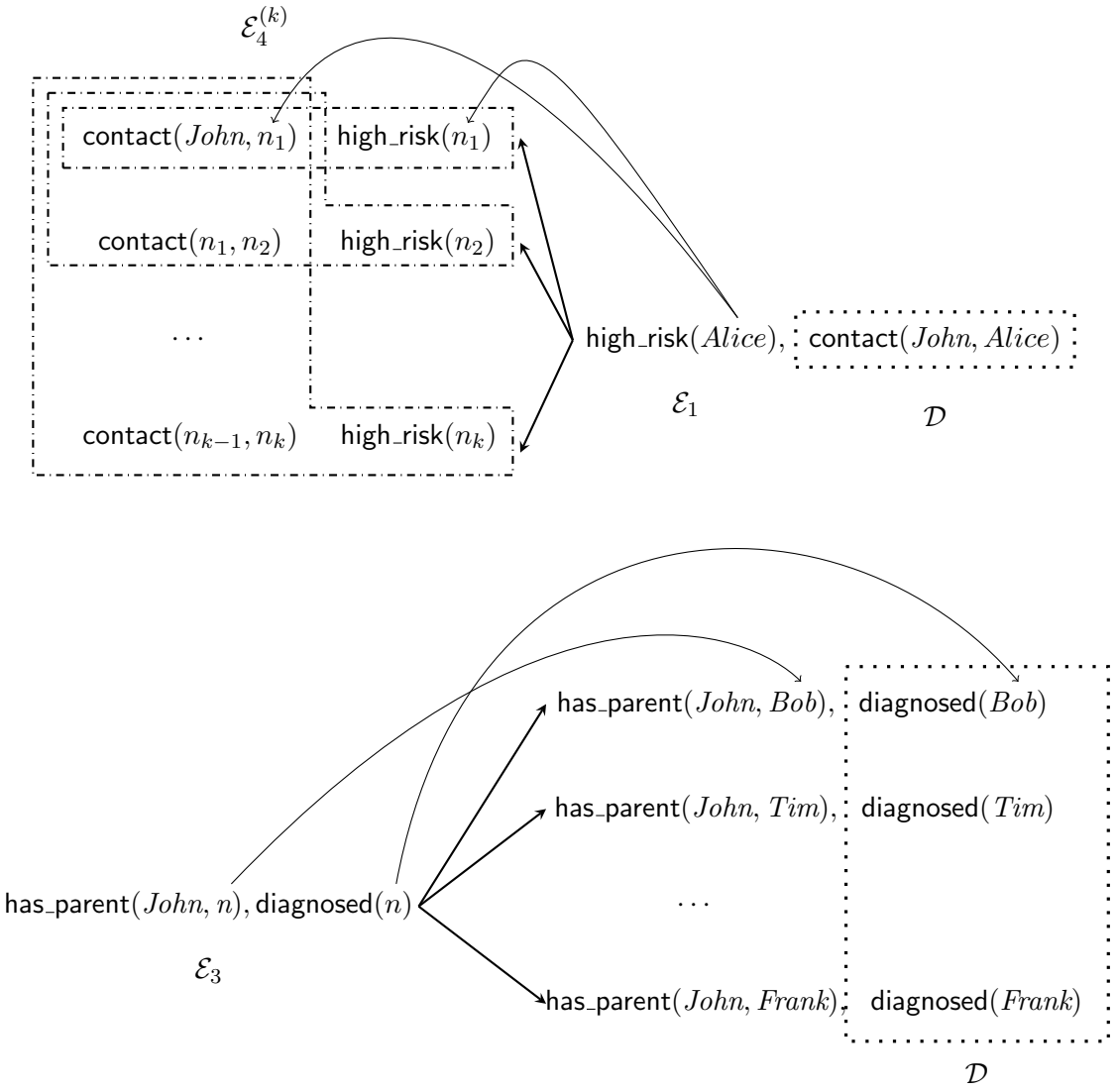


FIGURE 4.1: Preference relations for explanations in example 4.2

For another two explanations $\mathcal{E}_1 = \{\mathsf{high\_risk}(Alice)\}$ *and* $\mathcal{E}_1' = \{\mathsf{contact}(John, n'), \mathsf{high\_risk}(n')\}$. *There exists a substitution* $\sigma = \{n' \mapsto Alice\}$ *satisfying Conditions (1) – (3) for* $\mathcal{E}_1 \preceq_S \mathcal{E}_1'$, *and one can further verify that* $\mathcal{E}_1 \prec_S \mathcal{E}_1'$. *Intuitively,* $\mathcal{E}_1$ *is preferred as it reveals a concrete case, Alice, being a high risk person. Indeed, for any explanation of the form* $\mathcal{E}_4^{(k)} = \{\mathsf{contact}(John, n_1), \mathsf{contact}(n_1, n_2), \ldots, \mathsf{contact}(n_{k-1}, n_k), \mathsf{high\_risk}(n_k)\}$ *(*$k \geq 1$*), there is a substitution* $\sigma = \{n_i \mapsto Alice \mid \ 1 \leq i \leq k\}$ *satisfying Conditions (1) – (3) for* $\mathcal{E}_1 \preceq_S \mathcal{E}_4$, *and one can verify that* $\mathcal{E}_1 \prec_S \mathcal{E}_4$.

Note that the preference $\preceq_S$ *is over representative explanations, as otherwise a non-representative explanation* $\mathcal{E}_1'' = \{\mathsf{contact}(John, Bob), \mathsf{high\_risk}(Bob)\}$ *would be preferred*

than the representative explanation $\mathcal{E}'_1$ w.r.t. $\preceq_S$, contradicting the intuition of representative explanations.

We next discuss some properties of the preference relation $\preceq_S$, and the determination of the preference and the computation of selective explanations in some special cases. These properties are useful for understanding the definition and developing algorithms.

First of all, the preference relation $\preceq_S$ is reflexive and transitive.

Lemma 4.1. *For a QAP Λ and a selection criteria S, the relation $\preceq_S$ on explanations to Λ is a preorder.*

Proof. For any explanation $\mathcal{E}$ to $\mathcal{A}$, let σ be an empty substitution, then we have $\mathcal{E}\sigma|_{\Sigma_P} \subseteq (\mathcal{E} \cup D)|_{\Sigma_P}$, $\mathcal{E}\sigma|_{\Sigma_N} \subseteq (\mathcal{E} \cup D)|_{\Sigma_N}$ and because $\mathcal{E}$ is an explanation, $\mathcal{E} \cap D = \emptyset$, then $\mathcal{E}\sigma \setminus D = \mathcal{E}$, thus $\mathcal{E}|_{\Sigma_C} = (\mathcal{E}\sigma \setminus D)|_{\Sigma_C}$, we have $\mathcal{E} \preceq_R \mathcal{E}$, reflexivity is obtained. For transitivity, we assume $\mathcal{E} \preceq_S \mathcal{E}'$, $\mathcal{E}' \preceq_S \mathcal{E}''$, then with condition (1), there exist σ, σ' such that $\mathcal{E}|_{\Sigma_C} \subseteq \mathcal{E}'\sigma|_{\Sigma_C}$, and $\mathcal{E}'|_{\Sigma_C} \subseteq \mathcal{E}''\sigma'|_{\Sigma_C}$. Substituting both sides of the latter equation with σ, we have $\mathcal{E}'\sigma|_{\Sigma_C} \subseteq \mathcal{E}''\sigma'\sigma|_{\Sigma_C}$, thus $\mathcal{E}|_{\Sigma_C} \subseteq \mathcal{E}''\sigma'\sigma|_{\Sigma_C}$. Similarly, we have $\mathcal{E}|_{\Sigma_N} \subseteq \mathcal{E}''\sigma'\sigma|_{\Sigma_N}$ With condition (2), we have $\mathcal{E}\sigma|_{\Sigma_P} \subseteq (\mathcal{E}' \cup D)|_{\Sigma_P}$ and $\mathcal{E}'\sigma'|_{\Sigma_P} \subseteq (\mathcal{E}'' \cup D)|_{\Sigma_P}$, substituting both sides of the former equation with σ', we have $\mathcal{E}\sigma\sigma'|_{\mathcal{P}} \subseteq (\mathcal{E}' \cup D)\sigma'|_{\mathcal{P}} = (\mathcal{E}'\sigma' \cup D)|_{\mathcal{P}}$, thus $\mathcal{E}\sigma\sigma' \subseteq (\mathcal{E}'' \cup D)|_{\mathcal{P}}$, which suffice that $\mathcal{E} \prec_S \mathcal{E}''$ □

Next, the determination of the preference relation between two explanations is NP, which can be proved by a straightforward reduction from the homomorphism problem.

Lemma 4.2. *For two explanations $\mathcal{E}, \mathcal{E}'$ and a selection criteria S, deciding whether $\mathcal{E} \preceq_S \mathcal{E}'$ is NP-complete.*

Proof. (Membermship) Nondeterministically guess a substitution σ, condition (1), (2) and (3) can be checked in polynomial time.

(Hardness) For two sets of atoms $\mathcal{A}$ and $\mathcal{A}'$, we construct a QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$, where Π contains two rules $\mathsf{Q} \leftarrow \mathcal{A}$ and $\mathsf{Q} \leftarrow \mathcal{A}'$, $\mathcal{D} = \emptyset$, $q = \mathsf{Q}$ and $\Sigma = \mathsf{sig}(\Pi)$, then $\mathcal{A}$ and $\mathcal{A}'$ are explanations to Λ. Assume a selection criteria $S = (\emptyset, \Sigma, \emptyset)$, according to definition 4.1, we have that there is a homomorphism from $\mathcal{A}$ to $\mathcal{A}'$ iff $\mathcal{A} \prec_S \mathcal{A}$", the former is a NP-complete problem. □

While determining the preference relation between each pair of explanation is computationally expensive, we present next properties that will be useful for optimising the computation in some special cases. Given two sets of atoms $\mathcal{A}, \mathcal{A}'$, a dataset $\mathcal{D}$, and a most general substitution σ such that $\mathcal{A}' = \mathcal{A}\sigma \setminus \mathcal{D}$ with σ satisfying $(\mathcal{A} \setminus \mathcal{D})\sigma \cap \mathcal{D} \neq \emptyset$, then $\mathcal{A}$ is (strictly) *more abstract* than $\mathcal{A}'$ and $\mathcal{A}'$ is (strictly) *more specific* than $\mathcal{A}$ w.r.t. $\mathcal{D}$.

Proposition 4.1. *For a QAP Λ and a selection criteria $S = (\Sigma_C, \Sigma_P, \Sigma_N)$, given two representative explanations $\mathcal{E}, \mathcal{E}'$ to Λ, suppose $\mathcal{E}$ is more abstract than $\mathcal{E}'$ w.r.t. $\mathcal{D}$, then*

- *$\mathcal{E} \preceq_S \mathcal{E}'$ iff $\mathsf{sig}(\mathcal{E}) \setminus \mathsf{sig}(\mathcal{E}') \subseteq \Sigma_P$ and there is a renaming σ s.t. $\mathcal{E}\sigma|_{\Sigma_C} \subseteq \mathcal{E}'|_{\Sigma_C}$*
- *$\mathcal{E}' \preceq_S \mathcal{E}$ iff there is a renaming σ s.t. $\mathcal{E}'\sigma|_{\Sigma_P \cup \Sigma_N} \subseteq \mathcal{E}|_{\Sigma_P \cup \Sigma_N}$*

Proof. (1) $\mathcal{E} \preceq_S \mathcal{E}'$ iff $\mathsf{sig}(\mathcal{E}) \setminus \mathsf{sig}(\mathcal{E}') \subseteq \Sigma_P$ and there is a renaming σ s.t. $\mathcal{E}\sigma|_{\Sigma_C} \subseteq \mathcal{E}'|_{\Sigma_C}$.

($\Rightarrow$) Let $\mathcal{P} = \mathsf{sig}(\mathcal{E}) \setminus \mathsf{sig}(\mathcal{E})$, as $\mathcal{E}$ is more abstract than $\mathcal{E}'$, we have $\mathcal{P} \neq \emptyset$, if there is a $P \in \mathcal{P}$ such that $P \notin \Sigma_P$, then condition (1) and (3) for $\mathcal{E} \preceq_S \mathcal{E}'$ will not be satisfied, a contradiction. Because $\mathcal{E}$ is more abstract than $\mathcal{E}'$, we have a most general substitution σ s.t. $\mathcal{E}' = \mathcal{E}\sigma'' \setminus \mathcal{D}$, and as condition (1) is satisfied, then σ'' must not substitute terms in $\mathcal{E}|\Sigma_C$ to constants or to the same null, so σ'' is a renaming on Σ_C, thus we have $\mathcal{E}\sigma|_{\Sigma_C} \subseteq \mathcal{E}'|_{\Sigma_C}$, where σ is a renaming.

($\Leftarrow$) As $\mathcal{E}$ is more abstract than $\mathcal{E}'$ w.r.t $\mathcal{D}$, we have a substitution σ' such $\mathcal{E}\sigma' \setminus \mathcal{D} = \mathcal{E}'$, then $\mathcal{E}\sigma' = \mathcal{E}' \cup D$, because $\mathsf{sig}(\mathcal{E}) \setminus \mathsf{sig}(\mathcal{E}') \subseteq \Sigma_P$, we have $\mathcal{E}\sigma'|_{\Sigma_N} \subseteq \mathcal{E}'|_{\Sigma_N}$. Let $\sigma'' = \sigma' \cup \sigma$, then σ'' satisfies condition (1), (2) and (3).

(2) $\mathcal{E}' \preceq_S \mathcal{E}$ iff there is a renaming σ s.t. $\mathcal{E}'\sigma|_{\Sigma_P \cup \Sigma_N} \subseteq \mathcal{E}|_{\Sigma_P \cup \Sigma_N}$.

($\Rightarrow$) Assume no renaming σ exists s.t. $\mathcal{E}'\sigma|_{\Sigma_P \cup \Sigma_N} \subseteq \mathcal{E}|_{\Sigma_P \cup \Sigma_N}$, because $\mathcal{E}$ is more abstract than $\mathcal{E}$, there exists a substitution σ' such that $\mathcal{E}' \subseteq \mathcal{E}\sigma' \setminus \mathcal{D}$, then impossible to find a substitution σ'' such that σ'' is not a renaming and it satisfies condition (2) and (3), which is contradiction.

($\Leftarrow$) Similar to ($\Leftarrow$) of (1) □

Given a set of explanations $\mathbf{E}$ and a preorder $\preceq$ on $\mathbf{E}$, we use $\mathsf{min}_{\preceq}(\mathbf{E})$ to denote the set of explanations $\mathcal{E}$ in $\mathbf{E}$ such that for all $\mathcal{E}'$ in $\mathbf{E}$, if $\mathcal{E}' \preceq \mathcal{E}$, then $\mathcal{E} \preceq \mathcal{E}'$. The following

results are regarding the special cases where all abducibles are case, pattern, or normal abducibles.

Proposition 4.2. *For a QAP* $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ *and a selection criteria* $S = (\Sigma, \emptyset, \emptyset)$*, let* $\mathbf{E}$ *consists of all the most specific explanations in* $\mathsf{rexpl}(\Lambda)$ *over* Σ*. Then,* $\mathsf{rexpl}_S(\Lambda) \equiv_r \min_{\preceq_S}(\min_{\preceq_h}(\min_{\preceq_m}(\mathbf{E})))$*, which is always finite up to renaming.*

Proof. For equivalence, we only need to show every selective explanation $\mathcal{E}$ w.r.t $S = (\Sigma, \emptyset, \emptyset)$ must be a most specific explanations w.r.t $\mathcal{D}$. Assume $\mathcal{E}$ is not a most strict explanation, then there exists $\mathcal{E}'$ s.t. $\mathcal{E}'$ is more specific than $\mathcal{E}$, when all predicates are case abducibles, it satisfies that there exists σ s.t. $\mathcal{E}'|_{\Sigma_C} = \mathcal{E}\sigma|_{\Sigma_C} \setminus \mathcal{D} \subseteq \mathcal{E}\sigma|_{\Sigma_C}$, then $\mathcal{E}' \preceq_S \mathcal{E}$. And σ must be a renaming on nulls and substitute at least a null with constant, it is impossible to have $\mathcal{E} \preceq_S \mathcal{E}'$.

Next we show the set of selective explanations w.r.t S is always finite. Let $T = \mathsf{const}(\mathcal{D}) \cup \{*\}$, where $*$ is a null, and w be the maximum arity of a predicate in Σ. We consider a set explanations $\mathbf{E}^*$ which is constructed with predicates in Σ and terms in T, then the size of $\mathbf{E}^*$ is bounded by $2^{|\Sigma| \cdot |T|^w}$. Let σ be a substitution that for $t \in \mathsf{N}$, $t\sigma = *$, then for any explanation $\mathcal{E}$, there must exists an $\mathcal{E}' \in \mathbf{E}^*$ such that $\mathcal{E}\sigma = \mathcal{E}'$, according to definition 4.1, we have either $\mathcal{E}' \prec_S \mathcal{E}$, or $\mathcal{E}$ and $\mathcal{E}'$ are equivalent up to renaming, thus the set of selective explanations w.r.t S up to renaming is also bounded by $2^{|\Sigma| \cdot |T|^w}$. □

The above proposition shows that when all abducibles are case abducibles, all selective explanations are the most specific representative explanations. Furthermore, we can guarantee that the set of selective explanations up to renaming is always finite.

Proposition 4.3. *For a QAP* $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ *and a selection criteria* $S = (\emptyset, \Sigma, \emptyset)$*, let* $\mathbf{E}$ *consists of all the most abstract explanations in* $\mathsf{rexpl}(\Lambda)$ *over* Σ*. Then,* $\mathsf{rexpl}_S(\Lambda) \equiv_r \min_{\preceq_h}(\min_{\preceq_m}(\mathbf{E}))$.

Proof. We only need to show that every selective explanation $\mathcal{E}$ w.r.t $S = (\emptyset, \Sigma, \emptyset)$ is a most abstract explanation w.r.t $\mathcal{D}$. Assume there exists $\mathcal{E}'$ that is more abstract than $\mathcal{E}$ w.r.t $\mathcal{D}$, then there exists a substitution σ s.t. $\mathcal{E} = \mathcal{E}'\sigma \setminus \mathcal{D}$, then $\mathcal{E}'\sigma \subseteq \mathcal{E} \cup \mathcal{D}$. And because σ substitutes at least one null with a constant, and $|\mathcal{E}| > |\mathcal{E}'|$, it is impossible to have a substitution σ' s.t. $\mathcal{E} \subseteq \mathcal{E}'\sigma' \cup D$, thus we have $\mathcal{E}' \prec_S \mathcal{E}$, which is a contradiction. □

This proposition shows that when all abducibles are pattern abducibles, the selective explanations are all the most abstract representative explanations. While the following result shows selective explanations indeed generalize representative explanations.

Proposition 4.4. *For a QAP* $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ *and a selection criteria* $S = (\emptyset, \emptyset, \Sigma)$, $\mathsf{rexpl}_S(\Lambda) \equiv_r \mathsf{rexpl}(\Lambda)$.

A proof for the above proposition is straightforward. If there are no case and pattern abducibles, when comparing two explanations $\mathcal{E}$ and $\mathcal{E}'$, Condition (1) and (2) in definition 4.1 are trivially satisfied, and Condition (3) is exactly deciding the relation $\mathcal{E} \preceq_h \mathcal{E}'$, thus all representative explanations to Λ are in $\mathsf{rexpl}_S(\Lambda)$.

4.2 Computing Explanations

In this section, we introduce algorithms for computing selective explanations (and representative explanations) for existential rules. The algorithms are based on query rewriting, thus only work for first-order rewritable existential rules, more specifically, the algorithms terminates when the input ontologies are first-order rewritable. Our algorithms consist of three major steps: (1) To obtain the first-order rewriting of the given observation; (2) To instantiate the BCQs in the rewriting using constants from the dataset and removing those facts in the instantiations that are in the dataset, and (3) To filter out less preferred explanations.

We use $\mathsf{FORew}(\Pi, q)$ to denote the set of minimal first-order (UCQ) rewriting of q w.r.t Π, computed through the backward chaining rewriting procedure in Algorithm 1.

For convenience, when it is necessary, we assume there is a fixed bijective mapping between variables and labeled nulls, thus a BCQ can be regarded as an explanation (by substituting variables with distinct nulls).

Given a set of rules Π and a dataset $\mathcal{D}$, we use $\Pi \cup \mathcal{D}$ to denote the set of rules $\Pi \cup \{p \leftarrow \mid p \in \mathcal{D}\}$. According to the definition, the complete set of (selective or) representative explanations can contain multiple equivalent explanations, however for computation, there is no need to return all these explanations, it is more practical to return just one of them. The following lemma shows that the computation can be simplified if we just

require the result to be an equivalent set of explanations. Recall that $\mathsf{cover}(\mathbf{E})$ is a minimal subset $\mathbf{E}'$ of $\mathbf{E}$ satisfying that for every $\mathcal{E} \in \mathbf{E}$, there exist $\mathcal{E}' \in \mathbf{E}'$ s.t $\mathcal{E}' \preceq_h \mathcal{E}$.

Lemma 4.3. *Given a set of explanations* $\mathbf{E}$, $\mathsf{cover}(\mathbf{E}) \equiv \min_{\preceq_h}(\min_{\preceq_m}(\mathbf{E}))$.

Proof. If $\mathcal{E} \in \mathsf{cover}(\mathbf{E})$, then for any $\mathcal{E}' \in \min_{\preceq_h}(\min_{\preceq_m}(\mathbf{E}))$, $\mathcal{E} \preceq_h \mathcal{E}'$. According to the definition of $\min_{\preceq_m}$ and $\min_{\preceq_h}$, there exists $\mathcal{E}''$ s.t. $\mathcal{E}' \preceq_h \mathcal{E}'' \preceq_m \mathcal{E}$, because $\preceq_h$ is more general than $\preceq_m$ and transitive, we have $\mathcal{E}' \preceq_h \mathcal{E}$. The other direction is proved similarly. □

Proposition 4.5. *For a QAP* $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$, $\mathsf{rexpl}(\Lambda) \equiv \mathsf{FORew}(q, \Pi \cup \mathcal{D})|_\Sigma$.

Proof. Without loss of generality, we assume $\Sigma = \mathsf{sig}(\Pi) \cup \mathsf{sig}(q)$.

($\Rightarrow$) We show that if $\mathcal{E}$ is a representative explanation, then there exists a q' in $\mathsf{FORew}(q, \Pi \cup \mathcal{D})$ s.t. $\mathcal{E}$ is equivalent to q', i.e., $\mathcal{E} \preceq_h q'$ and $q' \preceq_h \mathcal{E}$. According to the definition of first-order rewriting, we have for any database $\mathcal{D}'$, $\Pi \cup \mathcal{D} \cup \mathcal{D}' \models q$ iff there exists a $q' \in \mathsf{FORew}(q, \Pi \cup \mathcal{D})$ s.t $\mathcal{D}' \models q'$. Let $\mathcal{D}' = \mathcal{E}$, because $\mathcal{E}$ is an explanation to Λ, we have $\Pi \cup \mathcal{D} \cup \mathcal{E} \models q$, thus we obtain $\mathcal{E} \models q'$, i.e. $q' \preceq_h \mathcal{E}$. Let δ be a bijective substitution from variables to nulls, we have $q'\delta \models q$, thus $\Pi \cup \mathcal{D} \cup (q'\delta) \models q$, $q'\delta$ is an explanation to Λ. Because $q' \preceq_h \mathcal{E}$ and δ is bijective, we also have $q'\delta \preceq_h \mathcal{E}$. And as $\mathcal{E}$ is representative, there does not exists an explanation $\mathcal{E}'$ such that $\mathcal{E}' \preceq_h \mathcal{E}$ and $\mathcal{E} \not\preceq_h \mathcal{E}'$, then we can guarantee that $\mathcal{E} \preceq_h q'\delta$, thus $\mathcal{E} \preceq_h q'$.

($\Leftarrow$) For a $q' \in \mathsf{FORew}(q, \Pi \cup \mathcal{D})$, we have $\Pi \cup \mathcal{D} \cup q'\delta \models q$, thus $q'\delta$ is an explanation to Λ. Assume there isn't a representative explanation $\mathcal{E}$ s.t. $\mathcal{E} \preceq_h q'$ and $q' \preceq_h \mathcal{E}$, then there must exist a representative explanation $\mathcal{E}'$ such that $\mathcal{E}' \preceq_h q'$ and $q' \not\preceq \mathcal{E}'$. According to the above result, there is a $q'' \in \mathsf{FORew}(q, \Pi \cup \mathcal{D})$ that is equivalent to $\mathcal{E}'$, thus we have $q'' \preceq_h q'$, however, $\mathsf{FORew}(q, \Pi \cup \mathcal{D})$ is a minimal rewriting set, which is a contradiction. □

Proposition 4.5 shows that to collect the representative explanations, we only need to compute the minimal rewriting set of the observation.

This result can be extended to compute selective explanations. According to the definition of selective explanations, one straightforward approach is to first obtain the set of representative explanations, and then remove those explanations that are less preferred

under the selection criteria. For a selection criteria S and a set of BCQs $\mathcal{Q}$, $\mathsf{cover}_{\preceq_S}(\mathcal{Q})$ is a minimal subset $\mathcal{Q}'$ of $\mathcal{Q}$ satisfying for all $q \in \mathcal{Q}$, there exists a $q' \in \mathcal{Q}'$ s.t. $q' \preceq_S q$.

Proposition 4.6. *For a QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ and a selection criteria S,*

$$\mathsf{rexpl}_S(\Lambda) \equiv \mathsf{cover}_{\preceq_S}(\mathsf{FORew}(q, \Pi \cup \mathcal{D})|_\Sigma).$$

Proof. For this proposition, we only need to prove for two explanations $\mathcal{E}$, $\mathcal{E}'$, if $\mathcal{E} \preceq_S \mathcal{E}'$ and $\mathcal{E}' \preceq_S \mathcal{E}$, then $\mathcal{E} \equiv \mathcal{E}'$, i.e, $\mathcal{E} \preceq_h \mathcal{E}'$ and $\mathcal{E}' \preceq_h \mathcal{E}$. According to definition 4.1, with condition (1), we have substitutions σ, σ' s.t. $\mathcal{E}'|_{\Sigma_C} \subseteq \mathcal{E}\sigma|_{\Sigma_C}$ and $\mathcal{E}|_{\Sigma_C} \subseteq \mathcal{E}'\sigma'|_{\Sigma_C}$, then σ and σ' must be a bijection from nulls to nulls, thus we have $\mathcal{E}|_{\Sigma_C} \preceq_h \mathcal{E}'|_{\Sigma_C}$ and $\mathcal{E}'|_{\Sigma_C} \preceq_h \mathcal{E}|_{\Sigma_C}$. Similarly, we can obtain the relation on Σ_P and Σ_N. □

The computation of selective explanations using a naive method can be very expensive, due to the large number of preference checking between explanations and the checking itself is NP-hard. Thus it is important to optimise the checking during the rewriting process. The following lemma connects Proposition 4.1 with the rewriting procedure.

Lemma 4.4. *For two BCQs q, q' and a dataset $\mathcal{D}$, if $q' \in \mathsf{FORew}(q, \mathcal{D})$ then q is more abstract than q'.*

This allows us to simplify the preference checking as our method rewrites (or more specifically substitutes the queries in the rewriting set of) the observation.

Next, we consider two special cases of the selection criteria and show the computation can be significantly simplified in these cases. That is, when all abducibles are case or pattern abducibles.

When all the abducibles are pattern ones, if an explanation is more specific than another explanation, then it is not a selective explanation, and if an explanation is rewritten with a fact, the result is always more specific. Therefore in the computation of this case, we don't need to involve the dataset as rewriting rules. The following proposition shows a simplified algorithm.

Proposition 4.7. *For a non-trivial QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ and $S = (\emptyset, \Sigma, \emptyset)$, $\mathsf{rexpl}_S(\Lambda) \equiv \mathsf{FORew}(q \setminus \mathcal{D}, \Pi)|_\Sigma$.*

Proof. It's easy to check that if Λ is non-trivial, then $\mathsf{FORew}(q\backslash\mathcal{D},\Pi) \subseteq \mathsf{FORew}(q,\Pi\cup\mathcal{D})$ and $\mathsf{FORew}(q\backslash\mathcal{D},\Pi)$ contains all the most abstract explanations in $\mathsf{FORew}(q,\Pi\cup\mathcal{D})$. Let $\mathbf{E}$ be the set of all most abstract explanations to Λ, we have $\mathsf{FORew}(q\backslash\mathcal{D},\Pi) \equiv \mathsf{cover}(\mathbf{E})$, then with proposition 4.3 and lemma 4.3, the conclusion is proved. □

The sizes of the datasets are often significantly larger than the sizes of the first-order rewritings, and a leap in the numbers of (representative) explanations occur when the rewritings are instantiated. In cases where it is important to extract explanations from an abstract level, i.e., patterns of potential explanations, by specifying all the abducibles to be pattern ones, the number of explanations can be greatly reduced. Also, this can be a first step of generating explanations based on other selection criteria, as by propositions 4.5 and 4.6, the selective explanations for any selection criteria S can be obtained from $\mathsf{FORew}(q,\Pi)|_\Sigma$ by first rewriting w.r.t. $\mathcal{D}$ and then filtering out less preferred ones via $\mathsf{cover}_{\preceq_S}$.

Another special case is when all abducibles are case ones. To compute such explanations, for each query in the rewriting set of the observation, the method tries to instantiate it w.r.t. the dataset as much as possible. For a BCQ q and a dataset $\mathcal{D}$, we define $\mathsf{FORew}^*(q,\mathcal{D})$ as the set consists of all the BCQs q' in $\mathsf{FORew}(q,\mathcal{D})$ such that q' cannot be rewritten by any rule (corresponding to a fact) in $\mathcal{D}$, that is, $\mathsf{FORew}^*(q,\mathcal{D})$ is the set of queries obtained by exhaustively rewriting q with $\mathcal{D}$. For example, consider a BCQ $q = \{\mathsf{A}(x,y),\mathsf{B}(y,z),\mathsf{C}(z)\}$ and a dataset $\mathcal{D} = \{\mathsf{A}(a,b),\mathsf{B}(b,c)\}$. Then $\mathsf{FORew}^*(q,\mathcal{D})$ consists of only $q_1 = \{\mathsf{C}(c)\}$, whereas neither $q_2 = \{\mathsf{B}(b,z),\mathsf{C}(z)\}$ nor $q_3 = \{\mathsf{A}(x,b),\mathsf{C}(c)\}$ is in $\mathsf{FORew}^*(q,\mathcal{D})$. For a set of existential rules Π, $\mathsf{FORew}^*(q,\Pi\cup\mathcal{D})$ is obtained from $\mathsf{FORew}(q,\Pi)$ by replacing each BCQ q' in $\mathsf{FORew}(q,\Pi)$ with $\mathsf{FORew}^*(q',\mathcal{D})$. The following proposition shows that to compute selective explanations in this special case, the algorithm can apply rules in $\mathcal{D}$ with higher priority than rules in Π.

Proposition 4.8. *For a QAP* $\Lambda = (\Pi,\mathcal{D},q,\Sigma)$ *and* $S = (\Sigma,\emptyset,\emptyset)$, $\mathsf{rexpl}_S(\Lambda) \equiv \mathsf{cover}_{\preceq_S}(\mathsf{FORew}^*(q,\Pi\cup\mathcal{D})|_\Sigma)$.

Proof. First, it can be checked that the following lemma holds.

Lemma 4.5. *For a BCQ* q*, a database* $\mathcal{D}$ *and* $S = (\Sigma,\emptyset,\emptyset)$, $\mathsf{FORew}^*(q,\mathcal{D}) = \min_{\preceq_S}(\mathsf{FORew}(q,\mathcal{D}))$.

Then we show that $\mathsf{FORew}(q, \Pi \cup \mathcal{D}) = \mathsf{FORew}(\mathsf{FORew}(q, \Pi), \mathcal{D})$, the proof is established by the following statement: for any query q' in $\mathsf{FORew}(q, \Pi \cup \mathcal{D})$, if q' is obtained from a rewriting sequence $qr_1 \dots r_n$, where r_i is the rule used for rewriting in the i'th iteration, then q' can always be obtained from a rewriting sequence $qr_{c_1} \dots r_{c_m} \dots r_{c_n}$ where r_{c_i} are rules from $\{r_1, \dots, r_n\}$, and for $i \leq m$, r_{c_i} are rules with non-empty bodies, otherwise facts. Consider a query q^* obtained from a rewriting sequence $qr_1 \dots r_{i-1} d_i r_{i+1} \dots r_n$, where d_i is a fact. Let q_1, q_2, q_3 be the result of $qr_1 \dots r_{i-1}$, $qr_1 \dots r_{i-1} d_i$ and $qr_1 \dots r_{i-1} d_i$ r_{i+1} respectively. Assume $q_1 = B \cup \{\mathsf{P}(\vec{v})\}$, where B is a set of atoms and P is the predicate of d_i. Let $\mathsf{P}(\vec{v})\sigma = d_i$, then $q_2 = B\sigma$, let $\mu = (B'\sigma, \tau)$ be the unifier used for rewriting q_2 to q_3, then $q_3 = B^-\sigma\tau \cup \mathsf{body}(r_2)\tau$, where $B^- = B \setminus B'$. We construct a substitution τ' s.t τ' agrees τ on variables not occurring in $\vec{v}$ and $\tau'\sigma = \tau$, as μ is a piece unifier between q_1 and $r_(i+1)$, we have $B^-\sigma\tau = H^-\tau$, where H' is the piece in $\mathsf{head}(r_{i+1})$, then $B^-\sigma\tau'\sigma = H^-\tau'\sigma \Rightarrow B^-\tau'\sigma = H^-\tau'\sigma \Rightarrow B^-\tau' = H^-\tau'$, thus $\mu' = (B^-, \tau')$ is a piece unifier $\mu' = (B', \tau')$ between q_1 and r_{i+1}, the result of which is $q'_2 = B^-\tau' \cup \mathsf{body}(r_{i+1})\tau' \cup \{P(\vec{v})\tau'\}$, let σ' be the substitution such that $\mathsf{P}(\vec{v})\tau'\sigma' = d_i$, then the result of rewriting q'_2 with d_i is $q'_3 = B^-\tau'\sigma' \cup \mathsf{body}(r_{i+1})\tau'\sigma'$. For $v \in \mathsf{vars}(B^-) \setminus \vec{v}$, we have $v\tau'\sigma' = v\tau' = v\tau'\sigma = v\sigma\tau = v\sigma\tau$, and for $v \in \vec{v}$, we have $v\tau'\sigma' = v\sigma = v\sigma\tau$, therefore $B^-\tau'\sigma' = B^-\sigma\tau$. Similarly, we have $\mathsf{body}(r_{i+1})\tau'\sigma' = \mathsf{body}(r_{i+1})\tau$, thus $q'_3 = q_3$, which means q^* can also be obtained from sequence $qr_1 \dots r_{i-1} r_{i+1} d_i \dots r_n$. Then the statement is proved.

According to the definition, $\mathsf{FORew}^*(q, \Pi \cup \mathcal{D}) = \{q'' \in \mathsf{FORew}^*(q', \mathcal{D}) \mid q' \in \mathsf{FORew}(q, \Pi)\}$ $= \{q'' \in \min_{\preceq_S}(\mathsf{FORew}(q', \mathcal{D})) \mid q' \in \mathsf{FORew}(q, \Pi)\}$, and because $\mathsf{FORew}(q, \Pi \cup \mathcal{D}) = \mathsf{FORew}(\mathsf{FORew}(q, \Pi), \mathcal{D}) = \{q'' \in \mathsf{FORew}(q', \mathcal{D}) \mid q' \in \mathsf{FORew}(q, \Pi)\}$, then we can conclude that for any $q' \notin \mathsf{FORew}^*(q, \Pi \cup \mathcal{D})$, q' is not a selective explanation w.r.t S, which suffice the proof. □

With the above results, we can produce a procedure for computing selective explanations as presented in Algorithm 3, which is adapted from the classical backward chaining rewriting algorithm. Particularly, Algorithm 3 distinguishes the two special cases from the general case for optimization. *ALL_PATTERN* denotes that all abducibles are pattern abducibles and *All_CASE* denotes that all abducibles are case abducibles. The following proposition can be directly proved with the results of proposition 4.6, 4.7, 4.8.

Algorithm 3: $\mathsf{abdu}(\Lambda, S)$

Input : A non-trivial QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$, a selective criteria S

```
1  begin
2      q ← (q \ D)|Σ;
3      F ← {q}; // result set
4      E ← {q}; // explore set
5      while E ≠ ∅ do
6          R ← FORew_1(E, Π); // one-step rewriting
7          if not ALL_PATTERN then
8              if All_CASE then
9                  R ← R ∪ FORew*(E, D);
10             else
11                 R ← R ∪ FORew(E, D);
12         R ← cover_{⪯h}(R ∪ F); // compute cover
13         R ← R|Σ; // remove explanations that are not over Σ
14         E ← R \ F; // next exploration set
15         F ← R;
16     if not ALL_PATTERN then
17         F ← cover_{⪯S}(F);
18     return F
```

Proposition 4.9. *Given a non-trivial QAP* $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$*, and a selective criteria* S*,* $\mathsf{abdu}(\Lambda, S)$ *returns the set of all selective explanations to* Λ *w.r.t* S*.* $\mathsf{abdu}(\Lambda, S)$ *terminates if* Π *is first-order rewritable,*

4.3 Compact Explanations

In Example 4.1, $\mathcal{E}_1 = \{\mathsf{high\_risk}(Alice)\}$ is a (representative) explanation, which is based on the fact $\mathsf{contact}(John, Alice)$ in the database $\mathcal{D}$. Now, suppose one adds to $\mathcal{D}$ facts of a similar form $\mathsf{contact}(John, Adam), \mathsf{contact}(John, Alex), \ldots$, there will be (representative) explanations $\mathcal{E}'_1 = \{\mathsf{high\_risk}(Adam)\}$, $\mathcal{E}''_1 = \{\mathsf{high\_risk}(Alex)\}$, ..., all following a similar pattern. From Proposition 4.5, these explanations can be obtained from a same query in a rewriting set of the observation, i.e., $q' \in \mathsf{FORew}(q, \Pi)$ of the form $\exists x.\mathsf{contact}(John, x) \wedge \mathsf{high\_risk}(x)$. And if an instance of the form $\mathsf{contact}(John, x\sigma)$ is found in $\mathcal{D}$ then $\{\mathsf{high\_risk}(x\sigma)\}$ is a (representative) explanation. Such a pattern can be captured by a rule $r : \mathsf{high\_risk}(x) \leftarrow \mathsf{contact}(John, x)$, and the above explanations can be derived by applying r to $\mathcal{D}$. Hence, rule r can be seen as a compact representation of the explanations $\mathcal{E}_1$, $\mathcal{E}'_1$, $\mathcal{E}''_1$, ... w.r.t. $\mathcal{D}$. Such a compact representation can

be constructed directly from the rewriting set before all the concrete explanations are generated.

Note that the compact representations as rules are different from the rules in the ontology. First, a compact representation $\mathsf{high\_risk}(x) \leftarrow \mathsf{contact}(John, x)$ may not hold in general (as ontological rules), but is specific to a QAP, i.e., if $\mathsf{contact}(John, a)$ is found in the given dataset for some constant a then $\{\mathsf{high\_risk}(a)\}$ is an explanation to the given observation. Also, each rule (as a compact representation) should not be used together with other rules nor applied recursively; for example, if $\{\mathsf{high\_risk}(a)\}$ is derived as an explanation then it does not make sense to applied other rules to it. Moreover, our experiments show the numbers of compact representations are much smaller than the total numbers of rules in the ontologies.

In this section, we propose a compact form for (representative and selective) explanations, which provides a simple and intuitive way for representing the explanations using rules. To achieve this, we need to extend rules with negations in the bodies, that is, we consider rules r of the form

$$\forall \vec{x}.\forall \vec{y}.[\varphi(\vec{x}, \vec{n}) \leftarrow \psi(\vec{x}, \vec{y}) \wedge \neg\psi_1(\vec{x}, \vec{y}, \vec{z}_1) \wedge \cdots \wedge \neg\psi_n(\vec{x}, \vec{y}, \vec{z}_n)],$$

where $\vec{x}$, $\vec{y}$ and $\vec{z}_i$ $(1 \leq i \leq n)$ are pairwise disjoint vectors of variables, $\vec{n}$ is a vector of labelled nulls, and $\varphi(\vec{x}, \vec{n})$, $\psi(\vec{x}, \vec{y})$, and $\psi_i(\vec{x}, \vec{y}, \vec{z}_i)$ $(1 \leq i \leq n)$ are conjunctions of atoms. Particularly, we use $\mathsf{body}_F(r)$ to denote the formula of extended body. Given a dataset $\mathcal{D}$, an instantiation σ from $\mathsf{vars}(r)$ to $\mathsf{const}(\mathcal{D})$ is a valid instantiation for r and $\mathcal{D}$ if $\mathcal{D} \models \mathsf{body}_F(r)$, Let $r(\mathcal{D}, \sigma) = \mathsf{head}(r)\sigma'$, we say $r(\mathcal{D}, \sigma)$ is an instance of r with $\mathcal{D}$ if σ is a valid instantiation. $r(\mathcal{D})$ denotes the set of all instances of r with $\mathcal{D}$, and for a set of rules $\mathcal{R}$, $\mathcal{R}(\mathcal{D}) = \bigcup_{r\in\mathcal{R}} r(\mathcal{D})$.

To generate a compact representation of explanations expressed as rules from the first-order rewritings of observations, we consider *bipartitions* of BCQs q of the form $\lambda = (q_D, q_E)$; intuitively, q_D is the part of q that can be homomorphically mapped to the dataset with some substitution σ, and q_E is the part of q that cannot be mapped to $\mathcal{D}$ under σ but can form an explanation. Note that q_D or q_E can be empty. For the BCQ $q' = \{\mathsf{contact}(John, x), \mathsf{high\_risk}(x)\}$ in the example above, a bipartition is $q'_D = \{\mathsf{contact}(John, x)\}$ and $q'_E = \{\mathsf{high\_risk}(x)\}$. Not all bipartitions will lead to explanations. For a QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ and a BCQ $q' \in \mathsf{FORew}(q, \Pi)$, a bipartition

$\lambda = (q'_D, q'_E)$ of q' is *solution-forming* to Λ if there exists a substitution σ satisfying $\mathsf{sig}(q'_E) \subseteq \Sigma$, $q'_D\sigma \subseteq \mathcal{D}$, and $q'_E\sigma \cap \mathcal{D} = \emptyset$.

We want to compute compact representations through (solution-forming) bipartitions. Given a solution-forming bipartition $\lambda = (q_D, q_E)$ to a QAP Λ, a rule can be obtained from λ, that is, $r_\lambda : \wedge q'_E \leftarrow \wedge q_D$ where q'_E is obtained from q_E by substituting each variable in $\mathsf{vars}(q_E) \setminus \mathsf{vars}(q_D)$ with a distinct labelled null.

An *instantiation* is a substitution from N_V to N_C.

Proposition 4.10. *Given a QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ and a solution-forming bipartition $\lambda = (q'_D, q'_E)$ to Λ, for each instantiation π from q'_D to $\mathcal{D}$, $r_\lambda(\mathcal{D}, \pi)$ is an explanation to Λ.*

Proof. The proof is straightforward. From the above discussion, we know for any $q' \in \mathsf{FORew}(q, \Pi)$, q' is an explanation to Λ. Let σ be a homomorphism from q'_D to $\mathcal{D}$, clearly $q'\sigma$ is also an explanation to Λ, as $q'\sigma = q'_D\sigma \cup q'_E\sigma$ and $q'_E\sigma \subseteq \mathcal{D}$, then we have $\Pi \cup q'_D\sigma \cup \mathcal{D} \models q$, so $q'_D\sigma$ is an explanation to Λ. □

Yet not all explanations represented by the rules are minimal or representative. Even for a single rule r_λ, it is possible that there exist two homomorphisms σ and σ' such that $r_\lambda(\mathcal{D}, \sigma)$ is a minimal (or representative) explanation but $r_\lambda(\mathcal{D}, \sigma')$ is not.

Example 4.3. *Consider $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ with $\mathcal{D} = \{\mathsf{A}(a), \mathsf{B}(a, a), \mathsf{B}(a, b)\}$ and $\Sigma = \{\mathsf{C}, \mathsf{D}\}$. Suppose two BCQs in a rewriting set of q by Π are $\{\mathsf{A}(x), \mathsf{C}(x, x, y)\}$ and $\{\mathsf{B}(x', y'), \mathsf{C}(x', y', z'), \mathsf{D}(z')\}$. There are two solution-forming bipartitions to Λ, $\lambda_1 = (\{\mathsf{A}(x)\}, \{\mathsf{C}(x, x, y)\})$ and $\lambda_2 = (\{\mathsf{B}(x', y')\}, \{\mathsf{C}(x', y', z'), \mathsf{D}(z')\})$. The following rules can be constructed,*

$$r_{\lambda_1} : \mathsf{C}(x, x, u) \leftarrow \mathsf{A}(x). \quad r_{\lambda_2} : \mathsf{C}(x', y', v) \wedge \mathsf{D}(v) \leftarrow \mathsf{B}(x', y').$$

Here u, v are labelled nulls.

Then, $r_{\lambda_1}(\mathcal{D}) = \{\mathcal{E}_1\}$ with $\mathcal{E}_1 = \{\mathsf{C}(a, a, u)\}$, and $r_{\lambda_2}(\mathcal{D}) = \{\mathcal{E}_2, \mathcal{E}_3\}$ with $\mathcal{E}_2 = \{\mathsf{C}(a, a, v), \mathsf{D}(v)\}$ and $\mathcal{E}_3 = \{\mathsf{C}(a, b, v), \mathsf{D}(v)\}$. $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ are all explanations, but $\mathcal{E}_2$ is not a representative explanation, as there exists $\sigma = \{u \mapsto v\}$ s.t. $\mathcal{E}_1\sigma \subset \mathcal{E}_2$. To capture only

the representative explanations, i.e., $\mathcal{E}_3$ but not $\mathcal{E}_2$, intuitively, r_{λ_2} need to be refined w.r.t r_{λ_1} as follows

$$r'_{\lambda_2} : \mathsf{C}(x', y', v) \wedge \mathsf{D}(v) \leftarrow \mathsf{B}(x', y') \wedge \neg\big(x' = y' \wedge \mathsf{A}(x')\big).$$

First, we demonstrate when a rule should be refined by another rule, so that the result of refinement can capture the minimal, representative, and selective explanations.

Definition 4.2. For a relation on explanations $\prec \in \{\prec_m, \prec_h, \prec_S\}$, a dataset $\mathcal{D}$, and two rules r and r' obtained from solution-forming bipartitions, a substitution ρ from $\mathsf{vars}(\mathsf{head}(r) \cup \mathsf{head}(r'))$ to $\mathsf{vars}(\mathsf{head}(r'))$ is a $\prec$-*association* from r to r' w.r.t. $\mathcal{D}$ if there exists an instantiation π such that (i) $(\mathsf{body}(r) \cup \mathsf{body}(r'))\rho\pi \subseteq \mathcal{D}$ and (ii) $\mathsf{head}(r)\rho\pi \prec \mathsf{head}(r')\rho\pi$.

In Example 4.3, $\rho = \{x' \mapsto x, y' \mapsto x\}$ is a $\prec_h$-association from r_{λ_1} to r_{λ_2} w.r.t. $\mathcal{D}$, as there exists a distinctive instantiation $\pi = \{x \mapsto a\}$ satisfying the conditions (i) and (ii) in Definition 4.2.

The following lemma shows that $\prec$-association can fully capture the relations between explanations instantiated from solution-forming rules. Note that each substitution σ defines an equivalence relation $\sim_\sigma$ over terms such that for two terms t_1, t_2, $t_1 \sim_\sigma t_2$ iff $t_1\sigma = t_2\sigma$. For a term t, $[t]_\sigma$ is the equivalent class for t under $\sim_\sigma$, i.e., $[t]_\sigma = \{t' \mid t \sim_\sigma t'\}$.

Lemma 4.6. *For a relation $\prec$, a dataset $\mathcal{D}$ and two solution-forming rules r, r', there exist $\mathcal{E} \in r(\mathcal{D})$ and $\mathcal{E}' \in r'(\mathcal{D})$ s.t. $\mathcal{E} \prec \mathcal{E}'$ if and only if there exists a $\prec$-association from r to r' w.r.t $\mathcal{D}$.*

Proof. ($\Leftarrow$) The left direction is straightforward. Let ρ be a $\prec$-association from r to r' w.r.t $\mathcal{D}$, and π a instantiation satisfying condition (i) and (ii) in definition 4.2. Let $\mathcal{E} = \mathsf{head}(r)\rho\pi$ and $\mathcal{E}' = \mathsf{head}((r))\rho\pi$, clearly $\mathcal{E} \in r(\mathcal{D})$ and $\mathcal{E}' \in r'(\mathcal{D})$, and according to proposition 4.10, they are explanations. With condition (ii), we have $\mathcal{E} \prec \mathcal{E}'$.

($\Rightarrow$) Let $\mathcal{E} \in r(\mathcal{D})$, $\mathcal{E}' \in r'(\mathcal{D})$ and $\mathcal{E} \prec \mathcal{E}'$, then there are instantiations π and π' s.t. $\mathcal{E} = \mathsf{head}(r)\pi$ and $\mathcal{E}' = \mathsf{head}(r')\pi'$. Particularly, we assume σ is the substitution that witnesses $\mathcal{E} \prec \mathcal{E}'$. Now we construct a substitution ρ from $\mathsf{vars}(\mathsf{head}(r) \cup \mathsf{head}(r'))$ to $\mathsf{vars}(\mathsf{head}(r'))$ as follows: for $v \in \mathsf{vars}(\mathsf{head}(r'))$, let v^* be a representative of the

equivalent class $[v]_\sigma$, for $v' \in [v]_\sigma$, we add $\{v' \mapsto v^*\}$ to ρ; for $v \in \mathsf{vars}(\mathsf{head}(r))$, $v' \in \mathsf{vars}(\mathsf{head}(r'))$ that satisfy $v\pi = v'\pi'$, let v^* be a representative of $[v']_\sigma$, we add $\{v \mapsto v^*\}$ to ρ. Then there exists an instantiation π'' s.t for $v \in \mathsf{head}(r)$, $v\rho\pi'' = v\pi$ and for $v' \in \mathsf{head}(r')$, $v'\rho\pi'' = v'\pi'$. It can be verified that ρ and π'' are substitutions satisfying condition (i) and (ii), thus ρ is a $\prec$-association from r to r' w.r.t $\mathcal{D}$. □

For a dataset $\mathcal{D}$ and two rules r, r', let $\Delta_\mathcal{D}(r, r', \prec)$ be the set of most general $\prec$-associations from r' to r w.r.t. $\mathcal{D}$, which is finite up to renaming.

For a $\prec$-association ρ from r to r' w.r.t $\mathcal{D}$, ρ is *most general*, if there does not exist another $\prec$-association ρ' from r to r' w.r.t $\mathcal{D}$ and a substitution σ s.t. $\rho'\sigma = \rho$. In the above example, it can be verified that ρ is a most general $\prec_h$-association from r_{λ_1} to r_{λ_2} w.r.t. $\mathcal{D}$.

Next we present the refined rule using $\prec$-associations. For a set of terms $T = \{t_i \mid 1 \leq i \leq n\}$, let $\varphi_\approx(T) = \bigwedge_{i=1}^{n-1} t_i = t_{i+1}$.

Definition 4.3. For a rule r, a set of rules $\mathcal{R}$, a database $\mathcal{D}$, and a preoroder $\prec$, let $\Delta_{r',r}$ be the set of all most general $\prec$-associations from r' to r w.r.t $\mathcal{D}$, the refinement of r with $\mathcal{R}$ and $\mathcal{D}$ under $\prec$, denoted $\mathsf{ref}_\prec(r, \mathcal{R}, \mathcal{D})$, is the following rule

$$\bigwedge \mathsf{head}(r) \leftarrow \bigwedge \mathsf{body}(r) \wedge \bigwedge_{r' \in \mathcal{R}, r' \neq r} \Big(\bigwedge_{\sigma \in \Delta_{r,r'}} \phi^{\neg}_{r',\sigma} \Big).$$

$$\phi^{\neg}_{r',\sigma} = \neg \Big(\bigwedge_{t \in \mathsf{vars}(r')} \varphi_\approx\big([t]_\sigma \cap \mathsf{vars}(r)\big) \wedge \mathsf{body}(r')\sigma \Big)$$

In Example 4.3, consider $\sigma = \{x' \mapsto x, y' \mapsto x\}$ and x, x', y' all belong to the same equivalent class. Taking x' as the representative for this equivalent class, let $\rho^\sigma_{\mathsf{vars}(r_{\lambda_2})} = \{x \mapsto x', y' \mapsto x'\}$. Also, $\varphi_\approx\big([x]_\sigma \cap \mathsf{vars}(r_{\lambda_2})\big)$ is $x' = y'$. Hence, r_{λ_2} is refined by r_{λ_1} w.r.t. σ to obtain $r'_{\lambda_2} : \mathsf{C}(x', y', v) \wedge \mathsf{D}(v) \leftarrow \mathsf{B}(x', y') \wedge \neg\big(x' = y' \wedge \mathsf{A}(x')\big)$.

Given a set of rules $\mathcal{R}$, let $\mathsf{ref}_\prec(\mathcal{R}, \mathcal{D}) = \{\mathsf{ref}_\prec(r, \mathcal{R}, \mathcal{D}) \mid r \in \mathcal{R}\}$.

Proposition 4.11. *For a QAP $\Lambda = (\Pi, \mathcal{D}, q, \Sigma)$ and a selection criteria S, let $\mathcal{R}$ consists of all the rules of the form r_λ with λ being a solution-form bipartition to Λ. Then, take $\mathcal{R}_r = \mathsf{ref}_{\prec_h}\big(\mathsf{ref}_{\prec_m}(\mathcal{R})\big)$ and $\mathcal{R}_s = \mathsf{ref}_{\prec_S}(\mathcal{R}_r)$, we have*

(1) $\mathsf{rexpl}(\Lambda) \equiv \mathcal{R}_r(\mathcal{D})$.

(2) $\mathsf{rexpl}_S(\Lambda) \equiv \mathcal{R}_s(\mathcal{D})$.

Proof. We show that $(\mathsf{ref}_{\prec_h}(\mathcal{R}))(\mathcal{D}) \equiv \min_{\preceq_h}(\mathcal{R}(\mathcal{D}))$, conclusions for other relation can be similarly proved, thus the result of the proposition follow.

We first show if $\mathcal{E} \notin (\mathsf{ref}_{\prec_h}(\mathcal{R}))(\mathcal{D})$, then there exists $\mathcal{E}' \in \mathcal{R}(\mathcal{D})$ s.t. $\mathcal{E}' \prec_h \mathcal{E}$. Let $\mathcal{E}$ be an instance of $r \in \mathcal{R}$ with instantiation σ, and $r^* = \mathsf{ref}_{\prec_h}(r, \mathcal{R}, \mathcal{D})$. Because $\mathcal{E} \notin (\mathsf{ref}_{\prec_h}(\mathcal{R}))(\mathcal{D})$, any instantiation agreeing σ is not a valid instantiation for r^* and $\mathcal{D}$, which means there exists a conjunct $\phi^{\neg}_{r',\rho}$ of r^* s.t. $\mathcal{D} \not\models \phi^{\neg}_{r',\rho}\sigma$. Because ρ is a $\prec_h$-association from r' to r w.r.t $\mathcal{D}$, any instantiation π agreeing ρ satisfies $\mathsf{head}(r')\rho\pi \prec_h \mathsf{head}(r)\rho\pi$. As $\mathcal{D} \not\models \phi^{\neg}_{r',\rho}$, then σ must agrees ρ and $\mathcal{D} \models \mathsf{body}(r')\sigma$, so $\mathcal{E}' = \mathsf{head}(r')\sigma$ is an instance of r'. And because σ agrees ρ, we have $\mathsf{head}(r')\sigma \prec_h \mathsf{head}(r)\sigma$, thus $\mathcal{E}' \prec_h \mathcal{E}$.

Next we show if $\mathcal{E} \notin \min_{\preceq_h}(\mathcal{R}(\mathcal{D}))$, then $\mathcal{E} \notin (\mathsf{ref}_{\prec_h}(\mathcal{R}))(\mathcal{D})$. Let $\mathcal{E}$ is an instance of $r \in \mathcal{R}$ with instantination σ, i.e., $\mathcal{E} = \mathsf{head}(r)\sigma$. As $\mathcal{E} \notin \min_{\preceq_h}(\mathcal{R}(\mathcal{D}))$, there exists $\mathcal{E}' \in \mathcal{R}(\mathcal{D})$ s.t. $\mathcal{E}' \prec_h \mathcal{E}$. Let $r' \in \mathcal{R}$ be the rule that produces $\mathcal{E}'$ with instantiation σ, then $\mathcal{E}' = \mathsf{head}(r')\sigma'$ and $\mathcal{D} \models \mathsf{body}_F(r)\sigma'$. Because $\mathcal{E}' \prec_h \mathcal{E}$, similar to the proof of lemma 4.6, we can construct a $\prec_h$-association ρ from r' to r w.r.t $\mathcal{D}$ s.t. $\rho\pi = \sigma$ on r and $\rho\pi = \sigma'$ on r', where π is an instantiation. It can seen that $\bigwedge_{t \in \mathsf{vars}(r')} \varphi{\approx}\big([t]_\rho \cap \mathsf{vars}(r)\big)$ is always satisfied if we substitutes variables in r with σ, which is less general than ρ. And as $\rho\pi = \sigma'$ on r' and $\mathsf{body}(r)\sigma' \subseteq \mathcal{D}$, we have $\mathsf{body}(r')\rho\sigma\pi = \mathsf{body}(r')\rho\pi = \mathsf{body}(r')\sigma' \subseteq \mathcal{D}$. Hence, $\phi^{\neg}_{r',\rho}\sigma$ can not be satisfied by $\mathcal{D}$. Then σ is not a valid instance of r after refinement, thus $\mathcal{E} \notin (\mathsf{ref}_{\prec_h}(\mathcal{R}))(\mathcal{D})$. □

Example 4.4. *Continue with Example 4.3, and consider a selection criteria* $S = (\{\mathsf{C}, \mathsf{D}\}, \emptyset, \emptyset)$. *Suppose there is another BCQ in the rewriting set of* q *by* Π *being* $\{\mathsf{A}(x), \mathsf{D}(x)\}$, *and a corresponding solution-forming bipartition is* $\lambda_3 = (\{\mathsf{A}(x)\}, \{\mathsf{D}(x)\})$. *It leads to a rule* $r_{\lambda_3} : \mathsf{D}(x'') \leftarrow \mathsf{A}(x'')$. *Then,* $\sigma = \emptyset$ *is a* $\prec_S$*-association from* r_{λ_3} *to* r_{λ_2}. *Hence,* $[x''] = \{x\}$ *and* $[x'']_\sigma \cap \mathsf{vars}(r_{\lambda_2}) = \emptyset$. *Also,* $\rho^{\sigma}_{\mathsf{vars}(\lambda_2)} = \emptyset$. *Hence,* r'_{λ_2} *will be further refined by* r_{λ_3} *to*

$$r''_{\lambda_2} : \mathsf{C}(x', y', v) \wedge \mathsf{D}(v) \leftarrow \mathsf{B}(x', y') \wedge \neg\big(x' = y' \wedge \mathsf{A}(x')\big) \wedge \neg\mathsf{A}(x'').$$

TABLE 4.1: Comparison between representative and selective explanations

Ontology	Query	Representative			Selective-Pattern		Selective-Case		
		#Rules	#FSets	Time	#FSets	Time	#Rules	#FSets	Time
LUBM	q1	4	557	323	2	9	2	555	368
	q2	5	13597	699	1	51	2	2167	155
	q3	12	18871	190	4	0	7	17093	240
	q4	10	40158	664	2	963	2	555	712
	q5	50	57546	6537	10	36	10	15086	322
Semintec	q1	4	1786	252	2	3	2	1784	108
	q2	6	2730	77	2	10	4	2728	51
	q3	6	3570	44	2	3	4	3568	46
	q4	6	18002	199	2	5	4	18000	202
	q5	6	3570	537	2	22	6	1788	87
Vicodi	q1	5	1259	322	3	3	3	1257	91
	q2	3	1257	42	1	11	2	1256	10
	q3	1	1	0	1	0	1	1	0
	q4	189	556596	4478	84	29	116	556060	5216
	q5	247	866896	7624	118	95	150	866336	5743
STB-128	q1	9	50306	794	4	3	6	39931	1001
	q2	3	9998	66	2	1	2	9997	127
	q3	3	10001	114	2	0	2	10000	131
	q4	15	80457	725	6	4	10	69963	1056
	q5	15	69773	644	6	1	10	69484	731
ONT-256	q1	9	39401	488	4	5	6	39284	779
	q2	9	39696	354	4	1	6	39502	613
	q3	9	49661	473	4	1	4	29680	409
	q4	11	39759	386	4	2	6	39336	582
	q5	9	39606	317	4	1	6	39264	462

4.4 Evaluation

We implemented a prototype system for the computation of both representative explanations and selective explanations on top of Drewer. Since the system in [Du et al., 2014] cannot handle several of the ontologies we evaluated, we use the representation explanations generated in our system as a baseline for comparison. All experiments were performed on a laptop with a processor at 2.20GHz CPU and 16GB of RAM.

LUBM, STB-128, ONT-256 and two new DL ontologies, Semintec and Vicodi are considered for evaluation. Semintec is about financial services and Vicodi is about European history. The numbers of rules in these ontologies range from 88 to 529, and the numbers of facts in their companying datasets range from 100k to 2M.

For each ontology, we used 5 BCQs as observations. Each of the BCQs is obtained from a CQ provided in existing benchmarks (benchmarks used in Chapter 3) by adding

an atom with a fresh predicate (not occurring in the ontology). It guarantees that none of the BCQs is entailed by the ontologies (together with their datasets), otherwise the abduction problem is trivial and there will be no difference between the results of different explanations. The difficulty of a QAP largely depends on the number of variables in the observation. Unlike many evaluations using observations with at most one variable, we challenge our systems with observations with 2-3 variables each. It should be noted that the obtained BCQs are essentially equivalent to the original CQs and the resulting explanations still make sense if the fresh predicate is ignored.

For each ontology and each BCQ observation, all the predicates in the ontology together with the fresh predicate introduced in the BCQ are specified as the abducibles.

For each QAP evaluated, we compare the numbers of representative explanations (Representative) and selective explanations, as well as the times (in milliseconds) used for generating them. In particular, for selective explanations, we consider the two special cases where all abducibles are pattern ones (Pattern) and all of them are case ones (Case). For the numbers of explanations, we report both the numbers of fact sets (#FSets), and the numbers of rules (#Rules) as the corresponding compact representations. For Pattern, as Proposition 4.7 shows that the selective explanations are all non-instantiated, their compact representations are rules with empty bodies and the numbers of rules coincide with the numbers of fact sets, hence we only report the numbers once. The results are shown in Table 4.1.

We can see that for Representative and Selective-Case, the numbers of explanations expressed as fact sets are often several magnitudes larger than those in the form of rules needed to represent them, which indicates most explanations share similar patterns. Also, the numbers of explanations for Pattern are significantly smaller than Representative and Case, due to the fact that such explanations are non-instantiated. Compared to Pattern, the numbers of explanations (as fact sets) for Case are much larger due to instantiations, but can still be significantly smaller than Representative (e.g., q2, q4, and q5 for LUBM, q5 for Semintec, q1 and q4 for STB-128, and q3 for ONT-256). While computing explanations for Case may take more time than for Representative due to more complex preference checking, for some cases (e.g., q2 and q5 for LUBM, q1 and q5 for Semintec, and q1 and q2 for Vicodi), the computation for Case is more efficient due to our optimisations for this special case.

Chapter 5

Inconsistency-tolerant Forgetting for Ontologies

Query answering can become rather inefficient when an ontology is large even for tractable languages. To attain efficient performance in real-world applications, it is necessary to control the sizes of ontologies by proper modularization and manipulations. Forgetting is an important technique to eliminate unwanted (non-logical) symbols from an ontology, while preserving the meaning of the remaining symbols. Similar notions have been widely studied under various names, such as forgetting [Lin and Reiter, 1994] in AI, uniform interpolation [Visser, 1996] in mathematical logic, variable elimination [Lang et al., 2003b] in propositional logic, and second-order quantifier elimination [Gabbay and Ohlbach, 1992] in second-order logic. In particular, due to its potentials in ontology applications, forgetting has been intensively studied for various description logics [Konev et al., 2009, Wang et al., 2010, Lutz and Wolter, 2011, Lutz et al., 2012, Nikitina and Rudolph, 2014, Ludwig and Konev, 2014, Koopmann and Schmidt, 2014, Zhao and Schmidt, 2016]. However, the approaches in these studies are no longer applicable if the considered logical theory is inconsistent. To the best of our knowledge, there is still no discussions about forgetting for inconsistent ontologies. It is a challenging and interesting topic, because it is difficult to tell what kind of information in an inconsistent ontology should be kept after forgetting and how to rebuild these information in the forgetting result is much more complex than cases where ontologies are consistent.

In this chapter, we present the first study about forgetting in inconsistent ontologies. We start by discussing possible properties that might be desired in inconsistency-tolerant forgetting, then we proposed three different definitions of inconsistency-tolerant forgetting for general ontologies, based on inconsistency-tolerant query answering. Considering the difficulty of this problem, instead of general existential rules, we study the existence of forgetting results using these definitions and propose algorithms for their computation in a light-weight language, DL-lite$_{core}$. Finally, we show through experiments that our proposed forgetting can be effectively computed for large inconsistent KBs.

5.1 Inconsistency-tolerant Forgetting Definitions

In this section, we will first discuss desired properties that need to be considered in inconsistency-tolerant forgetting, and then explore possible solutions for its definitions. In our discussions, we also adopt the same setting as in inconsistency-tolerant query answering, given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, we assume the background knowledge Π of $\mathcal{O}$ is always consistent and errors come from $\mathcal{D}$.

We recall the essential properties in traditional forgetting for consistent ontologies. To forget a set of symbols Σ from an ontology $\mathcal{O}$, a result should be a new ontology $\mathcal{O}'$ satisfying the following properties:

(P1) $\mathcal{O}'$ is expressed using the symbols in $\mathcal{O}$ but not in Σ.

(P2) $\mathcal{O}'$ is equivalent to (or inseparable from) $\mathcal{O}$ over the remaining symbols.

Property **(P1)** guarantees the symbols from Σ are successfully eliminated while no additional symbols are added (otherwise a renaming would suffice), and is defined in a standard way. On the other hand, various notions of equivalence (or inseparability) have been proposed for **(P2)** in the literature [Konev et al., 2009, Wang et al., 2010]. Equivalence between ontologies are typically defined in terms of their models or logical consequences. However, if the original ontology is logically inconsistent, it has no models and trivially entails everything. Thus, the existing definitions of forgetting are not suitable for inconsistent ontologies.

To keep the information that lead to inconsistencies in an ontology and in the mean time, avoid the trivialization of inference, a straightforward approach is to introduce

non-classical semantics or non-standard inference. Those logics applying non-classical semantics are also called *paraconsistent logics*, most of which adopt multi-valued semantics, such as Belnap's four-valued logics [Belnap, 1977], Kleene's three-valued semantics [Nguyen and Szalas, 2010]. *Here-And-There*(HT) logic can be regarded as a three-valued logic, which can be used to examine strong-equivalence between ASP programs [Lifschitz et al., 2001]. Intuitively, the tolerance of inconsistencies in multi-valued semantics is obtained by expanding the space of model candidates, such that even contradict formulas can match a generalized model. But these semantics sacrifice a lot of inference power, some important properties that coincide with human common senses, such as *disjunctive syllogism, resolution and intuitive equivalence* are no longer valid in these semantics. Though there are some studies trying to balance between inconsistent-tolerance and inference power, such as *quasi-classical logic* (QC logic) [Hunter, 2000], which is also extended for DL [Zhang et al., 2014], they are still not feasible for practical reasoning.

For application, it is important to retain the full inference power of ontologies. To satisfy this requirement, we can borrow the idea from inconsistency-tolerant query answering, that is, we can define equivalence over the repairs of ontologies. Recall that a repair of an ontology is a maximal consistent component of the ontology. The set of all repairs of $(\Pi, \mathcal{D})$ is denoted by $\mathsf{repair}_\Pi(\mathcal{D})$. As repairs are consistent subsets of ontologies, a model-based definition for inconsistency-tolerant forgetting can be immediately obtained by combing the models of repairs.

Definition 5.1 (Model-based Forgetting)**.** Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, a signature $\Sigma \subseteq \mathsf{sig}(\mathcal{O})$, an ontology $\mathcal{O}' = (\Pi', \mathcal{D}')$ is a *model-based forgetting* (or *m-forgetting*) of $\mathcal{O}$ about Σ, if

(1) $\mathsf{sig}(\mathcal{O}') \subseteq \mathsf{sig}(\mathcal{O}) \setminus \Sigma$;

(2) $\bigcup_{\mathcal{B}'_i \in \mathsf{repair}_{\Pi'}(\mathcal{D}')} \mathsf{Mod}((\Pi', \mathcal{B}'_i)) \equiv_\Sigma \bigcup_{\mathcal{B}_i \in \mathsf{repair}_\Pi(\mathcal{D})} \mathsf{Mod}((\Pi, \mathcal{B}_i))$.

In the above definition, the first condition is standard in forgetting, which corresponds to **(P1)**. While the second condition builds a model equivalence relation based on repairs. There can be different ways to composite the sets of models of repairs, taking the union of them is straightforward and rational: the ontology corresponding to the resulting set of models is strictly weaker than $\mathcal{O}$. It shall be noted that the results of

the above forgetting can be either consistent or inconsistent, which leads us to consider the following restriction.

(P3) $\mathcal{O}'$ is consistent after forgetting.

Restricting the forgetting results to be consistent can be controversial, as one may argue that why should forgetting a set of symbols Σ resolve inconsistencies that only concern symbols outside of Σ. If the symbols in Σ are totally unrelated to the causes of inconsistencies, then the forgetting results should be naturally inconsistent. However, considering that most existing OBDA systems are not capable to tolerate inconsistencies, when a consistent forgetting result exists, it is always better to produce a consistent forgetting result for efficient ontological reasoning. In such cases, forgetting can be regarded as a special repairing process for ontologies.

If we consider **(P3)** in the definition, we can obtain a stronger version of m-forgetting.

Definition 5.2 (Consistent Models-based Forgetting)**.** Given an ontology $\mathcal{O} = (\Pi, \mathcal{D})$, a signature $\Sigma \subseteq \mathsf{sig}(\mathcal{O})$, an ontology $\mathcal{O}'$ is a *consistent model-based forgetting* (or *cm-forgetting*) of $\mathcal{O}$ about Σ, if

(1) $\mathsf{sig}(\mathcal{O}') \subseteq \mathsf{sig}(\mathcal{O}) \setminus \Sigma$;

(2) $\mathsf{Mod}(\mathcal{O}') \equiv_\Sigma \bigcup_{\mathcal{B}_i \in \mathsf{repair}_\Pi(\mathcal{D})} \mathsf{Mod}((\Pi, \mathcal{B}_i))$.

As every repair $\mathcal{B}_i$ is Π-consistent, $\mathsf{Mod}(\Pi, \mathcal{B}_i)$ is not empty, then $\mathsf{Mod}(\mathcal{O}')$ must not be empty, thus $\mathcal{O}'$ is consistent.

Example 5.1. *Consider the ontology* $\mathcal{O} = (\Pi, \mathcal{D})$*, where* $\mathcal{D} = \{\mathsf{A}(a), \mathsf{R}(b, a)\}$ *and* Π *consists of the following rules,*

$$\mathsf{B}(x) \leftarrow \mathsf{A}(x). \quad \mathsf{D}(x) \leftarrow \mathsf{A}(x). \quad \mathsf{E}(x) \leftarrow \mathsf{A}(x).$$
$$\mathsf{D}(x) \leftarrow \mathsf{R}(y, x). \quad \bot \leftarrow \mathsf{R}(y, x) \wedge \mathsf{B}(x).$$

$\mathcal{O}$ is inconsistent because the constraint cannot be satisfied, and there are two repairs of $\mathcal{O}$, $\mathcal{B}_1 = \{A(a)\}$ and $\mathcal{B}_2 = \{R(b, a)\}$. Let $\Sigma_1 = \{\mathsf{B}\}$, according to definition 5.1, $\mathcal{O}' = (\Pi', \mathcal{D})$, where $\Pi' = \{\mathsf{D}(x) \leftarrow \mathsf{A}(x).\ \mathsf{E}(x) \leftarrow \mathsf{A}(x).\ \mathsf{D}(x) \leftarrow \mathsf{R}(y, x).\ \bot \leftarrow \mathsf{R}(y, x) \wedge$

$\mathsf{A}(x).\}$*, is an m-forgetting of* $\mathcal{O}$ *about* Σ_1*.* $\mathcal{O}'$ *is still inconsistent and it seems a rational result. If we consider* $\Sigma_2 = \{\mathsf{A}\}$*, an m-forgetting of* $\mathcal{O}$ *about* Σ_2 *is,* $\mathcal{O}'' = (\Pi'', \mathcal{D}')$*, where* $\Pi'' = \{\mathsf{D}(x) \leftarrow \mathsf{R}(y,x).\ \bot \leftarrow \mathsf{R}(y,x) \wedge \mathsf{B}(x).\ \bot \leftarrow \mathsf{R}(b,a) \wedge \mathsf{E}(a).\}$ *and* $\mathcal{D}' = \{\mathsf{D}(a), \mathsf{R}(b,a), \mathsf{B}(a), \mathsf{E}(a)\}$.

In the above example, if we want to produce a cm-forgetting of the ontology, we have to introduce disjunctions to represent the database, which is illegal in most ontologies. Besides, the rule $\bot \leftarrow \mathsf{R}(b,a), \mathsf{E}(a)$ in Π'' is bit awkward, it is only constructed to create conflict between $\mathsf{R}(b,a)$ and $\mathsf{E}(a)$, and it cannot be expressed in most DL ontologies.

Though the above definitions nicely define the equivalence relation for **(P2)**, model-based forgetting are unnecessarily too strong, their results are often inexpressible in less expressive ontology languages. For the scenarios of OBDA, where query answering is the main reasoning task, defining forgetting based on query answering seems more suitable. With the profound studies of inconsistency-tolerant query answering, inconsistency-tolerant forgetting can be straightforwardly defined taking advantage of inconsistency-tolerant semantics.

To demonstrate forgetting under inconsistency-tolerant semantics, we consider the following running example.

Example 5.2. *Consider a DL ontology* $\mathcal{O}_e$ *with the following TBox* $\mathcal{T}_e$*:*

$$\mathsf{Cat} \sqsubseteq \mathsf{Mammal}, \mathsf{Parrot} \sqsubseteq \mathsf{Bird}, \mathsf{Mammal} \sqsubseteq \neg\mathsf{Bird},$$
$$\mathsf{Cat} \sqsubseteq \mathsf{Pet}, \mathsf{Parrot} \sqsubseteq \mathsf{Pet}, \mathsf{Cat} \sqsubseteq \mathsf{Fluffy}$$

and the following ABox $\mathcal{A}_e$*:* $\mathsf{Cat}(a), \mathsf{Parrot}(a), \mathsf{Parrot}(b)$.

The ontology $\mathcal{O}_e$ *is inconsistent, as individual a cannot be a mammal and a bird at the same time. If using different inconsistency-tolerant semantics, the ontology can have different reasoning results, for example, (1) using the* CAR *or* ICAR *semantics to explore potential candidates for a pet, both* $\mathsf{Pet}(a)$ *and* $\mathsf{Fluffy}(a)$ *can be concluded; (2) using the more cautious* AR *semantics, only* $\mathsf{Pet}(a)$ *can be concluded; and (3) using the most sceptical* IAR *semantics, nothing about a can be concluded.*

While it is promising to define forgetting based on inconsistency-tolerant semantics, given the fact that there exist different semantics producing different querying answers,

it would be desirable for the definition of forgetting to be independent from each specific semantics.

(P4) The definition of forgetting is independent from a specific inconsistency-tolerant semantics.

The above property is appealing, however, because there is not a particular criteria for defining inconsistency-tolerant semantics, it is impossible to obtain a forgetting result that behaves the same as the original ontology under arbitrary semantics, therefore, it is necessary to specify the scope of inconsistency-tolerant semantics. In the following definition, we introduce a set of semantics as one of the inputs of forgetting.

Definition 5.3. For an ontology $\mathcal{O}$, a signature Σ and a set of inconsistency-tolerant semantics Γ, an ontology $\mathcal{O}'$ is a result of *semantics-independent forgetting* (or *si-forgetting*) of $\mathcal{O}$ about Σ and Γ if

(1) $\mathsf{sig}(\mathcal{O}') \subseteq (\mathsf{sig}(\mathcal{O}) \setminus \Sigma)$;

(2) $\mathcal{O}' \models_\gamma q$ iff $\mathcal{O} \models_\gamma q$ for each BCQ q over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$ and each $\gamma \in \Gamma$.

The definition performs forgetting independently from the specific semantics, in this way, the features of each semantics is preserved after forgetting. As for a concrete set of the input semantics, in general cases, we advocate to consider only the four basic semantics, $\{\mathsf{AR}, \mathsf{IAR}, \mathsf{CAR}, \mathsf{ICAR}\}$. We choose only these four semantics out of several reasons: (1) they have close relations between each other, which is essential for the satisfaction of Condition 2; (2) they don't need any other extra information or arguments to guide the entailment; (3) many other semantics stem from them, that means if Condition 2 is satisfied under these four semantics, it is of high chance to be satisfied under other semantics. We define $\Gamma_{\mathsf{basic}} = \{\mathsf{AR}, \mathsf{IAR}, \mathsf{CAR}, \mathsf{ICAR}\}$.

Example 5.3. *Consider $\mathcal{O}_e$ from the running exmaple, Example 5.2, let $\mathcal{T}' = \{\mathsf{Cat} \sqsubseteq \mathsf{Pet}, \mathsf{Parrot} \sqsubseteq \mathsf{Pet}, \mathsf{Cat} \sqsubseteq \mathsf{Fluffy}, \mathsf{Cat} \sqsubseteq \neg\mathsf{Parrot}\}$. Then $(\mathcal{T}', \mathcal{A}_e)$ is a result of si-forgetting about $\{\mathsf{Mammal}, \mathsf{Bird}\}$ and Γ_{basic} in $\mathcal{O}_e$.*

There are cases where a result of si-forgetting does not exist.

Example 5.4. *Consider again the running example, assume we try to construct a si-forgetting about* $\{\mathsf{Cat}\}$ *from* $\mathcal{O}_e$*. The TBox after forgetting consists of inclusion axioms* $\mathsf{Parrot} \sqsubseteq \mathsf{Bird}, \mathsf{Parrot} \sqsubseteq \mathsf{Pet}, \mathsf{Mammal} \sqsubseteq \neg\mathsf{Bird}$*. To construct the ABox after forgetting, because* $\mathcal{O} \models_{\mathsf{AR}} \mathsf{Pet}(a)$ *and* $\mathcal{O} \not\models_{\mathsf{IAR}} \mathsf{Pet}(a)$*, a result of forgetting should also have the same entailment. One can easily verify (by checking all possible results) that this cannot be achieved.*

We can also obtain a stronger version of si-forgetting by considering **(P3)**.

Definition 5.4. For an ontology $\mathcal{O}$, a signature Σ and a set of inconsistency-tolerant semantics Γ, an ontology $\mathcal{O}'$ is a result of *strong forgetting* (or *st-forgetting*) of $\mathcal{O}$ about Σ and Γ if

1. $\mathsf{sig}(\mathcal{O}') \subseteq (\mathsf{sig}(\mathcal{O}) \setminus \Sigma)$;
2. $\mathcal{O}' \models q$ iff $\mathcal{O} \models_\gamma q$ for each BCQ q over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$ and each $\gamma \in \Gamma$.

The definition is adapted from Definition 5.3 by replacing $\models_\gamma$ with $\models$ in Condition 2. Note that since $\mathcal{O} \not\models_\gamma \bot$, $\mathcal{O} \not\models \bot$, and $\mathcal{O}'$ is consistent, thus the definition satisfies **(P3)**.

Example 5.5. *Consider* $\mathcal{O}_e$ *from Example 5.2, let* $\mathcal{T}' = \{\mathsf{Cat} \sqsubseteq \mathsf{Mammal}, \mathsf{Parrot} \sqsubseteq \mathsf{Bird}, \mathsf{Mammal} \sqsubseteq \neg\mathsf{Bird}\}$*, then* $(\mathcal{T}', \{\mathsf{Parrot}(b)\})$ *is a st-forgetting of* $\mathcal{O}_e$ *about* $\{\mathsf{Pet}, \mathsf{Fluffy}\}$ *and* Γ_{basic}*.*

A result of st-forgetting is also a result of si-forgetting, which implies that a result of si-forgetting exists whenever it exists for st-forgetting. Nevertheless, as the following example shows, a result of si-forgetting may exist when st-forgetting does not.

Example 5.6. *In example 5.3,* $(\mathcal{T}', \mathcal{A}_e)$ *is result of si-forgetting of* $\mathcal{O}_e$ *about* $\{\mathsf{Mammal}, \mathsf{Bird}\}$ *and* Γ_{basic}*. However, there does not exist a result of st-forgetting about* $\{\mathsf{Mammal}, \mathsf{Bird}\}$ *and* Γ_{basic} *of* $\mathcal{O}_e$*.*

The forgetting results of st-forgetting seem ideally perfect, however, it is quite unrealistic to compute them in most inconsistent ontologies. As one can see, Condition 2 actually requires that the ontology produces the same answers under all given semantics, which can be barely achieved. In fact, if we consider Γ_{basic} as input, condition 2 can be satisfied only when it holds that $\mathcal{O}$ gives the same answers under IAR and CAR for any queries.

Lemma 5.1. *Given an ontology $\mathcal{O}$ and a signature Σ, a st-forgetting about of $\mathcal{O}$ about Σ exists only when the following condition holds: for any BCQ q over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$, $\mathcal{O} \models_{\mathsf{IAR}} q$ if $\mathcal{O} \models_{\mathsf{CAR}} q$.*

Proof. The proof is straightforward. If there exists q over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$ such that $\mathcal{O} \not\models_{\mathsf{IAR}} q$ and $\mathcal{O} \models_{\mathsf{CAR}} q$, to obtain an ontology $\mathcal{O}'$ satisfying $\mathcal{O}' \not\models_{\mathsf{IAR}} q$ and $\mathcal{O}' \models_{\mathsf{CAR}} q$, $\mathcal{O}'$ has to be inconsistent, then there is no way to obtain a st-forgetting about Σ and Γ_{basic} in $\mathcal{O}$. □

Though **(P4)** can guarantee that the forgetting results are highly close to the origin ontology over inconsistency-tolerant query answering, it is arguable that having a single notion of forgetting that fits various inconsistency-tolerant semantics could be unnecessary. Because in some application scenarios, users may only focus on only one particular semantics, then there is no need to acquire query answering equivalence for other semantics. Hence we look into notions of forgetting that are specific with different semantics. Dropping **(P4)**, we have the following definition of forgetting that takes a single semantics γ as a parameter.

Definition 5.5. For an ontology $\mathcal{O}$, a signature Σ and a inconsistency-tolerant semantics γ, an ontology $\mathcal{O}'$ is a result of *γ-forgetting* about Σ in $\mathcal{O}$ if

(1) $\mathsf{sig}(\mathcal{O}') \subseteq (\mathsf{sig}(\mathcal{O}) \setminus \Sigma)$;

(2) $\mathcal{O}' \models q$ iff $\mathcal{O} \models_\gamma q$ for each BCQ q over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$.

Example 5.7. *For a result of γ-forgetting about $\{\mathsf{Cat}, \mathsf{Parrot}\}$ in $\mathcal{O}_e$ with $\gamma \in \Gamma_{\mathsf{basic}}$, the TBox is $\{\mathsf{Mammal} \sqsubseteq \neg\mathsf{Bird}\}$. The ABox is as follows:*

- IAR-*forgetting:* $\{\mathsf{Pet}(b), \mathsf{Bird}(b)\}$
- ICAR-*forgetting:* $\{\mathsf{Pet}(b), \mathsf{Bird}(b), \mathsf{Pet}(a), \mathsf{Fluffy}(a)\}$
- AR-*forgetting:* $\{\mathsf{Pet}(b), \mathsf{Bird}(b), \mathsf{Pet}(a)\}$
- CAR-*forgetting:* $\{\mathsf{Pet}(b), \mathsf{Bird}(b), \mathsf{Pet}(a), \mathsf{Fluffy}(a)\}$

5.2 Inconsistency-tolerant forgetting for DL-lite

In this section, the computation of st-forgetting, si-forgetting and γ-forgetting will be studied. Because this work is the first attempt to consider forgetting for inconsistent ontologies, we choose a simple language for which both forgetting and inconsistency-tolerant query answering have been sufficiently discussed as a start point. We will have our focus on the light-weight DL language, DL-lite$_{core}$.

One common issue in query-based forgetting definitions is that they render no constraint on the shape of the resulting TBox. For instance, considering an ontology $(\mathcal{T}_e, \emptyset)$ where $\mathcal{T}_e$ is as in Example 5.2, to forget Mammal and Bird, both ontologys $(\emptyset, \emptyset)$ and $(\{\mathsf{Cat} \sqsubseteq \mathsf{Parrot}\}, \emptyset)$ would entail the same BCQs as the initial ontology. Yet the first result eliminates everything from the initial TBox and the second one even introduces an erroneous inclusion $\mathsf{Cat} \sqsubseteq \mathsf{Parrot}$.

Therefore, we slightly strengthen our definitions for DL-lite ontologies. First, we define a language extended from DL-lite$_{core}$ that is sufficient to capture the logical relationship specified by both TBoxes and ABoxes. The extended language, denoted DL-Lite$_n$, allows ABox assertions of the forms $A(t)$ or $P(t, t')$ with t, t' being labeled nulls from N_I or variables from N_V. We assume that when DL-lite$_n$ is used as a query language, variables in an assertion are considered existentially quantified; and when DL-lite$_n$ assertions are used to represented an ABox, they contains only constants and labeled nulls.

We state that the new language is sufficient to capture equivalence over DL-Lite$_{core}$ CQ answering.

Remark 1. For two DL-Lite$_{core}$ ontologies $\mathcal{O}_1, \mathcal{O}_2$, a signature Σ, and an inconsistency-tolerant semantics γ, suppose $\mathcal{O}_1 \models_\gamma \mathcal{Q}$ iff $\mathcal{O}_2 \models_\gamma \mathcal{Q}$ for each set of DL-Lite$_n$ axioms $\mathcal{Q}$ over Σ, then $\mathcal{O}_1 \models_\gamma q$ iff $\mathcal{O}_2 \models_\gamma q$ for each BCQ q over Σ.

Then the definitions of st-forgeting, is-forgetting and γ-forgetting for DL-lite$_{core}$ ontologies are strengthened by revising Condition 2: each occurrence of "BCQ q" is replaced by "set of DL-lite$_n$ axioms $\mathcal{Q}$"; and requiring the forgetting result $\mathcal{O}'$ to be expressed in DL-lite$_n$.

With the strengthened definition, it can be shown that a result of st-forgetting, once exists, is unique up to logical equivalence.

Proposition 5.1. *If $\mathcal{O}_1$ and $\mathcal{O}_2$ are both results of st-forgetting about a signature Σ and Γ of a DL-lite$_{core}$ ontology $\mathcal{O}$, then $\mathcal{O}_1 \equiv \mathcal{O}_2$.*

Proof. According to the definition, for each $\gamma \in \Gamma$, any set of DL-lite$_n$ axioms $\mathcal{Q}$ over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$, we have $\mathcal{O}_1 \models \mathcal{Q} \Leftrightarrow \mathcal{O} \models_\gamma \mathcal{Q} \Leftrightarrow \mathcal{O}_2 \models \mathcal{Q}$. Since $\mathsf{sig}(\mathcal{O}_2) \subseteq (\mathsf{sig}(\mathcal{O}) \setminus \Sigma)$, then every axiom in $\mathcal{O}_2$ can be entailed by $\mathcal{O}_1$, that means, $\mathcal{O}_1 \models \mathcal{O}_2$. Similarly, we also have $\mathcal{O}_2 \models \mathcal{O}_1$. □

Because γ-forgetting is actually a special case of st-forgetting (if we restrict Γ to contain only one semantics), γ-forgetting also satisfies uniqueness. While the following example shows that a result of si-forgetting is not necessarily unique.

Example 5.8 (Example 5.2 cont'd)**.** *$(\mathcal{T}', \{\mathsf{Parrot}(b)\})$ and $(\mathcal{T}', \mathcal{A}_e)$ are both si-forgetting about $\{\mathsf{Pet}, \mathsf{Fluffy}\}$ and Γ_{basic} in $\mathcal{O}_e$.*

Though there may be several results of si-forgetting, they share the same conflict-free part. A *conflict* in an ontology is a minimal component of data that leads to inconsistencies of the ontology.

Definition 5.6 (Conflict)**.** Given an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, a conflict in $\mathcal{O}$ is a minimal subset of $\mathcal{A}$ such that $\mathcal{T} \cup \mathcal{A}'$ is inconsistent.

We use $\mathsf{confs}(\mathcal{O})$ to denote the union of all conflicts in $\mathcal{O}$. The following lemma shows that the results of si-forgetting only differ in how conflicts are introduced.

Lemma 5.2. *If $\mathcal{O}_1$ and $\mathcal{O}_2$ are both results of si-forgetting about a signature Σ from an ontology $\mathcal{O}$, then $\mathcal{O}_1 \setminus \mathsf{confs}(\mathcal{O}_1) \equiv \mathcal{O}_2 \setminus \mathsf{confs}(\mathcal{O}_2)$.*

Proof. For any set DL-lite$_n$ axioms $\mathcal{Q}$, $\mathcal{O}_1 \setminus \mathsf{confs}(\mathcal{O}_1) \models \mathcal{Q} \Leftrightarrow \mathcal{O}_1 \setminus \mathsf{confs}(\mathcal{O}_1) \models_{\mathsf{IAR}} \mathcal{Q} \Leftrightarrow \mathcal{O} \models_{\mathsf{IAR}} \mathcal{Q} \Leftrightarrow \mathcal{O}_2 \setminus \mathsf{confs}(\mathcal{O}_2) \models_{\mathsf{IAR}} \mathcal{Q} \Leftrightarrow \mathcal{O}_2 \setminus \mathsf{confs}(\mathcal{O}_2) \models \mathcal{Q}$, thus $\mathcal{O}_1 \setminus \mathsf{confs}(\mathcal{O}_1) \equiv \mathcal{O}_2 \setminus \mathsf{confs}(\mathcal{O}_2)$. □

For the computation of st-forgetting, we show that it can be reduced to forgetting in consistent ontologies if only the semantics in Γ_{basic} are considered. As we have mentioned, even for query-based forgetting in consistent ontologies, the result is not guaranteed to exist in first-order logic if it is allowed to forget arbitrary symbols from the ontologies.

While it has been shown that a result of forgetting only concept names from DL-lite$_{core}$ ontologies always exists in DL-Lite$_n$, moreover, the result is unique (up to equivalence). Hence, in following discussion, we restrict ourselves to consider only forgetting of concepts. Note that, according to the previous discussions, even only concepts are forgotten, results of st-forgetting and si-forgetting are still not guaranteed to exist. Given a DL-lite$_{core}$ ontology and a set of concept names, we use $\mathsf{forget}(\mathcal{O}, \Sigma)$ to denote the result returned by Algorithm 2.

Proposition 5.2. *For an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ in DL-lite$_{core}$ and a set of concept names Σ, if $\mathcal{O}'$ is a result of st-forgetting about Σ and Γ_{basic} in $\mathcal{O}$, then $\mathcal{O}' \equiv \mathsf{forget}((\mathcal{T}, \mathcal{A} \setminus \mathsf{confs}(\mathcal{O})), \Sigma)$.*

Proof. Let $\mathcal{O}_\mathsf{f} = \mathsf{forget}((\mathcal{T}, \mathcal{A} \setminus \mathsf{confs}(\mathcal{O})), \Sigma)$. According to lemma 5.1, if $\mathcal{O}'$ is a result of st-forgetting about Σ in $\mathcal{O}$, then we have for any $\mathcal{Q}$ over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$, if $\mathcal{O} \models_{\mathsf{CAR}} q$, then $\mathcal{O} \models_{\mathsf{IAR}} q$, so to compute st-forgetting, we only need to check IAR equivalence. As $\mathcal{O} \models_{\mathsf{IAR}} \mathcal{Q}$ is equivalent to $(\mathcal{T}, \mathcal{A} \setminus \mathsf{confs}(\mathcal{O})) \models \mathcal{Q}$, and by the definition of forget, we also have $(\mathcal{T}, \mathcal{A} \setminus \mathsf{confs}(\mathcal{O})) \models \mathcal{Q}$ iff $\mathcal{O}_\mathsf{f} \models \mathcal{Q}$, so $\mathcal{O}_\mathsf{f}$ is a st-forgetting about Σ in $\mathcal{O}$. Then together with proposition 5.1, we have $\mathcal{O}' \equiv \mathcal{O}_\mathsf{f}$. □

The above proposition tells that if there exists a st-forgetting of the ontology, then we can compute it in two steps: (1) first remove all conflicts from the ontology; (2) then perform consistent forgetting to the resulting consistent ontology. The process is quite similar to the computation of query answering under IAR semantics. In fact, when the result of st-forgetting exists, we can immediately regard it as a result of γ-forgetting. It is not difficult to see that $\mathsf{forget}((\mathcal{T}, \mathcal{A} \setminus \mathsf{confs}(\mathcal{O})), \Sigma)$ is actually an IAR-forgetting of $\mathcal{O}$ about Σ.

While for si-forgetting, its computation is much more complicated. Given an ontology $\mathcal{O}$ in DL-lite$_{core}$, to compute its si-forgetting results about concepts, a straightforward approach is to guess a result $\mathcal{O}'$, then check whether $\mathcal{O}'$ satisfies Conditions 1 and 2 in Definition 5.3, which involves checking equivalence under each inconsistency-tolerant semantics $\gamma \in \Gamma_{\mathsf{basic}}$. Since checking AR and CAR entailment is already CoNP-complete w.r.t. data complexity [Lembo et al., 2010], the above process is highly intractable.

In what follows, we propose a sound (but not necessarily complete) PTime algorithm to compute si-forgetting. The algorithm attempts to find "substitutes" to replace the

conflicts that will be omitted during forgetting in a way that preserves equivalence under all concerned inconsistency-tolerant semantics. Given an ABox assertion α in DL-Lite$_n$ and a set of concept names Σ, a substitute of α should inherit all entailment from α (outside Σ). Recall that $\mathsf{HB}(\mathcal{O})$ the herbrand base of $\mathcal{O}$ is the set of all assertions that can be formed over $\mathsf{sig}(\mathcal{O})$ and $\mathsf{const}(\mathcal{O})$, which we slightly extend here to denote all such assertions in DL-Lite$_n$. Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, we define $\mathsf{clc}_{\mathcal{T}}^{\Sigma}(\alpha) = \{\ \beta \in \mathsf{HB}(\mathcal{O}) \mid \mathsf{sig}(\beta) \cap \Sigma = \emptyset, (\mathcal{T}, \{\alpha\}) \models \beta\ \}$ as the set of logical consequences of α outside of Σ under $\mathcal{T}$, and $\mathsf{conf}_{\mathcal{T}}(\alpha) = \{\beta \in \mathsf{assert}(\mathcal{O}) \mid \mathcal{T} \cup \{\alpha, \beta\} \models \bot\}$ be the set of all assertions that are in conflict with α under $\mathcal{T}$. A membership assertion $\beta \in \mathsf{HB}(\mathcal{O})$ is a *valid substitute* of α w.r.t. Σ, if $\mathsf{clc}_{\mathcal{T}}^{\Sigma}(\beta) \equiv \mathsf{clc}_{\mathcal{T}}^{\Sigma}(\alpha)$ and $\mathsf{conf}_{\mathcal{T}}(\beta) \equiv \mathsf{conf}_{\mathcal{T}}(\alpha)$.

Example 5.9. *Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T} = \{\mathsf{A} \sqsubseteq \mathsf{D}, \mathsf{B} \sqsubseteq \exists \mathsf{R}, \exists \mathsf{R} \sqsubseteq \mathsf{D}, \mathsf{A} \sqsubseteq \neg\exists \mathsf{R}\}$, $\mathcal{A} = \{\mathsf{A}(a), \mathsf{B}(a)\}$. Apparently, $\mathcal{O}$ is inconsistent as $\mathcal{A}$ itself is a conflict of $\mathcal{O}$. If we consider forgetting $\{\mathsf{B}\}$ from the ontology, because $\mathsf{clc}_{\mathcal{T}}^{\Sigma}(\mathsf{B}(a)) = \{\mathsf{R}(a, u)\} = \mathsf{clc}_{\mathcal{T}}^{\Sigma}(\mathsf{R}(a, u))$ and $\mathsf{conf}_{\mathcal{T}}(\mathsf{B}(a)) = \{\mathsf{A}(a)\} = \mathsf{conf}_{\mathcal{T}}^{\Sigma}(\mathsf{R}(a, u))$, where u is a labeled null, we have $\mathsf{R}(a, u)$ is a valid substitute of $\mathsf{B}(a)$ w.r.t $\{\mathsf{B}\}$,*

Lemma 5.3. *For an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ in DL-lite$_{core}$ and a signature Σ, if $\mathcal{O}'$ is obtained from $\mathcal{O}$ by replacing a subset of $\mathcal{A}$ with their valid substitutes w.r.t. Σ, then for each set of DL-Lite$_n$ axioms $\mathcal{Q}$ that $\mathsf{sig}(\mathcal{Q}) \cap \Sigma = \emptyset$, $\mathcal{O} \models_\gamma \mathcal{Q}$ iff $\mathcal{O}' \models_\gamma \mathcal{Q}$ for each $\gamma \in \Gamma_{\mathsf{basic}}$.*

Proof. Let vs be the function that maps $\mathcal{O}$ to $\mathcal{O}'$. We first prove that $\mathcal{B}$ is a repair of $\mathcal{O}$ if and only if $\mathsf{vs}(\mathcal{B})$ is a repair of $\mathcal{O}'$. Let $\mathcal{B}' = \mathsf{vs}(\mathcal{B})$. If $\mathcal{B}$ is a repair of $\mathcal{O}$, then for any other ABox assertion $\beta \in \mathcal{O}'$ that $\beta \notin \mathcal{B}'$, β must be in conflict with a certain $\beta' \in \mathcal{B}'$, otherwise, let $\mathsf{vs}(\alpha) = \beta$, then $\alpha \notin \mathcal{B}$, because $\mathsf{conf}_{\mathcal{T}}(\beta) = \mathsf{conf}_{\mathcal{T}}(\alpha)$, α is not in conflict with any $\alpha' \in \mathcal{B}$, which contradicts the fact that $\mathcal{B}$ is a maximal consistent component of $\mathcal{O}$. Thus $\mathcal{B}'$ is a maximal consistent component of $\mathcal{O}$, i.e., a repair of $\mathcal{O}$. The other direction can be similarly proved.

We claim that for each BCQ q outside of Σ, each $\mathcal{A}' \subseteq \mathcal{A}$, $(\mathcal{T}, \mathcal{A}') \models q$ iff $(\mathcal{T}, \mathsf{vs}(\mathcal{A}')) \models q$. Let q be a connected BCQ outside of Σ, if $(\mathcal{T}, \mathcal{A}') \models q$, then there exists a homomorphism h such that $h(q) \subseteq \mathcal{I}_{(\mathcal{T}, \mathcal{A}')}$, if the domain of $h(q)$ contains individuals, because $\mathsf{clc}_{\mathcal{T}}^{\Sigma}(\mathcal{A}') \equiv \mathsf{clc}_{\mathcal{T}}^{\Sigma}(\mathsf{vs}(\mathcal{A}'))$, then $h(q) \in \mathcal{I}_{(\mathcal{T}, \mathsf{vs}(\mathcal{A}'))}$, so $(\mathcal{T}, \mathsf{vs}(\mathcal{A}')) \models q$; if the domain of $h(q)$ does not contain individuals, there exists a cause atom $c \in q$ such that $\mathcal{T} \cup \mathcal{A}' \models c$ and $\mathcal{T} \cup c \models q$,

because we also have $\mathcal{T} \cup \mathsf{vs}(\mathcal{A}') \models c$, so $\mathcal{T} \cup \mathsf{vs}(\mathcal{A}') \models q$. The other direction can be similarly proved.

Finally, we show for each BCQ q that $\mathsf{sig}(\mathcal{Q}) \cap \Sigma = \emptyset$, $\mathcal{O} \models_\gamma q$ iff $\mathcal{O}' \models_\gamma q$ for each $\gamma \in \{\mathsf{AR}, \mathsf{IAR}, \mathsf{CAR}, \mathsf{ICAR}\}$. Let q be any BCQ outside of Σ,

1. AR: For any repair $\mathcal{B}$ of $\mathcal{O}$, we have $\mathsf{vs}(\mathcal{B})$ as a repair $\mathcal{O}'$, and because $(\mathcal{T}, \mathcal{B}) \models q$ iff $(\mathcal{T}, \mathsf{vs}(\mathcal{B})) \models q$, it follows that $\mathcal{O} \models_{\mathsf{AR}} q$ iff $\mathcal{O} \models_{\mathsf{AR}} q$.

2. IAR: Let $\mathcal{B}_0 = \bigcap_{\mathcal{B} \in \mathsf{rep}(\mathcal{O})}$, because $\mathcal{B} \in \mathsf{rep}(\mathcal{O})$ iff $\mathsf{vs}(\mathcal{B}) \in \mathsf{rep}(\mathcal{O})$, we have $\mathsf{vs}(\mathcal{B}_0) = \bigcap_{\mathcal{B} \in \mathsf{rep}(\mathcal{O}')}$. Then $(\mathcal{T}, \mathcal{B}_0) \models q$ iff $(\mathcal{T}, \mathsf{vs}(\mathcal{B}_0)) \models q$, it follows that $\mathcal{O} \models_{\mathsf{IAR}} q$ iff $\mathcal{O} \models_{\mathsf{IAR}} q$.

3. CAR: There is a little difference in proving the CAR case, let $\mathcal{O}_C$ and $\mathcal{O}'_C$ denote the ontologies extended from $\mathcal{O}$, $\mathcal{O}'$ by adding consistent logical consequences. we show that $\mathcal{B}$ is a repair of $\mathcal{O}_C$ if and only if there is a repair $\mathcal{B}'$ of $\mathcal{O}'_C$ such that $\mathcal{B} \sim_\Sigma \mathcal{B}'$, where $\sim_\Sigma$ denotes that two sets agree on all atoms outside of Σ. Let $\mathcal{B}$ be a repair of $\mathcal{O}_C$, we construct $\mathcal{B}'$ by deleting all $\alpha \in \mathcal{B}$ that $\alpha \notin \mathsf{clc}_\mathcal{T}(\mathsf{vs}(\mathcal{A}))$ (notice that such α must be inside of Σ), adding all $\beta \in \mathsf{clc}_\mathcal{T}(\mathsf{vs}(\mathcal{A}))$ that β is not in conflict with $\mathcal{B}$ under $\mathcal{T}$ (Possible β is also inside of Σ). So we have $\mathcal{B} \sim_\Sigma \mathcal{B}'$, R' is a repiar of $\mathcal{O}'_C$. Similar to the preceding discussion, with $R \sim_\Sigma R'$, we can conclude for any q outside of Σ, $(\mathcal{T}, R) \models q$ iff $(\mathcal{T}, R') \models q$, thus $\mathcal{O} \models_{\mathsf{CAR}} q$ iff $\mathcal{O} \models_{\mathsf{CAR}} q$.

4. ICAR: With the discussion in CAR case, we can easily have $\mathcal{B}_0 \sim_\Sigma \mathcal{B}'_0$, where $\mathcal{B}_0$ and $\mathcal{B}'_0$ are the intersections of all CAR-repair of $\mathcal{O}$ and $\mathcal{O}'$, so $\mathcal{O} \models_{\mathsf{ICAR}} q$ iff $\mathcal{O}' \models_{\mathsf{ICAR}} q$.

□

Algorithm 4 first performs consistent ontology forgetting over the conflict-free portion of the ontology (Line 1), which generates the complete resulting TBox (of forgetting) and a conflict-free subset of the resulting ABox. It then examines each initial ABox assertion that involved in a conflict (Line 2): If the assertion does not contain concepts to be forgotten, it is added to the result (Lines 3 – 4); or if a valid substitution is found on the remaining signature, add it to the result (Line 6 – 7); otherwise if a valid

Algorithm 4: Compute si-forgetting

Input : an ontology $\mathcal{O}$ in DL-lite$_{core}$ and a set of concept names Σ
Output: a result of is-forgetting $\mathcal{O}'$ or Failure

```
1  begin
2      O' := forget(O \ confs(O), Σ);
3      for α ∈ confs(O) do
4          if sig(α) ∩ Σ = ∅ then
5              O' := O' ∪ {α}
6          else
7              if a valid substitute β of α exists w.r.t Σ s.t. sig(β) ∩ Σ = ∅ then
8                  O' := O' ∪ {β};
9              else
10                 return Failure;
11     return O';
```

substitution cannot be found, return Failure (Line 9). The algorithm runs in polynomial time w.r.t. data complexity, note that $\mathsf{confs}(\mathcal{O})$ can be computed using query rewriting [Bienvenu et al., 2014b] and whether a valid substitute of α exists can be checked in polynomial time. The algorithm is not necessarily complete, that is, returning Failure does not necessarily means no si-forgetting result exists. The soundness of the algorithm is stated as follows.

Theorem 5.1. *For an ontology $\mathcal{O}$ in DL-lite$_{core}$ and a set of concept names Σ, an ontology $\mathcal{O}'$ returned by Algorithm 4 is a result of si-forgetting about Σ and Γ_{basic} of $\mathcal{O}$.*

Proof. If an ontology $\mathcal{O}'$ is returned, then for each $\alpha \in \mathsf{confs}(\mathcal{O})$, there exists a valid substitute of α about Σ, let $\mathcal{A}'_\bot$ denote the set of assertions substituting $\mathsf{confs}(\mathcal{O})$, and let $\mathcal{A}_\top = \mathcal{A} \setminus \mathsf{confs}(\mathcal{O})$, $(\mathcal{T}', \mathcal{A}'_\top) = \mathsf{forget}(\mathcal{O} \setminus \mathsf{confs}(\mathcal{O}), \Sigma)$. We next prove that for any BCQ q over $\mathsf{sig}(\mathcal{O}) \setminus \Sigma$, $(\mathcal{T}, \mathcal{A}'_\bot \cup \mathcal{A}'_\top) \models_\gamma q \Leftrightarrow \mathcal{O} \models_\gamma q$, which is achieved by modifying the proof for lemma 5.3. Notice that elements in $\mathcal{A}'_\top$ are not necessarily valid substitutes of elements in $\mathcal{A}_\top$, but with the computation of forget, we have $\mathsf{clc}^\Sigma_{\mathcal{T}}(\mathcal{A}'_\top) \equiv \mathsf{clc}^\Sigma_{\mathcal{T}}(\mathcal{A}_\top)$, and moreover, because $\mathcal{A}_\top$ is not in conflict with any other elements in $\mathcal{A}$, then so is $\mathcal{A}'_\top$, along with $\mathcal{A}'_\bot$ being the valid substitution of $\mathcal{A}_\bot$, we can obtain all the statements in lemma 5.3 using similar ideas. According to [Wang et al., 2008], $\mathcal{T}'$ is a model-based forgetting of $\mathcal{T}$, that is $\mathsf{Mod}(\mathcal{T}') \sim_\Sigma \mathsf{Mod}(\mathcal{T})$, then the equivalence $(\mathcal{T}, \mathcal{A}'_\bot \cup \mathcal{A}'_\top) \models_\gamma q \Leftrightarrow \mathcal{O} \models_\gamma q$ can be naturally reserved if we replace $\mathcal{T}$ with $\mathcal{T}'$, thus the return ontology $\mathcal{O}'$ is a si-forgetting about Σ in $\mathcal{O}$. □

Though neither st-forgetting nor si-forgetting is guaranteed to exist in DL-Lite$_n$, in what follows, we show that a result of γ-forgetting about concepts always exists.

Theorem 5.2. *For a DL-Lite$_{core}$ ontology $\mathcal{O}$ and a set of concept names Σ, a result of γ-forgetting about Σ of $\mathcal{O}$ always exists in DL-Lite$_n$ for each $\gamma \in \Gamma_{\mathsf{basic}}$.*

To show the above result, we first note that the TBox for a result of γ-forgetting is always the same (up to logical equivalence) regardless the semantics γ.

Proposition 5.3. *For an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ in DL-lite$_{core}$, a set of concept names Σ and a semantics γ, if $(\mathcal{T}', \mathcal{A}')$ is a result of γ-forgetting about Σ of $\mathcal{O}$ then $\mathcal{T}' \equiv \mathsf{forget}(\mathcal{T}, \Sigma)$.*

Proof. According to the definition of inconsistency-tolerant query answering, for any set of inclusion axioms (i.e., TBox axioms) $\mathcal{Q}$, $\mathcal{O} \models_\gamma \mathcal{Q} \Leftrightarrow \mathcal{O} \models \mathcal{Q}$. Therefore, $(\mathcal{T}', \mathcal{A}') \models \mathcal{Q} \Leftrightarrow \mathcal{O} \models \mathcal{Q} \Leftrightarrow \mathsf{forget}(\mathcal{T}, \Sigma) \models \mathcal{Q}$, thus $\mathcal{T}' \equiv \mathsf{forget}(\mathcal{T}, \Sigma)$. □

We have discussed that st-forgetting can be performed in two steps and the result of which is actually an IAR-forgetting. And particularly, $\mathcal{A} \setminus \mathsf{confs}(\mathcal{O})$ is the IAR repair of $\mathcal{O}$ defined in [Lembo et al., 2010].

Proposition 5.4. *For a DL-lite$_{core}$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and a set of concept names Σ, $\mathsf{forget}((\mathcal{T}, \mathcal{A} \setminus \mathsf{confs}(\mathcal{O}), \Sigma)$ is a result of IAR-forgetting about Σ of $\mathcal{O}$.*

To compute the ABox of a result of ICAR-forgetting, one can simply replace $\mathcal{A}$ with $\mathsf{clc}_\mathcal{T}(\mathcal{A})$ before computing the IAR-repair (which computes the ICAR-repair) and performing consistent ontology forgetting.

The cases for AR-forgetting and CAR-forgetting are much more complicated, as there no longer exists a single trivial AR- or CAR-repair that one can perform consistent KB forgetting on. Instead, we need to consider different repairs of the ontology and all the possible sets of DL-Lite$_n$ assertions (BCQs) that are entailed by all repairs, which is indeed a very challenging task.

Example 5.10. *Let* $\mathcal{O} = (\mathcal{T}, \mathcal{A})$*, where*

$$\begin{aligned}\mathcal{T} = \{&\mathsf{A} \sqsubseteq \exists \mathsf{P}, \quad \exists \mathsf{P}^- \sqsubseteq \mathsf{T}, \quad \mathsf{T} \sqsubseteq \mathsf{P} \\ &\mathsf{B} \sqsubseteq \exists \mathsf{R}, \quad \exists \mathsf{R}^- \sqsubseteq \exists \mathsf{P}, \quad \mathsf{A} \sqsubseteq \neg \mathsf{B}\} \\ \mathcal{A} = \{&\mathsf{A}(a), \mathsf{B}(a)\}\end{aligned}$$

$\mathcal{O}$ *is inconsistent and there are two repairs of* $\mathcal{O}$*,* $\mathcal{B}_1 = \{\mathsf{A}(a)\}$ *and* $\mathcal{B}_2 = \{B(a)\}$*. Let* $\mathcal{O}_1 = (\mathcal{T}, \mathcal{B}_1)$ *and* $\mathcal{O}_2 = (\mathcal{T}, \mathcal{B}_2)$*, to check the possible entailment of* $\mathcal{O}_1$ *and* $\mathcal{O}_2$*, we compute their canonical models, which are represented as sets of atoms,*

$$\begin{aligned}\mathcal{I}_{\mathcal{O}_1} =&\{\mathsf{A}(a), \mathsf{P}(a, u_1), \mathsf{T}(u_1, u_2), \mathsf{P}(u_2, u_3), \dots\}, \\ \mathcal{I}_{\mathcal{O}_2} =&\{\mathsf{B}(a), \mathsf{R}(a, u_1), \mathsf{P}(u_1, u_2), \mathsf{T}(u_2, u_3), \mathsf{P}(u_3, u_4), \dots\}\end{aligned}$$

here for neat presentation, instead of paths, u_i *are used to represent the labeled nulls in canonical models. Both canonical models are infinite, but we can tell that all BCQs of the form* $q_n = \{\mathsf{P}(x_0, x_1), \mathsf{T}(x_1, x_2), \dots, \mathsf{P}(x_n, x_{n+1}), \mathsf{T}(x_{n+1}, x_{n+2})\}$ $(n \geq 0)$ *can be entailed by* $\mathcal{O}$ *under the* AR *semantics. Then a direct ABox that captures all the* AR *entailment of* $\mathcal{O}$ *is an infinite set* $\{\mathsf{P}(u_0, u_1), \mathsf{T}(u_1, u_2), \mathsf{P}(u_2, u_3), \mathsf{T}(u_3, u_4) \dots\}$*. Actually, there is another simple ABox that can capture all* AR *entailment, which is* $\{\mathsf{P}(u_0, u_1)\}$*.*

In the above example, we can discover that the key to decide AR entailment is to find the 'converge points' of the canonical models $\mathcal{I}_{\mathcal{O}_i}$: $\mathsf{P}(x, y)$ is satisfied in both $\mathcal{I}_{\mathcal{O}_1}$ and $\mathcal{I}_{\mathcal{O}_2}$. The rest AR entailment can be completed by adding TBox: for any BCQ q, q is entailed by $(\{\mathsf{P}^- \sqsubseteq \mathsf{T},\ \mathsf{T} \sqsubseteq \mathsf{P}\}, \{\mathsf{P}(u_0, u_1)\})$ if and only q is entailed by $\mathcal{O}$ under the AR semantics.

However, to find such 'converge points', it is unnecessary to compute the whole canonical model, which is possibly infinite. We propose a finite model for DL-lite$_{core}$ ontologies called *semi-model*. The semi-model of an ontology $\mathcal{O}$, denoted $\mathcal{J}_{\mathcal{O}}$, is defined in a similar way as the canonical model $\mathcal{I}_{\mathcal{O}}$. The difference is that the domain of $\mathcal{J}_{\mathcal{O}}$, denoted $\Delta^{\mathcal{J}_{\mathcal{O}}}$ is a finite subset of $\Delta^{\mathcal{I}_{\mathcal{O}}}$. More specifically, $\Delta^{\mathcal{J}_{\mathcal{O}}}$ consists of paths $ac_{R_1} \cdots c_{R_n} \in \Delta^{\mathcal{I}_{\mathcal{O}}}$ such that $c_{R_i} \neq c_{R_j}$, for $1 \leq i \neq j \leq n$. That is , $\Delta^{\mathcal{J}_{\mathcal{O}}}$ consists of paths without any loop and thus the domain is finite. And the mapping function for $\mathcal{J}_{\mathcal{O}}$, $\cdot^{\mathcal{J}_{\mathcal{O}}}$ is defined as follows:

$$a^{\mathcal{J}_\mathcal{O}} = a \text{ for } a \in \mathsf{ind}(\mathcal{O}), \quad A^{\mathcal{J}_\mathcal{O}} = A^{\mathcal{I}_\mathcal{O}} \cap \Delta^{\mathcal{J}_\mathcal{O}},$$
$$P^{\mathcal{J}_\mathcal{O}} = P^{\mathcal{I}_\mathcal{O}} \cap (\Delta^{\mathcal{J}_\mathcal{O}} \times \Delta^{\mathcal{J}_\mathcal{O}}).$$

Example 5.11. *In example 5.10, we have*

$$\mathcal{J}_{\mathcal{O}_1} = \{\mathsf{A}(a), \mathsf{P}(a, u_1), \mathsf{T}(u_1, u_2)\},$$
$$\mathcal{J}_{\mathcal{O}_2} = \{\mathsf{B}(a), \mathsf{R}(a, u_1), \mathsf{P}(u_1, u_2), \mathsf{T}(u_2, u_3)\}$$

And $\mathsf{P}(x, y)$ *is satisfied in both* $\mathcal{J}_{\mathcal{O}_1}$ *and* $\mathcal{J}_{\mathcal{O}_2}$.

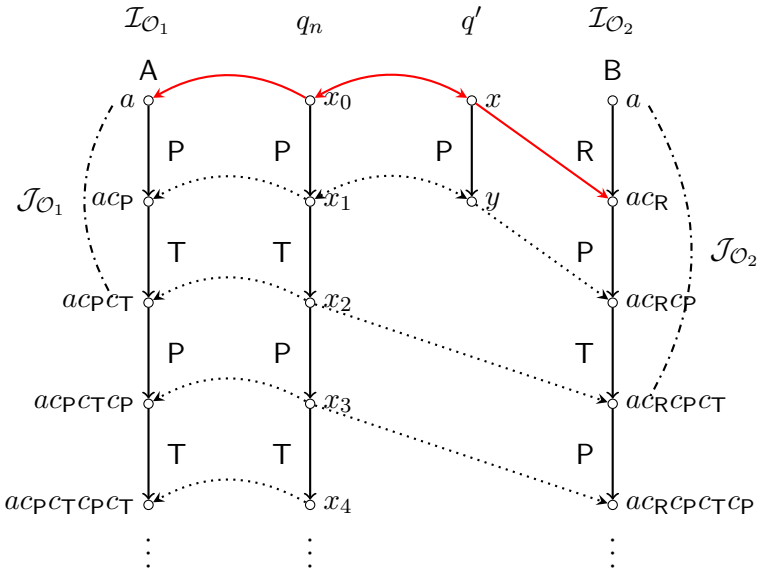


FIGURE 5.1: AR entailment of example 5.10

Now we show that semi-models are sufficient to capture all AR entailment for DL-lite$_{core}$ ontologies, the intuitive idea of which is, if there is a 'converge point' for the canonical models of the repaired ontologies, then it must be covered by the semi-models. Define $\mathcal{Q}_\mathcal{O}$ to be the set of all BCQ q satisfying that, for each repair $\mathcal{B}$ of $\mathcal{O}$, $\mathcal{J}_{(\mathcal{T},\mathcal{B})} \models q$.

Proposition 5.5. *For an ontology* $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ *in DL-lite*$_{core}$ *and each BCQ* q*, we have* $\mathcal{O} \models_{\mathsf{AR}} q$ *iff there exists* $q' \in \mathcal{Q}_\mathcal{O}$ *such that* $\mathcal{T} \cup q' \models q$.

Proof. As any BCQ is equivalent to a set of connected BCQs (i.e., any atom in the query has shared variable with another atom in the query), it is sufficient for us to discuss only connected BCQs.

The right direction is straightforward, we show the left direction. Let q be a connected BCQ, $\mathcal{O}_i = (\mathcal{T}, \mathcal{B}_i)$, where $\mathcal{B}_i \in \mathsf{rep}(\mathcal{O})$. If $\mathcal{O} \models_{\mathsf{AR}} q$, then $\mathcal{I}_{\mathcal{O}_i} \models q$ for every $\mathcal{O}_i$, i.e., there exists a homomorphism σ_i from q to $\mathcal{I}_{\mathcal{O}_i}$ such that $q\sigma_i \subseteq \mathcal{I}_{\mathcal{O}_i}$. Now we obtain a subset q_i of q for each $\mathcal{O}_i$ and show $\mathcal{T} \cup q_i \models q$:

1. If $q\sigma_i \cap \mathcal{J}_{\mathcal{O}_i} \neq \emptyset$, we make q_i be the subset of q that are mapped to $\mathcal{J}_{\mathcal{O}_i}$ with σ_i, i.e, $q_i\sigma_i = \mathcal{J}_{\mathcal{O}_i} \cap q\sigma_i$, it follows that $\mathcal{J}_{\mathcal{O}_i} \models q_i$. To explain $\mathcal{T} \cup q_i \models q$, we view q_i as a special set of facts and define σ' as follows, for $v \in \mathsf{var}(q_i)$, $v\sigma' = v$, and for $v' \in \mathsf{var}(q) \setminus \mathsf{var}(q_i)$, if $R(v'', v')$ is in q, then $v'\sigma' = (v''\sigma_i)c_R$, note that only one role relation can occur between v and v'' if v' is outside of q_i. It is easy to find that $q\sigma'$ is a subset of the canonical model of $\mathcal{T} \cup q_i$, so $\mathcal{T} \cup q_i \models q$.

2. If $q\sigma_i \subseteq (\mathcal{I}_{\mathcal{O}_i} \setminus \mathcal{J}_{\mathcal{O}_i})$, let $p = ac_{R_1} \dots c_{R_n}$ be the shortest path in the range of σ_i, we define $p\sigma' = ac_{R_1} \dots c_{R_{n'}}$, where $n' \leq n$, and for $l \neq m$, $c_{R_l} \neq c_{R_m}$. For any other path $p' = pp_t$ in the range of σ_i, we define $p'\sigma' = (p\sigma')p_t$. It can be checked that $q\sigma'\sigma_i$ is still a subset of $\mathcal{I}_{\mathcal{O}}$, and $q\sigma'\sigma_i \cap \mathcal{J}_{\mathcal{O}_i} \neq \emptyset$. Then we can obtain q_i as the above situation such that $\mathcal{J}_{\mathcal{O}_i} \models q$ and $\mathcal{T} \cup q_i \models q$.

w.l.o.g. we suppose $\mathcal{O}$ has two repairs $\mathcal{B}_1$, $\mathcal{B}_2$. Let q_1 and q_2 be the subsets of q obtained for $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively, we prove that there exists q' that $\mathcal{T} \cup q' \models q$ and $\mathcal{J}_{\mathcal{O}_1}, \mathcal{J}_{\mathcal{O}_2} \models q'$, i.e., $q' \subseteq \mathcal{Q}_{\mathcal{O}}$.

1. If $q_1 \cap q_2 = \emptyset$, then $q_1\sigma_2 \subseteq \mathcal{I}_{\mathcal{O}_2} \setminus \mathcal{J}_{\mathcal{O}_2}$, according to the preceding discussion, there exists $q' \subseteq q_1$ such that $\mathcal{T} \cup q'_1 \models q_1$, clearly, q' is also a subset of q_2, so $\mathcal{J}_{\mathcal{O}_2} \models q'$. And because $\mathcal{T} \cup q_1 \models q$, so $\mathcal{T} \cup q' \models q$.

2. If $q_1 \cap q_2 \neq \emptyset$, let $q' = q_1 \cap q_2$, then $\mathcal{J}_{\mathcal{O}_1}, \mathcal{J}_{\mathcal{O}_2}$ both entails q'. Because $(q \setminus q_2)\sigma_2 \subseteq \mathcal{I}_{\mathcal{O}_2} \setminus \mathcal{J}_{\mathcal{O}_2}$, so $(q_1 \setminus q_2)\sigma_2 \subseteq \mathcal{I}_{\mathcal{O}_2} \setminus \mathcal{J}_{\mathcal{O}_2}$, let $q'_2\sigma_2 \in q_2\sigma_2$ be the atom adjacent to $(q_1 \setminus q_2)\sigma_2$, then similarly, we have $\mathcal{T} \cup q'_2 \models q_1 \setminus q_2$. And because any atoms adjacent to $(q_1 \setminus q_2)\sigma_2$ must be in $q_1\sigma_2$, otherwise $q_1 \cap q_2 = \emptyset$ or q_1 is not connected, so $q'_2 \subseteq q_1 \cap q_2 = q'$, thus $\mathcal{T} \cup q' \models q' \cup (q_1 \setminus q_2) = q_1 \Rightarrow \mathcal{T} \cup q' \models q$.

Our claim immediately follows. □

Because $\mathcal{J}_{\mathcal{O}}$ is always finite, it can be shown that $\mathcal{Q}_{\mathcal{O}}$ can be represented by a finite subset.

Lemma 5.4. *There exists a finite subset* $\mathcal{Q}^*_{\mathcal{O}} \subseteq \mathcal{Q}_{\mathcal{O}}$ *such that* $\mathcal{Q}^*_{\mathcal{O}} \models \mathcal{Q}_{\mathcal{O}}$.

Proof. Let $\mathcal{O}_i = (\mathcal{T}, \mathcal{B}_i)$, where $\mathcal{B}_i$ is a repair of $\mathcal{O}$. We note that each $\mathcal{J}_{\mathcal{O}_i}$ is a finite set of atoms. For any q that can be homomorphically mapped to subsets of each $\mathcal{J}_{\mathcal{O}_i}$, we can always construct a finite q' whose size is bounded by the max size of $\mathcal{J}_{\mathcal{O}_i}$ such that q' is homomorphically mapped to each $\mathcal{J}_{\mathcal{O}_i}$ and $q' \models q$. With a bounded size (the max size of $\mathcal{J}_{\mathcal{O}_i}$), a bounded number of predicate symbols (the intersection of signatures of each $\mathcal{J}_{\mathcal{O}_i}$), a bounded number of constants, the number of q' that can be constructed for all $q \in \mathcal{Q}_{\mathcal{O}}$ is finite (up to variable renaming). Let $\mathcal{Q}^*_{\mathcal{O}}$ be the union of all such q', then $\mathcal{Q}^*_{\mathcal{O}}$ is a finite subset of $\mathcal{Q}_{\mathcal{O}}$, and $\mathcal{Q}^*_{\mathcal{O}} \models \mathcal{Q}_{\mathcal{O}}$. □

Now we can obtain a set of DL-lite$_n$ assertions $\mathcal{A}^*_{\mathcal{O}}$ from $\mathcal{Q}^*_{\mathcal{O}}$, by replacing each variable in $\mathcal{Q}^*_{\mathcal{O}}$ with distinct labeled nulls. Note that we assume different q in $\mathcal{Q}^*_{\mathcal{O}}$ uses a different set of variables. We claim that $\mathcal{A}^*_{\mathcal{O}}$ is consistent with the TBox.

Lemma 5.5. $\mathcal{A}^*_{\mathcal{O}}$ *is always consistent with* $\mathcal{T}$.

Proof. The claim is easy to show. For any BCQs q, if q can be entailed by $(\mathcal{T}, \mathcal{A})$ under the AR semantics, then q is entailed by $(\mathcal{T}, \mathcal{B}_i)$, where $\mathcal{B}_i$ is a repair of $(\mathcal{T}, \mathcal{A})$, then q must be consistent with $\mathcal{T}$, otherwise $(\mathcal{T}, \mathcal{B}_i)$ is inconsistent. While replacing variables with labeled nulls in the BCQs will not create conflicts in them. □

Then given a set of concept names Σ, $\mathsf{forget}((\mathcal{T}, \mathcal{A}^*_{\mathcal{O}}), \Sigma)$ is a result of AR-forgetting about Σ of $\mathcal{O}$.

Similarly, to compute the ABox of a result of CAR-forgetting, one can simply replace $\mathcal{A}$ with $\mathsf{clc}_{\mathcal{T}}(\mathcal{A})$ before computing $\mathcal{Q}_{\mathcal{O}}$. Thus we completes the proof of theorem 5.2.

5.3 Graph-based Implementation and Evaluation

From the above discussions, we can see that γ-forgetting possesses favourable properties w.r.t. computation. In particular, the computation of IAR- and ICAR-forgetting enjoys polynomial time data complexity. To evaluate its scalability over large ontologies, we implemented a prototype system for IAR-forgetting in DL-lite$_{core}$.

5.3.1 Graph-based Implementation

Instead of directly applying the constructive method used in (the proofs of) the previous section, we adopted a graph-based approach, which allows us to represent and store DL-lite ontologies in *graph databases* and perform forgetting using well optimised graph operations. We are inspired by the graph-based approaches for ontology revision [Qi et al., 2015, Fu et al., 2016], which were shown to be effective in handling conflicts. Yet in contract to ontology revision, where the update is small compared to the whole ontology, forgetting often requires graph operations on a much larger scale. We briefly introduce our graph-based approach for IAR-forgetting.

Given an ontology $\mathcal{O}$ in DL-lite$_{core}$, we first transform $\mathcal{O}$ into a directed binary graph $G_{\mathcal{O}}$ in a similar manner as [Qi et al., 2015], which can be stored in the high-performance graph database like Neo4j [1]. In $G_{\mathcal{O}}$, each node represents either a concept or an individual of $\mathcal{O}$. An edge represents either an inclusion axiom between concepts, which we call an inclusion edge, or a membership assertion between an individual and a concept, which we call a membership edge. To forget about a set of concepts Σ from $\mathcal{O}$, we perform the following operations on $G_{\mathcal{O}}$:

1. For each pair of nodes of the form B and $\neg B$, we search for their shared members and remove the corresponding membership edges (which represent conflicts in $\mathcal{O}$).
2. For each node of the form A or $\neg A$ with $A \in \Sigma$, we build edges between its predecessors and successors and then remove all the edges associated to it.

Then the ontology represented by the resulting $G_{\mathcal{O}}$ is an IAR-forgetting about Σ in $\mathcal{O}$.

[1] https://neo4j.com/

Figure 5.2 demonstrates $G_{\mathcal{O}_e}$ of $\mathcal{O}_e$ in Example 5.2 (each concept name is abbreviated using its first one or two letters). The operations to achieve IAR-forgetting (see Example 5.7) are also shown: a dotted line (or circle) represents the edge (resp., the node) is removed, and a bold line represent an edge is added. Note that all membership edges going out of a are removed because a is a common member of Bird and ¬Bird, and edges from b to Bird and Pet are added before Parrot is removed.

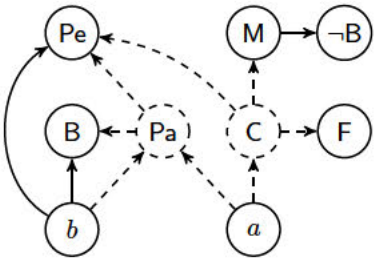


FIGURE 5.2: Graph-based procedure of IAR-forgetting for Example 5.7

5.3.2 Evaluation

We conducted experiments using the benchmark of inconsistent ontologies proposed in [Bienvenu et al., 2014b], which extends LUBM ontology with disjointness axioms. The benchmark also includes auto-generated ABoxes of a range of sizes containing conflicts in various ratios.

All experiments are performed on a server with Intel Xeon E5-1603 2.80 GHz CPU and 16GB of RAM, running Linux mint 17 OS, and Java 1.8 with 6GB of heap space.

We randomly select sets of concepts from the the signature of LUBM to forget, and each set contains 40 concepts. Note that selection of concepts has a considerable influence on the running time of computation and forgetting 40 concepts each time, which is nearly a half of the total number of concepts in LUBM, allows us to have good coverage of concepts. We evaluated 12 different concept sets on each ABox, and record the average running times for forgetting as well as the ABox sizes before and after forgetting.

The results are shown in Table 5.1, where $\mathcal{A}_m^n$ denotes an ABox about m universities containing conflicts with a ratio of n. The results show that our system is scalable over large ontologies. For an ABox with up to 2M assertions, our system is able to compute forgetting in around one minute. Also as shown in the results, the performance

ABox	original size	result size	Time (s)
$\mathcal{A}_5^{15e-4}$	500K	659K	21.9
$\mathcal{A}_5^{5e-2}$	508K	664K	22.6
$\mathcal{A}_5^{2e-1}$	532K	678K	21.0
$\mathcal{A}_{10}^{15e-4}$	931K	1.2M	32.5
$\mathcal{A}_{10}^{5e-2}$	945K	1.2M	33.6
$\mathcal{A}_{10}^{2e-1}$	989K	1.3M	34.7
$\mathcal{A}_{20}^{15e-4}$	2.0M	2.6M	63.4
$\mathcal{A}_{20}^{5e-2}$	2.0M	2.7M	64.9
$\mathcal{A}_{20}^{2e-1}$	2.1M	2.7M	67.2

TABLE 5.1: IAR-forgetting on inconsistent LUBM ontologies

is rather stable across different ration of conflicts, which is contributed by the graph-based approach that allows the conflicts to be handled quite efficiently. Finally, when forgetting eliminates concepts and conflicts from an ontology, which corresponds to the removal of nodes and edges in the graph, it also adds new axioms, which corresponds to the addition of edges between the predecessors and successors of the eliminated nodes. Since forgetting certain key concepts (which can be seen as hubs in a network) requires cross-linking many nodes, the forgetting operation does not necessarily reduce the size of the initial ontology. Indeed in our experiment, the average sizes of the ontology grew after forgetting, yet the growth is limited.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The prime aim of this thesis is to promote the application of OBDA, by studying typical reasoning problems over expressive ontologies and proposing practical approaches for these problems when they cannot be finely handled in real-world cases. We emphasize four important capabilities of a practical OBDA systems, which are the capabilities of efficient query answering over large volumes of data, explaining reasoning services, properly manipulating ontologies and handling possibly inconsistent knowledge. However, existing research and techniques are still far from meeting these requirements.

Because first-order rewriting suffers from poor scalability, which become more serious in expressive languages, a more compact rewriting should be considered for existential rules. Observing that current studies about datalog rewriting in existential rules are mostly theoretical, providing no practical algorithms for implementation and very few systems driven by datalog rewriting have been developed, we propose a practical approach and deliver a prototype system called *Drewer* for query answering and data rewriting in existential rules. Our rewriting approach is based on the notion of unfolding, while such technique usually generates a lot of redundant rules, we use a sophisticate strategy to reuse predicates in the construction of final rewriting. For datalog rewritability, we identify an abstract class called weakly separable rule sets, and show that a number of novel classes falling in this class can be obtained by combining existing well-accepted classes. We evaluate our system over a wide range of benchmarks and compare it with

state-of-the-art systems, the result of which shows superior performance of our system on both the compactness of rewriting and efficiency of query answering.

Regarding the capability of explaining query answering, current research of query abduction is still not sufficient for application as they are focusing on solving the problem, not providing appropriate strategies to satisfy the needs of different users in different scenarios. Moreover, most of these studies are discussed in less expressive languages. To provide suitable explaining services, explanations should be evaluated and selected in a more delicate way, for which we propose a novel framework for query abduction, where by further classification of abducibles, users are allowed to distinguish high-level pattern explanations from low-level concrete explanations. We show that our proposed explanation can be much more compact than current explanations, such as representative explanations. For its computation, we develop an algorithm based on backward chaining rewriting for existential rules. Evaluation result shows our approach can scale well over large databases.

When existing techniques of forgetting can not apply to complex real-world scenarios where inconsistencies might occur, we present the first study about forgetting over inconsistent ontologies. It is a challenging topic because when an ontology $\mathcal{O}$ is inconsistent, it is difficult to define the equivalence relation between $\mathcal{O}$ and its forgetting results. We propose three general definitions based on inconsistency-tolerant query answering, their properties and computation are discussed in DL-lite$_{core}$. We show through experiments that our proposed forgetting can be effectively computed for large inconsistent ontologies.

6.2 Future Work

We outlook possible future work.

1. As both chase-based and rewriting-based query answering have their pros and cons, many existing systems consider a combined approach, that is, using both rewriting and database saturation in the actual evaluations, a notable example of which is the Graal system. Such a combination can usually improve the performance of the system and let it handle a lot more ontologies even if they are not rewritable or

cannot be finitely chased. It also provide a certain kind of flexibility when users have different attitudes to the managements of data. We are planning to introduce this feature to *Drewer*.

2. It shall be noted that current works about explaining positive answers also suffer similar limitations as query abduction do. Our framework can also be adapted to explaining positive query answers. In addition, in our framework, we discriminate between predicates to select desired explanations, it would be interesting to consider a more delicate criteria based on the predicate positions to further filter the explaining results, which seems rational in real-world cases, as different users may have different requirements on the terms at different positions.

3. Regarding our proposed inconsistency-tolerant forgettings, we discuss their properties and computation methods only in DL-lite$_{core}$, it is necessary to extend our current work to cover more expressive languages like existential rules. Besides, though we have shown the existence of results of AR-forgetting in DL-lite$_{core}$, due to the complex nature of the problem, it is still very challenging to deliver an efficient algorithm for AR-forgetting. It seems that a promising approach to compute a result of AR-forgetting is to transform it into existing well-studied problems, such as answering set programming (ASP) [Gelfond and Lifschitz, 1988], where repairs can possibly be expressed by *stable models*.

References

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

[Ahmetaj et al., 2018] Ahmetaj, S., Ortiz, M., and Simkus, M. (2018). Rewriting guarded existential rules into small datalog programs. In *Proceedings of 21st International Conference on Database Theory (ICDT 2018)*, pages 4:1–4:24.

[Arenas et al., 1999] Arenas, M., Bertossi, L. E., and Chomicki, J. (1999). Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-99)*, pages 68–79.

[Arenas et al., 2016] Arenas, M., Botoeva, E., Calvanese, D., and Ryzhikov, V. (2016). Knowledge base exchange: The case of OWL 2 QL. *Artif. Intell.*, 238:11–62.

[Arocena et al., 2015] Arocena, P. C., Glavic, B., Ciucanu, R., and Miller, R. J. (2015). The ibench integration metadata generator. *Proc. VLDB Endow.*, 9(3):108–119.

[Baader et al., 2005] Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the EL envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 364–369.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*.

[Baader et al., 2008] Baader, F., Lutz, C., and Brandt, S. (2008). Pushing the EL envelope further. In *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions*.

[Baader and Suntisrivaraporn, 2008] Baader, F. and Suntisrivaraporn, B. (2008). Debugging SNOMED CT using axiom pinpointing in the description logic EL+. In *Proceedings of the Third International Conference on Knowledge Representation*, volume 410 of *CEUR Workshop Proceedings*.

[Bachmair et al., 2001] Bachmair, L., Ganzinger, H., McAllester, D. A., and Lynch, C. (2001). Resolution theorem proving. In Robinson, J. A. and Voronkov, A., editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 19–99. Elsevier and MIT Press.

[Baget et al., 2016] Baget, J., Benferhat, S., Bouraoui, Z., Croitoru, M., Mugnier, M., Papini, O., Rocher, S., and Tabia, K. (2016). A general modifier-based framework for inconsistency-tolerant query answering. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference (KR-16)*, pages 513–516.

[Baget et al., 2015] Baget, J., Leclère, M., Mugnier, M., Rocher, S., and Sipieter, C. (2015). Graal: A toolkit for query answering with existential rules. In *Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Proceedings*, volume 9202 of *Lecture Notes in Computer Science*, pages 328–344.

[Baget et al., 2011] Baget, J., Leclère, M., Mugnier, M., and Salvat, E. (2011). On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654.

[Bard and Rhee, 2004] Bard, J. B. and Rhee, S. Y. (2004). Ontologies in biology: design, applications and future challenges. *Nature Reviews Genetics*, 5(3):213–222.

[Beeri and Vardi, 1981] Beeri, C. and Vardi, M. Y. (1981). The implication problem for data dependencies. In *Automata, Languages and Programming, 8th Colloquium, Proceedings*, pages 73–85.

[Bellomarini et al., 2017] Bellomarini, L., Gottlob, G., Pieris, A., and Sallinger, E. (2017). Swift logic for big data and knowledge graphs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, (IJCAI-17)*, pages 2–10.

[Belnap, 1977] Belnap, N. D. (1977). A useful four-valued logic. In *Modern uses of multiple-valued logic*, pages 5–37. Springer.

[Benedikt et al., 2017] Benedikt, M., Konstantinidis, G., Mecca, G., Motik, B., Papotti, P., Santoro, D., and Tsamoura, E. (2017). Benchmarking the chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-17)*, pages 37–52.

[Benedikt et al., 2018] Benedikt, M., Motik, B., and Tsamoura, E. (2018). Goal-driven query answering for existential rules with equality. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 1761–1770.

[Bertaud-Gounot et al., 2012] Bertaud-Gounot, V., Duvauferrier, R., and Burgun, A. (2012). Ontology and medical diagnosis. *Informatics for Health and Social Care*, 37(2):51–61.

[Bienvenu and Bourgaux, 2016] Bienvenu, M. and Bourgaux, C. (2016). Inconsistency-tolerant querying of description logic knowledge bases. In *Reasoning Web: Logical Foundation of Knowledge Graph Construction and Query Answering - 12th International Summer School 2016, Tutorial Lectures*, volume 9885 of *Lecture Notes in Computer Science*, pages 156–202.

[Bienvenu et al., 2014a] Bienvenu, M., Bourgaux, C., and Goasdoué, F. (2014a). Querying inconsistent description logic knowledge bases under preferred repair semantics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, pages 996–1002.

[Bienvenu et al., 2014b] Bienvenu, M., Bourgaux, C., and Goasdoué, F. (2014b). Querying inconsistent description logic knowledge bases under preferred repair semantics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, pages 996–1002.

[Bienvenu et al., 2017] Bienvenu, M., Kikot, S., Kontchakov, R., Podolskii, V. V., Ryzhikov, V., and Zakharyaschev, M. (2017). The complexity of ontology-based data access with OWL 2 QL and bounded treewidth queries. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-17)*, pages 201–216.

[Bienvenu et al., 2014c] Bienvenu, M., ten Cate, B., Lutz, C., and Wolter, F. (2014c). Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44.

[Borgida et al., 2008] Borgida, A., Calvanese, D., and Rodriguez-Muro, M. (2008). Explanation in the dl-lite family of description logics. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1440–1457. Springer.

[Calì et al., 2012] Calì, A., Gottlob, G., and Lukasiewicz, T. (2012). A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83.

[Calvanese et al., 2007a] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2007a). Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429.

[Calvanese et al., 2007b] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2007b). Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429.

[Calvanese et al., 2011] Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2011). The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53.

[Calvanese et al., 2007c] Calvanese, D., Lembo, D., Lenzerini, M., Poggi, A., and Rosati, R. (2007c). Ontology-based database access. In *Proceedings of the 15th Italian Symposium on Advanced Database Systems (SEBD-07)*, pages 324–331.

[Calvanese et al., 2013] Calvanese, D., Ortiz, M., Šimkus, M., and Stefanoni, G. (2013). Reasoning about Explanations for Negative Query Answers in DL-Lite. *Journal of Artificial Intelligence Research*, 48:635–669.

[Ceylan et al., 2019] Ceylan, İ. İ., Lukasiewicz, T., Malizia, E., and Vaicenavicius, A. (2019). Explanations for query answers under existential rules. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pages 1639–1646.

[Chomicki, 2007] Chomicki, J. (2007). Consistent query answering: Five easy pieces. In *Proceedings of 11st International Conference on Database Theory (ICDT-07)*, pages 1–17.

[Donnelly, 2006] Donnelly, K. (2006). Snomed-ct: The advanced terminology and coding system for ehealth. *Studies in health technology and informatics*, 121:279.

[Du et al., 2011] Du, J., Qi, G., Shen, Y., and Pan, J. Z. (2011). Towards practical abox abduction in large OWL DL ontologies. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*.

[Du et al., 2014] Du, J., Wang, K., and Shen, Y. (2014). A tractable approach to abox abduction over description logic ontologies. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, pages 1034–1040.

[Du et al., 2015] Du, J., Wang, K., and Shen, Y. (2015). Towards tractable and practical abox abduction over inconsistent description logic ontologies. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 1489–1495.

[Eiter et al., 2012] Eiter, T., Ortiz, M., Simkus, M., Tran, T., and Xiao, G. (2012). Query rewriting for horn-shiq plus rules. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*.

[Elsenbroich et al., 2006] Elsenbroich, C., Kutz, O., and Sattler, U. (2006). A case for abductive reasoning over ontologies. In *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions*, volume 216 of *CEUR Workshop Proceedings*.

[Fagin et al., 2005] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. (2005). Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124.

[Fu et al., 2016] Fu, X., Qi, G., Zhang, Y., and Zhou, Z. (2016). Graph-based approaches to debugging and revision of terminologies in dl-lite. *Knowl.-Based Syst.*, 100:1–12.

[Gabbay and Ohlbach, 1992] Gabbay, D. M. and Ohlbach, H. J. (1992). Quantifier elimination in second-order predicate logic. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pages 425–435.

[Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. A. and Bowen, K. A., editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, (2 Volumes)*, pages 1070–1080.

[Gottlob et al., 2014a] Gottlob, G., Manna, M., and Pieris, A. (2014a). Polynomial combined rewritings for existential rules. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference (KR-14)*.

[Gottlob et al., 2015] Gottlob, G., Manna, M., and Pieris, A. (2015). Polynomial combined rewritings for linear existential rules and dl-lite with n-ary relations. In *Proceedings of the 28th International Workshop on Description Logics*.

[Gottlob et al., 2014b] Gottlob, G., Orsi, G., and Pieris, A. (2014b). Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46.

[Gottlob et al., 2014c] Gottlob, G., Orsi, G., and Pieris, A. (2014c). Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46.

[Gottlob et al., 2007] Gottlob, G., Pichler, R., and Wei, F. (2007). Efficient datalog abduction through bounded treewidth. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1626–1631.

[Gottlob et al., 2014d] Gottlob, G., Rudolph, S., and Simkus, M. (2014d). Expressiveness of guarded existential rule languages. In *Proc. of PODS-14*, pages 27–38.

[Gottlob et al., 2014e] Gottlob, G., Rudolph, S., and Simkus, M. (2014e). Expressiveness of guarded existential rule languages. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS-14)*, pages 27–38.

[Gottlob and Schwentick, 2012] Gottlob, G. and Schwentick, T. (2012). Rewriting ontological queries into small nonrecursive datalog programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference (KR-12)*.

[Grau et al., 2013] Grau, B. C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., and Wang, Z. (2013). Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res.*, 47:741–808.

[Guo et al., 2005] Guo, Y., Pan, Z., and Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. *J. Web Semant.*, 3(2-3):158–182.

[Hansen et al., 2015] Hansen, P., Lutz, C., Seylan, I., and Wolter, F. (2015). Efficient query rewriting in the description logic EL and beyond. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (ICJAI-15)*, pages 3034–3040.

[Hunter, 2000] Hunter, A. (2000). Reasoning with contradictory information using quasi-classical logic. *J. Log. Comput.*, 10(5):677–703.

[Kalyanpur et al., 2007] Kalyanpur, A., Parsia, B., Horridge, M., and Sirin, E. (2007). Finding all justifications of OWL DL entailments. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC07) (ASWC07)*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280.

[Konev et al., 2009] Konev, B., Walther, D., and Wolter, F. (2009). Forgetting and uniform interpolation in large-scale description logic terminologies. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 830–835.

[König et al., 2015a] König, M., Leclère, M., and Mugnier, M. (2015a). Query rewriting for existential rules with compiled preorder. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 3106–3112.

[König et al., 2013] König, M., Leclère, M., Mugnier, M., and Thomazo, M. (2013). Sound, complete, and minimal query rewriting for existential rules. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*, pages 3017–3021.

[König et al., 2015b] König, M., Leclère, M., Mugnier, M., and Thomazo, M. (2015b). Sound, complete and minimal UCQ-rewriting for existential rules. *Semantic Web*, 6(5):451–475.

[Kontchakov et al., 2010] Kontchakov, R., Lutz, C., Toman, D., Wolter, F., and Zakharyaschev, M. (2010). The combined approach to query answering in dl-lite. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference (KR-10)*.

[Kontchakov et al., 2011] Kontchakov, R., Lutz, C., Toman, D., Wolter, F., and Zakharyaschev, M. (2011). The combined approach to ontology-based data access.

In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 2656–2661.

[Kontchakov et al., 2008] Kontchakov, R., Wolter, F., and Zakharyaschev, M. (2008). Can you tell the difference between dl-lite ontologies? In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference (KR-08)*, pages 285–295.

[Koopmann and Schmidt, 2014] Koopmann, P. and Schmidt, R. A. (2014). Count and forget: Uniform interpolation of $\mathcal{SHQ}$-ontologies. In *Proceedings of the 7th International Joint Conference on Automated Reasoning*, pages 434–448.

[Koopmann and Schmidt, 2015] Koopmann, P. and Schmidt, R. A. (2015). Uniform interpolation and forgetting for ALC ontologies with aboxes. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 175–181.

[Krötzsch, 2012] Krötzsch, M. (2012). Owl 2 profiles: An introduction to lightweight ontology languages. In *Reasoning Web International Summer School*, pages 112–183. Springer.

[Lang et al., 2003a] Lang, J., Liberatore, P., and Marquis, P. (2003a). Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res.*, 18:391–443.

[Lang et al., 2003b] Lang, J., Liberatore, P., and Marquis, P. (2003b). Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res. (JAIR)*, 18:391–443.

[Lembo et al., 2010] Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2010). Inconsistency-tolerant semantics for description logics. In *Web Reasoning and Rule Systems - Fourth International Conference, RR 2010, Proceedings*, pages 103–117.

[Lembo et al., 2015] Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2015). Inconsistency-tolerant query answering in ontology-based data access. *J. Web Sem.*, 33:3–29.

[Leone et al., 2019] Leone, N., Manna, M., Terracina, G., and Veltri, P. (2019). Fast query answering over existential rules. *ACM Trans. Comput. Log.*, 20(2):12:1–12:48.

[Lifschitz et al., 2001] Lifschitz, V., Pearce, D., and Valverde, A. (2001). Strongly equivalent logic programs. *ACM Trans. Comput. Log.*, 2(4):526–541.

[Lin and Reiter, 1994] Lin, F. and Reiter, R. (1994). Forget it. In *Proceedings of the AAAI Fall Symposium on Relevance*, pages 154–159.

[Ludwig and Konev, 2014] Ludwig, M. and Konev, B. (2014). Practical uniform interpolation and forgetting for $\mathcal{ALC}$ TBoxes with applications to logical difference. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference (KR-14)*.

[Lukasiewicz et al., 2012] Lukasiewicz, T., Martinez, M. V., and Simari, G. I. (2012). Inconsistency handling in datalog+/- ontologies. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track*, pages 558–563.

[Lukasiewicz et al., 2013] Lukasiewicz, T., Martinez, M. V., and Simari, G. I. (2013). Complexity of inconsistency-tolerant query answering in datalog+/-. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Proceedings*, pages 488–500.

[Lutz et al., 2012] Lutz, C., Seylan, I., and Wolter, F. (2012). An automata-theoretic approach to uniform interpolation and approximation in the description logic $\mathcal{EL}$. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference (KR-12)*.

[Lutz and Wolter, 2011] Lutz, C. and Wolter, F. (2011). Foundations for uniform interpolation and forgetting in expressive description logics. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 989–995.

[McGuinness and Patel-Schneider, 1998] McGuinness, D. L. and Patel-Schneider, P. F. (1998). Usability issues in knowledge representation systems. In Mostow, J. and Rich, C., editors, *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, (AAAI-98) (IAAI-98)*, pages 608–614.

[McGuinness et al., 2004] McGuinness, D. L., Van Harmelen, F., et al. (2004). Owl web ontology language overview. *W3C recommendation*, 10(10):2004.

[Nenov et al., 2015] Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., and Banerjee, J. (2015). Rdfox: A highly-scalable RDF store. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Proceedings, Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 3–20.

[Nguyen and Szalas, 2010] Nguyen, L. A. and Szalas, A. (2010). Three-valued paraconsistent reasoning for semantic web agents. In *Agent and Multi-Agent Systems: Technologies and Applications, 4th KES International Symposium, KES-AMSTA 2010, Proceedings, Part I*, pages 152–162.

[Nikitina and Rudolph, 2014] Nikitina, N. and Rudolph, S. (2014). (Non-)Succinctness of uniform interpolants of general terminologies in the description logic $\mathcal{EL}$. *Artif. Intell.*, 215:120–140.

[Pérez-Urbina et al., 2010] Pérez-Urbina, H., Motik, B., and Horrocks, I. (2010). Tractable query answering and rewriting under description logic constraints. *J. Applied Logic*, 8(2):186–209.

[Qi et al., 2015] Qi, G., Wang, Z., Wang, K., Fu, X., and Zhuang, Z. (2015). Approximating model-based abox revision in dl-lite: Theory and practice. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 254–260.

[Rodriguez-Muro et al., 2013] Rodriguez-Muro, M., Kontchakov, R., and Zakharyaschev, M. (2013). Ontology-based data access: Ontop of databases. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 558–573.

[Rosati and Almatelli, 2010] Rosati, R. and Almatelli, A. (2010). Improving query answering over dl-lite ontologies. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference (KR-10)*.

[Sipser, 1996] Sipser, M. (1996). Introduction to the theory of computation. *SIGACT News*, 27(1):27–29.

[Soler-Toscano, 2019] Soler-Toscano, F. (2019). Which is the least complex explanation? abduction and complexity. *CoRR*, abs/1902.05479.

[Stefanoni et al., 2013] Stefanoni, G., Motik, B., and Horrocks, I. (2013). Introducing nominals to the combined query answering approaches for EL. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*.

[Suntisrivaraporn et al., 2008] Suntisrivaraporn, B., Qi, G., Ji, Q., and Haase, P. (2008). A modularization-based approach to finding all justifications for OWL DL entailments. In *The Semantic Web, 3rd Asian Semantic Web Conference, ASWC 2008, Proceedings*, volume 5367 of *Lecture Notes in Computer Science*, pages 1–15.

[Thomazo, 2013] Thomazo, M. (2013). *Conjunctive Query Answering Under Existential Rules - Decidability, Complexity, and Algorithms*. PhD thesis, Montpellier 2 University, France.

[Trivela et al., 2015] Trivela, D., Stoilos, G., Chortaras, A., and Stamou, G. B. (2015). Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49.

[Tsalapati et al., 2016] Tsalapati, E., Stoilos, G., Stamou, G. B., and Koletsos, G. (2016). Efficient query answering over expressive inconsistent description logics. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, pages 1279–1285.

[Urbani et al., 2016] Urbani, J., Jacobs, C. J. H., and Krötzsch, M. (2016). Column-oriented datalog materialization for large knowledge graphs. In Schuurmans, D. and Wellman, M. P., editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 258–264.

[Venetis et al., 2014] Venetis, T., Stoilos, G., and Stamou, G. B. (2014). Query extensions and incremental query rewriting for OWL 2 QL ontologies. *J. Data Semant.*, 3(1):1–23.

[Venetis et al., 2016] Venetis, T., Stoilos, G., and Vassalos, V. (2016). Rewriting minimisations for efficient ontology-based query answering. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (ICJAI-16)*, pages 1095–1102.

[Visser, 1996] Visser, A. (1996). *Uniform interpolation and layered bisimulation*, volume Volume 6 of *Lecture Notes in Logic*, pages 139–164. Association for Symbolic Logic, Berlin.

[Wan et al., 2016] Wan, H., Zhang, H., Xiao, P., Huang, H., and Zhang, Y. (2016). Query answering with inconsistent existential rules under stable model semantics. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 1095–1101.

[Wang et al., 2015] Wang, Z., Chitsaz, M., Wang, K., and Du, J. (2015). Towards scalable and complete query explanation with OWL 2 EL ontologies. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM-15)*, pages 743–752.

[Wang et al., 2008] Wang, Z., Wang, K., Topor, R. W., and Pan, J. Z. (2008). Forgetting concepts in dl-lite. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Proceedings*, pages 245–257.

[Wang et al., 2010] Wang, Z., Wang, K., Topor, R. W., and Pan, J. Z. (2010). Forgetting for knowledge bases in dl-lite. *Ann. Math. Artif. Intell.*, 58(1-2):117–151.

[Wang et al., 2018] Wang, Z., Wang, K., and Zhang, X. (2018). Forgetting and unfolding for existential rules. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 2013–2020.

[Zhang et al., 2014] Zhang, X., Xiao, G., Lin, Z., and den Bussche, J. V. (2014). Inconsistency-tolerant reasoning with OWL DL. *Int. J. Approx. Reasoning*, 55(2):557–584.

[Zhao and Schmidt, 2016] Zhao, Y. and Schmidt, R. A. (2016). Forgetting concept and role symbols in $\mathcal{ALCOIH}_{\mu}^{+}(\nabla, \sqcap)$-ontologies. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, pages 1345–1353.

[Zhou et al., 2015] Zhou, Y., Grau, B. C., Nenov, Y., Kaminski, M., and Horrocks, I. (2015). Pagoda: Pay-as-you-go ontology query answering using a datalog reasoner. *J. Artif. Intell. Res.*, 54:309–367.