DOCTORAL THESIS

Approaches to Abductive and Inductive Reasoning in Lightweight Ontologies

Mahsa Chitsaz
Master of Computer Science
Bachelor of Computer Engineering

School of Information and Communication Technology
Griffith University

Submitted in fulfilment of the requirements of the degree of

Doctor of Philosophy

February 2015
Abstract

Ontologies have been widely used in various application domains such as the Semantic Web, information systems and bio-informatics. Ontology is often modelled using description logics (DLs) to provide the mechanisms for representing knowledge of a domain, defining its semantics and reasoning upon it. Lightweight DL ontologies are those which purposefully designed to have high performance on standard reasoning tasks, including satisfiability, subsumption and query answering, while sacrificing to certain degree their expressive power. Along with the vast use of ontologies, non-standard reasoning tasks have started to emerge, namely abductive and inductive reasoning. Abductive reasoning is the process of inferring the best explanation from the given observations according to a background knowledge. Inductive reasoning is the process of deriving general logical theory from the given example data.

For abductive reasoning, we study one important explanation problem for DL ontologies, referred to as query abduction. Ontology-based data access applications show that query abduction is among the most important tools that reasoning systems have to be equipped to support debugging incomplete data. Hence, we study query abduction that aims to find explanations for the given observations (i.e. a query with answers) that together with the knowledge base will be satisfied if the explanations add to the knowledge base.

However, existing query abduction approaches either cannot scale over relative large data sets or are not guaranteed to compute all desired explanations. To the best of our knowledge, no existing approach can efficiently and completely explain conjunctive queries (CQs) over lightweight DL ontologies. Therefore, the first aim of this thesis is to propose such a query abduction approach. We achieved this aim through ontology approximation and query rewriting. The present study makes several significant contributions to query abduction for general CQs in lightweight DL ontologies. First, our query abduction approach is sound and complete that supports both general CQs as observations and ontologies with genuine existential rules as background knowledge. Moreover, we introduced a novel method for module extraction in query abduction, by applying data summarisation techniques, and we developed a pruning strategy that largely enhances the efficiency of explanation generation. Finally, we implemented a prototypical system (ABEL) based on our abductive reasoning
approach. The experimental results of ABEL show scalability and practicality of our approach.

For inductive reasoning, we investigated a learning technique, called concept learning in DLs. Concept learning aims to drive new concept definitions from given example data of an ontology. One of the most significant current discussions in the Semantic Web is how to (semi-)automatically construct ontology from structured or semi-structured data. This construction is an onerous task even for knowledge engineers. Furthermore, the new added information may have diverse presentations among different engineers. Therefore, one can construct an ontology by discovering new concept description through concept learning.

Despite recent developments in proposing concept learning systems for DL ontologies, none of learning systems are scalable in practice. This is because of using the fully-fledged reasoners for evaluation of a concept description. Therefore, the second aim of this thesis is to propose a scalable concept learning approach over lightweight DLs ontologies. We achieved this aim by developing several novel techniques. First, an intelligent initialisation of the search was proposed to prune the search space of concept descriptions effectively. Second, a sound and complete instance checker for concept learning is employed through ontology approximation to be scalable in practice. Finally, we implemented a new prototypical learning system (CL-EL) based on our inductive reasoning approach. We conducted experimental evaluations on CL-EL over different concept learning problems. The results show that CL-EL performs well and handles lightweight ontologies with large data sets.
I, Mahsa Chitsaz, declare that this thesis titled, ‘Approaches to Abductive and Inductive Reasoning in Lightweight Ontologies’ and the work presented in it are my own. I confirm that:

- 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: 
“We can achieve happiness only then when we have a beauty; and we have a beauty thanks to philosophy. The truth is that only because of philosophy we can achieve happiness.”

-Alpharabius
Acknowledgements

I thank my principal supervisor, Professor Kewen Wang, for granting me the freedom to develop and pursue my research ideas, for constructive feedback about my ideas, for scholarly guidance and support throughout my PhD candidature, and for providing funding for study. I also want to thank Dr Zhe Wang my associate supervisor, with whom I had many interesting discussions which greatly directed me when I was in a general muddle, and who inspired me with suggestions for my work during the past two years. I sincerely hope that we will find opportunities to work together again in the future. I would like to thank Professor Michael Blumenstein, my associate supervisor for supporting me in early stage of my PhD study.

I want to thank all my friends in my research group for always being supportive and have been glad to be part of this group. I thank Yifan, Akshay, Marco, Sebastian, and Melissa. I thank other researchers that I have met along my PhD journey, from students to experienced professors that have offered encouragement, helpful advice, or welcomed a chat. Specifically, I thank Dr Guilin Qi and Dr Jianfeng Du for providing thoughtful feedback on our research papers. I also thank Prof. David Warren for the insightful communication regarding abductive reasoning and XSB.

I thank the Griffith University for study and travel funding. I thank GU staff for their kind support and generous help during my PhD career. Specifically, I thank Natalie Dunstan for being always friendly to my endless questions.

Finally and most importantly, I am grateful to my parents, Mehran and Mohsen for enriching my life beyond my scientific endeavours. Words fail me to express my gratitude to my husband Hadi for his love, constant support, a great deal of patience and comforting me by his own subtle blend of wit and charm in this stressful period.
List of Publications


Contents

Abstract iii
Statement of Originality v
Acknowledgements vii
List of Publications viii
Contents ix
List of Figures xi
List of Tables xiii

1 Introduction 1
   1.1 Ontology 1
   1.2 Ontology Languages and Description Logics 2
   1.3 What is Abductive and Inductive Reasoning? 5
       1.3.1 Abductive Reasoning in DLs 6
       1.3.2 Inductive Reasoning in DLs 7
       1.3.3 Why lightweight DL ontologies? 9
   1.4 Aims of the Study and Research Objectives 10
   1.5 Thesis Structure 11

2 Preliminaries and State of the Art 13
   2.1 Ontology Languages 13
       2.1.1 OWL 2 EL 17
   2.2 First-Order Logic 18
   2.3 Datalog Family 22
   2.4 Conjunctive Queries 23
   2.5 Query Answering in Lightweight Description Logics 25
       2.5.1 Query Rewriting Approach 26
       2.5.2 Combined Approach 27
   2.6 Query Abduction 28
### Contents

2.7 Concept Learning ........................................... 31

3 Query Abduction ............................................. 37
3.1 Query Abduction Problem .................................. 39
3.2 Ontology Transformations .................................. 41
3.3 Observation Rewriting ..................................... 47

4 Optimisations and Implementations ......................... 59
4.1 Optimisations ................................................ 60
  4.1.1 Module Extraction ....................................... 60
  4.1.2 Solution Search with Pruning .......................... 65
4.2 Implementation .............................................. 69
4.3 Experimental Results ....................................... 72
  4.3.1 Test Cases ............................................... 73
  4.3.2 Comparison ............................................. 75
  4.3.3 Scalability Test ......................................... 76

5 Concept Learning for $\mathcal{ELH}^\bot$ Ontology .......... 81
5.1 Refinement Operator ....................................... 83
5.2 CL-EL (Concept Learning for EL) .......................... 92
  5.2.1 Initialising CL-EL ....................................... 95
  5.2.2 CL-EL with Trimming Procedure ...................... 99
5.3 Scalable Instance Checker .................................. 102
5.4 Experimental Results ..................................... 103
  5.4.1 Test Cases ............................................... 104
  5.4.2 Comparisons ........................................... 107
  5.4.3 Scalability Test ........................................ 110

6 Conclusions ................................................... 113
6.1 Summary of Results ...................................... 113
  6.1.1 Results on Query Abduction ......................... 113
  6.1.2 Results on Concept Learning ......................... 115
6.2 Directions for Future Work ................................ 117

Bibliography ................................................... 121
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The overview of the developed query abduction system</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>A fork-shaped model introduced by the datalog approximation.</td>
<td>46</td>
</tr>
<tr>
<td>3.3</td>
<td>A cyclic model enforced by the datalog approximation.</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>Scalability tests on generated queries.</td>
<td>79</td>
</tr>
<tr>
<td>5.1</td>
<td>The overview of the proposed concept learning approach</td>
<td>83</td>
</tr>
<tr>
<td>5.2</td>
<td>The refinement operator $\rho$</td>
<td>87</td>
</tr>
<tr>
<td>5.3</td>
<td>The set-enumeration tree of the set ${A_1, A_2, A_3}$</td>
<td>100</td>
</tr>
<tr>
<td>5.4</td>
<td>Description of trains in Michalski’s trains problem [1]</td>
<td>105</td>
</tr>
<tr>
<td>5.5</td>
<td>The scalability test on LSTW($n$) ontologies.</td>
<td>110</td>
</tr>
<tr>
<td>5.6</td>
<td>Scalability test on LSTW ontology by increasing example sets and ABox sizes.</td>
<td>111</td>
</tr>
</tbody>
</table>
# List of Tables

1.1 The main objectives of this thesis and its references .......................... 11
2.1 Knowledge base axioms for description logics ................................. 15
3.1 Transformation from $\mathcal{ELH}_\bot$ to datalog$^\pm$ .................... 42
3.2 Transformation from datalog$^\pm$ to datalog. ................................. 43
4.1 Ontology statistics. .............................................................. 74
4.2 Compare DuQAP and ABELs. .................................................. 76
4.3 Scalability tests on standard queries. ......................................... 78
5.1 Ontology statistics. .............................................................. 104
5.2 Compare CL-EL and DL-Learner ELTL. ...................................... 108
To my beloved family...
Chapter 1

Introduction

“The genius of culture is to create an ontological system so compelling that what is inside and outside of a person are viewed as of a piece, no seams and patches noticeable.”

–Richard Shweder

1.1 Ontology

The term ontology originates in philosophy, a device used by Aristotle to classify things in the world [2]. More recently, ontology has been widely used in the Artificial Intelligence research community for different purposes such as intelligent information integration, information retrieval on the Internet, and knowledge management. For instance, ontologies have been applied to a wide range of practical domains such as e-science, e-commerce, bio-informatics, and the Semantic Web [3]. Ontology as a formal model of some domain knowledge of the world provides a shared vocabulary relevant to the domain, and specifies the formalisation of the domain knowledge as well as the meaning (semantics)
of the formalisation. In other words, an ontology provides a formal and un-
ambiguous representation of a particular domain of interest [4]. The ontology
is helpful to describe different concepts and their relationships to a domain.
This provides a mechanism to enable machines to process information.

Ontologies have been employed on the Semantic Web very effectively. The
Semantic Web is a web of data, also known as a web of linked data. It includes
standards for integration and combination of the data from diverse sources
found on the web. It also contains standard languages to record the data
relationships on real world objects. Typically, current web documents are
provided for humans capable of using the web to carry out different tasks. For
instance, humans can access the latest news about a global crisis, reserve a
hotel room with their criteria or search for a car’s specifications using search
criteria. However, machines cannot automatically accomplish all of these tasks.
The vision for the Semantic Web is to make the information on the web pages
readable for machines so that they can perform more complex tasks involved
in finding, combining, and reasoning on the web.

The current Semantic Web technologies provide the possibility for applications
to represent, query, and reason on the web data using ontologies. In recent
years, different ontology languages have been introduced in the Semantic Web
communities. These languages encapsulate the expressive power of ontologies
while allowing effective reasoning.

1.2 Ontology Languages and Description Logics

The first major step towards introducing a machine-processable ontology lan-
guage came in early 1999 when the initial Resource Description Framework
(RDF) became a World Wide Web Consortium (W3C) [5] recommendation. RDF provides a conceptual model to present data in the form of subject-predicate-object expressions known in RDF terminology as triples. Subsequently, the RDF Schema (RDFS) [6] was published as a W3C recommendation in early 2004. The core idea was to extend the semantic of RDF and allow for semantics to attach user-defined classes and properties.

The *Web Ontology Language* (OWL) [7] has become the standard ontology language and has been accepted as the W3C recommendation by improving the expressiveness power of RDFS. In 2009, OWL has evolved into a new standard OWL 2 [8] by W3C OWL Working Group. The OWL 2 standard provides different profiles that trade some expressive power for efficiency of reasoning. OWL 2 consists of three ontology language profiles:

- **OWL 2 EL** is based on the *EL* family [9]. It is suitable for applications employing ontologies that contain very large numbers of properties and classes. Efficient classification is its most important reasoning task.

- **OWL 2 QL** is based on the DL-Lite family [10]. It is aimed at applications that use very large volumes of data (might be stored in relational databases). Query answering is its most important reasoning task.

- **OWL 2 RL** is based on *Description Logic Programs (DLP)* [11] and pD* [12]. It is designed for applications that trade the full expressibility in order to have an efficient reasoning using rule-based technologies.

There are many *ontology editors* that help the user to manually construct or explore the ontology including Fluent Editor [13], NeOn Toolkit [14], Protégé [15], etc.

The OWL and its latest version OWL 2 are based on *description logics (DLs)*, where an ontology is formalised as a DL knowledge base. DL is a decidable
fragment of first-order logics. In DLs, knowledge is represented by defining the concept description of a knowledge domain, and then using these concepts to specify properties of objects and individuals. A DL ontology (or simply an ontology) consists of terminological knowledge (or schema information) represented in a TBox, and membership assertions (or data information) stored in an ABox. Lightweight DLs are DL-Lite family and $\mathcal{EL}$ family of DLs underlying the ontology language profiles of OWL2 QL and OWL2 EL, respectively. Reasoning can be performed on a knowledge base to derive both new TBox and ABox knowledge using DL reasoners. Some examples of DL reasoners are Pellet [16], Hermit [17], Racer [18], CEL [19], and ELK [20], where the last two specifically designed for lightweight DLs.

Currently, the DL reasoners are equipped with standard reasoning tasks to infer automatically implicit knowledge from the knowledge that is explicitly contained in TBox and ABox. The standard reasoning tasks which are supported by DL reasoners are as follows.

- **Satisfiability**: whether a concept description is non-contradictory (or satisfiable). Consistency of a knowledge base can be defined by satisfiability, i.e. if all concept descriptions in a knowledge base are satisfiable, the knowledge base is consistent.

- **Subsumption**: whether one concept description is more general than another one, i.e. the first subsumes the second.

- **Instance checking**: whether an individual is an instance of a concept in the given ontology.

- **Query answering**: finding all answers of a query that represented in first-order logic and its vocabulary is shared with the ontology.
Along with the vast use of DL ontologies, non-standard reasoning tasks have started to emerge. This is inevitable because users of ontology demanded reasoning tasks that cannot perform by above-mentioned tasks such as abductive and inductive reasoning. These two major non-standard reasoning tasks are explained in the following section.

1.3 What is Abductive and Inductive Reasoning?

Abductive reasoning [21] is the process of inferring the best explanation from given observations according to a background knowledge. A conclusion derived by abductive reasoning is an intelligent guess because it may come from an incomplete body of observation.

Inductive reasoning [22] is the process of deriving general logical theory from the given conclusions. An inference derived by inductive reasoning has a predictive power which can make predictions for similar cases in the future.

In summary, both abductive and inductive reasoning start from a background knowledge and given logical conclusions. The aim then is to find a logical theory that, together with the background knowledge, entails the logical conclusions. However, in description logics, the logical theory that abductive reasoning produces is often in a form of membership assertions. In inductive reasoning, the logical theory is the form of logical axioms. Therefore, in the following sections, we introduce two typical cases of abductive and inductive reasoning tasks in DLs. Subsequently, the theoretical gaps in the literature are discussed.
1.3.1 Abductive Reasoning in DLs

The DL reasoners provide a wide variety of standard reasoning tasks which often involve complicated chains of reasoning steps. Interpreting the results can be confusing even for knowledge engineer in description logics. This is often the case when application domains are unfamiliar or when the chain of reasoning steps are long. Additionally, non-expert users may require explanation tools to understand those chain of reasoning even though it may appear simple to experts. These two scenarios raise a requirement for current reasoning systems to be equipped with an automatic explanation facility. Therefore, in Chapter 3 and Chapter 4, we study one important explanation problem for DL ontologies, referred to as query abduction. Query abduction is formalised as abductive reasoning and it aims to find explanations for the given observations (i.e. queries with provided answers) where are not derived by the knowledge base. If such explanations add to the knowledge base, the observations will be satisfied.

As an example of query abduction, suppose we have a TBox that consists of this knowledge:

\[
\begin{align*}
\text{‘A postgraduate student is a student’} & \quad \leadsto \quad \text{Postgrad} \sqsubseteq \text{Student} \\
\text{‘An undergraduate student is a student’} & \quad \leadsto \quad \text{Undergrad} \sqsubseteq \text{Student}
\end{align*}
\]

where the right part shows the statement in DL syntax. There are two possible explanations for the observation ‘Is Tom a student?’ (i.e. Student(Tom)), namely ‘Tom is a postgraduate student’ (i.e. Postgrad(Tom)) and ‘Tom is an undergraduate student’ (i.e. Undergrad(Tom)).

What are the theoretical gap in current query abduction approaches?
Existing approaches to query abduction in DLs vary according to their underlying DLs and types of queries. For instance, the DL languages in the work by Mcguinness et al. [23, 24] are based on $\mathcal{ALCN}$, $\mathcal{ALC}$ DL language is the base of the work by Klarman et al. [25], Halland and Britz [26], and Ma et al. [27]. Finally, the work proposed by Borgida et al. [28] and Calvanese et al. [29] are based on DL-Lite, while Du et al. [30] used $\mathcal{SHIOQ}$ as the base DL language. Regarding queries used for observations, both atomic queries and conjunctive queries (CQs) are considered in the literature.

Some approaches [23, 24, 31, 32] are proposed for query abduction in DL ontologies without considering the ABox where it is not desirable in this thesis. Others [28, 29] have considered the computational complexity of the query abduction in lightweight DLs, where an efficient algorithm is often missing. Some researchers [25, 26, 27] have investigated tableau-based algorithms for abduction problem. However, algorithms such as the tableau-based ones are not efficient enough to deal with large ABox of realistic ontologies. As well, it might not terminate in some cases. To find explanations of a query abduction problem, Du et al. [33] proposed a scalable system but incomplete for computing explanations of general observations (which we consider in this thesis) over $\mathcal{ELH}_\bot$ ontologies. Hence, in Chapter 3 and Chapter 4, we develop a new query abduction approach that is sound and complete for supporting both general CQs and $\mathcal{ELH}_\bot$ ontologies.

### 1.3.2 Inductive Reasoning in DLs

Ontology languages are intrinsic tools which help to construct our knowledge of a particular domain in a machine-readable language. However, manually constructing an ontology is an onerous task even for knowledge engineers considering the fact that there are many different concepts and properties in the ontology. To maximise the usability of an ontology, at least two fundamental
challenges have to be addressed: (1) building tools which construct ontologies (semi-)automatically with the high performance on very large-scale data sets (ontology construction), (2) developing frameworks to enrich the schema of an ontology (ontology enrichment).

In this thesis, we investigate concept learning in DLs to address these two challenges, using techniques analogous to those in inductive logic program [22]. Concept learning aims to find a concept description for given example sets, which are sets of individuals occurring in the ontology, that describes the example sets. Therefore, concept learning approaches can be employed for ontology construction or enrichment by discovering new concept description from an ontology.

As an example of concept learning, suppose we have an ontology that describes the family relationships as follows:

\[
\begin{align*}
\text{‘A female is a human’} & \implies \text{Female } \sqsubseteq \text{ Human} \\
\text{‘Mary is a female’} & \implies \text{Female(Mary)} \\
\text{‘John is a human’} & \implies \text{Human(John)} \\
\text{‘Mary has a child named John’} & \implies \text{hasChild(Mary, John)}
\end{align*}
\]

There is at least one description for the concept of mother by the given example of \{Mary\} that is \text{Female } \sqcap \exists \text{hasChild.Human} which means that mother is a female and has at least one human child.

**What are theoretical gaps in current concept learning approaches?**

Currently, most of the approaches to concept learning for DLs are an extension of inductive logic programming (ILP) methods. For instance, there is little research on concept learning in DLs [34] that transfer DL axioms to logic
programs, which then apply the ILP method in order to learn a concept. This approach is too expensive in terms of computation time. Another approach to address the concept learning problem in DLs is by employing a Machine Learning approach such as Genetic Programming [35] and kernels [36]. Experimental results of these approaches show that longer concept descriptions are generated compared than in ILP based methods. In the area of concept learning in DLs, promising research [37, 38, 39] has been conducted. All of these approaches have been proposed for expressive DLs such as $\mathcal{ALC}$. However, none of these are scalable to work with real world ontologies.

In terms of learning a concept description in lightweight DLs, research [40, 41, 42, 43] is limited. Some approaches [43] investigated the computational complexity of learning problem without proposing any practical approach. The rest of the approaches are not generally scalable in practice but still might be efficient for small ontologies. Therefore, in Chapter 5 a scalable concept learning framework is proposed for lightweight DL ontologies.

1.3.3 Why lightweight DL ontologies?

The need to equip the current DL reasoners with more sophisticated tools to provide explanations for queries and learning new concept descriptions is rising. Notably, developing such scalable approaches is challenging in different aspects.

First, the current approaches of query abduction and concept learning are mostly focused on expressive DLs, where it is not possible to have a scalable system. This is mainly because the underlying reasoners are not scalable. One natural option for obtaining scalable systems is to use a tractable ontology language. $\mathcal{ELH}_\bot$ is attractive due to at least three key facts: (1) it is a lightweight description logic, in particular, the classification is tractable even
in the presence of general concept inclusion axioms [44]; (2) many practical ontologies can be represented in $\mathcal{ELH}_\perp$ such as SNOMED CT [45], the Gene ontology [46], and large parts of GALEN [47], and (3) several efficient reasoners for $\mathcal{ELH}_\perp$ ontologies are available, e.g. CEL [19], ELK reasoner [48], and Snorocket [49].

Another major challenge in both query abductions and concept learning is the requirement that ABox reasoning is necessary to find a solution whereby both TBox and ABox have to be taken into account. However, the dedicated reasoners of lightweight DLs do not fully support ABox reasoning. Furthermore, the fully-fledged reasoners are not scalable enough for lightweight DL ontologies. To overcome these limitations, we approximated the OWL 2 EL ontology to OWL 2 RL, then we used an RL reasoner.

Hence, the focus of this study is on lightweight DL ontology, mainly $\mathcal{ELH}_\perp$ ontology, that involves facilitation of scalable abductive and inductive reasoning systems.

### 1.4 Aims of the Study and Research Objectives

With respect to the afore-mentioned challenges and theoretical gaps in the literature, the main aims of this thesis are to propose scalable approaches on query abduction and concept learning for $\mathcal{ELH}_\perp$ ontologies. Subsequently, the research questions addressed in this thesis are:

1. How to develop a scalable and complete query abduction approach for general conjunctive queries as observations that handles practical $\mathcal{ELH}_\perp$ ontologies?
2. How to develop a scalable concept learning approach over $\mathcal{ELH}_\bot$ ontologies with large data sets?

To propose scalable approaches, both theoretical and pragmatic aspects of the approaches have to be investigated. A summary of the specific objectives of this thesis is listed in Table 1.1 to answer both research questions of this thesis.

**Table 1.1: The main objectives of this thesis and its references**

<table>
<thead>
<tr>
<th>Objective</th>
<th>addressed in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Abduction in $\mathcal{ELH}_\bot$</td>
<td></td>
</tr>
<tr>
<td>to develop and implement a sound and complete query abduction algorithm for conjunctive queries over $\mathcal{ELH}_\bot$ ontologies</td>
<td>Chapter 3</td>
</tr>
<tr>
<td>to develop different optimisations for the benefit of query abduction</td>
<td>Section 4.1</td>
</tr>
<tr>
<td>to implement our approach using a resolution-based system</td>
<td>Section 4.2</td>
</tr>
<tr>
<td>to compare our implemented query abduction system (ABEL) with the current abduction system</td>
<td>Section 4.3.2</td>
</tr>
<tr>
<td>to show the scalability of ABEL over large $\mathcal{ELH}_\bot$ ontologies</td>
<td>Sections 4.3.3</td>
</tr>
<tr>
<td>Concept Learning in $\mathcal{ELH}_\bot$</td>
<td></td>
</tr>
<tr>
<td>to develop a refinement operator for concept learning in $\mathcal{ELH}_\bot$</td>
<td>Section 5.1</td>
</tr>
<tr>
<td>to develop and implement a concept learning algorithm for $\mathcal{ELH}_\bot$ ontologies</td>
<td>Section 5.2</td>
</tr>
<tr>
<td>to employ a scalable instance checker for the benefit of concept learning in $\mathcal{ELH}_\bot$</td>
<td>Section 5.3</td>
</tr>
<tr>
<td>to compare our implemented concept learning system (CL-EL) with the current available learning systems for DL ontologies</td>
<td>Section 5.4.2</td>
</tr>
<tr>
<td>to show the scalability of CL-EL over large $\mathcal{ELH}_\bot$ ontologies</td>
<td>Sections 5.4.3</td>
</tr>
</tbody>
</table>

### 1.5 Thesis Structure

Chapter 2 introduces the preliminaries necessary to understand the content of this thesis. It covers ontology languages such as description logics, datalog
language and part of first-order logics which are needed for the proofs in the following chapters. Besides an overview of current work in the query abduction and the concept learning for the $\mathcal{ELH}_\perp$ ontologies, we provide formal definitions of these problems in the previous work.

A basic approach of query abduction is presented in Chapter 3. The details of the approach which involves different transformations and query rewriting are given. Additionally, all proofs of the soundness and the completeness of the proposed approach for query abduction supporting conjunctive queries and ontologies containing existential rules are provided.

Based on our approach, a query abduction system (ABEL) for $\mathcal{ELH}_\perp$ is proposed in Chapter 4. The system involves different optimisations to allow it to be scalable and efficient. In Section 4.3, the experimental results of ABEL is provided which includes its comparison with the current state of the art system and its scalability test. Therefore, Chapter 3 and Chapter 4 address the research question 1.

In Chapter 5, the concept learning system (CL-EL) is described to address the research question 2. First, we start with the definition of a new refinement operator and proofs of its completeness and local finiteness. Then, we explain how to do a heuristic search among all possible concept description of $\mathcal{ELH}_\perp$ in solving the concept learning problem. Subsequently, a data summarisation approach is explained which is used in initialisation of the learning algorithm. Furthermore, we describe our instance checker which used a datalog reasoner in order to handle the real-world ontologies. Finally, we report the experimental results of CL-EL.

Finally, some conclusions are drawn and further enhancements to this thesis are mentioned in Chapter 6.
Chapter 2

Preliminaries and State of the Art

“It really is worth the trouble to invent a new symbol if we can thus remove not a few logical difficulties and ensure the rigour of the proofs.”

– Gottlob Frege

In this chapter, we introduce all necessary preliminary definitions for further references. Additionally we present the state of the art of topics such as, but not limited to, query abduction and concept learning.

2.1 Ontology Languages

The W3C ontology working group\(^1\), by introducing ontology languages, provides the possibility for applications to reason and query over ontologies. In the past decades, different ontology languages which encapsulated the expressive

\(^1\)http://www.w3.org/2001/sw/WebOnt/
power of ontologies while allowing effective reasoning were introduced. The Web Ontology Language (OWL)\(^2\) is standard for ontology language recommended by the W3C and its latest version is OWL 2. The formal foundation is description logics (DLs). OWL 2 standard provides different profiles, namely OWL 2 EL, OWL 2 QL, and OWL 2 RL, that trade some expressive power for efficiency of reasoning.

Description logic [50] is a family of logic-based knowledge representation formalisms that represents knowledge by defining the concept description of the knowledge domain, and then using these concepts to specify properties of objects. In particular, the knowledge is represented in terms of objects (or constants, or individuals), concepts (or unary predicate), and roles (or binary predicate, or properties). Concepts describe a class of information in an application domain, e.g. the concept of father can be represented as “a man has at least a child” (\(Male \sqcap \exists hasChild.\top\) in DLs). A concept is interpreted as a set of objects while roles are binary relations between objects. Usually, an ontology in description logics formalism consists of a terminology box, TBox, which represents the relationship among concepts and roles in the ontology and an assertion box, ABox which stores the instances of the represented concepts and roles. An axiom is a logical sentence in either TBox or ABox of an ontology.

We consider countably infinite and mutually disjoint sets \(N_C, N_R, N_I,\) and \(N_V\) of concept names, role names, objects, and variables respectively. A DL knowledge base \(\mathcal{K} = \mathcal{T} \cup \mathcal{A}\) consists of a set \(\mathcal{T}\) of terminological axioms (TBox), and a set \(\mathcal{A}\) of assertional axioms (ABox). We introduce the syntax and semantics of a general DL, which is a proper fragment of OWL and is shown in Table 2.1. Here \(a\) and \(b\) denote object names, \(r\) denotes role name, \(C\) and \(D\) denote concept descriptions.

\(^2\)http://www.w3.org/TR/owl-features/
Table 2.1: Knowledge base axioms for description logics

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>⊤</td>
<td>⊤Δ^2</td>
</tr>
<tr>
<td>bottom</td>
<td>⊥</td>
<td>⊥∅</td>
</tr>
<tr>
<td>atomic concept</td>
<td>A</td>
<td>A ⊆ Δ^2</td>
</tr>
<tr>
<td>atomic role</td>
<td>r</td>
<td>r^I ⊆ Δ^2 × Δ^2</td>
</tr>
<tr>
<td>nominal</td>
<td>{a}</td>
<td>{a^I} ⊆ Δ^2,</td>
</tr>
<tr>
<td>inverse role</td>
<td>r⁻</td>
<td>(r⁻)^I = {(a,b)</td>
</tr>
<tr>
<td>disjunction</td>
<td>C ⊔ D</td>
<td>C^I ⊔ D^I</td>
</tr>
<tr>
<td>conjunction</td>
<td>C ∩ D</td>
<td>C^I ∩ D^I</td>
</tr>
<tr>
<td>negation</td>
<td>¬C</td>
<td>Δ^I \ C^I</td>
</tr>
<tr>
<td>value restriction</td>
<td>∀r.C</td>
<td>{a ∈ Δ^I</td>
</tr>
<tr>
<td>existential restriction</td>
<td>∃r.C</td>
<td>{a ∈ Δ^I</td>
</tr>
<tr>
<td>concept assertion</td>
<td>C(a)</td>
<td>a^I ∈ C^I</td>
</tr>
<tr>
<td>role assertion</td>
<td>r(a,b)</td>
<td>(a^I,b^I) ∈ r^I</td>
</tr>
<tr>
<td>GCI</td>
<td>C ⊑ D</td>
<td>C^I ⊆ D^I</td>
</tr>
<tr>
<td>RI</td>
<td>r_1 ◦ ... ◦ r_k ⊑ r</td>
<td>r_1^I ◦ ... ◦ r_k^I ⊑ r^I</td>
</tr>
<tr>
<td>domain restriction</td>
<td>domain(r) ⊆ C</td>
<td>(r)^I ⊆ C^I × Δ^2</td>
</tr>
<tr>
<td>range restriction</td>
<td>range(r) ⊆ D</td>
<td>(r)^I ⊆ Δ^2 × C^I</td>
</tr>
</tbody>
</table>

In general, a TBox includes axioms of the form C ⊑ D, is called general concept inclusion (GCI), and r_1 ⊑ r_2, is called role inclusion (RI), where C and D are concepts and r_1 and r_2 are role names [50]. A TBox can contain equivalence axioms, C ≡ D, that is an abbreviation for C ⊑ D and D ⊑ C. Additionally, C ⊑ D is equivalent to C ⊆ D and D ⊈ C. And C ⊑ T is an abbreviation of T |= C ⊆ D. Note that RI generalizes three important means of expressivity in ontology applications: role hierarchies r ⊑ s, transitive roles, which can be expressed by writing r ◦ r ⊑ r and right-identity rules r ◦ s ⊑ s. Moreover, the bottom concept in combination with GCI can be used to express disjointness of complex concept descriptions C ⊓ D ⊑ ⊥ says that C, D are disjoint. An ABox is a finite set of concept and role assertions shown in Table 2.1. The following example shows a DL knowledge base.
Example 2.1. Consider the DL knowledge base whose TBox and ABox are specified in Table 2.1.

<table>
<thead>
<tr>
<th>TBox</th>
<th>ABox</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1) Male ⊑ Person</td>
<td>A1) Father(Christopher)</td>
</tr>
<tr>
<td>T2) hasSon ⊑ hasChild</td>
<td>A2) hasChild(Christopher, James)</td>
</tr>
<tr>
<td>T3) ∃hasSon ∼ ⊑ Male</td>
<td>A3) hasChild(Victoria, James)</td>
</tr>
<tr>
<td>T4) Parent ≡ ∃hasChild.⊤</td>
<td>A4) hasChild(James, Marry)</td>
</tr>
<tr>
<td>T5) Father ≡ Male ∩ Parent</td>
<td>A5) HappyFather(James)</td>
</tr>
</tbody>
</table>

In the above knowledge base, Male, Person, Parent and Father are concept names. hasSon and hasChild are role names, and Christopher, Victoria, James, and Marry are objects. The axiom T1 says that a Male is a Person, and T2 expresses that x has a son y, then x has a child y. The axiom T3 shows that if there exists someone has a son, the son is a Male. The axiom T4 says that a Parent is someone who has a child and vice versa. T5 expresses that a Father is a Male and a Parent, and vice versa. In the ABox, the sentence ‘Christopher is a Father’ is represented by A1. A2 says that ‘Christopher has James as a child’. A3 and A4 can be expressed similarly to A2. Finally, A1 represents the statement ‘James is a happy father (HappyFather)’.

The semantics of description logic is defined in terms of an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty domain of $\mathcal{I}$, and $\cdot^\mathcal{I}$ is a function mapping each concept name $A$ (resp. role name $r$) to a subset $A^\mathcal{I}$ (resp. a subset $r^\mathcal{I}$) of $\Delta^\mathcal{I}$ (resp. $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$), and each individual name $a$ to $a^\mathcal{I} \in \Delta^\mathcal{I}$ [50]. The interpretation function can be extended to arbitrary concept and role descriptions as shown in right part of Table 2.1.

A concept description $C$ is satisfiable if there exists an interpretation $\mathcal{I}$ such that $C^\mathcal{I} \neq \emptyset$. We say $C$ is subsumed by $D$, denoted as $C \sqsubseteq D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all interpretations $\mathcal{I}$. 
An interpretation $\mathcal{I}$ is a model of a TBox, denoted by $\mathcal{I} \models T$, (resp. an ABox, denoted by $\mathcal{I} \models A$) if $\mathcal{I}$ satisfies each axiom in $T$ (resp., each fact in $A$). An interpretation $\mathcal{I}$ is a model of a knowledge base $\mathcal{K} = T \cup A$, $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I} \models T$ and $\mathcal{I} \models A$. We use $\text{mod}(\mathcal{K})$ (respectively, $\text{mod}(T)$ and $\text{mod}(A)$) to denote the set of all models of $\mathcal{K}$ (respectively, $T$ and $A$).

A knowledge base $\mathcal{K}$ is called consistent if it has at least one model, such that \( \text{mod}(\mathcal{K}) \neq \emptyset \). $\mathcal{K}$ is coherent if there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that, for each concept name $A$ and role name $r$ occurring in $\mathcal{K}$, $A^\mathcal{I} \neq \emptyset$ and $r^\mathcal{I} \neq \emptyset$.

The standard reasoning tasks for a knowledge base $\mathcal{K}$ are as follows.

- **Consistency**: Check whether $\mathcal{K}$ is consistent.
- **Subsumption**: Given two concepts $C$ and $D$, check whether $\mathcal{K} \models C \sqsubseteq D$.
- **Instance Checking**: Given a concept assertion $C(a)$ or a role assertion $r(a, b)$. Check whether $\mathcal{K} \models C(a)$ or $\mathcal{K} \models r(a, b)$.

### 2.1.1 OWL 2 EL

The OWL 2 EL profile is designed as a profile of OWL 2 which is formulated in $\mathcal{EL}^{++}$ [51]. DL $\mathcal{EL}^{++}$ is a tractable fragment of DLs that allows powerful reasoning technologies to be applied. It is particularly suitable for applications employing ontologies that define very large numbers of classes and properties.

We introduce $\mathcal{ELH}_\bot$ as a fragment of the $\mathcal{EL}^{++}$ family in this section. In $\mathcal{ELH}_\bot$, a concept description (or simply concept) $C$ is defined inductively using the following constructors:

$$
C ::= \top \mid \bot \mid A \mid C_1 \cap C_2 \mid \exists r.C,
$$

where $A \in N_C$, $r \in N_R$, and $C_1$ and $C_2$ are concepts. An $\mathcal{ELH}_\bot$ TBox includes axioms of the form $C \sqsubseteq D$, and an $\mathcal{ELH}_\bot$ ABox includes concept assertions...
$A(a)$ and role assertions $r(a, b)$, where $C$ and $D$ are concepts, $A$ is an atomic concept, $r$ is a role name, and $a$ and $b$ are objects. An $\mathcal{ELH}_\perp$ knowledge base (KB) contains an $\mathcal{ELH}_\perp$ TBox and an $\mathcal{ELH}_\perp$ ABox. If a concept description $C$ is expressible by combinations of the above-mentioned constructors, then $C$ is an $\mathcal{ELH}_\perp$-concept (description).

An $\mathcal{ELH}_\perp$ TBox is normalized if it consists of only axioms of the forms $A \sqsubseteq B$, $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq A$, $A_1 \sqsubseteq \exists r.A$, $\exists r.A_1 \sqsubseteq A$, and $r \sqsubseteq s$, where $A(i) \in N_C \cup \{ \top \}$, $B \in N_C \cup \{ \top, \bot \}$, and $r$ and $s$ are in $N_R$. An $\mathcal{ELH}_\perp$ TBox can be transformed in polynomial time to a normalized TBox that is equivalent to the TBox [9]. Without loss of generality, we consider only TBoxes in normal form.

### 2.2 First-Order Logic

Description logics are decidable fragments of first-order logic (FOL) [52], therefore it is possible to transform a DL ontology to an equivalent FOL program. Then, we can benefit from basic characteristics of semantics and reasoning mechanism in FOL. Thus, in this section, we present the syntax and semantics of first-order logic in addition to basics of proof procedures where can be further referenced.

We use the standard notion of quantifiers as $\exists$ (there exists, the existential quantifier) and $\forall$ (for all, the universal quantifiers) and connectives as $\neg$ (negation), $\land$ (and), $\lor$ (or), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence).

A first-order language is a tuple $L(P, F, V)$, where $P$ is a non-empty and finite set of predicate symbols, $F$ is a finite set of function symbols, and $V$ is a countable set of variables. Each predicate (resp. function) symbol has a natural number, called its arity, assigned to it which this is the number of arguments the predicate (resp. function) has. So, we say $P \in P \ (f \in F)$ is an
n-ary predicate (function) symbol if $P$ (resp. $f$) has $n$ arguments. A constant is a 0-ary function symbol.

A term is a constant, a variable, or an $n$-ary function $f(t_1, t_2, \cdots, t_n)$ where $t_1, t_2, \cdots, t_n$ are terms. A ground term is a term which does not contain any variables. Using terms, we can construct formulas. An atom, $P(t_1, t_2, \cdots, t_n)$, is the smallest possible formula which constructed by placing the $n$ arguments of an $n$-ary predicate symbol with $n$ terms, also called atomic formula. If $\phi$ is a formula, then $\neg\phi$, $\phi \circ \psi$, $\exists x.\phi$, and $\forall x.\phi$ are formulas, where $\circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}$, and $x$ is a variable.

For a formula $\exists x.\phi$ or $\forall x.\phi$, we denote that $x$ is a the bound variable. An occurrence of a variable which is not bound, is called free. A closed formula is a formula with no free variable occurrences.

A predicate with arity 1 and 2 called unary and binary predicate, respectively. For a (set of) formula(e) $\phi$, $\text{pred}(\phi)$, $\text{term}(\phi)$, $\text{const}(\phi)$ denote the set of predicates, terms, and constants occurring in $\phi$.

For an FOL $L(P, F, V)$, let $T$ be the set of terms occurring in $L$. A substitution is a mapping function $\sigma : V \mapsto T$ from the set of variables $V$ to the set of terms $T$. The result of applying a substitution $\sigma$ to a term $t$, denoted by $t\sigma$, is defined recursively as follows: $c\sigma = c$, and $[f(t_1, \cdots, t_n)]\sigma = f(t_1\sigma, \cdots, t_n\sigma)$, for a constant symbol $c$, and an $n$-ary function symbol $f$. An application of a substitution $\sigma$ to a formula $\phi$, denoted by $\phi\sigma$, is defined as follows: $[P(t_1, \cdots, t_n)]\sigma = P(t_1\sigma, \cdots, t_n\sigma)$; $[-\phi]\sigma = -\phi\sigma$; $(\phi \circ \psi)\sigma = \phi\sigma \circ \psi\sigma$ for $\circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}$; $[\forall x.\phi]\sigma = \forall x.(\phi\sigma)$; and $[\exists x.\phi]\sigma = \exists x.(\phi\sigma)$.

A composition of substitutions $\sigma$ and $\tau$, denoted by $\sigma\tau$, is a substitution such that for each variable $x$, $x(\sigma\tau) = (x\sigma)\tau$. A substitution $\sigma$ is called a variable renaming if it contains only mappings of the form $x \mapsto y$, where both $x$ and $y$ are variables. A substitution $\sigma$ is equivalent to $\theta$ up to variable renaming
if there is a variable renaming $\tau$ such that $\theta = \sigma \tau$ (which is automatically $\theta$ equivalent to $\sigma$).

A substitution $\sigma$ is a unifier of terms $t$ and $s$ if $t \sigma = s \sigma$. A unifier $\sigma$ is called a most general unifier (MGU) if there is no unifier $\theta$ of terms $t$ and $s$ such that there is a substitution $\tau$, $\theta = \sigma \tau$. If there is a MGU $\sigma$ of $s$ and $t$, it is unique up to variable renaming.

The semantic of a first-order logic language $L(P, F, V)$ defines as follows. An interpretation $I$ is a pair $I = (D, .^I)$ where $D$ is a non-empty set, called domain, and .$^I$ is a mapping function that associates to every $n$-ary predicate symbol $P \in P$, some $n$-ary relation $P^I : D^n \to \{true, false\}$, as well as to every $n$-ary function symbol $f \in F$, some $n$-ary function $f^I : D^n \to D$. A variable assignment $A$ in an interpretation $I$ is a mapping from the set of variables to the set of domain $D$. We denote $v^A$ as the image of variable $v$ under an assignment $A$. The assignment $B$ is an $x$-variant of the assignment $A$, if both assignments $A$ and $B$ assign the same values to every variable except possibly $x$. To each term $t$ of language $L(P, F, V)$, we associate a value $t^I_A$ under $I$ and $A$ in the domain $D$ as follows: for a variable or a constant $x$, $x^I_A = x^A$, and for a function symbol $f$, $[f(t_1, \cdots, t_n)]^I_A = f^I(t_1^A, \cdots, t_n^A)$.

To each formula $\phi$ of $L(P, F, V)$, we associate a truth value $\phi^I_A$ under $I$ and $A$ as follows: $[\top]^I_A = true$; $[\bot]^I_A = false$, $[P(t_1, \cdots, t_n)]^I_A = true \Leftrightarrow (t_1^I, \cdots, t_n^I) \in P^I$; $[\neg \phi]^I_A = \neg[\phi]^I_A$; $[\phi \circ \psi]^I_A = \phi^I \circ \psi^I_A$; $[\forall x. \phi]^I_A = true \Leftrightarrow \phi^{I_B} = true$ for every assignment $B$ in $I$ that is an $x$-variant of $A$; and $[\exists x. \phi]^I_A = true \Leftrightarrow \phi(\vec{x})^{I_B} = true$ for some assignment $B$ in $I$ that is an $x$-variant of $A$. If $\phi$ is a ground formula, then $\phi^I_A$ does not depend on $A$, so we simply write $\phi^I$.

For a formula $\phi$, an interpretation $I$ is a model of $\phi$, denoted by $I \models \phi$, if $\phi^I_A = true$ for all assignments $A$. A formula $\phi$ is valid, denoted by $\models \phi$, if $I \models \phi$ for all interpretations $I$. A formula $\phi$ is satisfiable if $I \models \phi$ for at
least one interpretation $I$, and $\phi$ is unsatisfiable if no such interpretation $I$ exists. A formula $\phi_1$ entails a formula $\phi_2$, denoted by $\phi_1 \models \phi_2$, if $I \models \phi_2$ for each interpretation $I$ such that $I \models \phi_1$. $\phi_1$ and $\phi_2$ are equivalent, denoted by $\phi_1 \Leftrightarrow \phi_2$, if both $I \models \phi_1$ and $\phi_2 \models \phi_1$.

A literal is either an atom or a negation of an atom. A clause is a finite disjunction of zero or more literals. A formula is in conjunctive normal form (CNF) if it has the following form: $q_1x_1 \cdots q_nx_n(C_1 \land \cdots \land C_m)$, where each $q_i$ is either $\exists$ or $\forall$, $x_1, \cdots, x_n$ are all the variables occurring in the formula, and each $C_j$ is a clause. In fact, any formula $\phi$ can be transformed into an equivalent formula $\psi$, which is in CNF. Let $\phi$ be a CNF formula, then a Skolemized form of $\phi$ is a formula $\text{sk}(\phi)$ obtained by defining recursively as follows: if $q_i$ is the left-most existential quantifier in $\phi$, and $x_{i_1}, \cdots, x_{i_j}$ are the variables on the left of $x_i$, then a new $j$-ary Skolem function symbol $f_{sk}$ will be added to the language $L$, and each occurrence of $x_i$ is replaced by the term $f_{sk}(x_{i_1}, \cdots, x_{i_j})$. If the $j$ is zero, we call $f_{sk}$ as Skolem constant. So, this whole process of removing existential quantifiers is called Skolemization. It is important to note that the alphabet of $L(P,F,V)$ is extended by new Skolem function or constant was not previously in the alphabet. For example, let $\phi$ be $\exists x.\forall y.\exists z.P(x,y,z)$ then the $\text{sk}(\phi)$ is $\forall y.P(c_{sk}, y, f_{sk}(y))$, where $c_{sk}$ is a Skolem constant and $f_{sk}$ is a Skolem function added to the language. The result of Skolemization is unique up to renaming of Skolem function symbols.

Herbrand universe ($HU_L$) of $L$ is the set of all ground terms which can be constructed by constants and function symbols in $L$. The Herbrand Base ($HB_L$) of $L$ is the set of all ground atoms constructed over the predicate symbols in $L$ and the terms in $HU_L$. A Herbrand interpretation $I$ is an interpretation such that the domain $D$ is $HU_L$, and for each ground term $t$, $t^I = t$. A formula $\phi$ is satisfiable, if $\text{sk}(\phi)$ is satisfiable in a Herbrand interpretation.
2.3 Datalog Family

It is plausible to approximate a DL ontology to another formalism which has efficient reasoning tools. In our approaches, we have transformed an $\mathcal{ELH}_\bot$ ontology to a datalog program as an approximation of $\mathcal{ELH}_\bot$ ontology. Thus, in this section we present the syntax and semantics of datalog language. Furthermore, a transformation of an $\mathcal{ELH}_\bot$ ontology to $\text{datalog}^\pm$ program has been used in our approach. So, the syntax of $\text{datalog}^\pm$ is presented as follows.

Datalog [53] is a decidable fragment of first-order logic which there is no function of arity more than 0. $\text{datalog}^\pm$ [53] is developed as a family of rule languages that extend datalog with existential rules and are sufficient to capture several lightweight DLs including $\mathcal{ELH}_\bot$. A $\text{datalog}^\pm$ rule is a sentence of the form

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

where $\phi(\vec{x}, \vec{y})$ and $\psi(\vec{x}, \vec{z})$ are conjunctions of function-free atoms and $\vec{x}$, $\vec{y}$ and $\vec{z}$ are pairwise disjoint. For convenience of presentation, universal quantifiers are often omitted. Formula $\phi$ is the body and formula $\psi$ is the head of the rule. Note that a fact is a special $\text{datalog}^\pm$ rule whose head is a ground atom and whose body is empty. A $\text{datalog}^\pm$ program is a finite set of $\text{datalog}^\pm$ rules. We refer to existential rules as $\text{datalog}^\pm$ rules that $\vec{z}$ is not empty. The semantics of a $\text{datalog}^\pm$ program can be defined similar to the semantics of an FOL program presented in previous section.

A $\text{datalog}$ rule is the form of $\forall \vec{x}. \forall \vec{y}. \lbrack \phi(\vec{x}, \vec{y}) \rightarrow \psi(\vec{x}) \rbrack$ where $\phi(\vec{x}, \vec{y})$ and $\psi(\vec{x})$ are conjunctions of function-free atoms and $\vec{x}$ and $\vec{y}$ are disjoint.

A guarded rule is a $\text{datalog}^\pm$ rule which at least one of the atoms in its body has a guard atom, that has among its arguments all the body variables. A linear rule restricts $\text{datalog}^\pm$ rule bodies to contain a single atom only (which
Chapter 2. Preliminaries

is then automatically a guard rule). Guarded datalog\(^{\pm}\) is more general than \(\mathcal{ELH}_{\perp}\), but the linear datalog\(^{\pm}\) encompasses DL-Lite\(_{core}\).

It is possible to define the semantics of a datalog program by FOL semantics, however in our framework we used datalog program semantic from Logic Programming. In logic programming, the semantics of a datalog program \(\mathcal{D}\) is given by the least Herbrand model. Given a set \(\mathcal{F}\) of facts, let \(\mathcal{D}(\mathcal{F})\) be the smallest superset of \(\mathcal{F}\) such that for each rule \(\phi \rightarrow \psi\) in \(\mathcal{D}\) and each ground substitution \(\sigma\) mapping the variables occurring in the rule to the constants in \(\mathcal{F}\), if \(\sigma(\phi) \subseteq \mathcal{F}\) then \(\sigma(\psi) \subseteq \mathcal{D}(\mathcal{F})\). Let \(\mathcal{F}_0\) be the set of facts in \(\mathcal{D}\), \(\mathcal{F}_{i+1} = \mathcal{D}(\mathcal{F}_i)\) for \(i \geq 0\), and \(\mathcal{M} = \bigcup_{i \geq 0} \mathcal{F}_i\). Then, \(\mathcal{M}\) seen as a first-order interpretation is the least Herbrand model of \(\mathcal{D} \cup \mathcal{F}\).

Calì et al. [53] are shown that \(\mathcal{ELH}_{\perp}\) can be captured by datalog\(^{\pm}\) rules. In particular, given an \(\mathcal{ELH}_{\perp}\) TBox \(\mathcal{T}\), \(\mathcal{T}\) can be transformed into an datalog\(^{\pm}\) program \(\mathcal{D}_\mathcal{T}\) that is semantically equivalent to \(\mathcal{T}\) in FOL.

2.4 Conjunctive Queries

In Chapter 1, we have talked about query, conjunctive query (CQ), and instantiated query. In this section, we formally define all those terms. Subsequently, we present different approaches to find an answer set of a query in a datalog program. In the next section, we introduce such approaches for lightweight DL ontologies.

A conjunctive query \(Q(\vec{x})\) (or simply \(Q\)) is a first-order formula \(\exists y. \phi(\vec{x}, \vec{y})\) with \(\phi(\vec{x}, \vec{y})\) a conjunction of function-free atoms over \(N_C \cup N_R\). A first-order query (FO query) is a first-order formula, \(\exists y. \phi(\vec{x}, \vec{y})\), that \(\phi(\vec{x}, \vec{y})\) contains only function-free literals over the predicates of an FOL language. Variables \(\vec{x}\) are answer variables, denoted \(avar(Q)\), and \(\vec{y}\) are quantified variables, denoted
A CQ is called boolean if \( \text{avar}(Q) \) is an empty set. A CQ can also be denoted as a datalog rule \( \phi(\vec{x}, \vec{y}) \rightarrow Q(\vec{x}) \). An ground query \( Q \) is a CQ where \( \text{qvar}(Q) \) is an empty set. An atomic query is a ground query which its body has only single atom. For a tuple of constants \( \vec{a} \) with the same arity as \( \text{avar}(Q) \), we write \( Q(\vec{a}) \) to denote the Boolean CQ obtained from \( Q(\vec{x}) \) by substituting the variables \( \vec{x} \) with constants \( \vec{a} \). A tuple of constants \( \vec{a} \) is a positive answer (or simply an answer) of a CQ \( Q(\vec{x}) \) over a KB \( K = \mathcal{T} \cup \mathcal{A} \), if the arity of \( \vec{a} \) agrees with that of \( \vec{x} \) and \( K \models Q(\vec{a}) \). Otherwise, if \( K \not\models Q(\vec{a}) \) then \( \vec{a} \) is a negative answer of \( Q(\vec{x}) \) over \( K \). \( \text{ans}(Q, K) \) (we also use \( \text{ans}(Q, \mathcal{T} \cup \mathcal{A}) \)) denotes the set of answers of \( Q \) over \( K \) (or \( \mathcal{T} \) and \( \mathcal{A} \)).

Conjunctive query answering for \( \mathcal{EL}^{++} \) guaranteed to be in polynomial-time and for DL-Lite\(_{\text{core}}\) guaranteed to be \( \text{AC}_0 \) in terms of data complexity. This means that based on a fixed TBox, a fixed query can be answered in polynomial and \( \text{AC}_0 \) time over variable ABoxes for \( \mathcal{EL}^{++} \) and DL-Lite\(_{\text{core}}\) ontologies, respectively.

For a datalog program \( D \), let \( M \) be the least Herbrand model of \( D \cup F \), it is well known that for a tuple \( \vec{a}, \vec{a} \in \text{ans}(Q, D \cup F) \) if and only if \( M \) satisfies \( Q(\vec{a}) \).

Answering CQs w.r.t. a datalog program can be achieved either through a forward chaining procedure to compute the least Herbrand model, as implemented in most datalog engine, or through a backward chaining procedure based on SLD-solution, which we formally present as follows. A goal is a conjunction of atoms \( \alpha_1 \land \cdots \land \alpha_m \). The SLD-resolution rule takes as premises a goal and a datalog rule, and it produces a new goal as follows

\[
\frac{\alpha_1 \land \cdots \land \alpha_m, \ \phi \rightarrow \beta}{\sigma(\alpha_2) \land \cdots \land \sigma(\alpha_m) \land \sigma(\phi)}
\]

where \( \sigma \) is the most general unifier of \( \alpha_1 \) and \( \beta \). An SLD-proof of a goal \( G_0 \) w.r.t. a datalog program \( D \) and a set \( F \) of facts is a sequence of goals
G_0, \ldots, G_n, where G_n \subseteq \mathcal{F} and each G_{i+1} is obtained from G_i by a single SLD-resolution. SLD-resolution is sound and complete for datalog: for a datalog program \mathcal{D}, a set \mathcal{F} of fact, a CQ \mathcal{Q}(\vec{x}) = \exists \vec{y}. \phi(\vec{x}, \vec{y}) and a tuple \vec{a} of constants, 
\mathcal{D} \cup \mathcal{F} \models \mathcal{Q}(\vec{a}) iff there exists an SLD-proof of the goal \phi(\vec{a}, \vec{y}) w.r.t. \mathcal{D} and \mathcal{F}.
In this case, we call the SLD-proof (of \phi(\vec{a}, \vec{y})) an SLD-proof of \mathcal{Q}(\vec{a}) w.r.t. \mathcal{D} and \mathcal{F}. For a rule r \in \mathcal{D}, denote r \in SLD(\mathcal{Q}, \mathcal{D}, \mathcal{F}) if r is involved in an SLD-proof of \mathcal{Q} w.r.t. \mathcal{D} and \mathcal{F}; and for a fact \alpha, denote \alpha \in SLD(\mathcal{Q}, \mathcal{D}, \mathcal{F}) if there exists an SLD-proof G_0, \ldots, G_n of \mathcal{Q}(\vec{a}) such that \alpha \in G_n.

2.5 Query Answering in Lightweight Description Logics

Recently, ontology-based data access (OBDA) [54] is proposed as an architectural principle for data integration and maintenance over lightweight DL ontologies. In the OBDA setting, ontologies are used as virtual schemas over datasets to enrich query answering with ontological knowledge through logical reasoning, while making use of practical relational database management systems (RDBMS) to scale over very large datasets. Currently, there are two main approaches to find answers to a query over a lightweight DL ontology, or simply find this set \text{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})). We informally describe these approaches, but in the following sections, these will be formally defined.

The first approach is called \textit{query rewriting}, where the query is rewritten according to a TBox and independently of an ABox. This method is sound and complete for DL-Lite ontologies, but might generate exponential number of queries in rewriting phase. Also, it does not terminate for a general ELH⊥ TBox (e.g. axiom \exists R.A \sqsubseteq A is in TBox). Another approach, namely \textit{combined approach}, materialises an ABox over a TBox as an off-line phase. Then, in the query time, the query is rewritten in polynomial-time to the size of the query.
This method is also sound and complete for DL-Lite$_{core}$ ontologies, as well as $\mathcal{ELH}_\perp$ ontologies. The rewriting in this approach does not depend on either TBox or ABox, but only relates to the query itself. However, it has not yet been elucidated if such a rewriting is available for more expressive ontologies than DL-Lite and $\mathcal{ELH}_\perp$. In section 2.5.2, we introduce a different combined approach that has a simpler query rewriting phase.

2.5.1 Query Rewriting Approach

The query rewriting approach is proposed for applications with the assumption that the ABox of an ontology is very large and can be stored in a database or a data store. In that way, we can then find answers to a query that has a signature that is over the concept and role names in the ontology through rewriting the query. The approach works in two steps. First the given query $Q$ is compiled into a union of CQs denoted as $Q'$ using the TBox $\mathcal{T}$ of the ontology and independently of the ABox $\mathcal{A}$. Second $Q'$ will be evaluated over any incomplete ABox stored in a database, and its results yield exactly the same result as $Q$ evaluated against $\mathcal{T}$ and $\mathcal{A}$. We formally define the notion of first-order rewritable ontology as follows.

**Definition 2.1.** For a TBox $\mathcal{T}$, $\mathcal{T}$ is first-order rewritable (FO-rewritable) if for each CQ $Q$, $Q$ and $\mathcal{T}$ can be compiled into a first-order query $Q'$ such that for each ABox $\mathcal{A}$, $\mathcal{T} \cup \mathcal{A} \models Q$ iff $\mathcal{A} \models Q'$.

If an ontology has the property of FO-rewritable, then answering queries is in the class AC$_0$ in data complexity, since query answering over first-order queries is in AC$_0$ in data complexity [54]. It was shown that the members of the DL-Lite family and the slightly more expressive language linear datalog$^\pm$ are FO-rewritable. However, $\mathcal{EL}^{++}$ TBoxes are not generally FO-rewritable.
2.5.2 Combined Approach

Combined approaches for query answering [55, 56, 57, 58] are proposed to overcome the inherent limitation of the pure query rewriting approach. In particular, for ontologies that are not FO-rewritable, the ontological knowledge cannot be embedded into the query only. Combined approaches encode the ontological knowledge into both the data and the query in order to apply the state of the art relational database techniques for ontological query answering. A combined approach answers a CQ over an ontology in three steps: first, it encodes the ontological knowledge through either transformation to datalog rules [58] or partial materialisation of the data [55, 56]; second, it rewrites the CQ by adding filtering constraints; and third, it evaluates the rewritten query over the datalog rules and (materialised) data.

We define the class of ontologies in which the combined approach is applicable.

**Definition 2.2.** For a TBox $T$ and a CQ $Q$, a *combined rewriting* of the pair $(T, Q)$ is another pair $(D_T, Q')$, where $D_T$ is a datalog program and $Q'$ is a CQ logically stronger than $Q$ (i.e., $\forall \vec{x}. [Q \rightarrow Q']$ is a tautology), such that for each ABox $A$, $\text{ans}(Q, T \cup A) = \text{ans}(Q', D_T \cup A)$. $T$ is *(polynomially) combined rewritable* if for each CQ $Q$, a combined rewriting of $(T, Q)$ can be obtained (in polynomial time).

A combined rewriting transforms TBox axioms into datalog rules, and adds filter constraints to the initial query to ensure correctness of query answering. If an ontology can be translated into a datalog program that is (polynomially) combined rewritable, one can make use of datalog engines for direct query answering. Analogously, one can first materialise the data, then make use of relational database systems for query answering. Moreover, transforming the ontology into a datalog program allows one to use resolution-based approach for query answering and in particular for query abduction. Note that our
definition of combined rewriting is different from those in [55, 56, 57], where the ontological knowledge is fully embedded into the data and hence $D_T$ is empty. A large portion of ontologies in OWL 2 profiles, in particular for OWL 2 QL [56] and OWL 2 EL [55, 58] are polynomially combined rewritable, while OWL 2 RL ontologies can be equivalently translated into datalog programs and hence are directly applicable. Moreover, results in [57] imply that a datalog$^\pm$ program is polynomially combined rewritable if it processes the so-called polynomial witness property. A concrete class satisfying the property is the collection of linear datalog$^\pm$ with predicates of bounded arities.

2.6 Query Abduction

Efficient DL reasoners have been developed for reasoning tasks such as classification and query answering. In order to meet usability requirements set by domain users, efforts have been made to equip DL-based ontology systems with explanation algorithms for reasoning services[23, 24, 29, 31, 32, 59]. One important explanation problem for DL ontologies, referred to as query abduction, can be formalised as abductive reasoning investigated in AI [60], or more precisely, query abduction in DLs. Since this type of reasoning is dependent on the ABox of an ontology, it is also called ABox abduction. Informally, given an ontology and an observation (a set of ABox axioms or a query with instantiated answer variables), a solution to the query abduction problem (a.k.a. an explanation for the query answer) is a set of facts (i.e., an ABox) that together with the ontology can entail the observation. It can be formally defined as follows.

**Definition 2.3** (ABox Abduction Problem). Let $\mathcal{K}$ be a consistent DL knowledge base, $\Sigma$ be a finite subset of concept and role names in $\mathcal{K}$ and $\Phi$ be a set of ABox axioms composed of individuals in $\mathcal{K}$ and concepts or roles in $\Sigma$. The pair $\langle \mathcal{K}, \Phi \rangle$ is an ABox abduction problem if $\mathcal{K} \not\models \Phi$ and $\mathcal{K} \cup \Phi \not\models \bot$. 
The solution to an abductive problem $\langle K, \Phi \rangle$ is a set of ABox assertions $E$ iff $K \cup E \models \Phi$. Moreover, $E$ is called:

**consistent** iff $K \cup E \not\models \bot$.

**relevant** iff $E \not\models \Phi$.

**minimal** iff there is no solution $E'$ to $\langle K, \Phi \rangle$ where $E \subset E'$.

The observation in the above definition is a set of ABox axioms where can be presented as conjunctions of ground atoms. Existing approaches to such an abductive problem in DLs vary according to their underlying DLs. For instance,\cite{23, 24} are based on $\mathcal{ALCN}$, \cite{25, 26, 27} are based on $\mathcal{ALC}$, \cite{28, 29} are based on DL-Lite, while \cite{30} is based on $\mathcal{SHIQ}$.

Regarding different types of observations used in abductive problem, both atomic queries and **conjunctive queries** (CQs) are considered in the literature. Moreover, a query answer can be either **positive** or **negative**. A query answer is positive w.r.t. a given ontology and a query if it occurs in the answer set of the query over the ontology; otherwise, it is negative. Negative answers occur when a user expects some tuple to be among the answers but it turns out to be not. Previous research focused on the explanation for positive atomic queries until Borgida, Calvanese and Rodriguez-Muro \cite{28} advocated the importance of ontology-based systems to be capable of explaining **negative query answers**, who claimed that such reasoning service is essential for users to understand why a desired answer is missing from the query answers and studied the problem of explaining negative answers for CQs over DL-Lite ontologies. Therefore, a more fine-grained abductive problem proposed by Calvanese et al. \cite{29}. In the following definition, a solution can contain any constant, including arbitrarily many fresh constants not presents in the initial ABox but it is a $\Sigma$-ABox which means that all assertions composed of only predicates in $\Sigma$. 


Definition 2.4 (Query abduction problem [29]). Let $\mathcal{T}$ and $\mathcal{A}$ be TBox and ABox of a DL ontology, $q(\bar{x})$ a query, $\bar{c}$ a tuple of individuals of arity $|\bar{x}|$, and $\Sigma$ a non-empty finite subset of $N_C \cup N_R$. The tuple $\langle \mathcal{T}, \mathcal{A}, q, \bar{c}, \Sigma \rangle$ is called Query Abduction Problem (QAP). An explanation for $\langle \mathcal{T}, \mathcal{A}, q, \bar{c}, \Sigma \rangle$ is a $\Sigma$-ABox $E$ such that

1. $\mathcal{T} \cup \mathcal{A} \cup E \not\models \bot$
2. $\mathcal{T} \cup \mathcal{A} \cup E \models q(\bar{c})$

Some researchers have investigated tableau-based algorithms for query abduction [25, 26, 27] as well as computational complexity of query abduction [29]. However, algorithms such as the tableau-based ones, are not efficient enough to deal with query abduction for realistic ontologies. The problem of developing practical algorithms for query abduction supporting CQs is rarely studied except for [30, 33, 61]. In [30, 33], an algorithm is developed for query abduction for ground CQs and it is shown that the algorithm is able to handle some large ontologies such as LUMB ontologies [62]. Besides several optimization techniques, a major idea in their algorithm is to encode a query abduction problem as a Prolog program. However, in their transformation, existential rules have been absorbed in the KAON2 [63] transformation to ensure the completeness for the ground CQs. While this brute force approach to handle existential rules is simple, the resulting Prolog program is too weak to find some solutions for query abduction with DL ontologies containing (essentially) existential rules. For example, consider the ontology $\mathcal{K}$ consisting of only one TBox axiom $A \sqsubseteq \exists r.B$ and one ABox assertion $B(a)$, and the query $Q(x) = \exists y. r(x, y)$. Note that $a$ is a negative answer to the query as $Q(a)$ is not derivable from the ontology. Intuitively, the new ABox $\{A(a)\}$ is an explanation for $a$ as $\mathcal{K} \cup \{A(a)\} \models Q(a)$. However, the algorithm developed in [30, 33] discards the axiom $A \sqsubseteq \exists r.B$ in KAON2 transformation and thus the resulting Prolog program is insufficient for obtaining the solution $\{A(a)\}$. In [61], the authors suggested computing
representative explanations to reduce the number of explanations to be computed. However, the approach can only address first-order rewritable TBoxes, and many TBoxes expressed in common ontology languages, such as $\mathcal{EL}$, are not first-order rewritable in general. For example, let ontology $\mathcal{K}$ consist of a TBox with one axiom $\exists r.A \sqsubseteq A$ and an empty ABox. For observation $A(a)$, as defined in [61], there are an infinite number of representative explanations of the form $\{r(a, u_1), r(u_1, u_2), \ldots, r(u_{n-1}, u_n), A(u_n)\}$.

In Chapter 3, we first define a slightly different query abduction problem, then we propose a practical algorithm over $\mathcal{ELH}_\perp$ ontologies to address all those issues in the previous work.

### 2.7 Concept Learning

Data on the web is growing rapidly from an already a large volume of documents. For example, in September 2011, there were 31 billion RDF triples and around 504 million RDF links for the 295 data sets available online. Accordingly, the Web is rapidly evolving into a large scale platform for publishing and sharing knowledge using ontologies. For instance, there were at least 7,000 ontologies used in the Watson ontology search engine in 2008 [64]. Consequently, one of the main challenges in working with structured data is to construct ontologies of the semantically annotated data on the web, which is an onerous task even for knowledge engineers. Additionally, this construction may have diverse presentations among different engineers, and the data on the web is not always complete.

Concept learning is important in different application domains. One of the applications of learning a concept description is to assist knowledge engineers to construct an ontology harmoniously. On the other hand, a knowledge engineer can use a concept learning system to enrich a knowledge base. Another
application of learning concept description is to give a compact answer to a query. For example, an answer to the question of “Which rivers flow through more than 5 cities in Germany?” is “All the rivers which flow through Duisburg.” [65]; such answers provide additional knowledge to the user about the question. Finally, debugging the ontology is a possible application of concept learning, when the learned concept shows data inconsistencies and indicates where there are the missing data.

Concept learning in DLs concerns learning a general hypothesis, as a concept description, from the given example data of a background knowledge that we want to learn. There are two kinds of examples: positive examples, which are true, and negative examples, which are false. The positive and negative examples are given as subsets of \( N_I \), denoted as \( E^+_G \) and \( E^-_G \), respectively.

Literally, if one assumes a set \( A' \) can be viewed as a finite example set of the goal concept \( G \) (or the learned concept). That is, \( A' = \{ G(p_1), G(p_2), \ldots, G(p_i), \neg G(n_1), \neg G(n_2), \ldots, \neg G(n_j) \} \), consequently \( E^+_G = \{ p_1, p_2, \ldots, p_i \} \) and \( E^-_G = \{ n_1, n_2, \ldots, n_j \} \). In practice, it might not be possible to give both sets of \( E^+_G \) and \( E^-_G \) so one can assume \( E^+_G \) is always given and \( E^-_G \) is a subset of \( N_I \backslash E^+_G \). In the query answering, the positive and negative examples can be also seen as positive and negative answers of the query of the goal concept. We denote \( \text{concept}(T) \), \( \text{role}(T) \), \( \text{Ind}(A) \) as a set of concept names, role names, and individuals occurring in an ontology, respectively. The signature of a concept \( C \) denoted by \( \text{sig}(C) \) is a set of all concept and role names occurring in \( C \).

Analogously, we can define \( \text{sig}(T) \). The size of a concept \( C \), denoted by \( \lvert C \rvert \), is all concepts, roles, and connectives including \( \{ \cap, \exists, ., (,) \} \) occurred in \( C \).

**Definition 2.5** (Concept Learning Problem (CLP)). The concept learning problem over a language \( L \) is a four tuple \( \langle T, A, E^+_G, E^-_G \rangle \), where \( T \) is an \( L \) TBox, \( A \) is an \( L \) ABox, and both \( E^+_G \) and \( E^-_G \) are subsets of \( \text{Ind}(A) \). The solution of a CLP is a concept description \( G \) where \( \text{sig}(G) \subseteq \text{sig}(T) \) such that:
Chapter 2. Preliminaries

1. $\mathcal{T} \cup \mathcal{A} \models G(p)$ for each $p \in E^+_G$

2. $\mathcal{T} \cup \mathcal{A} \not\models G(n)$ for each $n \in E^-_G$

The first constraint is called the **completeness** of $G$ w.r.t. $E^+_G$, and the second constraint is **consistency** of $G$ w.r.t. $E^-_G$. Overall, $G$ is **correct**, if $G$ is complete and consistent w.r.t. $E^+_G$ and $E^-_G$. We denote $\mathcal{T}_G$ as $\mathcal{T} \cup \{G \equiv C_G\}$ in this thesis. We call a concept description $C_G$ is **overly general** w.r.t. $E^+_G$ and $E^-_G$, if $C_G$ is complete w.r.t. $E^+_G$ but not consistent w.r.t. $E^-_G$. $C_G$ is **overly specific** w.r.t. $E^+_G$ and $E^-_G$, if $C_G$ is consistent w.r.t. $E^-_G$ but not complete w.r.t. $E^+_G$. $C_G$ is **too weak** w.r.t. $E^+_G$, if $C_G$ is not complete w.r.t. $E^+_G$. $C_G$ is **too strong** w.r.t. $E^-_G$, if $C_G$ is not consistent w.r.t. $E^-_G$.

**Example 2.2.** If one wants to learn the concept of Grandfather for the ontology mentioned in Example 2.1 with a positive example, $E^+_G = \{\text{Christopher}\}$, and a negative example, $E^-_G = \{\text{James}\}$, a correct solution is:

$$G = \text{Male} \sqcap \exists \text{hasChild}.\text{Parent}$$

Generally, finding a correct concept description in a language $\mathcal{L}$ is a search algorithm to examine the correctness of possible $\mathcal{L}$-concept descriptions. Concept learning approaches can usually be classified in a natural way as **bottom-up** or **top-down**, depending on the general direction of its search algorithm. The **bottom-up** approaches start with a $G$ such that $G$ is overly specific and generalise $G$. The **top-down** approaches start with a $G$ such that $G$ is overly general, and specialise it. Nevertheless, it is possible to have a specification step in a bottom-up approach. Such a specification step may be needed to correct an earlier generalisation step, which made the $G$ too strong. Analogously, a generalisation step may be needed in a top-down approach.

Currently, most of the approaches to concept learning for DLs are an extension of inductive logic programming (ILP) methods. In concept learning in
description logics, promising research has been investigated and described in [37, 38, 39]. All of these approaches have been proposed for expressive DLs such as ALC. One of the most significant concept learning system for DLs is DL-Learner [39] which has different heuristics to explore the search space with a built-in instance checker to employ Close World Assumption (CWA). However, none of these systems are scalable to work with real world ontologies.

Nevertheless, there is little research on concept learning in DLs that transfer DL axioms to logic programs (LP), which applies the ILP method in order to learn a concept [34]. On one hand, this approach is too expensive in terms of computation time. On the other, it is not always guaranteed that this conversion is possible. Another approach to tackle the concept learning problem in DL is by employing a Machine Learning approach such as Genetic Programming [35] and kernels [36]. Lehmann et al. [35] performed genetic programming (GP) in concept learning in two different ways. First, standard GP was used in concept learning; however, the results were not satisfactory because the subsumption properties of concepts could not be employed in the model. Then, a hybrid system that uses refinement operators in GP was proposed. The results for the hybrid system outperformed the standard GP, although it generated longer hypotheses.

In terms of learning a concept description in less expressive DLs, research is limited. A learner for DL EL, proposed by Lehmann and Haase [42], uses minimal trees to construct DL EL axioms and refines these by refinement operators. The DLs axioms were converted to trees with four different operators defined to refine those trees. Apart from these ILP-based approaches, Rudolph [40] proposed a method based on Formal Concept Analysis (FCA) to generate a hypothesis. Further Baader et al. [41] used FCA to complete a knowledge base. Both of these methods used a less expressive DLs, where the former used FLE, while the latter used a fragment of DLs which is less expressive.
than FL. These approaches demand many interactions of a knowledge engineer as an oracle of the system, which is not applicable in most scenarios. In this thesis, an automated system to learn concept descriptions more efficiently will be developed.

In terms of bottom-up or top-down classifications of current concept learning approaches, the bottom-up approaches \cite{38, 40, 41} are computationally more expensive than top-down approaches \cite{34, 35, 37, 39, 42}. Since, for generalising a concept description in DLs, the most specific concept (msc) (or least common subsumer (lcs) in ILP) has to be computed where it does not exist even for concepts represented in lightweight DL such as $\mathcal{ELH}_\perp$ \cite{66}. Therefore, our proposed approach is kind of a top-down search algorithm.

The above-mentioned approaches mostly focused on concept learning in expressive DLs, where it is not possible to have a scalable learner due to the fact that the underlying reasoners are not scalable. Thus, in Chapter 5, we propose a concept learning problem which produces the concept description in DL $\mathcal{ELH}_\perp$. Our approach is based on refinement operators to create a search space, then we use a top-down approach to find a correct concept description.
Chapter 3

Query Abduction

“But every error is due to extraneous factors (such as emotion and education); reason itself does not err.”

–Kurt Gödel

In this chapter, we first introduce a new query abduction problem which considers general conjunctive queries (CQs) and make it possible to have a finite number of solutions even for non-FO rewritable ontologies such as $\mathcal{ELH}_{\perp}$ [9] ontology. We choose DL $\mathcal{ELH}_{\perp}$ as a test case of our query abduction approach for several reasons. First, $\mathcal{ELH}_{\perp}$ underlies the OWL 2 EL. Also, many practical ontologies such as SNOMED CT [45] can be expressed in $\mathcal{ELH}_{\perp}$. Moreover, $\mathcal{ELH}_{\perp}$ is a core fragment of several expressive DLs.

Figure 3.1 shows the outline of our new practical algorithm for query abduction. In our approach, we consider general CQs and do not simply discard existential rules in the ontologies. These two factors make the goal of efficiently computing solutions for query abduction very challenging. This is achieved through two steps of ontology transformations and one step of observation rewriting.
First, we equivalently translate a given $\mathcal{ELH}_\bot$ ontology into a datalog$^\pm$ program using a translation introduced in [53]. Then, in order to employ some off-the-shelf Prolog engines, such as XSB [67], the resulting datalog$^\pm$ program is further transformed into a plain datalog program based on a technique introduced for query answering in [58]. Specifically, we rewrite each existential rule into two datalog rules by instantiating existential variables with fresh constants (cf. Table 3.1), while the soundness will be lost. To ensure the soundness of our approach, we adapt a query rewriting technique to rewrite the observation so as to filter out incorrect solutions. As a result, the query abduction problem is reduced to an abduction problem in datalog, and we show that our approach is sound and complete for query abduction in $\mathcal{ELH}_\bot$. Finally, the abduction problem is encoded as a Prolog program with an extended encoding of [33], and thus, by executing the Prolog program on XSB, all the minimal solutions for query abduction are computed.

Therefore, we adapt a resolution-based approach from logic programming to compute the (minimal) solutions to a QAP. Although an $\mathcal{ELH}_\bot$ KB can be translated into a datalog$^\pm$ program without affect CQ answering, a resolution procedure is not guaranteed to terminate on a datalog$^\pm$ program. Hence, we approximate the datalog$^\pm$ program to a plain datalog program, for which a resolution algorithm (with tabling) is guaranteed to terminate [68].
3.1 Query Abduction Problem

Before presenting our query abduction algorithm, we need to define a different query abduction problem to have an efficient and scalable system. Therefore, we consider the query abduction problem (QAP) over $\mathcal{ELH}_\perp$ ontologies where a general CQ and an answer tuple are given as the observation and a solution to the problem (or equivalently an explanation to the query answer) is a set of facts. Formally, given a background ontology consisting of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, taking an instantiated CQ (or a BCQ) $Q(\bar{a})$ as the observation, a solution to the query abduction problem is a (possibly empty) set $E$ of facts such that $E$ is consistent with $\mathcal{T} \cup \mathcal{A}$ and $\bar{a}$ is an answer to $Q(\bar{x})$ over $\mathcal{T} \cup \mathcal{A}$ (only) together with $E$. In classical abduction setting, to reduce the search space and retrieve only relevant solutions, the predicates allowed in the solutions are often pre-specified, which are called abducibles and denoted $\Sigma$. Furthermore, for the ease of understanding, solutions should be as succinct as possible, and the minimal solutions (w.r.t. set containment) are desired. The formal definition of QAP and a solution to the QAP is given as follows. Afterwards, whenever we use the QAP we refer to the following definition, otherwise it is stated.

**Definition 3.1 (QAP for $\mathcal{ELH}_\perp$ ontology).** An instance of a query abduction problem (QAP) is a five tuple $P = \langle \mathcal{T}, \mathcal{A}, Q(\bar{a}), \Sigma, \Delta \rangle$, where $\mathcal{T}$ is an $\mathcal{ELH}_\perp$ TBox, $\mathcal{A}$ is an $\mathcal{ELH}_\perp$ ABox, $Q(\bar{x})$ is a CQ, $\bar{a}$ a tuple of constants with the same arity as $\bar{x}$, $\Sigma \subseteq N_C \cup N_R$ is a finite set of predicates, and $\Delta$ is a finite set of constants. BCQ $Q(\bar{a})$ is the observation, $\Sigma$ is the set of abducibles, and $\Delta$ is the domain.

A solution $E$ to the QAP $P$ is a set of facts such that

1. $\text{pred}(E) \subseteq \Sigma$ and $\text{const}(E) \subseteq \Delta$;
2. $\mathcal{T} \cup \mathcal{A} \cup E \models Q(\bar{a})$;
3. $T \cup A \cup E \not\models \bot$; and

4. $E \not\models Q(\bar{a})$;

Conditions 3 and 4 are called non-triviality conditions for a solution. We denote $\text{sol}(\mathcal{P})$ as the set of solutions to $\mathcal{P}$. Moreover, $E$ is minimal if there is no solution $E' \in \text{sol}(\mathcal{P})$ s.t. $E' \subset E$.

Query abduction problem in datalog$^\pm$ or datalog programs are defined in the same way as in $\mathcal{ELH}_\bot$ ontologies, and for query abduction of datalog$^\pm$ or datalog rules, we extend the definition QAP to allow $Q$ in observations to be an FO query.

In contrast to the definition 2.4 in [29, 61], the above definition has a finite domain $\Delta$ as an additional attribute, and $\Delta$ contains all the constants occurring in a solution. In [29, 61], a solution can contain any constant, including arbitrarily many fresh constants not presents in the initial ABox. In this case infinitely many (minimal) solutions always exist for a QAP, and hence it is impossible to compute all the (minimal) solutions. For example, given a datalog program $\mathcal{R} = \{R(x,y) \land A(y) \rightarrow A(x)\}$, an empty set of facts, and a BCQ $A(a)$, then for the QAP with abducibles $\{A, R\}$, each set $\{R(a,u), A(u)\}$ with $u$ an arbitrary constant is a minimal solution. Hence if arbitrarily many constants are allowed to occur in a solution then the minimal solutions are infinitely many. In [61], the authors suggest to compute representative solutions which ignore renaming of fresh constants and are minimal up to variable renaming, still infinitely many such representative solutions may exist in general.

A sufficient condition for finiteness is the ontology to be FO rewritable [61]. However, many simple (and practical) ontologies are not FO rewritable, for instance OWL 2 EL ontologies. The existential rule in the above example corresponds to an $\mathcal{ELH}_\bot$ axiom $\exists R.A \sqsubseteq A$ and the BCQ is a simple atomic query, yet an infinite number of representative solutions exist that are of the
form \( \{ R(a, u_1), R(u_1, u_2), \ldots, R(u_{n-1}, u_n), A(u_n) \} \) for \( n \geq 1 \). On the other hand, allowing uncontrolled introduction of fresh constants to QAP solutions is arguably of mere theoretical interest. In query abduction, a finite number of constants are often sufficient.

Hence, we assume a finite domain \( \Delta \) in a QAP, which can either be specified by the user, or by default set to be the set of all the constants in the initial ABox \( A \) and/or the constants in the observation \( Q \). Given that the predicates and constants allowed in a solution are both finite and pre-specified, the total number of solutions to a QAP is always finite (even for ontologies that are not FO rewritable).

### 3.2 Ontology Transformations

Now we are ready to introduce our new practical algorithm according to the QAP definition in the previous section. As shown in Figure 3.1, the first part of our algorithm is ontology transformation.

Therefore, for an \( \mathcal{ELH}_\perp \) KB \( \mathcal{K} = T \cup A \) with a normalized TBox, we will first transform \( T \) into a datalog\( ^\pm \) program and then further transform it into a datalog program. The primary goal of using these transformations here is to reduce the query abduction problem in \( \mathcal{ELH}_\perp \) into the problem of abduction in datalog.

An \( \mathcal{ELH}_\perp \) TBox \( T \) can be transformed into a datalog\( ^\pm \) program \( R_T \) as shown in Table 3.1 [53].

\( T \) and \( R_T \) are equivalent theories in first-order logic [9], and the datalog\( ^\pm \) program \( R_K = R_T \cup A \) is the datalog\( ^\pm \) translation of \( K = T \cup A \). Thus, the two corresponding QAPs \( \langle T, A, Q(\bar{a}), \Sigma, \Delta \rangle \) and \( \langle R_T, A, Q(\bar{a}), \Sigma, \Delta \rangle \) are
equivalent too, for any KB $\mathcal{K} = T \cup A$, BCQ $Q(\overline{a})$, sets $\Sigma$ and $\Delta$ of abducibles and domain respectively.

**Proposition 3.2.** Let $T$ and $A$ be an $\mathcal{ELH}_\bot$ TBox and an $\mathcal{ELH}_\bot$ ABox, $Q(\overline{a})$ be a BCQ, $\Sigma \subseteq N_C \cup N_R$, and $\Delta \subseteq N_I$. Then,

$$sol(T, A, Q(\overline{a}), \Sigma, \Delta) = sol(R_T, A, Q(\overline{a}), \Sigma, \Delta)$$

**Proof.** For each solution $E \in sol(T, A, Q(\overline{a}), \Sigma, \Delta)$, by Definition 3.1, we have $pred(E) \subseteq \Sigma$, and $const(E) \subseteq \Delta$. Also, $T \cup A \cup E$ is consistent and $T \cup A \cup E \models Q(\overline{a})$. Since $R_T$ and $T$ have the same set of first-order models, $T \cup A \cup E$ and $R_T \cup A \cup E$ have the same set of models. That is, $R_T \cup A \cup E$ is consistent and $R_T \cup A \cup E \models Q(\overline{a})$. Thus, $E \in sol(R_T, A, Q(\overline{a}), \Sigma, \Delta)$, and that is, $sol(R_T, A, Q(\overline{a}), \Sigma, \Delta) \subseteq sol(R_T, A, Q(\overline{a}), \Sigma, \Delta)$. In the same way, we can show $sol(R_T, A, Q(\overline{a}), \Sigma, \Delta) \subseteq sol(T, A, Q(\overline{a}), \Sigma, \Delta)$. \qed

The above transformation reduces a QAP in $\mathcal{ELH}_\bot$ into a QAP in $\text{datalog}^\pm$ but there is no efficient $\text{datalog}^\pm$ abduction engine available. For this reason, we use another transformation to reduce the QAP in $\text{datalog}^\pm$ into a QAP in datalog. Formally, the resulting $\text{datalog}^\pm$ program $R_T$ is transformed into a $\text{datalog}$ program $D_T$ as specified in Table 3.2 [58]. That is, $D_T$ is obtained by

<table>
<thead>
<tr>
<th>$\mathcal{ELH}_\bot$ axiom</th>
<th>datalog$^\pm$ rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \subset \bot$</td>
<td>$\vartriangleright$ A(x) $\rightarrow$ $\bot$</td>
</tr>
<tr>
<td>$A_1 \subseteq A$</td>
<td>$\vartriangleright$ $A_1(x) \rightarrow A(x)$</td>
</tr>
<tr>
<td>$A_1 \cap A_2 \subseteq A$</td>
<td>$\vartriangleright$ $A_1(x) \land A_2(x) \rightarrow A(x)$</td>
</tr>
<tr>
<td>$A_1 \subseteq \exists r. A$</td>
<td>$\vartriangleright$ $A_1(x) \rightarrow \exists y.(r(x, y) \land A(y))$</td>
</tr>
<tr>
<td>$A_1 \subseteq \exists r. \top$</td>
<td>$\vartriangleright$ $A_1(x) \rightarrow \exists y.(r(x, y))$</td>
</tr>
<tr>
<td>$\exists r. A_1 \subseteq A$</td>
<td>$\vartriangleright$ $r(x, y) \land A_1(y) \rightarrow A(x)$</td>
</tr>
<tr>
<td>$\exists r. \top \subseteq A$</td>
<td>$\vartriangleright$ $r(x, y) \rightarrow A(x)$</td>
</tr>
<tr>
<td>$r \subseteq s$</td>
<td>$\vartriangleright$ $r(x, y) \rightarrow s(x, y)$</td>
</tr>
</tbody>
</table>

**Table 3.1:** Transformation from $\mathcal{ELH}_\bot$ to $\text{datalog}^\pm$. 
replacing each existential rule in datalog$^+$ program $\mathcal{R}_T$ with two datalog rules.

<table>
<thead>
<tr>
<th>datalog$^+$ rule</th>
<th>datalog rules</th>
<th>$\Rightarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1(x) \rightarrow \exists y. (r(x,y) \land A(y))$</td>
<td>$A_1(x) \rightarrow r(x,c_{r,A})$</td>
<td>$A_1(x) \rightarrow A(c_{r,A})$</td>
</tr>
<tr>
<td></td>
<td>$c_{r,A}$ is a fresh constant.</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Transformation from datalog$^+$ to datalog.

It is clear that $\mathcal{D}_T$ can be computed in linear time to the size of $\mathcal{T}$ [58].

The following example illustrates the two steps of transformations.

**Example 3.1.** Let TBox $\mathcal{T}$ consists of the following three axioms:

$T1)$ $RA \sqsubseteq Person \sqcap \exists works.RG$

$T2)$ $Student \sqsubseteq Person \sqcap \exists takes.Course$

$T3)$ $Person \sqcap \exists takes.Course \sqsubseteq Student$

Axiom $T1$ says that a research assistant (RA) is a person and she works in some research group (RG), and axioms $T2$ and $T3$ say that a student is a person who takes some courses.

The corresponding datalog$^+$ program $\mathcal{R}_T$ has five rules $R1$–$R6$:

$R1)$ $RA(x) \rightarrow Person(x)$

$R2)$ $RA(x) \rightarrow \exists y. (works(x,y) \land RG(y))$

$R3)$ $Student(x) \rightarrow Person(x)$

$R4)$ $Student(x) \rightarrow \exists y. (takes(x,y) \land Course(y))$

$R5)$ $Person(x) \land takes(x,y) \rightarrow tempRole(x,y)$

$R6)$ $tempRole(x,y) \land Course(y) \rightarrow Student(x)$

where $R5$ and $R6$ represent $T3$, the auxiliary role $tempRole$ is introduced via normalization.
Then, \( R_T \) is transformed into the datalog program \( D_T \) containing \( R1, R3, R5 \), and the following datalog rules \( S1-S4 \):

\[
\begin{align*}
S1) \quad & RA(x) \rightarrow works(x, c_1) \\
S2) \quad & RA(x) \rightarrow RG(c_1) \\
S3) \quad & Student(x) \rightarrow takes(x, c_2) \\
S4) \quad & Student(x) \rightarrow Course(c_2)
\end{align*}
\]

where \( c_1, c_2 \) are fresh constants, \( S1 \) and \( S2 \) are transformed from \( R2 \), and \( S3 \) and \( S4 \) from \( R4 \).

For an \( \mathcal{ELH}_\bot \) KB \( \mathcal{K} = \mathcal{T} \cup \mathcal{A} \), the program \( D_K = D_T \cup \mathcal{A} \) is referred to as its datalog approximation. The following result states that the datalog approximation of each KB is complete for query abduction. That is, each solution to the QAP w.r.t. KB \( \mathcal{T} \cup \mathcal{A} \) is a solution to the corresponding QAP w.r.t. datalog program \( D_T \cup \mathcal{A} \).

**Proposition 3.3.** Let \( \mathcal{P} = \langle \mathcal{T}, \mathcal{A}, \mathcal{Q}(\bar{a}), \Sigma, \Delta \rangle \) be a QAP in \( \mathcal{ELH}_\bot \), and \( \mathcal{P}' \) be the QAP obtained from \( \mathcal{P} \) by replacing \( \mathcal{T} \) with \( D_T \). Then, \( \text{sol}(\mathcal{P}) \subseteq \{ \mathcal{E} \in \text{sol}(\mathcal{P}') \mid D_T \cup \mathcal{A} \cup \mathcal{E} \not\models \bot \} \).

Let \( \mathcal{R}_T \) be the existential program obtained from \( \mathcal{T} \) in the same way as \( \mathcal{D}_T \), except that each axiom \( A_1 \sqsubseteq \exists R.A \) in \( \mathcal{T} \) is transformed into \( A_1(x) \rightarrow \exists y. [R(x, y) \land A(y)] \).

To prove Proposition 3.3, we first show the following lemma. This lemma says that \( D_T \) can be used for consistency check of \( \mathcal{T} \) together with any set of facts, and for CQ answering.

**Lemma 3.4.** For an \( \mathcal{ELH}_\bot \) TBox \( \mathcal{T} \) and an ABox \( \mathcal{A} \), \( \mathcal{T} \cup \mathcal{A} \models \bot \iff D_T \cup \mathcal{A} \models \bot \). Whenever \( \mathcal{T} \cup \mathcal{A} \not\models \bot \), for each BCQ \( \mathcal{Q} \), \( \mathcal{T} \cup \mathcal{A} \models \mathcal{Q} \) implies \( D_T \cup \mathcal{A} \models \mathcal{Q} \).

**Proof.** Let \( \mathcal{T}' \) be the result of removing each axiom \( A \sqsubseteq \bot \) in \( \mathcal{T} \), and \( \mathcal{R}_T' \) (\( D_T' \)) be the corresponding program by removing \( A(x) \rightarrow \bot(x) \) in \( \mathcal{R}_T \) (\( D_T \)). It is clear that \( \mathcal{T}' \) and \( \mathcal{R}_T' \) are equivalent theories in first-order logic. Also,
$T \cup A \models \bot$ iff $T' \cup A \models \exists x. A(x)$ for some $A$ occurring in $T \setminus T'$; as otherwise a model of $T' \cup A$ is a model of $T \cup A$. Hence, $T \cup A \models \bot$ iff $R_T' \cup A \models \exists x. A(x)$ for some $A$ occurring in $R_T \setminus R_T'$, iff $R_T \cup A \models \bot$. It is shown in [58] that $R_T \cup A \models \bot$ iff $D_T \cup A \models \bot$. Whenever $T \cup A \not \models \bot$, $T \cup A \models Q$ iff $T' \cup A \models Q$, and $D_T \cup A \models Q$ iff $D_T' \cup A \models Q$. We want to show that $D_T' \models R_T'$, as $R_T'$ is equivalent to $T'$. Note that $D_T'$ is obtained by replacing each rule in $R_T'$ of the form $R_1 : A_1(x) \rightarrow \exists y. [r(x,y) \land A(y)]$ with two rules $R_2 : A_1(x) \rightarrow r(x,c)$ and $R_3 : A_1(x) \rightarrow A(c)$ where $c$ is a constant. For each first-order model $I$ of $D_T'$, $I$ satisfies rules $R_2$ and $R_3$, and thus $I$ also satisfies rule $R_1$. Thus, $I$ is a model of $R_T'$.

Now, we show Proposition 3.3.

*Proof.* For each solution $E \in \text{sol}(P)$, by Definition 3.1, we have $\text{pred}(E) \subseteq \Sigma$, $\text{const}(E) \subseteq \Delta$, and $E \not \models Q(\vec{a})$. Also, $T \cup A \cup E \not \models \bot$ and $T \cup A \cup E \models Q(\vec{a})$. By Lemma 3.4, $D_T \cup A \cup E \models Q(\vec{a})$, and hence, $E \in \text{sol}(P')$. Also, $T \cup A \cup E \not \models \bot$ implies $D_T \cup A \cup E \not \models \bot$. 

However, the converse of Proposition 3.3 may not hold, that is, the approximation is not necessarily sound. We demonstrate this in the following two examples, respectively.

**Example 3.2.** Consider $KB K = T \cup A$ where $T$ is the TBox in Example 3.1 and $A$ is empty, observation $Q(john, julie)$ where

$$Q(x,y) = \exists z. [\text{works}(x,z) \land \text{works}(y,z)],$$

abducibles and domain be $\Sigma = \{RA\}$ and $\Delta = \{john, julie\}$ respectively. A solution to $QAP \langle D_T, A, Q(john, julie), \Sigma, \Delta \rangle$ is $E = \{RA(john), RA(julie)\}$, which is not a solution to $QAP \langle T, A, Q(john, julie), \Sigma, \Delta \rangle$. 
Chapter 3. Query Abduction

The cause of the incorrect solution is illustrated by Figure 3.2. In particular, in each model (A) of $D_K$, all the research assistants work in the same research group $c$, whereas in a model (B) of $K$, two research assistants can work in two distinct research groups.

![Diagram](image)

**Figure 3.2:** A fork-shaped model introduced by the datalog approximation.

**Example 3.3.** Consider a KB $K$ whose TBox $T$ consists of axioms $T1$ and $T2$. The datalog approximation for $T$ consists of three rules $R1$–$R3$ as follows:

- $T1)$ $A \sqsubseteq B \rightsquigarrow R1) A(x) \rightarrow B(x)$.
- $T2) B \sqsubseteq \exists r.B \rightsquigarrow R2) B(x) \rightarrow r(x,c)$.
- $R3) B(x) \rightarrow B(c)$.

Let the ABox be empty. Consider the CQ

$$Q(x) = \exists y.[r(x,y) \land r(y,y)].$$

Then, a solution to QAP $\langle D_T, A, Q(a), \{A\}, \{a\} \rangle$ is $E = \{A(a)\}$, which is not a solution to the QAP $\langle T, A, Q(a), \{A\}, \{a\} \rangle$. The problem is depicted in Figure 3.3, where $D_K$ enforces an $r$-cycle in each of its models.

![Diagram](image)

**Figure 3.3:** A cyclic model enforced by the datalog approximation.
3.3 Observation Rewriting

In the last section, we have been able to reduce a QAP instance in $\mathcal{ELH}_\bot$ into a QAP instance in a datalog program. However, the procedure may bring in some incorrect solutions. To retain the soundness of our ontology transformation, we propose to rewrite observations in a QAP. In particular, our goal is to rewrite CQ $Q(\vec{x})$ into a FO query $Q^*(\vec{x})$ independently of the initial ABox, $\Sigma$ and $\Delta$, such that for any observation $Q(\vec{a})$, the solutions to QAP $\langle T, A, Q(\vec{a}), \Sigma, \Delta \rangle$ are exactly those to QAP $\langle D_T, A, Q^*(\vec{a}), \Sigma, \Delta \rangle$. Our rewriting approach is adapted from [55], where a rewriting method is used to filter spurious query matching. While the rewriting in [55] is shown to be sufficient for query answering, it was unknown whether it works for abduction. Also, it is worth noting that our rewriting is simpler than that of [55], as we use a different program transformation from theirs. In this section, we show that the rewriting technique can be used in abduction and in particular, our transformation-based procedure and the adapted observation rewriting form a sound and complete algorithm for query abduction in $\mathcal{ELH}_\bot$.

We first introduce unary predicates $Ind$ and $Aux$ to assert respectively the individuals occurring in the initial ABox and the auxiliary constants (i.e., those fresh constants generated in the approximation phase shown in Table 3.2). All individuals and only individuals are asserted to be members of $Ind$, and all auxiliary constants and only those constants are asserted to be members of $Aux$ in the extended ABox.

Then, to prevent spurious solutions introduced by fork-shaped or cyclic structures in the least Herbrand model of $D_\mathcal{K}$ (but not in models of $\mathcal{K}$), for a CQ $Q$, we define the equivalence relation $\sim$ over the terms (i.e., variables and constants) in $Q$ exactly as in [55]. Formally, $\sim \subseteq \text{term}(Q) \times \text{term}(Q)$ is the smallest reflexive, symmetric and transitive relation that satisfies the following condition:
(*) if \( r_1(s,t), r_2(s',t') \in Q \) with \( t \sim t' \) then \( s \sim s' \).

Intuitively, the relation \( \sim \) will be used to guarantee that if an instantiation of the query is satisfied by the least Herbrand model of \( D_K \), then the instantiation does not contain a fork or cycle introduced by auxiliary constants (e.g., as in Figures 3.2 and 3.3). Relation \( \sim \) can be computed in polynomial time to the size of \( Q \) \([55]\). For each equivalent class \( \xi \) of \( \sim \), a representative \( t_\xi \in \xi \) is chosen.

For a CQ \( Q \), the following notions help us to identify substructures in \( Q \) that can possibly lead to spurious solutions, as their instantiations may contain forks and cycles:

- **Fork** is the set of pairs \( (pre(\xi), \xi) \) with \( \xi \) being an equivalence class of \( \sim \), \( pre(\xi) = \{ t \in \text{term}(Q) \mid \exists r \in N_R, \exists t' \in \xi \text{ such that } r(t,t') \in Q \} \), and \(|pre(\xi)| \geq 2\).

- **Cyc** is the set of variables \( v \in qvar(Q) \) such that there are \( r_0(t_0, t'_0), \cdots, r_m(t_m, t'_m), \cdots, r_n(t_n, t'_n) \in Q \) \((n, m \geq 0)\), with \( v \sim t_i \) for some \( 0 \leq i \leq n \), \( t'_i \sim t_{i+1} \) for all \( i < n \), and \( t'_n \sim t_m \).

**Fork** and **Cyc** can be also computed in time polynomial to the size of \( Q \) \([55]\).

Now, we introduce the rewriting of observations. Let CQ \( Q(x) = \exists y. \phi(x, y) \), its *rewriting* \( Q^*(x) \) is the FO query \( \exists \vec{y}. [\phi \land \phi_1 \land \phi_2] \) where

\[
\phi_1 := \bigwedge_{(t_1, \cdots, t_k, \xi) \in \text{Fork}} \left( \text{Aux}(t_\xi) \rightarrow \bigwedge_{1 \leq i < k} t_i = t_{i+1} \right)
\]

\[
\phi_2 := \bigwedge_{v \in \text{Cyc}} \neg \text{Aux}(v).
\]

Intuitively, filter \( \phi_1 \) says that fork-shaped substructures in \( Q \) should be instantiated in a way that all the legs merge into one, and \( \phi_2 \) says that a cyclic substructure in \( Q \) can only be instantiated with no auxiliary constants.
Since we have finite number of individuals in the ontology, and constants are fresh individuals not occurring in the ontology, then \( \neg \text{Aux}(v) \) equals to \( \text{Ind}(v) \). Therefore, the rewriting of \( \text{Cyc} \) query is semantically equivalent to the following:

\[
\phi_2 := \bigwedge_{v \in \text{Cyc}} \text{ind}(v)
\]

Example 3.4 (Cont’d Example 3.2). There are two equivalence classes \( \{x, y\} \) and \( \{z\} \) with respect to the relation \( ~ \). The rewriting of \( \text{CQ} Q(x, y) \) is

\[
Q^*(x, y) = \exists z. [\text{works}(x, z) \land \text{works}(y, z) \land (\text{Aux}(z) \rightarrow x = y)]
\]

It guarantees that \( \mathcal{E} = \{\text{RA}(\text{john}), \text{RA}(\text{julie})\} \) is not a solution to the QAP \( \langle D_T, A, Q^*(\text{john, julie}), \{\text{RA}\}, \{\text{john, julie}\} \rangle \), as \( D_T, A \cup \mathcal{E} \not\models Q^*(\text{john, julie}) \). Indeed, no solution exists to the QAP \( \langle D_T, A, Q^*(\text{john, julie}), \{\text{RA}\}, \{\text{john, julie}\} \rangle \), which is also the case to QAP \( \langle T, A, Q(\text{john, julie}), \{\text{RA}\}, \{\text{john, julie}\} \rangle \).

Example 3.5 (Cont’d Example 3.3). There is only one equivalence classes \( \{x, y\} \) with respect to the relation \( ~ \). The rewriting of \( \text{CQ} Q(x) \) is

\[
Q^*(x) = \exists y. [r(x, y) \land r(y, y) \land (\text{Aux}(y) \rightarrow x = y) \land \text{Ind}(y)].
\]

Then, \( \mathcal{E} = \{A(a)\} \) is not a solution to the QAP \( \langle D_T, A, Q^*(a), \{A\}, \{a\} \rangle \), and no solution exists to QAP \( \langle D_T, A, Q^*(a), \{A\}, \{a\} \rangle \), which is the same case to QAP \( \langle T, A, Q(a), \{A, a\} \rangle \).

Note that our rewriting is simpler than that of [55], largely due to the datalog transformation we adopt in the paper, which provides a tighter approximation on the models. For example, given an axiom \( A \sqsubseteq \exists r.B \sqcap \exists s.B \), it is transformed into four datalog rules \( A(x) \rightarrow r(x, c_{r,B}), A(x) \rightarrow B(c_{r,B}), A(x) \rightarrow s(x, c_{s,B}), \) and \( A(x) \rightarrow B(c_{s,B}) \). In an approximated model in [55], however informally,
$c_{r,B}$ and $c_{s,B}$ are not distinguished. Thus, it is not hard to see that we do not need the filter queries related to $\text{Fork}_x$ and $\text{Fork}_H$ in [55].

The following theorem states that the ontology transformation combined with observation rewriting is sound and complete in $\mathcal{ELH}_\perp$, which was unknown before.

**Theorem 3.5.** Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{A}, Q(\vec{a}), \Sigma, \Delta \rangle$ be a QAP in $\mathcal{ELH}_\perp$ and $\mathcal{P}' = \langle \mathcal{D}_\mathcal{T}, \mathcal{A}, Q^*(\vec{a}), \Sigma, \Delta \rangle$. Then, $\text{sol}(\mathcal{P}) = \{ E \in \text{sol}(\mathcal{P}') \mid \mathcal{D}_\mathcal{T} \cup \mathcal{A} \cup E \not\models \perp \}$. 

To prove Theorem 3.5, the following result shows the observation rewriting guarantees the correctness of query answering over our datalog approximation.

**Proposition 3.6.** Let $\mathcal{T}$ be an $\mathcal{ELH}_\perp$ TBox, $\mathcal{A}$ be an ABox, $Q(\vec{x})$ be a CQ, and $\vec{a}$ be a tuple of constants. Then, whenever $\mathcal{T} \cup \mathcal{A} \not\models \perp$, $\mathcal{T} \cup \mathcal{A} \models Q(\vec{a})$ iff $\mathcal{D}_\mathcal{T} \cup \mathcal{A} \models Q^*(\vec{a})$.

To prove Proposition 3.6, we need some preparation. In particular, we show some useful properties of the least Herbrand model $\mathcal{M}$ of $\mathcal{D}_\mathcal{T} \cup \mathcal{A}$, and provide a way to construct a canonical model of $\mathcal{T} \cup \mathcal{A}$ from $\mathcal{M}$. First, let $\text{ind}(\mathcal{A})$ be the set of individuals occurring in $\mathcal{A}$, and $\mathcal{M}$ can be naturally seen as a DL interpretation as follows

\[
\Delta^\mathcal{M} = \{ a \mid a \in N_I \text{ occurs in } \mathcal{M} \}, \\
a^\mathcal{M} = a \text{ for each } a \in \text{ind}(\mathcal{A}), \\
A^\mathcal{M} = \{ a \mid A(a) \in \mathcal{M} \}, \text{ and } R^\mathcal{M} = \{ (a, b) \mid R(a, b) \in \mathcal{M} \}.
\]

Moreover, $\mathcal{M}$ is a model of $\mathcal{T} \cup \mathcal{A}$ as shown by the following lemma.

**Lemma 3.7.** Let $\mathcal{T} \cup \mathcal{A}$ be a consistent $\mathcal{ELH}_\perp$ KB and $\mathcal{M}$ be the least Herbrand model of $\mathcal{D}_\mathcal{T} \cup \mathcal{A}$. Then, $\mathcal{M}$ is a model of $\mathcal{T} \cup \mathcal{A}$.
Proof. From the definition, $\mathcal{M}$ satisfies the ABox, and each TBox axiom of the form $A \sqsubseteq B$, $A_1 \sqcap A_2 \sqsubseteq A$, $\exists R.A_1 \sqsubseteq A$, or $R \sqsubseteq S$, as the corresponding rules in $\mathcal{D}_T$ are exactly as in $\mathcal{R}_T$. For each TBox axiom of the form $A_1 \sqsubseteq \exists R.A$, from the datalog transformation, $A(x) \rightarrow R(x, c_{R,B})$ and $A(x) \rightarrow B(c_{R,B})$ are in $\mathcal{D}_T$. Suppose $d \in A^M_1$, which is $A_1(d) \in \mathcal{M}$, then by the definition of $\mathcal{M}$, $R(d, c_{R,B}), B(c_{R,B}) \in \mathcal{M}$. That is, $d \in (\exists R.A)^M$, and hence, $\mathcal{M}$ satisfies this axiom.

As outlined in [69], the interpretation $\mathcal{M}$ can be extended to concepts Ind and Aux by letting $\text{Ind}^\mathcal{M} = \{a^\mathcal{M} \mid a \in \text{ind}(A)\}$ and $\text{Aux}^\mathcal{M} = \Delta^\mathcal{M} \setminus \text{Ind}^\mathcal{M}$, and the extended interpretation (which we still denote as $\mathcal{M}$) satisfies the following properties useful to the proof of Proposition 3.6. Let $\mathcal{I}$ be a DL interpretation in which $\text{Ind}^\mathcal{I} = \{a^\mathcal{I} \mid a \in \text{ind}(A)\}$ and $\text{Aux}^\mathcal{I} = \Delta^\mathcal{I} \setminus \text{Ind}^\mathcal{I}$. A path in $\mathcal{I}$ is a finite sequence $d_0R_1d_1 \cdots R_nd_n \ (n \geq 0)$, where $d_0 \in \text{Ind}^\mathcal{I}$, $R_{i+1} \in N_R$ and $(d_i, d_{i+1}) \in R^\mathcal{I}_{i+1}$ for $0 \leq i < n$. For each path $p$, $\text{tail}(p)$ denotes the last element $e_n$ in $p$. $\text{paths}(\mathcal{I})$ denotes the set of all the paths in $\mathcal{I}$. The interpretation $\mathcal{I}$ is called ABox-connected if for each $d \in \Delta^\mathcal{I}$, $d = \text{tail}(p)$ for some path $p \in \text{paths}(\mathcal{I})$; and $\mathcal{I}$ is split if $d \in \text{Aux}^\mathcal{I}$ and $(d, e) \in R^\mathcal{I}$ for some $R \in N_R$ imply $e \in \text{Aux}^\mathcal{I}$.

Lemma 3.8. [69] Let $\mathcal{T} \cup \mathcal{A}$ be a consistent $\mathcal{ELH_\bot}$ KB and $\mathcal{M}$ be the least Herbrand model of $\mathcal{D}_T \cup \mathcal{A}$. Then, $\mathcal{M}$ is both ABox connected and split.

Further, we present a notion of unravelling for $\mathcal{M}$ and show that the unravelling forms a canonical model of $\mathcal{T} \cup \mathcal{A}$. In particular, the unravelling of $\mathcal{M}$ is a DL
interpretation \( \mathcal{J} \) defined as follows:

\[
\begin{align*}
\Delta^\mathcal{J} &= \text{paths}(\mathcal{M}), \text{ and } a^\mathcal{J} = a \text{ for each } a \in \text{ind}(\mathcal{A}); \\
A^\mathcal{J} &= \{ p \mid \text{tail}(p) \in A^\mathcal{M} \}; \text{ and} \\
R^\mathcal{J} &= \{ (d,e) \mid d,e \in \text{Ind}^\mathcal{M}, (d,e) \in R^\mathcal{M} \} \cup \\
& \quad \{ (p,p \cdot Se) \mid p,p \cdot Se \in \text{paths}(\mathcal{M}), \mathcal{T} \models S \subseteq R \},
\end{align*}
\]

where ‘·’ denotes concatenation. The following connection between \( \mathcal{M} \) and \( \mathcal{J} \) can be easily shown via an induction on the concept structure.

**Lemma 3.9.** Let \( \mathcal{T} \cup \mathcal{A} \) be a consistent \( \mathcal{ELH}_\perp \) KB, \( \mathcal{M} \) be the least Herbrand model of \( \mathcal{D}_\mathcal{T} \cup \mathcal{A} \), and \( \mathcal{J} \) be the unravelling of \( \mathcal{M} \). Then, the following statements hold:

1. \( p \in C^\mathcal{J} \) iff \( \text{tail}(p) \in C^\mathcal{M} \) for each \( p \in \Delta^\mathcal{J} \) and each \( \mathcal{ELH}_\perp \)-concept \( C \);

and

2. \( (p_1,p_2) \in R^\mathcal{J} \) iff \( \text{tail}(p_1),\text{tail}(p_2) \) \( \in R^\mathcal{M} \) for each pair \( p_1,p_2 \in \Delta^\mathcal{J} \) and each role \( R \in N_R \).

For an DL interpretation \( \mathcal{I} \) and a FO query \( \mathcal{Q} \), a mapping \( \pi : \text{term}(\mathcal{Q}) \to \Delta^\mathcal{I} \) with \( \pi(a) = a^\mathcal{I} \) for all \( a \in N_I \cap \text{term}(\mathcal{Q}) \) is called a match for \( \mathcal{Q} \) in \( \mathcal{I} \) if \( \mathcal{I} \) satisfies \( \mathcal{Q} \) under the variable assignment agreeing with \( \pi \). We write \( \mathcal{I} \models^\pi \mathcal{Q} \), and sometimes omit \( \pi \) for simplicity while inferring its implicit existence. It is clear that \( \mathcal{KB} \models \mathcal{T} \cup \mathcal{A} \models \mathcal{Q} \) if and only if \( \mathcal{I} \models \mathcal{Q} \) for each model \( \mathcal{I} \) of \( \mathcal{T} \cup \mathcal{A} \). The following result shows that \( \mathcal{J} \) is a canonical model of \( \mathcal{T} \cup \mathcal{A} \), and in particular for any CQ \( \mathcal{Q} \), answering \( \mathcal{Q} \) over \( \mathcal{T} \cup \mathcal{A} \) amounts to evaluating \( \mathcal{Q} \) against \( \mathcal{J} \).

**Lemma 3.10.** Let \( \mathcal{T} \cup \mathcal{A} \) be a consistent \( \mathcal{ELH}_\perp \) KB and \( \mathcal{J} \) be the unravelling of the least Herbrand model of \( \mathcal{D}_\mathcal{T} \cup \mathcal{A} \). Then, (1) \( \mathcal{J} \) is a model of \( \mathcal{T} \cup \mathcal{A} \), and (2) for each BCQ \( \mathcal{Q}(\bar{a}) \), \( \mathcal{T} \cup \mathcal{A} \models \mathcal{Q}(\bar{a}) \) iff \( \mathcal{J} \models^\pi \mathcal{Q}(\bar{a}) \) for some \( \pi \).
Proof. For (1), let $\mathcal{M}$ be the least Herbrand model of $\mathcal{D}_T \cup A$. By Lemma 3.7, $\mathcal{M}$ is a model of $\mathcal{T} \cup \mathcal{A}$. From Lemma 3.9, it is clear that $\mathcal{J}$ satisfies each axiom in $\mathcal{T}$ and each assertion in $\mathcal{A}$. That is, $\mathcal{J}$ is a model of $\mathcal{T} \cup \mathcal{A}$.

For (2), the “only if” direction is trivial by (1), and we only need to show the “if” direction. Suppose $\mathcal{J} \models^\pi Q(\bar{a})$ for some $\pi$. We show that $\mathcal{J}$ is a canonical model of $\mathcal{T} \cup \mathcal{A}$, that is, $\mathcal{J}$ can be mapped into each model $\mathcal{I}$ of $\mathcal{T} \cup \mathcal{A}$ with a mapping $\delta: \Delta^\mathcal{J} \to \Delta^\mathcal{I}$ satisfying the following conditions:

1. $\delta(a^\mathcal{J}) = a^\mathcal{I}$ for each $a \in \text{ind}(\mathcal{A})$;
2. $p \in A^\mathcal{J}$ implies $\delta(p) \in A^\mathcal{I}$ for each $p \in \Delta^\mathcal{J}$ and $A \in N_C$;
3. $(p_1, p_2) \in R^\mathcal{J}$ implies $(\delta(p_1), \delta(p_2)) \in R^\mathcal{I}$ for each pair $p_1, p_2 \in \Delta^\mathcal{J}$ and role $R \in N_R$.

For each $p \in \Delta^\mathcal{J}$, suppose $p = p'_0 \cdot d_0 d_1 \cdots R_k d_k$ with $d_0$ being the last element in the sequence such that $d_0 \in \text{Ind}^\mathcal{M}$. Let $\text{dep}(p) = k$, and we define $\delta(p)$ by an induction on $\text{dep}(p)$. For $\text{dep}(p) = 0$, as $d_0 = a^\mathcal{M}$ for some $a \in \text{ind}(\mathcal{A})$, define $\delta(p) = a^\mathcal{I}$. For $\text{dep}(p) = k \geq 1$, let $p_k = d_0 R_1 d_1 \cdots R_k d_k \in \Delta^\mathcal{J}$. We will define $\delta(p_k)$ and assign $\delta(p) = \delta(p_k)$. Let $p_{k-1} = d_0 R_1 d_1 \cdots R_{k-1} d_{k-1}$, then $p_{k-1} \in \Delta^\mathcal{J}$ and $\text{dep}(p_{k-1}) < k$. From the definition of $\mathcal{J}$, $(d_{k-1}, d_k) \in R_k^\mathcal{M}$, $d_k = c_{R,A}$ for some $R \in N_R$ and $A \in N_C$.

To introduce $c_{R,A}$, there must be two rules $A_1(x) \to R(x, c_{R,A})$ and $A_1(x) \to A(c_{R,A})$ in $\mathcal{D}_T$. By the datalog transformation, there is an axiom $A_1 \sqsubseteq \exists R.A$ in $\mathcal{T}$. Also, $\mathcal{D}_T \cup \mathcal{A} \models A_1(d_{k-1})$ needs to hold. Thus, $d_{k-1} \in A_1^\mathcal{M}$, and from Lemma 3.9, $p_{k-1} \in A_1^\mathcal{J}$. Since $\text{dep}(p_{k-1}) < k$, $\delta(p_{k-1})$ is defined. By (b), $\delta(p_{k-1}) \in A_1^\mathcal{I}$, and as $\mathcal{I}$ satisfies $A_1 \sqsubseteq \exists R.A$, $\delta(p_{k-1}) \in (\exists R.A)^\mathcal{I}$. Moreover, from the definition of $\mathcal{M}$ and the shapes of the rules in $\mathcal{D}_T$ (see Tables 3.2), $(d_{k-1}, c_{R,A}) \in R_k^\mathcal{M}$ only if there is a sequence of roles $S_0, \ldots, S_l$ ($l \geq 0$) with $R = S_0$, $S_i(x, y) \rightarrow S_{i+1}(x, y) \in \mathcal{D}_T$ for $0 \leq i \leq l$, and $S_l = R_k$. As these
rules are transformed from role inclusions in \( \mathcal{T} \), \( \mathcal{T} \models R \sqsubseteq R_k \). Thus, \( \delta(p_{k-1}) \in (\exists R_k.A)^\mathcal{T} \). That is, there exists some \( e \in \Delta^\mathcal{T} \) such that \( (\delta(p_{k-1}), e) \in R_k^\mathcal{T} \) and \( e \in A^\mathcal{T} \). We assign \( \delta(p) = \delta(p_k) = e \).

To see that \( \delta(p) \) satisfies (a)–(c), note that (a) follows directly from the above construction. For (c), suppose \((p_1, p_2) \in R^\mathcal{T} \). From the definition of \( J \), there are two cases: (i) Suppose \( p_1, p_2 \in \text{Ind}^\mathcal{M} \), that is \( p_i = a_i^\mathcal{M} \) for some \( a_i \in \text{ind}(\mathcal{A}) \) (for \( i = 1, 2 \)), then \( R(a_1, a_2) \) is either in \( \mathcal{A} \) or derivable from some \( S(a_1, a_2) \in \mathcal{A} \) and role inclusions. In both cases, \( (\delta(p_1), \delta(p_2)) = (a_1^T, a_2^T) \in R^2 \). (ii) Suppose \( p_2 = p_1 \cdot S_e \) for some \( S \) with \( \mathcal{T} \models S \sqsubseteq R \) and some \( e \). Then, from the above construction of \( \delta(p_2) \), \( (\delta(p_1), \delta(p_2)) \in S^\mathcal{T} \). As \( I \) satisfies \( S \sqsubseteq R \), \( (\delta(p_1), \delta(p_2)) \in R^\mathcal{T} \). For (b), suppose \( p \in A^\mathcal{T} \), that is \( d_k \in A^\mathcal{M} \), we want show that \( \delta(p) \in A^\mathcal{T} \). Let \( \mathcal{F}_0 = \mathcal{A} \), and suppose \( \mathcal{M} \) is constructed through a sequence \( \mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_l \) by applying rules in \( D_\mathcal{T} \) as outlined in Section 2.4. We show that by an induction on \( 0 \leq i \leq l \) that \( A(d_k) \in \mathcal{F}_i \) implies \( \delta(p) \in A^\mathcal{T} \). (*)

This clearly holds for \( \mathcal{F}_0 \) which contains only constants in \( \text{ind}(\mathcal{A}) \). Suppose (*) holds for \( i \geq 0 \), we show that (*) holds for \( i + 1 \). There are three cases:

(i) Suppose \( A(d_k) \) is added to \( \mathcal{F}_{i+1} \) by applying rule \( A_1(x) \rightarrow A(x) \) (or \( A_1(x) \land A_2(x) \rightarrow A(x) \)), then \( A_1 \sqsubseteq A \in \mathcal{T} \) (\( A_1 \sqcap A_2 \sqsubseteq A \in \mathcal{T} \)) and \( A_1(d_k) \in \mathcal{F}_i \) (and \( A_2 \in \mathcal{F}_i \)). By induction hypothesis, \( \delta(p) \in A^\mathcal{T}_1 \) (and \( \delta(p) \in A^\mathcal{T}_2 \)). Thus, as \( I \) satisfies \( A_1 \sqsubseteq A_1 \sqcap A_2 \sqsubseteq A \) in \( \mathcal{T} \), \( \delta(p) \in A^\mathcal{T} \).

(ii) Suppose \( A(d_k) \) is added to \( \mathcal{F}_{i+1} \) by applying rule \( A_1(x) \rightarrow A(c_{R,A}) \), then \( d_k = c_{R,A} \). From the above construction of \( \delta \), \( \delta(p) = e \) for some \( e \in A^\mathcal{T} \).

(iii) Suppose \( A(d_k) \) is added to \( \mathcal{F}_{i+1} \) by applying rule \( R(x, y) \land A_1(y) \rightarrow A(x) \), then \( \exists R.A_1 \sqsubseteq A \in \mathcal{T} \), and \( R(d_k, e) \) and \( A_1(e) \) must be in \( \mathcal{F}_i \) for some \( e \). Then \( p' = p \cdot Re \) is in \( \text{paths}(\mathcal{M}) \). From (c), \( (\delta(p), \delta(p')) \in R^\mathcal{T} \). By the induction hypothesis, \( \delta(p') \in A^\mathcal{T}_1 \). Thus, as \( I \) satisfies \( \exists R.A_1 \sqsubseteq A \) in \( \mathcal{T} \), \( \delta(p) \in A^\mathcal{T} \).
Let $\pi'$ be the composition of $\pi$ with $\delta$, i.e., $\pi'(t) = \delta(\pi(t))$ for each $t \in \text{term}(Q(\vec{a}))$. Then, $\mathcal{J} \models \pi' Q(\vec{a})$ implies that $\mathcal{I} \models \pi' Q(\vec{a})$. We have shown the "if" direction of (2).

Now, we are ready to present the proof for Proposition 3.6.

**Proof.** By Lemma 3.10, we only need to show that $\mathcal{J} \models Q(\vec{a})$ iff $\mathcal{M} \models Q^*(\vec{a})$.

To show this, we use a similar result established in [55], which is however, is stated for a more complex rewriting $Q'$ of $Q$ than our rewriting $Q^*$. In particular, $Q'$ contains more filter queries (as conjuncts) than $Q^*$, and thus $\mathcal{I} \models Q'$ implies $\mathcal{I} \models Q^*$ (but not necessarily vice versa) for any DL interpretation $\mathcal{I}$. It is shown in [55] that for an ABox-connected and split DL interpretation $\mathcal{I}$ and its unravelling $\mathcal{I}'$, $\mathcal{I}' \models Q(\vec{a})$ iff $\mathcal{I} \models Q^*(\vec{a})$. From Lemma 3.8, $\mathcal{M}$ is ABox-connected and split, and their result is sufficient to show the "only if" direction of Proposition 3.6, i.e., $\mathcal{J} \models Q(\vec{a})$ only if $\mathcal{M} \models Q^*(\vec{a})$. We only need to prove the "if" direction.

For the "if" direction, suppose $\mathcal{M} \models \pi Q^*(\vec{a})$ for some mapping $\pi$. We want to construct a mapping $\tau$ such that $\mathcal{J} \models \tau Q(\vec{a})$. Let $\sim_\pi$ be the transitive closure of

\[
\{(t,t) \mid t \in \text{term}(Q(\vec{a}))\} \cup \\
\{(t_1,t_2) \in \text{term}(Q(\vec{a}))^2 \mid t_1 \sim t_2 \text{ and } \pi(t_1), \pi(t_2) \in \text{Aux}^\mathcal{M} \} \cup \\
\{(t_1,t_2) \in \text{term}(Q(\vec{a}))^2 \mid R_1(t_1,t_1'), R_2(t_2,t_2') \in Q(\vec{a}) \text{ s.t. } \pi(t_1') \in \text{Aux}^\mathcal{M} \text{ and } t_1' \sim t_2' \}.
\]

It is shown in [55] that $\sim_\pi$ is an equivalence relation and $\pi(t_1) = \pi(t_2)$ whenever $t_1 \sim_\pi t_2$ for any $t_1, t_2 \in \text{term}(Q(\vec{a}))$. Let $Q_\pi$ be obtained from $Q(\vec{a})$ by replacing each term in some $\sim_\pi$ equivalence class $\xi$ with a fixed representative $t_\xi \in \xi$. It is also shown that $\mathcal{M} \models \pi Q_\pi$ and the follows hold:
(a) if \( v \in \text{qvar}(\mathcal{Q}_\pi) \) and \( \pi(v) \in \text{Aux}^\mathcal{M} \), then there is at most one \( t \in \text{term}(\mathcal{Q}_\pi) \) such that \( R(t, v) \in \mathcal{Q}_\pi \) for some \( R \in \mathcal{N}_R \);

(b) if \( \{R_1(t_0, t_1), \ldots, R_n(t_{n-1}, t_n)\} \subseteq \mathcal{Q}_\pi \) with \( t_0 = t_n \), then \( \pi(t_i) \in \text{Ind}^\mathcal{M} \) for all \( 0 \leq i \leq n \).

We inductively define mapping \( \tau : \text{term}(\mathcal{Q}_\pi) \rightarrow \Delta^\mathcal{J} \) such that \( \text{tail}(\tau(t)) = \pi(t) \) for all \( t \in \text{term}(\mathcal{Q}_\pi) \). For the initial step:

- for each \( t \in \text{term}(\mathcal{Q}_\pi) \) with \( \pi(t) \in \text{Ind}^\mathcal{M} \), set \( \tau(t) = \pi(t) \);

- for each \( v \in \text{qvar}(\mathcal{Q}_\pi) \) with \( \pi(v) \in \text{Aux}^\mathcal{M} \) such that there is no \( R(t, v) \in \mathcal{Q}(\bar{a}) \) for some \( \mathcal{M} \) is ABox-connected, there is a path \( p \in \Delta^\mathcal{J} \) such that \( \text{tail}(p) = \pi(v) \); set \( \tau(v) = p \in \Delta^\mathcal{J} \).

For the induction step, if \( \tau(v) \) is undefined and there exists \( R(t, v) \in \mathcal{Q}_\pi \) with \( \tau(t) \) defined. Since \( \pi(v) \in \text{Aux}^\mathcal{M}, \pi(v) = c_{S,A} \) for some \( S \in \mathcal{N}_R \) and \( A \in \mathcal{N}_C \).

Since \( (\pi(t), \pi(v)) \in \mathcal{R}^\mathcal{M} \), from the shapes of the rules in \( \mathcal{D}_T \), it can only be the case that \( (\pi(t), \pi(v)) \in S^\mathcal{M} \) and \( \mathcal{T} \models S \subseteq R \) (as shown in the proof of Lemma 3.10). Then, set \( \tau(v) = \tau(t) \cdot S \cdot \pi(v) \). From (a), \( \tau(v) \) is uniquely defined. By induction hypothesis, \( \text{tail}(\tau(t)) = \pi(t) \), and as \( (\pi(t), \pi(v)) \in S^\mathcal{M}, \tau(v) \in \Delta^\mathcal{J} \). In this way, \( \tau(t) \) can be defined for all \( t \in \text{term}(\mathcal{Q}_\pi) \). To show this, towards a contradiction, suppose \( \tau(t) \) is undefined for some \( t \in \text{term}(\mathcal{Q}_\pi) \).

Since \( \tau(t) \) is not defined in the initial step, \( \pi(t) \in \text{Aux}^\mathcal{M} \) and there exists \( R(s, t) \in \mathcal{Q}(\bar{a}) \). Since \( \tau(t) \) is not defined in the induction step, \( \tau(s) \) is undefined. We can trace back the undefined terms in an infinite chain. However, \( \mathcal{Q}_\pi \) is finite, and thus there must be a loop of undefined terms, which contradicts (b).

It is easy to show that \( \mathcal{J} \models^* \mathcal{Q}_\pi \). For each atom \( A(t) \in \mathcal{Q}_{\pi^i}, \pi(t) \in A^\mathcal{M} \). Since \( \text{tail}(\tau(t)) = \pi(t) \), and by Lemma 3.9, \( \text{tail}(\tau(t)) \in A^\mathcal{M} \) implies \( \tau(t) \in A^\mathcal{J} \). For each atom \( R(t, t') \in \mathcal{Q}_\pi \), if \( \pi(t), \pi(t') \in \text{Ind}^\mathcal{M} \), then \( \tau(t) = \pi(t) \) and \( \tau(t') = \pi(t') \), and thus \( (\tau(t), \tau(t')) \in R^\mathcal{J} \). Otherwise, as \( \mathcal{M} \) is split, \( \pi(t') \in \text{Aux}^\mathcal{M} \).
From the construction of $\tau$, $\tau(t') = \tau(t) \cdot S \cdot \pi(t')$ with $T \models S \subseteq R$. By the definition of $J$, $(\tau(t), \tau(t')) \in R^J$.

Finally, we can extend $\tau$ to a match of $Q(\bar{a})$ in $J$ by setting for each term $t \in \text{term}(Q(\bar{a})) \setminus \text{term}(Q_\pi)$, $\tau(t) = \tau(t')$ for some $t' \in \text{term}(Q_\pi)$ with $t \sim_\pi t'$.

It is clear that $J \models^\tau Q(\bar{a})$ under such an extension. 

Now, we are ready to show Theorem 3.5

Proof. For each solution $E \in \text{sol}(P)$, by Definition 3.1, we have $\text{pred}(E) \subseteq \Sigma$, $\text{const}(E) \subseteq \Delta$, and $E \not\models Q(\bar{a})$. As $Q^*(\bar{a})$ strengthens $Q(\bar{a})$ (with more conjuncts), $E \not\models Q^*(\bar{a})$. Also, $T \cup A \cup E \not\models \bot$ and $T \cup A \cup E \models Q(\bar{a})$.

By Proposition 3.6, $D_T \cup A \cup E \models Q^*(\bar{a})$. Thus, $E \in \text{sol}(P')$. Further, by Lemma 3.4, $D_T \cup A \cup E \not\models \bot$. Hence, $\text{sol}(P) \subseteq \{E \in \text{sol}(P') | D_T \cup A \cup E \not\models \bot\}$.

Conversely, for each solution $E \in \text{sol}(P')$ such that $D_T \cup A \cup E \not\models \bot$, $\text{pred}(E) \subseteq \Sigma$, $\text{const}(E) \subseteq \Delta$, and $E \not\models Q^*(\bar{a})$. Since $\phi_1$ and $\phi_2$ in $Q^*(\bar{a})$ concern only (spurious matches caused by) auxiliary constants and $E$ does not contain such constants, it is clear that $E \not\models Q(\bar{a})$. Also, $D_T \cup A \models Q^*(\bar{a})$. By Lemma 3.4, $T \cup A \cup E \not\models \bot$. Then, by Proposition 3.6, $T \cup A \cup E \models Q(\bar{a})$. That is, $E \in \text{sol}(P)$. Thus, $\{E \in \text{sol}(P') | D_T \cup A \cup E \not\models \bot\} \subseteq \text{sol}(P)$. 

$\square$
Chapter 4

Optimisations and Implementations

“When somebody has learned how to program a computer ... You’re joining a group of people who can do incredible things. They can make the computer do anything they can imagine.”

–Tim Berners-Lee

Reducing a QAP in $\mathcal{ELH}_\perp$ to an equivalent one in datalog allows us to use a resolution-based procedure, like SLD-resolution, to generated solution candidates $\mathcal{E}$. Let $\Lambda$ be the set of all facts constructed from the predicates in $\Sigma$ and the constants in $\Delta$. Then, all the candidates for minimal solutions can be generated by enumerating the SLD-proofs of goal $Q(\vec{a})$ w.r.t. $\mathcal{D}_T$ and $\mathcal{A} \cup \Lambda$. We will speak informally about (SLD-)resolution proofs of $Q^*(\vec{a})$, which are SLD-proofs of $Q(\vec{a})$ validated against the additional filtering conditions in $Q^*(\vec{a})$. Practically, $Q^*(\vec{a})$ can be encoded in Prolog, and hence it makes sense to talk about resolution proofs of $Q^*(\vec{a})$ and the solution candidates generated from them. We will omit $\vec{a}$ in the (rewritten) observation for simplicity. To
verify the non-triviality of a solution $\mathcal{E}$, existing query abduction algorithms [70] often use a DL reasoner like HermiT or Pellet$^1$. Theorem 3.5 shows that we can use a datalog engine for such verifications, which implements a forward chaining procedure and is often more efficient in verification than a backward chaining procedure like resolution.

4.1 Optimisations

4.1.1 Module Extraction

Although reducing a QAP in $\mathcal{ELH}_\bot$ to a QAP in datalog allows us to apply rule-based reasoning and highly efficient rule-based systems to compute solutions, a resolution procedure can still be computationally very expensive especially when handling a large number of facts. In such cases, even if there are a small number of datalog rules, a resolution procedure may suffer a great computation overhead in attempting to find valid substitution among the many constants. This is illustrated with the following example.

Example 4.1. Consider the BCQ $Q = \exists x. [A(x) \land B(x) \land C(x)]$, datalog program $\mathcal{D} = \{R(x,y) \land D(y) \rightarrow B(x), E(x) \rightarrow F(x)\}$, and ABox $\mathcal{A} = \{A(a), C(a), D(b), A(c_1), \ldots, A(c_m), E(d_1), \ldots, E(d_n)\}$. Let $\Sigma = \{R\}$ and $\Delta$ consist of all the individuals in $\mathcal{A}$. Then the QAP $\mathcal{P} = \langle \mathcal{D}, \mathcal{A}, Q, \Sigma, \Delta \rangle$ has a single solution $\{R(a,b)\}$. Yet in the computation of (all minimal) solutions, a resolution procedure will attempt to resolve $A(x)$ to each $A(c_i)$ ($1 \leq i \leq m$). Then, after resolving $B(c_i)$ to $R(c_i, y) \land D(y)$ via the rule $R(x, y) \land D(y) \rightarrow B(x)$, it will attempt to resolve $y$ in $R(c_i, y)$ with all the individuals including

$^1$We used fully-fledged reasoner since the current version of ELK, which is an OWL EL reasoner, does not support full ABox reasoning as well as domain and range axioms, we did not use it for our experiments. The reason that we choose Pellet rather than HermiT is that, we are going to have a fair comparison of ABEL with previous system which used Pellet as its main DL reasoner.
all the \( d_j \)'s (1 ≤ j ≤ n). Such an effort only fails eventually when trying to resolve say \( D(d_j) \) and \( C(c_i) \).\(^2\)

When the ABox is large, a huge number of similar unfortunate attempts may occur, and as a result the computation suffers.

Hence, we want to reduce the number of input facts by extracting modules of the initial QAP components that are sufficient for the query abduction, which would essentially reduce the search spaces for resolution. We achieve this through data summarisation, a technique used in query answering [71, 72, 73]. Yet instead of using the result of summarisation directly for reasoning, we apply a novel method to extract modules of the initial QAP components. The main idea of data summarisation is to group the individuals in the ABox according to their occurrences in the ABox and merge individuals within the same group, such that one representative individual remains for each group in the result of summarisation. Individuals are grouped according to their types in the ABox, and a type is a finite set of concept names an individual is asserted to be a member. For an ABox \( \mathcal{A} \) and an individual \( a \) in \( \mathcal{A} \), the type of \( a \) in \( \mathcal{A} \) is \( \tau = \{ A \mid A(a) \in \mathcal{A}, A \in N_C \} \). Then, an ABox summary is defined as follows.

**Definition 4.1.** For an ABox \( \mathcal{A} \), for each type \( \tau \) of an individual in \( \mathcal{A} \), assign a distinct fresh individual \( c_\tau \) to \( \tau \). Let \( \delta \) map each individual \( a \) in \( \mathcal{A} \) to \( c_\tau \) where \( \tau \) is the type of \( a \) in \( \mathcal{A} \). Then, \( \delta(\mathcal{A}) \) is the summary of \( \mathcal{A} \).

In practice, the number of types of individuals in a dataset is often much smaller than the number of individuals. As individuals with the same type tend to behave in a similar way in reasoning, merging them in data summarisation often leads to a compact and tight approximation of the initial ABox.

\(^2\)We assume the resolution proceeds from left to right as implemented in most Prolog engines, which may not be the case for every resolution procedure. Indeed, certain optimisation with ordering could help in this specific example, yet the example is to demonstrate a general issue not necessarily tied to ordering.
For a QAP $\mathcal{P}' = \langle \mathcal{D}_T, \mathcal{A}, \mathcal{Q}^*, \Sigma, \Delta \rangle$, we first show how to use the ABox summary to extract two modules of the ABox $\mathcal{A}$, one for generating QAP solutions and the other for verifying non-triviality. After that, we will show how to extract modules of other components of the QAP. We extend mapping $\delta$ to individuals $a \in \Delta$ not occurring in $\mathcal{A}$ in a way that $\delta(c) = c$. Define the following modules of $\mathcal{A}$ using ABox summary:

$$\mathcal{A}_Q = \{ \alpha \in \mathcal{A} \mid \delta(\alpha) \in SLD(\delta(\mathcal{Q}^*), \mathcal{D}_T, \delta(\mathcal{A} \cup \Lambda)) \},$$

$$\mathcal{A}_\bot = \{ \alpha \in \mathcal{A} \mid \delta(\alpha) \in SLD(\bot, \mathcal{D}_T, \delta(\mathcal{A} \cup \Lambda)) \}.$$

Module $\mathcal{A}_Q$ is for generating candidate solutions, whereas module $\mathcal{A}_\bot$ is for verifying non-triviality. To extract $\mathcal{A}_Q$, we start with QAP $\mathcal{P}'' = \langle \mathcal{D}_T, \delta(\mathcal{A}), \delta(\mathcal{Q}^*), \Sigma, \delta(\Delta) \rangle$, and generate solution candidates to $\mathcal{P}''$ with a resolution-based procedure. Note that using ABox summary $\delta(\mathcal{A})$ instead of $\mathcal{A}$ can largely reduces the number of input facts. In the process of generating solution candidates, we keep track of the facts in $\delta(\mathcal{A})$ that are involved in the resolution. Finally, the corresponding facts in $\mathcal{A}$ are extracted to form the module $\mathcal{A}_Q$. Module $\mathcal{A}_\bot$ is extracted analogously.

In the same resolution and solution generation processes, we can also extract modules for other components of the QAP. In particular, define

$$\mathcal{D}_Q = \{ r \in \mathcal{D}_T \mid r \in SLD(\delta(\mathcal{Q}^*), \mathcal{D}_T, \delta(\mathcal{A} \cup \Lambda)) \},$$

$$\mathcal{D}_\bot = \{ r \in \mathcal{D}_T \mid r \in SLD(\bot, \mathcal{D}_T, \delta(\mathcal{A} \cup \Lambda)) \},$$

$$\Sigma_Q = \{ P \in \Sigma \mid \text{a fact } \alpha \text{ containing } P \text{ exists s.t. }$$

$$\alpha \in SLD(\delta(\mathcal{Q}^*), \mathcal{D}_T, \delta(\mathcal{A} \cup \Lambda)) \},$$

$$\Delta_Q = \{ a \in \Delta \mid \text{a fact } \alpha \text{ containing } \delta(a) \text{ exists s.t. }$$

$$\alpha \in SLD(\delta(\mathcal{Q}^*), \mathcal{D}_T, \delta(\mathcal{A} \cup \Lambda)) \}. $$
Note that $A_Q$, $D_Q$, $\Sigma_Q$ and $\Delta_Q$ can be extracted in one go via the resolution of $\delta(Q^*)$, and $A_\bot$ and $D_\bot$ can be extracted via the resolution of $\bot$.

**Example 4.2** (Cont’d Example 4.1). There are four types in $A$: $\{A,C\}$ for $a$, $\{D\}$ for $b$, $\{C\}$ for each $c_i$ ($1 \leq i \leq m$), and $\{E\}$ for each $d_j$ ($1 \leq j \leq n$). Using $a$ and $b$ themselves as representatives, and $c$ and $d$ as representatives for $c_i$’s and $d_j$’s, respectively, $\delta(A) = \{A(a), C(a), D(b), A(c), E(d)\}$. Then, $A_Q = \{A(a), C(a), D(b)\}$, $A_\bot = \emptyset$, $D_Q = \{R(x,y) \land D(y) \rightarrow B(x)\}$, $D_\bot = \emptyset$, $\Sigma_Q = \{R\}$, $\Delta_Q = \{a,b\}$.

The following theorem shows that the above modules are sufficient for query abduction.

**Theorem 4.2.** Let $\mathcal{P} = \langle \mathcal{T}, A, Q, \Sigma, \Delta \rangle$ be a QAP in $\mathcal{ELH}_\bot$. Then, $\mathcal{E}$ is a minimal solution in $\text{sol}(\mathcal{P})$ iff the following conditions hold:

1. $\text{pred}(\mathcal{E}) \subseteq \Sigma_Q$ and $\text{const}(\mathcal{E}) \subseteq \Delta_Q$;
2. $D_Q \cup A_Q \cup \mathcal{E} \models Q^*$;
3. $D_\bot \cup A_\bot \cup \mathcal{E} \not\models \bot$;
4. $\mathcal{E} \not\models Q$; and
5. no $\mathcal{E}' \subset \mathcal{E}$ exists satisfying conditions 1–4.

In Example 4.2, $\mathcal{E} = \{R(a,b)\}$ and it satisfies the conditions in Theorem 4.2. $\mathcal{E}$ can be obtained via a SLD-proof of $Q$ (the same as $Q^*$) w.r.t. $D_Q$ and $A_Q \cup \Lambda_Q$, where $\Lambda_Q$ is the set of all facts over $\Sigma_Q$ and $\Delta_Q$. Then, $\mathcal{E}$ is verified against non-triviality conditions $D_\bot \cup A_\bot \cup \mathcal{E} \not\models \bot$ and $\mathcal{E} \not\models Q$, and is returned as a solution.

For a BCQ $Q$, a datalog program $D$, and a set $\mathcal{F}$ of facts, denote an SLD-proof of $Q$ w.r.t. $D$ and $A$ as $\Gamma = G_0 \not\models^r_{\sigma_1} G_1 \not\models^r_{\sigma_2} \cdots \not\models^r_{\sigma_n} G_n$, where each
\(G_i\) is obtained from \(G_{i-1}\) and \(r_i \in D\) with unifier \(\sigma_i\) for \(1 \leq i \leq n\). Denote \(R(\Gamma) = \{r_1, \ldots, r_n\}\) and \(F(\Gamma) = G_n\). We say \(\Gamma\) is a SLD-proof of \(Q^*\) if the unifications applied in \(\Gamma\) satisfies the filter conditions in \(Q^*\).

To prove Theorem 4.2, we first show the following results. We extend substitution \(\delta\) in data summary such that for each term \(t\) (i.e., constant or variable) not occurring in the initial ABox \(A\) such that \(\delta(t) = t\).

**Lemma 4.3.** For each SLD-proof \(\Gamma\) of \(Q^*\) w.r.t. \(D_T\) and a set \(F\) of facts, \(\delta(\Gamma)\) is a SLD-proof of \(\delta(Q^*)\) w.r.t. \(D_T\) and \(\delta(F)\), such that \(R(\delta(\Gamma)) = R(\Gamma)\) and \(F(\delta(\Gamma)) = \delta(F(\Gamma))\).

The above lemma is easy to see, as for \(\Gamma = G_0 \sim_{\sigma_1} G_1 \sim_{\sigma_2} \ldots \sim_{\sigma_n} G_n\), \(\delta(\Gamma) = \delta(G_0) \sim_{\delta(\sigma_1)} \delta(G_1) \sim_{\delta(\sigma_2)} \ldots \sim_{\delta(\sigma_n)} \delta(G_n)\) is a SLD-proof with each \(\delta \cdot \sigma_i\) a most general unifier. Moreover, since filter conditions concern only auxiliary constants introduced in Table 3.2, \(\Gamma\) satisfying these conditions implies that \(\delta(\Gamma)\) also satisfies them.

**Proposition 4.4.** Let \(T, A, Q, \Sigma, \Delta\) be as in Theorem 4.2. Then, for a set \(E\) of facts such that \(\text{pred}(E) \subseteq \Sigma\) and \(\text{const}(E) \subseteq \Delta\), \(D_T \cup A \cup E \models Q^*\) iff \(D_Q \cup A_Q \cup E \models Q^*\); and if \(E\) is a minimal set satisfying the above condition then \(\text{pred}(E) \subseteq \Sigma_Q\) and \(\text{const}(E) \subseteq \Delta_Q\).

**Proof.** The “if” direction is clear from monotonicity. We only need to show the “only if” direction. From the completeness of SLD-resolution, \(D_T \cup A \cup E \models Q^*\) implies that there is an SLD-proof \(\Gamma\) of \(Q^*\) w.r.t. \(D_T\) and \(A \cup E\), and \(R(\Gamma) \cup F(\Gamma) \models Q^*\). By Lemma 4.3, \(\delta(\Gamma)\) is a SLD-proof of \(\delta(Q^*)\) w.r.t. \(D_T\) and \(\delta(A) \cup \delta(E)\). Since \(R(\delta(\Gamma)) \subseteq SLD(\delta(Q^*), D_T, \delta(A \cup \Lambda))\), \(R(\Gamma) \subseteq D_Q\). Similarly, as \(F(\delta(\Gamma)) \subseteq SLD(\delta(Q^*), D_T, \delta(A \cup \Lambda))\), we have \(F(\Gamma) \subseteq A_Q \cup E\). Hence, \(D_Q \cup A_Q \cup E \models Q^*\). Moreover, given the minimality of \(E\), we have \(\delta(E) \subseteq F(\delta(\Gamma))\), and thus \(\text{pred}(E) \subseteq \Sigma_Q\) and \(\text{const}(E) \subseteq \Delta_Q\). \(\square\)
Chapter 4. Optimisations and Implementations

Note that Proposition 4.4 applies when $Q = \exists x. \bot(x)$, using the same proof as $Q^* = Q$. Now, we are ready to prove Theorem 4.2.

Proof. The “if” direction: For each set $E$ satisfying the conditions, by conditions 1 and 4, clearly $\text{pred}(E) \subseteq \Sigma$, $\text{const}(E) \subseteq \Delta$, and $E \not\models Q^*$. Also, by condition 2 and monotonicity, $\mathcal{D}_\mathcal{T} \cup \mathcal{A} \cup E \models Q^*$. That is, $E \in \text{sol}(\mathcal{P}')$ where $\mathcal{P}'$ is as in Theorem 3.5. Further, by condition 3 and Proposition 4.4, $\mathcal{D}_\bot \cup \mathcal{A}_\bot \cup E \not\models \bot$ implies that $\mathcal{D}_\mathcal{T} \cup \mathcal{A} \cup E \not\models \bot$. By Theorem 3.5, $E \in \text{sol}(\mathcal{P})$. And $E$ is a minimal solution by condition 5.

The “only if” direction: For each solution $E \in \text{sol}(\mathcal{P})$, by Definition 3.1, $\text{pred}(E) \subseteq \Sigma$, $\text{const}(E) \subseteq \Delta$, and $E \not\models Q$. Condition 4 is satisfied. By Theorem 3.5, we have $\mathcal{D}_\mathcal{T} \cup \mathcal{A} \cup E \models Q^*$ and $\mathcal{D}_\mathcal{T} \cup \mathcal{A} \cup E \not\models \bot$. By monotonicity, $\mathcal{D}_\bot \cup \mathcal{A}_\bot \cup E \not\models \bot$; that is condition 3. Further, by Proposition 4.4, $\mathcal{D}_Q \cup \mathcal{A}_Q \cup E \models Q^*$; and that is condition 2. Also, by the minimality of $E$ $\text{pred}(E) \subseteq \Sigma_Q$ and $\text{const}(E) \subseteq \Delta_Q$; that is condition 1. Finally, condition 5 holds, as otherwise from the proof of the “if” direction, $E' \in \text{sol}(\mathcal{P})$ and $E$ is not minimal.

4.1.2 Solution Search with Pruning

After preprocessing of the initial QAP, we obtain a datalog QAP of relatively small size. Our basic approach systematically searches for solution candidates using a resolution-based procedure, verifies each candidate using a datalog engine, and keeps only the minimal solutions to display to the user. Besides systematically enumerating the resolution proofs, the search for solutions can largely benefit from advanced pruning methods. In particular, when a solution $E$ is computed, instead of forgetting about $E$ and constructing a new resolution proof, a pruning method further exploits $E$ to generate another candidate solution $E'$. When $E'$ is verified to be a solution, the resolution proofs for $E'$
and its supersets are pruned. Various heuristics can be applied in pruning. In [30], a pruning strategy that exploits the subsets of $E$ is proposed, which allows the minimality of a solution to be checked immediately after its generation. Although such a pruning strategy for query abduction may require a larger number of verifications than the basic approach, such verifications are often less expensive than resolution. Given that the candidates $E'$ obtained from $E$ are close enough to valid solutions and can be enumerated efficiently, search with pruning can largely outperform a pure resolution one.

Once a minimal solution $E$ is computed, we propose a new pruning strategy that exploits $E$ and the hierarchical knowledge from the TBox. A hierarchical axiom in $E LH$ is of the form $A \sqsubseteq B$ or $R \sqsubseteq S$. Such hierarchical axioms often constitute a large portion of practical ontology axioms, and can cause computation difficulty in resolution. This is illustrated in the following example.

**Example 4.3.** Consider the BCQ $Q = A_1(a_1) \land \ldots \land A_1(a_m)$ and the datalog program $D = \{ A_{i+1}(x) \rightarrow A_i(x) \mid 1 \leq i \leq n \}$. Let $A = \emptyset$, $\Sigma = \{ A_1, \ldots, A_n \}$, and $\Delta = \{ a_1, \ldots, a_m \}$. Then, the QAP $P = \langle D, A, Q, \Sigma, \Delta \rangle$ has $n^m$ minimal solutions of the form $\{ A_{j_1}(a_1), \ldots, A_{j_m}(a_m) \mid 1 \leq j_1, \ldots, j_m \leq n \}$.

Constructing a resolution proof for each of the minimal solution is a tedious job and involves a large amount of repetition. However, from one minimal solution $\{ \ldots, A_j(a_i), \ldots \}$, it is very convenient to directly construct another minimal solution $\{ \ldots, A_{j+1}(a_i), \ldots \}$ using the hierarchical rule $A_{j+1}(x) \rightarrow A_j(x)$.

Based on this observation, we propose the following pruning technique that makes use of the hierarchical knowledge on concept names and role names in the TBox. Consider the QAP $P' = \langle D_T, A, Q', \Sigma, \Delta \rangle$ and the modules of its components $D_Q$, $A_Q$, $\Sigma_Q$, and $\Delta_Q$. For a solution $E \in \text{sol}(P)$, a weakening of $E$ is obtained by replacing some $A(a) \in E$ with $B(a)$ such that $B \in \Sigma_Q$ and $T \models A \sqsubseteq B$, or by replacing some $R(a,b) \in E$ with $S(a,b)$ such that
Chapter 4. Optimisations and Implementations

67

\[ S \in \Sigma_Q \text{ and } T \models R \subseteq S; \] a strengthening of \( \mathcal{E} \) is obtained by replacing some \( A(a) \in \mathcal{E} \) with \( B(a) \) such that \( B \in \Sigma_Q \) and \( T \models B \subseteq A \), or by replacing some \( R(a,b) \in \mathcal{E} \) with \( S(a,b) \) such that \( S \in \Sigma_Q \) and \( T \models S \subseteq R \). \( \mathcal{E}_\uparrow \) and \( \mathcal{E}_\downarrow \) denote a weakening and a strengthening of \( \mathcal{E} \), respectively.

**Proposition 4.5.** Let \( \mathcal{P} = \langle T, A, Q, \Delta \rangle \) be a QAP in \( \mathcal{ELH}_\perp \). For a solution \( \mathcal{E} \in \text{sol}(\mathcal{P}) \), the following conditions hold:

- \( \mathcal{E}_\uparrow \in \text{sol}(\mathcal{P}) \) iff \( D_Q \cup A_Q \cup \mathcal{E}_\uparrow \models \mathcal{Q}^* \) and \( \mathcal{E}_\uparrow \not\models \mathcal{Q} \); and \( \mathcal{E}_\uparrow \) is minimal if \( \mathcal{E} \) is minimal.

- \( \mathcal{E}_\downarrow \in \text{sol}(\mathcal{P}) \) iff \( D_\perp \cup A_\perp \cup \mathcal{E}_\downarrow \not\models \perp \) and \( \mathcal{E}_\downarrow \not\models \mathcal{Q} \); and \( \mathcal{E}_\downarrow \) is minimal only if \( \mathcal{E} \) is minimal.

To prove Proposition 4.5, we have the following observation, which says a weakening of \( \mathcal{E} \) is logically weaker than \( \mathcal{E} \) whereas a strengthening of \( \mathcal{E} \) is logically stronger than \( \mathcal{E} \).

**Lemma 4.6.** For a TBox \( T \) and a set \( \mathcal{E} \) of facts, \( T \cup \mathcal{E} \models \mathcal{E}_\uparrow \) and \( T \cup \mathcal{E}_\downarrow \models \mathcal{E} \).

Now by Lemma 4.6 according to our observation, we can prove Proposition 4.5.

**Proof.** For the first condition, to show the “if” direction, suppose \( D_Q \cup A_Q \cup \mathcal{E}_\uparrow \models \mathcal{Q}^* \) and \( \mathcal{E}_\uparrow \not\models \mathcal{Q} \), by Proposition 4.4, \( D_T \cup A \cup \mathcal{E}_\uparrow \models \mathcal{Q}^* \). Since \( \mathcal{E} \in \text{sol}(\mathcal{P}) \), by Theorem 3.5, \( \text{pred}(\mathcal{E}) \subseteq \Sigma, \text{const}(\mathcal{E}) \subseteq \Delta \), and \( D_T \cup A \cup \mathcal{E} \not\models \perp \). Then, \( \text{pred}(\mathcal{E}_\uparrow) \subseteq \Sigma \) and \( \text{const}(\mathcal{E}_\uparrow) \subseteq \Delta \). That is, \( \mathcal{E}_\uparrow \in \text{sol}(\mathcal{P}') \), where \( \mathcal{P}' \) is as in Theorem 3.5. By Lemma 4.6, \( D_T \cup A \cup \mathcal{E}_\uparrow \not\models \perp \). By Theorem 3.5, \( \mathcal{E}_\uparrow \in \text{sol}(\mathcal{P}) \). To show the “only if” direction, suppose \( \mathcal{E}_\uparrow \in \text{sol}(\mathcal{P}) \), by Theorem 3.5, \( D_T \cup A \cup \mathcal{E}_\uparrow \models \mathcal{Q}^* \) and \( \mathcal{E}_\uparrow \not\models \mathcal{Q}^* \). That is, \( \mathcal{E}_\uparrow \not\models \mathcal{Q} \). Also, by Proposition 4.4, \( D_Q \cup A_Q \cup \mathcal{E}_\uparrow \models \mathcal{Q}^* \).

To show the second half of the first condition, suppose \( \mathcal{E}_\downarrow \) is not minimal, then a subset \( \mathcal{E}' \subseteq \mathcal{E}_\downarrow \) exists such that \( \mathcal{E}' \in \text{sol}(\mathcal{P}) \). We want to show that \( \mathcal{E} \) is
not minimal either. By Theorem 3.5, \( D_T \cup A \cup E' \models Q^* \). Without loss of generality, suppose \( E_\uparrow = E \setminus \{A(a)\} \cup \{B(a)\} \). If \( B(a) \not\in E' \) then \( E' \subset E \) and \( E \) is not minimal. Otherwise, let \( E'_\downarrow = E' \setminus \{B(a)\} \cup \{A(a)\} \), and thus \( E'_\downarrow \subset E \). Clearly, \( \text{pred}(E'_\downarrow) \subseteq \Sigma \) and \( \text{const}(E'_\downarrow) \subseteq \Delta \). By monotonicity, \( E'_\downarrow \not\models Q \) and \( D_T \cup A \cup E'_\downarrow \not\models \bot \) as \( E \) does. By Lemma 4.6, \( D_T \cup A \cup E'_\downarrow \models Q^* \) as \( E' \) does. That is, \( E'_\downarrow \in \text{sol}(P') \) and by Theorem 3.5, \( E'_\downarrow \in \text{sol}(P) \). Hence, \( E \) is not minimal.

For the second condition, to show the “if” direction, suppose \( D_Q \cup A_Q \cup E_\downarrow \not\models \exists x. \text{bot}(x) \) and \( E_\downarrow \not\models Q \), by Proposition 4.4, \( D_T \cup A \cup E_\downarrow \not\models \exists x. \text{bot}(x) \). Since \( E \in \text{sol}(P) \), by Theorem 3.5, \( \text{pred}(E) \subseteq \Sigma \), \( \text{const}(E) \subseteq \Delta \), and \( D_T \cup A \cup E \models Q^* \). Then, \( \text{pred}(E_\downarrow) \subseteq \Sigma \) and \( \text{const}(E_\downarrow) \subseteq \Delta \). By Lemma 4.6, \( D_T \cup A \cup E_\downarrow \models Q^* \). By Theorem 3.5, \( E_\downarrow \in \text{sol}(P) \). To show the “only if” direction, suppose \( E_\downarrow \in \text{sol}(P) \), by Theorem 3.5, \( D_T \cup A \cup E_\downarrow \not\models \bot \) and \( E_\downarrow \not\models Q^* \). That is, \( E_\downarrow \not\models Q \). Also, by Proposition 4.4, \( D_Q \cup A_Q \cup E_\downarrow \not\models \bot \).

To show the second half of the second condition, suppose \( E \) is not minimal, then a subset \( E' \subset E \) exists such that \( E' \in \text{sol}(P) \). We want to show that \( E_\downarrow \) is not minimal either. By Theorem 3.5, \( D_T \cup A \cup E' \models Q^* \). Without loss of generality, suppose \( E_\downarrow = E \setminus \{A(a)\} \cup \{B(a)\} \). If \( B(a) \not\in E' \) then \( E' \subset E_\downarrow \) and \( E_\downarrow \) is not minimal. Otherwise, let \( E'_\downarrow = E' \setminus \{B(a)\} \cup \{A(a)\} \), and thus \( E'_\downarrow \subset E_\downarrow \). Clearly, \( \text{pred}(E'_\downarrow) \subseteq \Sigma \) and \( \text{const}(E'_\downarrow) \subseteq \Delta \). By monotonicity, \( E'_\downarrow \not\models Q \) and \( D_T \cup A \cup E'_\downarrow \not\models \bot \) as \( E_\downarrow \) does. By Lemma 4.6, \( D_T \cup A \cup E'_\downarrow \models Q^* \) as \( E' \) does. That is, \( E'_\downarrow \in \text{sol}(P') \) and by Theorem 3.5, \( E'_\downarrow \in \text{sol}(P) \). Hence, \( E_\downarrow \) is not minimal.

Note that in the second condition, the minimality of \( E_\downarrow \) is not guaranteed by the second half of the condition. For example, let \( Q = D(a) \), \( D_Q = \{B(x) \rightarrow A(x), A(x) \land C(x) \rightarrow D(x), B(x) \rightarrow D(x)\} \), \( \Sigma_Q = \{A, B, C\} \), \( \Delta_Q = \{a\} \), and \( D_\bot \) and \( A_Q \cup A_\bot \) are both empty. Then, a minimal solution is \( E = \{A(a), C(a)\} \), whereas \( E_\downarrow = \{B(a), C(a)\} \) is a solution but not minimal.
Once a solution $E$ is computed, our pruning method checks the minimality of $E$ as in [30]. If $E$ is minimal, the method enumerates and verifies whether each $E^\uparrow$ and each $E^\downarrow$ is a minimal solution. $E^\uparrow$ and $E^\downarrow$ are enumerated using the concept and role hierarchies pre-computed from $T$ using a OWL (EL) reasoner. Verifications of $E^\uparrow$ and $E^\downarrow$ are performed efficiently using a datalog engine.

### 4.2 Implementation

We have shown that a QAP in $\mathcal{ELH}_\bot$ can be transformed into a QAP in datalog such that the solutions to the datalog QAP are exactly the solutions to the initial QAP. Like many other procedures of computing abductive solutions, a resolution-based algorithm is needed for our transformation-based procedure. However, most existing datalog engines are not resolution-based. We implemented our new procedure of computing QAP solutions using Prolog and make use of the in-built list structure in Prolog engine XSB [67]. To compute (minimal) solutions to a QAP $\langle D_T, A, Q^*(\vec{a}), \Sigma, \Delta \rangle$, we encode datalog program $D_T$ and FO query $Q^*(\vec{a})$ into Prolog rules, and use the list structure in Prolog to store the solutions generated during the resolution. Our encoding is adapted from and extended that of the approach proposed by Du et al. [30, 33], and we have also employed the database feature of XSB to allow the initial ABox to be stored in a database which significantly improves the efficiency.

Similarly to the work [30] by Du et al., we assign a binary predicate $p_A$ for each $A \in N_C$ and a ternary predicate $p_r$ for each $r \in N_R$. The increase of arity is for an extra parameter to store solutions. In particular, the solutions are stored and manipulated in an in-built list structure in Prolog. Hence, we have atoms of the forms $p_A(X,L)$ and $p_r(X,Y,L)$ where $X, Y$ are variables and $L$ is a list. To encode $D_T$, we encode each datalog rule as a Prolog rule.
For example, a datalog rule $r(x, y) \land B(y) \rightarrow A(x)$ is encoded as

$$p_A(X, L) : -p_r(X, Y, L1), p_B(Y, L2), \text{merge}(L1, L2, L).$$

where $\text{merge}(L1, L2, L)^3$ merges lists $L1$ and $L2$ to $L$.

In contrast to the previous encoding [30, 33], our datalog rules may contain constants, i.e., in rules of the forms $A(x) \rightarrow r(x, c)$ and $A(x) \rightarrow B(c)$. Firstly, for each auxiliary constant $c$, we assert $p_{\text{Aux}}(c)$ as a fact. Also, the rule $A(x) \rightarrow B(c)$ often dramatically affects the efficiency of the resolution. More specifically, when both $r(a, c)$ and $B(c)$ are present, the resolution procedure attempts to resolve $x$ with (all) other constants $b$ regardless whether $r(b, c)$ is presented or not. Such attempts often result in non-minimal solutions but generate a large number of redundant facts $A(b)$ consuming a large amount of time and memory. To address this issue, we encode the rule $A(x) \rightarrow B(c)$ as a fact $p_B(c, \text{['asserted']}])$. Such a fact needs no special treatment during the resolution. Instead, we check during the post-processing whether a fact $A(a)$ exists for some constant $a$ in the initial ABox or the solution. The solution is valid only if such a fact $A(a)$ exists; otherwise the solution is discarded.

Facts in the initial ABox like $r(a, b)$ and $A(a)$ are encoded as $p_r(a, b, [])$. and $p_A(a, [])$, respectively, where [] is an empty list. Moreover, a fact $p_{\text{Ind}}(a)$. is added to the program for each individual $a$ occurring in the initial ABox. For the lists to store solutions consisting of only facts constructed from predicates and constants in $\Sigma$ and $\Delta$, we introduce a unary predicate $\text{ab\_ind}$ to assert the individuals in $\Delta$, and add the following Prolog rules:

- for each individual $a \in \Delta$, $\text{ab\_ind}(X)$.
- for each concept name $A \in \Sigma$, $p_A(X, [p_A(X)]) :- \text{ab\_ind}(X)$.

---

^3We use $\text{merge}/3$ instead of $\text{append}/3$ as the former returns a merged list without repetitive elements, which is convenient for computing minimal solutions.
• for each role name \( r \in \Sigma \), \( p_r(X,Y,[p_r(X,Y)]) :- ab\_ind(X),\ ab\_ind(Y). \)

To encode observation \( Q^*(\vec{a}) \), we first show how to encode CQ \( Q(\vec{x}) \). Suppose 
\( Q(\vec{x}) = \exists \vec{y}. [\bigwedge_{1 \leq i \leq n} P_i(\vec{t}_i)] \) where each \( P_i(\vec{t}_i) \) is an atom, it can be directly encoded as a Prolog rule

\[
p_Q(\vec{X},L) :- p_{P_1}(\vec{T}_1,L_1), \ldots, p_{P_n}(\vec{T}_n,L_n), \text{merge}(L_1,\ldots,L_n,L).
\]

(4.1)

where \( \vec{X} \) and \( \vec{T}_i \) correspond to \( \vec{x} \) and \( \vec{t}_i \) (\( 1 \leq i \leq n \)), and \( \text{merge}(L_1,\ldots,L_n,L) \) is an abbreviation, e.g., \( \text{merge}(L_1,L_2,L_3,L) \) abbreviates \( \text{merge}(L_1,L_2,L_{12}) \) and \( \text{merge}(L_{12},L_3,L) \). For the filter queries \( \phi_1 \) and \( \phi_2 \), the sets \textbf{Fork} and \textbf{Cyc} are pre-computed. The encoding of \( Q^*(\vec{x}) \) is obtained by adding the following atoms to the body of (4.2):

• for each pair \( \{\{t_1,\ldots,t_k\},\xi\} \) in \textbf{Fork}, add

\[ p_{\text{Aux}}(T_{\xi}) \rightarrow (T_1 == T_2,\ldots,T_{k-1} == T_k);\ \text{true} \]

where \( T_{\xi} \) and \( T_i \) correspond to \( t_{\xi} \) and \( t_i \) (\( 1 \leq i \leq k \)), respectively; and

• for each variable \( v \) in \textbf{Cyc}, add \( p_{\text{Ind}}(V) \), where \( V \) corresponds to \( v \).

The symbol \( \rightarrow \) in Prolog is an \textit{If-Then-Else} command, that is, the sentence before arrow (\( \rightarrow \)) is the \textit{If} part; the sentence between arrow (\( \rightarrow \)) and the semicolon (\( ; \)) is the \textit{Then} part; and the reminder is the \textit{Else} part. If the \textit{Else} part is empty, the rule fails, therefore the \textbf{true} sentence is used.

Finally, the following rule starts the execution.

\[ \text{go} :- p_Q(\vec{a},L), \text{add}\_answer(L),\ \text{fail}. \]

The \textbf{add}\_answer\( (L) \) directive verifies \( L \) is a set minimal solution according to Definition 3.1. The \textbf{fail} directive is necessary to traverse all solutions.
Example 4.4 (Cont’d Examples 3.1 and 3.4). The QAP $\langle D_T, A, Q^*(\text{john}, \text{julie}), \Sigma, \Delta \rangle$ is encoded by the following Prolog program. In particular, rules P1–P10 encode datalog rules R1, R3, R5, R6, and S1–S4. Rules F1–F2 encode the constants in $\Delta$, and rule I initializes the solutions. Recall the rewritten CQ $Q^*(x, y) = \exists z. [\text{works}(x, z) \land \text{works}(y, z) \land (\text{Aux}(z) \rightarrow x = y)]$, and the corresponding observation $Q^*(\text{john}, \text{julie})$ is encoded as O1–O2.

\[
\begin{align*}
P1) & \quad p_{\text{Person}}(X,L) :- p_{\text{RA}}(X,L). \\
P2) & \quad p_{\text{Person}}(X,L) :- p_{\text{Student}}(X,L). \\
P3) & \quad p_{\text{tempRole}}(X,Y,L) :- p_{\text{Person}}(X,L_1), p_{\text{takes}}(X,Y,L_2), merge(L_1,L_2,L). \\
P4) & \quad p_{\text{Student}}(X,L) :- p_{\text{tempRole}}(X,Y,L_1), p_{\text{Course}}(Y,L_2), merge(L_1,L_2,L). \\
P5) & \quad p_{\text{works}}(X,c_1,L) :- p_{\text{RA}}(X,L). \\
P6) & \quad p_{\text{RG}}(c_1,[\text{RA asserted}]). \\
P7) & \quad p_{\text{Aux}}(c_1). \\
P8) & \quad p_{\text{takes}}(X,c_2,L) :- p_{\text{Student}}(X,L). \\
P9) & \quad p_{\text{Course}}(c_2,[\text{Student asserted}]). \\
P10) & \quad p_{\text{Aux}}(c_2). \\
F1) & \quad \text{ab\_ind}(\text{john}). \\
F2) & \quad \text{ab\_ind}(\text{julie}). \\
I) & \quad p_{\text{RA}}(X,[p_{\text{RA}}(X)]) :- \text{ab\_ind}(X). \\
O1) & \quad p_{\text{Q}}(X,Y,L) :- p_{\text{works}}(X,Z,L_1), p_{\text{works}}(Y,Z,L_2), (p_{\text{Aux}}(Z) \rightarrow (X==Y); \text{true}), merge(L_1,L_2,L). \\
O2) & \quad \text{go} :- p_{\text{Q}}(\text{john},\text{julie},L), \text{add\_answer}(L), \text{fail}.
\end{align*}
\]

4.3 Experimental Results

We have implemented a prototype QAP solver ABEL (ABduction for EL), which integrates the KARMA system [58] for ontology transformation, Prolog engine XSB 3.4 [67] for generating solutions, and OWL reasoner Pellet$^4$ as

$^4$clarkparsia.com/pellet
well as datalog engine RDFox\textsuperscript{5} for verifying solution candidates. We encoded QAP instances in Prolog and used the in-built list structure in Prolog to store solutions. We have evaluated our system on three ontology benchmarks and under different settings. The experimental results demonstrate that ABEL is able to efficiently process complex CQs over realistic ontologies and large data sets. All the experiments were performed on a PC with an Intel Xenon 2.8 GHz processor and 16 GiB RAM. The system and testing data are available online\textsuperscript{6}.

4.3.1 Test Cases

In our experiments, we have used the ontologies and data sets described next. All ontologies are widely used for benchmarking ontology-based CQ answering and for abduction \cite{30, 58, 74}.

Visual Contextualization of Digital Content (VICODI) ontology The VICODI ontology describes European history. We have used this benchmark since it has a simple TBox that can be mostly employed for evaluating atomic queries which is a fair ontology to compare ABEL with the previous system.

Financial ontology The financial ontology developed for the SEMINTEC project\textsuperscript{7} at the University of Poznan using the financial data. This ontology is designed for a bank manager to improve the offering services to private persons. This ontology has a more complex TBox with genuine existential rules. The ABox is moderately large that the query abduction problem is more challenging.

Extended version of Lehigh University Benchmark The LSTW \cite{74} is an extended version of the Lehigh University benchmark LUBM \cite{62} with more

\textsuperscript{5}\url{www.cs.ox.ac.uk/isg/tools/RDFox}

\textsuperscript{6}\url{http://www.ict.griffith.edu.au/~kewen/AbductionEL/}

\textsuperscript{7}\url{http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm}
existential rules. LSTW is more complex than LUBM, as it is shown that loading and reasoning of LSTW ontology is approximately 2-4 times harder than LUBM ontology [75]. For LSTW, ABoxes can be automatically generated in a range of sizes using EUGen [74]. LSTW(n) denotes the variant with an ABox of n universities. We mostly used this ontology for evaluating the scalability of ABEL. We have also modified the TBox of LSTW to contain the axiom $\exists_{\text{advisor}}.\text{Person} \sqsubseteq \text{Person}$ to make the TBox non-FO rewritable.

Table 4.1 shows some statistics on the aforementioned ontologies: the numbers of their concept names (#C), role names (#R), TBox axioms ($|\mathcal{T}|$), genuine existential rules (#E), ABox facts (|$\mathcal{A}$|), and individuals (#I).

| Ontology  | #C | #R | $|\mathcal{T}|$ | #E | $|\mathcal{A}|$ | #I |
|-----------|----|----|---------------|----|---------------|----|
| VICODI    | 194| 10 | 214          | 0  | 116,181       | 32,238|
| Financial | 60 | 16 | 207          | 8  | 65,244        | 17,945|
| LSTW(n)   | 132| 32 | 223          | 29 | $\approx10^5n$ | $\approx17\times10^3n$|

Table 4.1: Ontology statistics.

We have used three kinds of queries in our experiments.

**Standard Queries.** The LUBM ontology comes with 14 standard queries. We have adopted these queries for LSTW. LSTW does not have any data type property, so we treated all data type properties occurring in the queries as object properties.

**Comparison Queries.** For comparing ABEL with the previous system, we’ve generated 40 atomic concept or role queries from each ontology.

**Generated Queries.** We have used the system SyGENiA [76], a query generator for benchmarking CQ answering, to generate related CQs for LSTW ontology. As SyGENiA failed to terminate on some instances of LSTW ontology, so we have set a bound on the chase algorithm of the SyGENiA to generate CQs with up to three atoms. Total number of generated queries are
Observations. To generate observations, we substituted the answer variables in each query with randomly chosen individuals from the corresponding ABox and verified that each substitution was indeed a negative answer to the query.

We evaluated ABEL in three different configurations.

ABEL-Naive is the basic resolution-based system without module extraction or local search. In the ABEL-Naive, Pellet is used for the reasoning tasks such as non-triviality check and query entailment.

ABEL is the system which enables pruning. Also, ABEL checking non-triviality through RDFox and used Pellet for computing concept and role hierarchies.

ABEL-Sum has the same specifications as ABEL plus the module extraction through data summarisation.

4.3.2 Comparison

In the first set of experiments, we have compared our system with the system developed by Du et al. [70] (DuQAP). DuQAP accepts input observations as atomic queries and ontologies in $\mathcal{SHIQ}$, therefore this system is generally incomplete for answering CQs even for an $\mathcal{ELH}_\bot$ ontology. We have included all concepts in the ontology as abducibles for each ontology, and the domain includes all individuals in the ABox. We have randomly generated 40 different observations for each ontology, therefore the total number of test cases in this experiment were 200 ($40 \times 5$ ontologies). Table 4.2 shows the comparison results. The first column is the ontology name, the next one shows the average number of solutions computed ($\#\mathcal{E}$). The following columns show the average...
computation times (in seconds) for finding all minimal solutions. Note that all the three systems could complete the query abduction task on all the test cases, except that DuQAP could not handle queries over the LSTW with more than 5 universities. Also, it is worth noting that ABEL-Naive already outperformed DuQAP, and this superiority is largely enhanced with our pruning technique. The pruning method was practical since all abducibles are concept names and once XSB found one solution the rest of solutions could be found with little burden on XSB.

<table>
<thead>
<tr>
<th>Ontologies</th>
<th>#E</th>
<th>ABEL-Naive</th>
<th>ABEL</th>
<th>DuQAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTW(1)</td>
<td>8.6</td>
<td>5.1</td>
<td>0.4</td>
<td>7.8</td>
</tr>
<tr>
<td>LSTW(5)</td>
<td>8.6</td>
<td>24.8</td>
<td>1.1</td>
<td>46.9</td>
</tr>
<tr>
<td>LSTW(10)</td>
<td>8.6</td>
<td>85.9</td>
<td>5.2</td>
<td>N/A</td>
</tr>
<tr>
<td>Vicodi</td>
<td>12.4</td>
<td>5.8</td>
<td>0.1</td>
<td>9.5</td>
</tr>
<tr>
<td>Financial</td>
<td>0.4</td>
<td>1</td>
<td>0.1</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Table 4.2: Compare DuQAP and ABELs.

### 4.3.3 Scalability Test

To test the scalability of our system in finding the minimal solutions, we have conducted experiments for standard queries using LSTW data sets of increasing sizes (1, 10, 50, and 100 universities). For each query, we generated 5 negative observations. The abducibles are all concept names in the ontology and all role names in the query. The domains are limited to constants in the observations with sizes ranging from 5 to 15. The Table 4.3 shows the results. \( t_{\text{avg}} \) is the average computation time. The column \(|\mathcal{A}_Q|\) shows the average sizes of the ABox module for each query, and \( t_{\text{mod}} \) shows the time taken to compute the modules for all 5 observations of each query. The \( \mathcal{A}_\perp \) module was always empty since there is no disjointness axiom in the TBox. TO means that the system could not find all solutions in 10 minutes. All times are in seconds.
All systems could successfully process LSTW(1), while ABEL was more efficient than ABEL-Naive for most of the queries (12 out of 14). In the case of LSTW(10), ABEL could return all solutions in 10 minutes, and outperformed ABEL-Naive for all queries.

For larger LSTW ontologies, we had memory issue for both Pellet reasoner and XSB. Therefore, we store the ontology in a MySQL database to skip loading the ontology in the memory. We used ABEL-Sum to extract modules $A_Q$ of the ABox for each query, and input the module to XSB as well as RDFox to generate the solutions and verify them. We computed and stored the ABox summary off-line and retrieved relevant information in run-time. ABEL-Sum could successfully compute solutions for all queries, whereas ABEL failed for the queries Q1 and Q2 over LSTW(50) and LSTW(100). Therefore, we only included all details related to ABEL-Sum.

To challenge the ABEL, we have also conducted another round of experiments on LSTW ontology with 120 CQs generated by SyGENiA. A large portion of these queries have existentially quantified variables which are more difficult CQs rather than the CQs used in the previous experiments. We have added 5 disjointness axioms to the TBox to enforce more non-triviality checks. Namely, $Course \sqcap Person \sqsubseteq \bot$, $Professor \sqcap Organization \sqsubseteq \bot$, $Professor \sqcap Publication \sqsubseteq \bot$, $Professor\sqcap ResearchAssistant \sqsubseteq \bot$, and $Professor \sqcap Student \sqsubseteq \bot$. Note that there are 20 sub-concepts for each concept of $Student$, $Professor$, and $Course$ which will be affected by these disjointness axioms.

We also increased the abducibles to be all concept and role names in the TBox. The domain is limited to the constants in the observations plus up to 5 individuals from the ABox. The generated queries were too difficult for ABEL-Naive to handle, so we didn’t include it in this experiments. We firstly ran each test case with ABEL, in a case that it failed to find solutions we
ran it again with ABEL-Sum. Figure 4.1 shows the results. The first two columns show the success rate of ABEL and ABEL-Sum. The $t_{avg}$ is the average computation time of successful test cases. It includes the time taken to compute the modules. $t_{mod}$ is the average time taken to compute both $A_Q$.

<table>
<thead>
<tr>
<th>Query</th>
<th>ABEL-Naive</th>
<th>ABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LSTW(1)</td>
<td>LSTW(10)</td>
</tr>
<tr>
<td>Q1</td>
<td>1.68</td>
<td>37.56</td>
</tr>
<tr>
<td>Q2</td>
<td>6.03</td>
<td>118.43</td>
</tr>
<tr>
<td>Q3</td>
<td>0.74</td>
<td>20.12</td>
</tr>
<tr>
<td>Q4</td>
<td>71.62</td>
<td>TO</td>
</tr>
<tr>
<td>Q5</td>
<td>1.57</td>
<td>17.31</td>
</tr>
<tr>
<td>Q6</td>
<td>18.54</td>
<td>313.33</td>
</tr>
<tr>
<td>Q7</td>
<td>1.16</td>
<td>32.77</td>
</tr>
<tr>
<td>Q8</td>
<td>58.41</td>
<td>TO</td>
</tr>
<tr>
<td>Q9</td>
<td>6.48</td>
<td>47.6</td>
</tr>
<tr>
<td>Q10</td>
<td>0.73</td>
<td>36.1</td>
</tr>
<tr>
<td>Q11</td>
<td>1.27</td>
<td>29.48</td>
</tr>
<tr>
<td>Q12</td>
<td>1.67</td>
<td>34.01</td>
</tr>
<tr>
<td>Q13</td>
<td>2.1</td>
<td>20.06</td>
</tr>
<tr>
<td>Q14</td>
<td>18.9</td>
<td>320.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>ABEL-Sum</th>
<th>LSTW(50)</th>
<th></th>
<th>LSTW(100)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_{avg}$</td>
<td>$t_{mod}$</td>
<td>$</td>
<td>A_Q</td>
<td>$</td>
</tr>
<tr>
<td>Q1</td>
<td>0.21</td>
<td>11.51</td>
<td>0.0%</td>
<td>0.0</td>
<td>22.58</td>
</tr>
<tr>
<td>Q2</td>
<td>1.42</td>
<td>15.65</td>
<td>4.616%</td>
<td>4.31</td>
<td>27.26</td>
</tr>
<tr>
<td>Q3</td>
<td>4.23</td>
<td>16.77</td>
<td>21.86%</td>
<td>22.6</td>
<td>31.17</td>
</tr>
<tr>
<td>Q4</td>
<td>5.45</td>
<td>10.69</td>
<td>0.004%</td>
<td>5.67</td>
<td>20.13</td>
</tr>
<tr>
<td>Q5</td>
<td>0.63</td>
<td>12.05</td>
<td>4.444%</td>
<td>1.68</td>
<td>24.6</td>
</tr>
<tr>
<td>Q6</td>
<td>0.62</td>
<td>11.22</td>
<td>0.0%</td>
<td>0.6</td>
<td>21.81</td>
</tr>
<tr>
<td>Q7</td>
<td>0.41</td>
<td>12.51</td>
<td>1.473%</td>
<td>0.63</td>
<td>22.96</td>
</tr>
<tr>
<td>Q8</td>
<td>2.07</td>
<td>11.34</td>
<td>0.011%</td>
<td>2.21</td>
<td>21.74</td>
</tr>
<tr>
<td>Q9</td>
<td>6.1</td>
<td>40.93</td>
<td>0.565%</td>
<td>6.22</td>
<td>51.28</td>
</tr>
<tr>
<td>Q10</td>
<td>0.21</td>
<td>11.55</td>
<td>1.263%</td>
<td>0.34</td>
<td>22.36</td>
</tr>
<tr>
<td>Q11</td>
<td>0.08</td>
<td>11.22</td>
<td>0.003%</td>
<td>0.08</td>
<td>21.98</td>
</tr>
<tr>
<td>Q12</td>
<td>0.17</td>
<td>11.14</td>
<td>0.011%</td>
<td>0.17</td>
<td>21.33</td>
</tr>
<tr>
<td>Q13</td>
<td>1.06</td>
<td>13.13</td>
<td>6.452%</td>
<td>3.28</td>
<td>24.8</td>
</tr>
<tr>
<td>Q14</td>
<td>0.59</td>
<td>11.67</td>
<td>0.0%</td>
<td>0.61</td>
<td>22.05</td>
</tr>
</tbody>
</table>

Table 4.3: Scalability tests on standard queries.
and $\mathcal{A}_\perp$ for all queries. All times are in seconds. The next two columns are the size percentages of computed modules $|\mathcal{A}_Q|$ and $|\mathcal{A}_\perp|$ respectively.

| Ontology    | ABEL (%) | ABEL-Sum (%) | $t_{avg}$ | $t_{mod}$ | $|\mathcal{A}_Q|$ (%) | $|\mathcal{A}_\perp|$ (%) |
|-------------|----------|--------------|-----------|-----------|----------------------|--------------------------|
| LSTW(1)     | 100      |              | 19.2      |           |                      |                          |
| LSTW(50)    | 48       | 46           | 59.1      | 18.5      | 0.03                 | 0.016                    |
| LSTW(100)   | 42       | 50           | 66.2      | 24.4      | 0.03                 | 0.014                    |

**Figure 4.1:** Scalability tests on generated queries.

The overall success rates of our system for 1, 50 and 100 universities are 100%, 94% and 92% respectively. ABEL found all explanations for 120 queries over the LSTW(1) ontology, therefore we did not use ABEL-Sum for this ontology. Moreover, our system could compute some explanations for all queries over LSTW(50), but failed to compute any explanation for 2 queries over LSTW(100).
Chapter 5

Concept Learning for $\mathcal{ELH}_\bot$

Ontology

“In a certain sense all knowledge is inductive. We can only learn the laws and relations of things in nature by observing those things.”

–William Stanley Jevons

In concept learning, the aim is to find correct concept descriptions to satisfy all positive examples and none negative examples. In this way, it can be seen as a search algorithm to examine as least as possible concept descriptions in a specific DL while eventually finding correct concept descriptions $w.r.t.$ the example sets. Therefore, a concept learning approach based on a top-down search starts with the most general concept which is $\top$ concept in DLs, then will specialise the concept in order to satisfy all positive examples and none negative examples. In each step of specialisation, the concept is evaluated to find out how much strong or week is. If the search terminates, a list of all found correct concept descriptions will be returned. Then a knowledge engineer who might be familiar with the domain can decide which description is suitable to be added to the original knowledge base.
In this chapter, we present a top-down approach on concept learning that is scalable for $\mathcal{ELH}_\perp$ ontologies. First, we have to create a search space that includes all possible concept descriptions in DL $\mathcal{ELH}_\perp$. For this purpose, a downward refinement operator for $\mathcal{ELH}_\perp$ is adapted which is complete and locally finite. This operator is also designed to specialise an $\mathcal{ELH}_\perp$-concept description whenever it is overly general w.r.t. the example sets.

A key issue encountered in concept learning is how to efficiently explore the search space created by refinement operators to eventually find a correct concept description w.r.t. the given example sets. To address this issue, we employ a data summarisation approach in order to initialise the top-down search algorithm. This initialisation yields to tremendously prune the search space at the start of the search. We also introduced a new heuristic to shorten a correct concept description in order to prune the search space afterwards. This yields to have a lesser number of solutions for each learning problem, while includes all possible short solutions, which makes the task of a knowledge engineer easier in order to choose a correct concept description.

Another issue in concept learning is how to evaluate a concept description. For this issue, we first transfer the $\mathcal{ELH}_\perp$ ontology to a datalog program, shown in Section 3.2, then materialise the ontology to complete the ABox according to the approximated TBox. It can be done in a matter of few seconds as off-line processing with the current scalable off-the-shelf datalog reasoner such as RDFox. This materialisation is needed to be computed only once before the learning approach starts or every time the ABox is changed during the learning phase. Then, we use the materialised data store for instance checking of a concept description which is sound and complete for $\mathcal{ELH}_\perp$ ontologies.

The whole approach is developed by a heuristic search algorithm which designed to find all possible shortest solutions for the given concept learning
Chapter 5. *Concept Learning for $\mathcal{ELH}_\bot$ ontology*

83

problem. Figure 5.1 shows the overview of our concept learning approach, called CL-EL, for $\mathcal{ELH}_\bot$ ontology.

![Figure 5.1: The overview of the proposed concept learning approach](image)

In the following sections, we present our adapted downward refinement operators to construct the search space. In the following sections, we first present our top-down search algorithm that describes the whole procedure of finding a concept description. Subsequently, we show a new approach to initialise the search algorithm which in most cases it does not start from $\top$ concept as it is the case for previous work [37, 38, 39, 42]. Following this, we describe a *trimming procedure* which prune the search space of concept descriptions dramatically. In the next section, we define our instance checker for concept learning using ABox materialisation. The final section discusses the experimental results of the CL-EL system.

### 5.1 Refinement Operator

In any learning system, there is a large space of concept descriptions that would be traversed to reach a correct concept description. This can not be achieved through a simple search algorithm, unless an external heuristic is employed to traverse the search space flexibly. *Refinement operators*, were
first defined in *inductive logic programming* (ILP), introduce such heuristic to be used in the search algorithm. Downward (upward) refinement operators construct specialisations (generalisation) of a concept description \[22\].

The following definition formally introduces a downward(upward) refinement operator and its properties. Completeness and finiteness properties are very important in proposing a refinement operator which we prove ours satisfies those. The completeness property is highly important since it guaranties all possible concept descriptions in a language can be constructed by an operator. The finiteness property is important since it guaranties any concept description in every depth of the search space is computable beforehand. Further, the redundancy and properness properties of an operator are introduced in Definition 5.1 which are not as important as the previous two properties.

We recall the notion \(\langle L, R \rangle\) from Chapter 2 that is used in Definition 5.1. The pair \(\langle L, R \rangle\) is a *quasi-ordered set*, if a relation \(R\) on a set \(L\) is reflexive and transitive. If \(\langle L, R \rangle\) is a quasi-ordered set, a *downward refinement operator* for \(\langle L, \geq \rangle\) is a function \(\rho\), such that \(\rho(C) \subseteq \{D | C \geq D\}\) for every \(C \in L\).

**Definition 5.1.** Let \(\langle L, \geq \rangle\) be a quasi-ordered set and \(\rho\) is a downward refinement operator for \(\langle L, \geq \rangle\). The followings are some properties of a refinement operator which can be used to evaluate their usefulness for finding a concept description.

- The sets of *one-step refinements*, *n-step refinements* and *refinements* of some \(C \in L\) are respectively:
  
  \[
  \begin{align*}
  \rho^1(C) &= \rho(C), \\
  \rho^n(C) &= \{D | \text{there is an } E \in \rho^{n-1}(C) \text{ such that } D \in \rho(E)\}, n \geq 2, \\
  \rho^*(C) &= \rho^1(C) \cup \rho^2(C) \cup \rho^3(C) \ldots
  \end{align*}
  \]
• A refinement chain from \( C \) to \( D \) is a finite sequence \( C_0 \bowtie C_1 \bowtie \ldots \bowtie C_n \) of concepts such that \( C_i \in \rho(C_{i-1}) \) for every \( 1 \leq i \leq n \).

• \( \rho \) is (locally) finite if \( \rho(C) \) is finite and computable for every concept \( C \).

• \( \rho \) is complete if for all \( C \) and \( D \) with \( C > D \), there is \( E \in \rho^*(C) \) such that \( D \approx E \) (i.e., \( D \) and \( E \) are equivalent in the \( \geq \)-order).

• \( \rho \) is weakly complete if for all \( C \in L, C \in \rho^*(\top) \).

• \( \rho \) is redundant if there exists a refinement chain from \( C_1 \) to \( D \) not going through \( C_2 \) and a refinement chain from \( C_2 \) to \( D \) not going through \( C_1 \).

• \( \rho \) is proper if for all \( C \in L, \rho(C) \subseteq \{ D \mid C > D \} \).

• \( \rho \) is ideal if it is finite, complete, and proper.

(Dually, we can define and present the properties of an upward refinement operator, denoted by \( \delta \), by replacing to a generality order.)

We now present our refinement operator, \( \langle S_\mathcal{E}, \sqsubseteq_T \rangle \), where \( S_\mathcal{E} \) is a set of all possible concept descriptions that can be constructed in \( \mathcal{ELH}_\bot \) language except \( \bot \) concept. This refinement operator is adapted from the work [39] by Lehmann and Haanse. We further prove that this operator is a downward, locally finite and complete refinement operator. This refinement operator is different from the one introduced by Lehmann et al. [39, 42] in terms of creating a search space for concept discretions in \( \mathcal{ELH}_\bot \) only. It also provides the flexibility to generate concept descriptions with shorter size in our search algorithm.

We recall from preliminaries that \( \text{concept}(T) \), \( \text{role}(T) \) and \( \text{Ind}(A) \) are set of all concept names, role names, and individuals occurred in the ontology, respectively. We also use the subscript \( T \) for each connective \( \sqsubseteq, \sqsubseteq \) and \( \equiv \) to emphasise that \( T \) models each axiom that has the connective, e.g \( C \sqsubseteq_T D \) equals to \( T \models C \sqsubseteq D \). In this chapter, we assume that an \( \mathcal{ELH}_\bot \) ontology
includes the domain and range restrictions in its TBox. Then, for each \( A \in \text{concept}(\mathcal{T}) \) and \( r \in \text{role}(\mathcal{T}) \), we define \( \mathcal{H}_A \) and \( \mathcal{H}_r \) respectively:

\[
\mathcal{H}_A = \{ B \mid B \sqsupseteq_T A \text{ where } A \in \text{concept}(\mathcal{T}) \text{ and there is no } A' \in \text{concept}(\mathcal{T}) \text{ s.t. } B \sqsupseteq_T A' \sqsupseteq_T A \} \cup \{ \exists r. \top \mid \exists r. \top \sqsupseteq_T A \text{ where } r \in \text{role}(\mathcal{T}) \text{ and } \exists A' \in \text{concept}(\mathcal{T}) \text{ such that } \exists r. \top \sqsupseteq_T A' \sqsupseteq_T A \}
\]

\[
\mathcal{H}_r = \{ s \mid s \sqsubseteq_T r \text{ where } s \in \text{role}(\mathcal{T}) \text{ and there is no } s' \in \text{role}(\mathcal{T}) \text{ s.t. } s \sqsubseteq_T s' \sqsubseteq_T r \}
\]

The fist set of \( \mathcal{H}_A \) is simply constructed by classifying all concepts occurring in the ontology. The second set of \( \mathcal{H}_A \) can be computed by checking \( |\text{concept}(\mathcal{T})| \times |\text{role}(\mathcal{T})| \) concept subsumptions. \( \mathcal{H}_r \) is also constructed by classifying all roles occurring in the ontology.

In Figure 5.2, the refinement operators \( \rho \) is defined for any \( \mathcal{ELH}_\perp \) concept description. The refinement operator \( \rho \) computes refinements of a concept inductively. For each atomic concept or \( \top \) concept, \( B \), its refinements are the set \( \mathcal{H}_B \) by the rule number 1 in Figure 5.2. For any existential axiom that has an atomic concept as its range, the role name \( r \) in the existential axiom will be refined by \( \mathcal{H}_r \). The other case is an existential axiom that has a complex concept as its range. Then the complex concept will be refined by one of its refinements. In a case that we have conjunctions of concepts, we can refine any conjunction by the rule number 4 in Figure 5.2. Finally, we can always add a conjunction to any given concept by the rule number 5 in Figure 5.2.

To have a better understanding of the refinement operator \( \rho \), we provide the following example.
Chapter 5. Concept Learning for $\mathcal{ELH}_\bot$ ontology

<table>
<thead>
<tr>
<th>concept</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) $B = \top$ or $A$ where $A \in \text{concept}(T)$</td>
<td>$\mathcal{H}_B$</td>
</tr>
<tr>
<td>2) $D = \exists r. B$</td>
<td>${\exists s. B</td>
</tr>
<tr>
<td>3) $E = \exists r. C$</td>
<td>${\exists r. C'</td>
</tr>
<tr>
<td>4) $F = C \cap L$ where $L = B$ or $D$</td>
<td>${C \cap L'</td>
</tr>
<tr>
<td>5) $C$ where $C \not\sqsubseteq \top$</td>
<td>${C \cap H</td>
</tr>
</tbody>
</table>

Figure 5.2: The refinement operator $\rho$

Example 5.1. Let $TBox$ $T$ includes the following axioms where describes our knowledge about a university domain:

$T = \{\text{Professor} \sqsubseteq \text{Faculty}, \text{Graduate Student} \sqsubseteq \text{Student};$

$\text{Graduate Course} \sqsubseteq \text{Course}; \text{Student} \sqsubseteq \text{Person};$

$\text{Faculty} \sqsubseteq \text{Person}; \text{University} \sqsubseteq \text{Organisation};$

$\text{Person} \cap \text{Course} \not\sqsubseteq \bot; \text{Person} \cap \text{Organisation} \not\sqsubseteq \bot;$

$\text{Course} \cap \text{Organisation} \not\sqsubseteq \bot; \text{worksFor} \sqsubseteq \text{memberOf};$

$\text{range(takesCourse)} \sqsubseteq \text{Course}; \text{range(memberOf)} \sqsubseteq \text{Organisation}$

$\text{domain(takesCourse)} \sqsubseteq \text{Student}; \text{domain(memberOf)} \sqsubseteq \text{Person}\}$

Then the $\rho(T)$ is the following set:

$\{\text{Person}, \text{Course}, \text{Organisation}, \exists \text{takesCourse}.\top, \exists \text{memberOf}.\top\}$

The $\rho(\text{Person} \cap \exists \text{memberOf}.\top)$ is the following set:

$\{\text{Student} \cap \exists \text{memberOf}.\top, \text{Faculty} \cap \exists \text{memberOf}.\top,$

$\text{Person} \cap \exists \text{worksFor}.\top, \text{Person} \cap \exists \text{memberOf}.\text{Organisation},$

$\text{Person} \cap \exists \text{memberOf}.\exists \text{memberOf}.\top,$

$\text{Person} \cap \exists \text{memberOf}.\top \cap \exists \text{takesCourse}.\top, \ldots\}$
In our concept learning approach, the heuristic search algorithm is doing a top-down search. Therefore, we need to show that the proposed $\rho$ has the property of being a downward refinement operator in order to be used in our top-down search algorithm. It is achieved by the following proposition.

**Proposition 5.2.** $\rho$ is a downward refinement operator.

**Proof.** We have to show that if $D \in \rho(C)$ implies $D \sqsubseteq_T C$. We prove using structural induction of $\mathcal{ELH}_\bot$ concepts for each cases in Figure 5.2.

$C = \top : D \subseteq \top$ is trivially true;

$C = A \in \text{concept}(\mathcal{T}) : D \in \rho(C)$ implies that $D$ has one of the following forms, an atomic concept, $A \cap C'$, or $\exists r.\top$. In the case of an atomic concept $D \sqsubseteq_T C$ by the first set in $\mathcal{H}_C$. The second case is trivial. And the last case, we have $\exists r.\top \sqsubseteq_T C$ by the second set in $\mathcal{H}_C$. Thus in all cases $D \sqsubseteq_T C$ holds.

$C = \exists r.E : D \in \rho(C)$ implies that $D$ has one of the form $\exists r.E \cap C'$, $\exists r.E'$ or $\exists r'.E$. The first case is trivial. The second case we have $E' \sqsubseteq_T E$ by induction, so $\exists r.E' \sqsubseteq_T \exists r.E$ holds. The last case, because $r' \sqsubseteq_T r$, we have $\exists r'.E \sqsubseteq_T \exists r.E$. Thus in all cases $D \sqsubseteq_T C$ holds.

$C = C_1 \cap \ldots \cap C_n :$ As shown in all above cases, by refining one conjunction, we have $D \sqsubseteq_T C$ by induction.

Since we are interested to find all possible correct concept descriptions w.r.t. the example set, we have to provide a search space that includes all possible concept descriptions in $\mathcal{ELH}_\bot$. Therefore, we show that our downward refinement operator $\rho$ is complete.
Proposition 5.3. $\rho$ is complete.

To prove this proposition, we first show $\rho$ is a weakly complete refinement operator by Lemma 5.4.

Lemma 5.4. $\rho$ is a weakly complete refinement operator.

Proof. We have to show that for any concept $H \in S_\mathcal{E}$, then $H \in \rho^*(\top)$, where $H$ is the form of $H_1 \sqcap \ldots \sqcap H_n$. We prove this statement by induction. Assume that $H_1$ is either of these form:

A: where $A \in \mathit{concept}(\mathcal{T}) \cup \top$. Since $A \sqsubseteq_\mathcal{T} \top$, there is a subsumption hierarchy from $\top$ concept to $A$ which is shown by the following refinement chain using the concept from $\mathcal{H}_\top$: $\top \sim A_1 \sim \ldots \sim A_m \sim A$ This refinement chain has a finite refinement steps since there are finite number of concepts in $\mathit{concept}(\mathcal{T})$. The refinement step 1 at Figure 5.2 create such a refinement chain.

$\exists r.B$: where $B \in \mathit{concept}(\mathcal{T}) \cup \top$. we use the same argument as above for traversing a role subsumption hierarchy so the refinement chain is like:

$\top \sim A_1. \top \sim \ldots \sim \exists r_m. \top \sim \exists r. \top$ while $r \sqsubseteq r_m$, $r_m \sqsubseteq r_{m-1}$, ..., $r_2 \sqsubseteq r_1$ is in the TBox. The refinement step 2 at Figure 5.2 creates such a refinement chain.

$\exists r.C$: by above cases we can have a refinement chain $\top \sim \exists r_1. \top \sim \ldots \sim \exists r.B$. Then using the refinement step 3 at Figure 5.2, we have the refinement chain $\exists r.B \sim \exists r.B_1 \sim \ldots \sim \exists r.C$ where $B_1 \in \rho(B)$, ..., $C \in \rho(B_m)$. It is possible that $B_i$ be of the form $B_i \sqcap B_{i_2} \sqcap \ldots \sqcap B_{i_m}$. Then by the refinement step 4 at Figure 5.2, we can refine either of $B_{i_j}$ by above cases.
Hence, we can reach $H_1$ from $\top$ concept by above cases, then add each conjunct stepwise using the refinement steps 5 at Figure 5.2 as follows:

\[ \top \leadsto \ldots \leadsto H_1 \leadsto \ldots \leadsto H_1 \cap H_2 \leadsto \ldots \leadsto H_1 \cap \ldots \cap H_n \]

Thus, $\rho$ is a weakly complete refinement operator.

Now we show the completeness proof of $\rho$.

**Proof.** For the completeness, let $C, D \in S_\mathcal{E}$ such that $D \sqsubseteq_{\mathcal{T}} C$. We have to show that there is a concept $E \equiv_{\mathcal{T}} D$ where $E \in \rho^*(C)$. If $E$ is $C \cap D$, given $D \sqsubseteq_{\mathcal{T}} C$, then $E = C \cap D \equiv_{\mathcal{T}} D$, we obviously have $E \in \rho(C)$ by only one refinement step. We know that $\rho$ provides to extend concept descriptions by adding a conjunction in refinement steps of the top concept. Hence, we know that $D \cap C$ can be reached from $D \cap \top$ by the weak completeness result for $\rho$. Thus, $\rho$ is complete.

Another important property that our refinement operator $\rho$ must have is being locally finite. This is needed for terminating our heuristic search algorithm. Thus, we show that the $\rho^l(C)$ is computable for any concept description $C$ and natural number $l$. The size of a concept $C$, denoted by $|C|$, is all concepts, roles, and connectives including $\{\cap, \exists, \ldots, (, )\}$ occurred in $C$. For example, let $C$ be $A \cap \exists r.(B \cap \exists s.C)$, then the size of $C$ is 13.

**Proposition 5.5.** For any concept $C$ in $\mathcal{ELH}_\perp$, and any natural number $l$, the set $\{D \mid D \in \rho(C), |D| \leq l\}$ can be computed in finite time.

To prove this proposition, we first show that all elements of $\rho(C)$ have the same or bigger size than $|C|$ and it is computable in finite time.

**Lemma 5.6.** Let $D \in \rho(C)$. Then there is no infinite refinement chain of the form $C_1 \rightarrow C_2 \rightarrow \ldots$ with $|C_1| = |C_2| = \ldots$
Chapter 5. Concept Learning for $\mathcal{ELH}_\bot$ ontology

Proof. To show that the size of concept descriptions are not always the same, we have to observe the refinement steps in Figure 5.2. There are 3 different refinements:

- **refine atomic concept**: This step either does not change the size of $C$ if $C$ refines by another atomic concept or increases its size if $C$ refines by a concept the form $\exists r.T$.

- **refine complex concept**: There are two possibilities if $C$ is of the form $\exists r.B$. First, the role might be refined by another role which does not change the size of $C$. Second possibility is that the concept $C'$ in $C = \exists r.C'$ is refined which is similar to the above case.

- **add conjunction**: This step will always increase the size of $C$ since it add another conjunct to it.

Since $\text{concept}(T)$ and $\text{role}(T)$ are finite sets, there are only finite many occurrences of an atomic concept within any concept. Hence, there are no infinite refinement chains using $\rho(C)$ where does not change the size of $C$. Thus, after a finite number of refinements, $\rho(C)$ produces a longer concept.  

Now we can prove Proposition 5.5.

Proof. Due to Lemma 5.6, for any concept $D$ in the set, there exists an $n$ such that $|D| > |C|$ with $D \in \rho^n$. If we consider all refinement chains of a concept $C$ by $\rho$ up to size $l$, then there are only finitely many concepts of size $\leq l$ and any such concept can be reached by a finite refinement chain. Thus, the set $\{D \mid D \in \rho(C), |D| \leq l\}$ can be computed in finite time.
5.2 CL-EL (Concept Learning for EL)

In this section, we propose a new concept learning approach which does a top-down search among concept descriptions created by the downward refinement operator $\rho$. We adapt a heuristic search algorithm to develop a top-down search and eventually find a correct concept description w.r.t. the example sets.

In our concept learning approach, a problem-solving agent (or simply an agent) interacts with its environment via perception and action. On each step of interaction the agent determines the current state of the environment. The agent then chooses an action to generate an output. The action changes the state of the environment and the value of this state transition is evaluated through a reward function. The agent’s behaviour should lead the agent to choose actions that tend to reach the goal state.

Subsequently, we define all states, actions and reward function accordingly. First of all, the environment is a search space of all concept descriptions in $\mathcal{ELH}_\perp$. Then a state is any concept description in the search space. An action is a member of all possible refinements in a given state. Therefore, action function is employing the downward refinement operator $\rho$ in addition to some heuristics which helps to effectively explore the search space. The reward function is a score function that shows how much a concept description is either too strong w.r.t. the negative examples or too weak w.r.t. the positive examples. The goal state is a concept description which is correct w.r.t. the example sets.

Now we formally define state, action and reward function of our approach.

**Definition 5.7 (State).** Let $S_\xi$ be a set of all possible concept descriptions in $\mathcal{ELH}_\perp$. Assume $C \in S_\xi$ as a concept description. Then $C$ is a state.
The actions are based on the downward refinements that defined in Figure 5.2. These actions are members of $\rho(C)$ where $C$ is a state. However, there are infinite number of states, but according to Proposition 5.5, the $\rho(C)$ is always computable.

**Definition 5.8** (Action). Let $C$ be a state. An action is a pair $(C, D)$ where $D$ is a concept description such that $D \in \rho(C)$.

The reward is the way of communicating to the agent in order to achieve the goal state which is finding a correct concept description w.r.t. the example sets. Therefore, we define a reward function to return a reward $r_C$ to the agent as an evaluation of the current state $C$.

**Definition 5.9** (Reward). Let $p$ and $n$ be the number of individuals in $E^+_G$ and $E^-_G$, respectively. And $C$ be a state. Let $n^+$ be the size of set $\{a \in E^+_G | T \cup A \models C(a)\}$, and $n^-$ is the size of set $\{b \in E^-_G | T \cup A \models C(b)\}$. Then the reward function returns a real number between 0 and 1, denoted as a reward $r_C$ of the state $C$ as follows:

$$r_C(C) = \begin{cases} 
0 & \text{if } n^+ < p \\
2 \times \frac{n^+}{n^+ + n^- + p} & \text{if } n^+ = p \text{ and } 0 < n^- \leq n \\
1 & \text{if } n^+ = p \text{ and } n^- = 0
\end{cases}$$

The reward function can be written differently according to the user preference. In Definition 5.9, the reward function returns a higher value to those concept descriptions that are overly general w.r.t. the example sets. This definition is plausible since all of actions to refine a concept description are from downward refinement operators. So after a refinement, if the refined concept is too weak w.r.t. the positive examples, it does not make sense to refine it by a downward refinement operator again.
Algorithm 1 Concept Learning in $\mathcal{ELH}_\bot$

**Input:** $\mathcal{T}, A, E_G^+, E_G^-$, $l$ and $Init$

**Output:** a list of all correct concept descriptions

$\text{result} \leftarrow \emptyset$

$\text{Insert}(Init,\ queue)$

while queue is not empty do

$State \leftarrow \text{pop}(queue)$

$r_{State} \leftarrow \text{Reward}(State, \mathcal{T}, A, E_G^+, E_G^-)$

$n \leftarrow \text{which\ refinement\ step}(State, Init)$

if $0 < r_{State} < 1$ and $n \leq l$ then

$actions \leftarrow \text{find\ all\ actions}(State)$

$\text{InsertAll}(actions, queue)$

end if

if $r_{State} = 1$ then

$\text{result} \leftarrow \text{result} \cup \{State\}$

$actions \leftarrow \text{find\ all\ actions\ same\ size}(State)$

$\text{InsertAll}(actions, queue)$

end if

end while

return $\text{result}$

Now we can define the goal state using the reward function. The goal state is the state that its reward is 1. The initial state is usually $\top$ concept but we will define it in the next section.

In Algorithm 1, the whole procedure for the proposed CL approach is described. The inputs of the algorithm are a knowledge base, positive and negative examples, as well as a maximum length of a refinement chain. We also give the initiative concept description $Init$ as an initial state. The output result is a list of all correct concept descriptions w.r.t. the example sets. We use a data structure queue for storing actions.

Initially, the initiative concept $Init$ is inserted into the queue by Insert function. In the while loop, the last element of queue is popped out which is the current state $State$. The $State$ is evaluated by the Reward function. The Reward function returns a value $r_{State}$ according to Definition 5.9 for each $State$.
Chapter 5. Concept Learning for $\mathcal{ELH}_\bot$ ontology

using $\mathcal{T}, \mathcal{A}, E^+_G$ and $E^-_G$. If $r_{State}$ has the value of 1, State is a goal state and it is added to result.

The function of which refinement step returns a natural number of $n$ where $State \in \rho^n(Init)$. We recall from Definition 5.1 that $\rho^1(C) = \rho(C)$, and $n$-step refinements of $C$ are $\rho^n(C) = \{D \mid \text{there is an } E \in \rho^{n-1}(C) \text{ such that } D \in \rho(E)\}$. We can keep each $i$-step refinements of $Init$ to efficiently find the $n$ for each $State$.

If $r_{State}$ has value of bigger than 0 but 1 and $n$ is less than $l$, it means that the State is overly general w.r.t. the example set and we have not reached the maximum length of a refinement chain for $Init$. So we can find all possible actions for $State$ by $\text{find\_all\_actions}$ function which will be inserted to $queue$ by $\text{InsertAll}$ function. $\text{InsertAll}$ function is similar to $\text{Insert}$ function with the exception that it inserts all elements of a set to the queue.

In a case that $r_{State}$ has the value of 1, we can further refine $State$ to find other correct concept descriptions. However, we do not need add any other conjunction to $State$ by refinement step 5 at Figure 5.2 since we are interested in correct concept descriptions with shorter size. Therefore, $\text{find\_all\_actions\_sameSize}$ function will find all actions that have the same size with $|State|$. All these actions are inserted to $queue$ by $\text{InsertAll}$ function.

This procedure continues until the $queue$ is empty. This means that the search finishes after examining all concept descriptions of set $\rho^l(Init)$.

5.2.1 Initialising CL-EL

In the previous sections, we proposed a downward refinement operator as well as a new CL algorithm to traverse the search space in a top-down manner. In this section, we present a new approach to initialise the Algorithm 1 using the data summarisation technique. By this technique, the initiative concept
\(\text{Init}\) will be found which reduces the search space dramatically. Normally, in the previous approaches [37, 39, 42] based on a top-down search, the initiative concept is \(\top\) concept where a score function validates each node in the search space to be further expanded. Their approach is promising, but there are lots of reasoning efforts to validate each node, so a more sophisticated method is required to initialise the search algorithm. We introduce such an initialisation through defining \textit{individual type} for all positive examples.

Now we define the type of an individual introduced using the definition introduced by Glimm et al. [72]. For an ABox \(\mathcal{A}\) and an individual \(a\) in \(\text{Ind}(\mathcal{A})\), the type \(\vartheta\) of \(a\) is a pair \(\vartheta(a) = \{\tau_{\text{concept}}(a), \tau_{\text{role}}(a)\}\) where \(\tau_{\text{concept}}(a) = \{C \mid C(a) \in \mathcal{A}, C \in \text{concept}(T)\}\) and \(\tau_{\text{role}}(a) = \{r \mid \exists b \in \text{Ind}(\mathcal{A}) : r(a,b) \in \mathcal{A}, r \in \text{role}(T)\}\). This new definition of type is included more details regarding the role assertions from the ABox compare to our previous definition (at Section 4.1.1) which ignored all role assertions in the definition of an individual type. By the following definition, we formally define the \textit{initiative concept} using type \(\vartheta\).

**Definition 5.10.** The initiative concept is an \(\mathcal{ELH}_\perp\) concept description of the form \(\bigcap_{A \in \Gamma} A \cap \bigcap_{r \in \Omega} \exists r. \top\), where \(\Gamma\) is \(\{A \mid A \not\equiv \top \land A \in \{\bigcap_{a \in E^+_G} \tau_{\text{concept}}(a)\}\}\) and \(\Omega\) is \(\{r \mid r \in \{\bigcap_{a \in E^+_G} \tau_{\text{role}}(a)\}\}\).

If both \(\Gamma\) and \(\Omega\) are empty sets, the initiative concept is \(\top\) concept.

The following example helps to have a better understanding of how to compute the initiative concept.
Example 5.2. Consider the TBox from Example 5.1 and the ABox includes the following assertions:

\[
\begin{align*}
\text{GraduateStudent}(GS_1), \ldots, \text{GraduateStudent}(GS_{100}), \\
\text{GraduateCourse}(GC_1), \ldots, \text{GraduateCourse}(GC_{20}), \\
\text{Professor}(P_1), \ldots, \text{Professor}(P_{15}), \text{ResearchGroup}(RG_1), \ldots, \\
\text{ResearchGroup}(RG_{10}), \text{worksFor}(P_i, GU) 1 \leq i \leq 15, \\
isPartOf(RG_i, GU) 1 \leq i \leq 10, \text{worksFor}(GS_i, RG_i) 1 \leq i \leq 10 \\
takesCourse(GS_i, GC_j) (1 \leq i \leq 100 \text{ and } j_i \subseteq \{1,2,\ldots,20\}), \\
\text{teacherOf}(P_{j_i}, GC_i) (1 \leq i \leq 20 \text{ and } j_i \subseteq \{1,2,\ldots,15\})
\end{align*}
\]

We want to learn a concept description of Employee having \( E^+_G = \{GS_1, \ldots, GS_{10}, P_1, \ldots, P_{15}\} \) and \( E^-_G = \{GS_{11}, \ldots, GS_{100}\} \). The types of all positive examples are:

\[
\begin{align*}
\vartheta(GS_i) &= (\{\text{GraduateStudent}\}, \{\text{worksFor, takesCourse}\}) 1 \leq i \leq 10 \\
\vartheta(P_j) &= (\{\text{Professor}\}, \{\text{worksFor, teacherOf}\}) 1 \leq j \leq 15
\end{align*}
\]

So \( \Gamma = \emptyset \) and \( \Omega = \{\text{worksFor}\} \), then the initiative concept is \( \exists \text{worksFor.} \top \).

As another example, we want to find a concept description of ResearchAssistant having \( E^+_G = \{GS_1, \ldots, GS_{10}\} \) and \( E^-_G = \{GS_{11}, \ldots, GS_{100}, P_1, \ldots, P_{15}\} \). The types of all positive examples are mentioned above.

In this case, \( \Gamma = \{\text{GraduateStudent}\} \) and \( \Omega = \{\text{worksFor, takesCourse}\} \), so the initiative concept is:

\[
\text{GraduateStudent} \sqcap \exists \text{worksFor.} \top \sqcap \exists \text{takesCourse.} \top
\]
The above example shows that starting with initiative concept the search space is reduced. In both cases the initiative concept is a correct concept description w.r.t. the example sets although it can be further refined to be more specific. For example, another correct concept description of Employee can be $\exists \text{worksFor} . \text{Organisation}$ by refining initiative concept $\exists \text{worksFor} . \top$.

It also shows that for refining State by adding a conjunction (via refinement step 5 at Figure 5.2) in Algorithm 1, we only need to add conjunctions that have been already included in $\Omega$ of positive examples type. For example, we never add the conjunction of $\exists \text{takesCourse} . \top$ while refining initiative concept $\exists \text{worksFor} . \top$ for finding all correct concept descriptions of Employee since the role name takesCourse is not in $\Omega$.

Admittedly, we found in our experimental that the initiative concept description includes irrelevant conjunctions most of the time. The irrelevant conjunctions are those conjunctions in a concept description that if it is discarded from the description, the new concept description is neither too week nor too strong w.r.t. the example sets. For example, the initiative concept for ResearchAssistant was GraduateStudent $\sqcap \exists \text{worksFor} . \top \sqcap \exists \text{takesCourse} . \top$, where the conjunct $\exists \text{takesCourse} . \top$ is an irrelevant conjunction. It means that GraduateStudent $\sqcap \exists \text{worksFor} . \top$ is also a correct concept description after removing takesCourse.$\top$ from initiative concepts.

In the next section, we show a post-processing procedure in order to trim a correct concept description by removing irrelevant conjunctions. This trim procedure prunes the search space effectively, since those irrelevant conjunctions will be no further refined. For example, the algorithm does not refine $\exists \text{takesCourse} . \top$ concept for ResearchAssistant any more and this concept never will be added as a conjunction further in the refinement steps.
5.2.2 CL-EL with Trimming Procedure

As it is described in the previous sections, our CL approach starts with an initiative concept that is computed by types of all positive examples. This initiative concept accelerates the search of finding a correct concept description w.r.t. example sets. Our approach will further refine a found correct concept description in order to find other correct concept descriptions as shown in Algorithm 1. However, it is expected that the found correct concept description includes irrelevant conjunctions most of the time. Therefore, we introduce a trimming procedure in order to remove irrelevant conjunctions from the found correct concept description. This trimming procedure helps to find other concept descriptions quickly by pruning the search space of states.

In the trimming procedure of a concept description, we have to be sure that the concept is already a correct concept description w.r.t. the example sets. If it is not, this trimming step will be postponed until the first correct concept description found.

Initially, a found correct concept description, $C$, is split to a set of all conjunctions occurring in it, denoted by $\mathbf{C}$. Literally, if $C$ is of the form $C_1 \sqcap C_2 \sqcap \ldots \sqcap C_n$, then $\mathbf{C} = \{C_1, C_2, \ldots, C_n\}$. The trimming procedure on a correct concept description $C$ is to find all minimal subsets of $\mathbf{C}$ that are correct concept descriptions.

To find all minimal subsets, a set-enumeration tree is used to find all the subsets of $\mathbf{C}$ in a systematic way. The root of a set-enumeration tree is $\mathbf{C}$, and each node stores a subset of $\mathbf{C}$. Each level of tree stores the subsets of the same size. For example, Figure 5.3 shows the set-enumeration tree for the set $\{A_1, A_2, A_3\}$. 
Subsequently, we evaluate each node of set-enumeration tree for $C$. If it is not a correct concept description, we do not evaluate its children. Thus, we eventually find all minimal subsets of $C$, $M(C)$, that are correct concept descriptions \textit{w.r.t.} the example sets.

After having all minimal subsets $M(C)$, we construct complex concepts for each minimal subset by conjuncting all members of any subset. Formally, a complex concept $C_i$ is $\bigcap_{D \in m} D$ where $m \in M(C)$ for any correct concept $C$. Then, we call Algorithm 2 for each $C_i$ to find all correct concept descriptions that have the same size with $|C_i|$.

In Algorithm 2, \textit{queue} is initialised to $Correct$ which is a correct concept description. In the while loop, the current state $State$ is computed by popping out an element of \textit{queue}. It is evaluated by Reward function according to Definition 5.9. If it is a correct concept description ($r_{State}$ has the value of 1), and its size is less or equal to $|Correct|$, the possible actions of $State$ will be computed by $\text{find\_all\_actions\_by\_Size}$ where all actions have the same size with $|Correct|$. All these actions will be inserted to \textit{queue} by InsertAll function. The algorithm terminates when the \textit{queue} is empty.
Algorithm 2 Find all correct solutions with the same size

**Input:** $T, A, E^+_C, E^-_C$ and $Correct$

**Output:** a list of all correct concept descriptions

```pseudocode
1: result ← ∅
2: Insert(Correct, queue)
3: while queue is not empty do
4:   State ← pop(queue)
5:   rState ← Reward(State, $T, A, E^+_C, E^-_C$)
6:   if $|State| \leq |Correct|$ and $rState = 1$ then
7:      result ← result ∪ {State}
8:      actions ← find_all_actions_bySize(State, $|Correct|$)
9:      InsertAll(actions, queue)
10: end if
11: end while
12: return result
```

Consequently, we modify Algorithm 1 which is shown in Algorithm 3 in order to include the trimming procedure. Algorithm 3 works the same as Algorithm 1 except in the case that a `State` is a correct concept description w.r.t. the example sets. In this case, `State` will be added to the `result` set, and it will be trimmed by `Trim` function. The Trim function will return a set, i.e. $M(\text{State}) = \text{minimal\_subsets}$, which includes all minimal subset of `State` that are also correct concept descriptions. Subsequently, for each subset $m \in \text{minimal\_subsets}$, we first create an `m\_concept` by `conjunctAll_concepts` function that is the conjunctions of all elements of $m$. Then, we compute all other correct concept descriptions that have the same size with $|m\_concept|$ using Algorithm 2. All correct concept descriptions returned by Algorithm 2 will be added to the `result` set. Eventually, we remove all actions in `queue` that have bigger size than `State` by `removeAll_bigger_actions` function. Since we have already found that `State` is a correct concept description, so we prune the search space of states that have bigger size than $|State|$. This is essential because we are interested to find as short as possible correct concept descriptions.
Algorithm 3 CL-EL Approach

**Input:** $T, A, E^+_G, E^-_G, l$ and $Init$

**Output:** a list of all correct concept descriptions

1: $result \leftarrow \emptyset$
2: Insert($Init, queue$)
3: while $queue$ is not empty do
4:    $State \leftarrow$ pop($queue$)
5:    $r_{State} \leftarrow$ Reward($State, T, A, E^+_G, E^-_G$)
6:    $n \leftarrow$ which_refinement_step($State, Init$)
7:    if $0 < r_{State} < 1$ and $n \leq l$ then
8:      $actions \leftarrow$ find_all_actions($State$)
9:      InsertAll($actions, queue$)
10: end if
11: if $r_{State} = 1$ then
12:    $result \leftarrow result \cup \{State\}$
13:    minimal_subsets $\leftarrow$ Trim($State$)
14:    $result \leftarrow result \cup minimal_subsets$
15: for all $m \in minimal_subsets$ do
16:    $m_{concept} \leftarrow$ conjuctAll_concepts($m$)
17:    $sameSize \leftarrow$ Algorithm2($T, A, E^+_G, E^-_G, m_{concept}$)
18:    $result \leftarrow result \cup sameSize$
19: end for
20: removeAll_bigger_actions($queue, |State|$)
21: end if
22: end while
23: return $result$

5.3 Scalable Instance Checker

In DL-Learner [39], an instance checker was proposed which basically computes all concept and role assertions of the given ontology by the Pellet reasoner and keep all these inferences in the memory. Then, in run time, they use these cached assertions to find all individuals of a concept description. A major difference of our instance checker with the one proposed in DL-Learner is that we do not assume that the ABox is fixed (since they only use the cache data generated by the reasoner in initialisation phase). Because currently fully-fledged reasoners are not equipped by incremental reasoning, so whenever new assertions added or some assertions deleted, all inferences has to be recomputed.
from scratch. This problem does not exist for datalog-based reasoner. Another difference is that, using the fully-fledge reasoners for instance checking does not guarantee scalability for ontologies with very large ABoxes.

In this section, we present an approach how to efficiently find all individuals which are members of a concept description over $\mathcal{ELH}_\bot$ ontology. First the $\mathcal{ELH}_\bot$ ontology is transformed to its approximated datalog program according to Table 3.2. Further, we employ an off-the-shelf datalog reasoner [77] to materialise the ABox w.r.t. the datalog program.

We recall from Chapter 2 that ABox $\mathcal{A}$ is concept materialised if $A(a) \in \mathcal{A}$ whenever $T \cup \mathcal{A} \models A(a)$ for $A \in \text{concept}(T)$, and $\mathcal{A}$ is role materialised if $r(a,b) \in \mathcal{A}$ whenever $\cup \mathcal{A} \models r(a,b)$ for $r \in \text{role}(T)$. The ABox $\mathcal{A}$ is materialised if it is both concept and role materialised.

RDFox system is employed to compute the materialisation for datalog approximation of any $\mathcal{ELH}_\bot$ ontology. Now we can convert each $\mathcal{ELH}_\bot$ concept description $C$ where created during the learning algorithm to a datalog rule. It can be achieved by introducing a fresh concept name $Q$ that is not in $\text{concept}(T)$ and transforming the axiom $C \sqsubseteq Q$ inductively using transformation shown in Table 3.1. Finally, by querying $Q$ through RDFox, we find all instances of an $\mathcal{ELH}_\bot$ concept. Since, $\mathcal{ELH}_\bot$ concept description are not the forms of $\text{Fork}$ and the $\text{Cyc}$, our system is sound and complete to find instances of any $\mathcal{ELH}_\bot$ concept description.

5.4 Experimental Results

We have implemented a prototype learning system, CL-EL (Concept Learning for EL), which employs the OWL reasoner Pellet\footnote{clarkparsia.com/pellet} through OWL API\footnote{http://owlapi.sourceforge.net/} for
concept and role classifications as well as datalog engine RDFox\(^3\) for verifying the concept descriptions.

In the following, we present the results of CL-EL system on different ontologies and its comparison with other learning systems in description logics. Moreover, we performed another set of experiments on CL-EL in order to evaluate its scalability. All the experiments were performed on a PC with an Intel Xenon 2.8 GHz processor and 16 GiB RAM.

### 5.4.1 Test Cases

There is no standard benchmark for evaluating DL concept learning algorithms, although test cases have been borrowed from Machine Learning community\(^4\) and Lehmann and Haase \([39]\) transferred those test cases to DL ontologies which are available online\(^5\). There are two main challenges for us to use all of these test cases. First of all, most of the ontologies are expressed in expressive DLs, and solutions of a learning problem is not expressible by an \(\mathcal{ELH}_{\perp}\) concept description. Secondly, the aim of this research is to have a scalable learning framework which the sizes of these data sets are not close to the real-world ontologies since the largest ontology in these test cases has less than a million ABox assertions.

\(^3\)www.cs.ox.ac.uk/isg/tools/RDFox
\(^4\)http://archive.ics.uci.edu/ml/
\(^5\)http://sourceforge.net/projects/dl-learner/files/DL-Learner/
Therefore, for the comparison of CL-EL to the current learning framework, we use some data sets from DL-Learner project, and for the scalability test, the extended version of LUMB benchmark\(^6\) used where its ABox could include more than millions of assertions. Table 5.1 shows some statistics on the ontologies used in our experiments: the numbers of their concept names (\(\#C\)), role names (\(\#R\)), TBox axioms (\(|T|\)), ABox assertions (\(|A|\)), and individuals (\(\#I\)). The following describes each ontology:

**Train** This ontology is constructed manually by Lehmann and Haase [39] to represent the Michalski’s train problem [1]. The train ontologies is described two sets of trains, Eastbound and Westbound (respectively the left and the right part of Figure 5.4) by 11 features such as which carriage is in front of another carriage, whether a car is short or long, closed or open, jagged or not, which shape car contains and the car’ loading status.

![Description of trains in Michalski’s trains problem [1]](image)

**Poker** Another manually constructed ontology is the Poker ontologies where describes different poker hands (i.e. straight and pair) that includes different cards, cards’ suits, and card’ ranking. The ontology for straight hands has 1326 ABox assertions and 311 individuals. The ABox size of poker hand ontology is 1410 and it contains 347 individuals.

**Family** The family ontology is combination of forte ontology [39] and Basic-Family ontology [37] which is artificially constructed for test purpose.

---

\(^6\)[http://swat.cse.lehigh.edu/projects/lubm/](http://swat.cse.lehigh.edu/projects/lubm/)
Extended version of Lehigh University Benchmark The LSTW [74] is an extended version of the Lehigh University benchmark LUBM [62] with more axioms. However, we modified the TBox to have lesser axioms since to challenge CL-EL. For LSTW, ABoxes can be automatically generated in a range of sizes using EUGen [74]. LSTW(n) denotes the variant with an ABox of n universities. We mostly used this ontology for evaluating the scalability of CL-EL.

We have used different learning problems for each ontologies in our experiments with the condition that outcomes are expressible in $\mathcal{ELH}_\bot$. For the first three ontologies, we used the known problems from DL-Learner project [39]. For the learning problems in LSTW, the original TBox of LSTW was modified for all learning problems in order to be challenging for CL-EL to find a concept description. The TBox modifications include removing all equivalence axioms from TBox. Then the CL-EL will be employed to learn some of missing concepts.

Train. The Michalski’s train problem is to describe descriptions for each Eastbound and Westbound trains in Figure 5.4. The positive examples are the trains on the left in Figure 5.4 and the negative examples are the trains on the right. The shortest description for Eastbound trains is $\exists \text{hasCar.} (\text{Short} \sqcap \text{ClosedCar})$.

Poker. The first Poker problem is to find a description for a straight hand, and the second one is to find a description for a pair hand. Therefore, two poker ontologies are given that for learning a straight hand there are 1410 ABox axioms, and 347 individuals, and for learning a pair hand the ontology includes 1326 and 311 ABox axioms and individuals respectively. Possible shortest descriptions for a pair hand and a straight hand are $\exists \text{hasCard.} (\exists \text{sameRank.card})$ and $\exists \text{hasCard.} (\exists \text{nextRank.} (\exists \text{nextRank.} (\exists \text{nextRank.} (\exists \text{nextRank.} (\exists \text{nextRank.} (\exists \text{nextRank.} (\exists \text{nextRank.} \text{card}) )))) )$, respectively.
**Family.** The learning problems in Family ontology are concepts of *Grandfather* and *Mother*. Possible shortest descriptions for a Grandfather and a mother are $\text{Male} \sqcap \exists \text{hashChild} \sqcap \exists \text{hasChild} \sqcap \top$ and $\text{Female} \sqcap \exists \text{hasChild} \sqcap \top$.

**LSTW.** The first (I) learning problem for LSTW ontology is to find a description for someone who has an advisor who teaches a graduate course. The second problem (II) is to find a description for a chair of a department.

### 5.4.2 Comparisons

There are different learning system candidates that could be used to compare with CL-EL. The first candidates are ILP tools where the input background knowledge is represented as a Logic Program. The second candidates are the state-of-the-art learning systems for DL ontologies.

However, we did not compare CL-EL to the ILP tools for two main reasons. First, ILP tools are not designed to fully take into account the schema level of a background knowledge base (i.e. TBox in DL ontologies), mostly their methods are bottom-up approaches that starts from example sets to find a general concept description. Therefore the outcome descriptions are longer than it could be expected and its readability is not easy for human as shown by Lehmann and Haase [39]. For example, if we have an axiom in the TBox for *Parent* as $\text{Parent} \equiv \exists \text{hasChild} \sqcap \text{Person}$, then if one wants to learn the concept of *GrandParent*, the most desirable one is $\exists \text{hasChild} \sqcap \exists \text{hasChild} \sqcap \text{Person}$ rather than $\exists \text{hasChild} \sqcap \exists \text{hasChild} \sqcap \text{Person}$. Second, since CL-EL is specifically designed to work on $\mathcal{ELH}_1$ ontologies and ILP tools are more expressive than $\mathcal{ELH}_1$, it is possible in most cases the outcome would not be expressible in $\mathcal{ELH}_1$ which makes comparisons incomprehensible.

There are mainly three DL-based learning systems such as YinYang [37], DL-Foil [38] and DL-Learner [39]. Although both former systems support only
\(\mathcal{ALC}\) ontologies and it is not possible to set the program to find only \(\mathcal{ELH}_\perp\) concept descriptions. Nevertheless, DL-Foil has not been publicly available so far.

Thus, we decide to evaluate CL-EL performance among DL-based learning systems that supports \(\mathcal{ELH}_\perp\) concept descriptions as solutions. Therefore, we compare CL-EL with DL-Learner system which has the possibility to specifically find concept descriptions expressible in \(\mathcal{ELH}_\perp\).

DL-Learner has different algorithms and options to customise the expressibly of a learned concept description. We set the DL-Learner’s algorithm type to “eltl” which specifically designed for \(\mathcal{EL}\) ontologies (we further refer to it as \(DL-Learner\ ELTL\) or \(ELTL\)). CL-EL is also set to find a concept description that each atom has at most the depth of 8. DL-Learner ran out of memory for LSTW(\(n\)) where \(n\) was equal to 5. Table 5.2 shows the comparison result of CL-EL and DL-Learner ELTL on the aforementioned learning problems: the numbers of positive examples (\(|E^+_G|\)), negative examples (\(|E^-_G|\)), correct solutions (\(#S\)) and time taken for learning in seconds (Time). The \(\text{avg}\) is the average time to find a solution. It computed by dividing the total time by the number of solutions.

| Ontologies | Problem         | \(|E^+_G|\) | \(|E^-_G|\) | CL-EL #S | Time | \(\text{avg}\) | ELTL #S | Time | \(\text{avg}\) |
|------------|----------------|------------|------------|---------|------|-------------|---------|------|-------------|
| Train      | Eastbound      | 5          | 5          | 11      | 0.09 | 0.008       | 100     | 10.24 | 0.102       |
| Poker      | Straight Pair  | 4          | 51         | 33      | 0.18 | 0.005       | 100     | 11.69 | 0.117       |
|            |                 | 20         | 29         | 11      | 0.11 | 0.010       | 100     | 10.13 | 0.101       |
| Family     | Grandfather    | 16         | 59         | 42      | 0.20 | 0.005       | 100     | 10.21 | 0.102       |
|            | Mother         | 31         | 59         | 9       | 0.08 | 0.009       | 100     | 11.47 | 0.115       |
| LSTW(1)    | I              | 495        | 577        | 14      | 1.88 | 0.134       | 39      | 10.34 | 0.265       |
|            | II             | 15          | 434        | 33      | 1.25 | 0.038       | 100     | 10.64 | 0.106       |

Table 5.2: Compare CL-EL and DL-Learner ELTL.
Both CL-EL and DL-Learner ELTL successfully found shortest correct solutions for all learning problems except the description of *Grandfather* that DL-Learner ELTL could not find the concept description $\text{Male} \sqcap \exists \text{hasChild.}$ $\exists \text{hasChild.} \top$. Moreover, DL-Learner ELTL takes longer to terminate for all learning problems. The reason that CL-EL is faster because of our efficient instance checker as well as the more fine-grained initialisation step.

It is also a disadvantage of DL-Learner ELTL that the number of its returned correct solutions were very large compare to the size of result sets from CL-EL. Having many solutions is troublesome for even a knowledgeable user to choose a concept description among those. Indeed, the usability of the system increases when a shorter list of correct solutions is available and this short list includes all shortest possible concept descriptions.

The reason that CL-EL returned a shorter list of concept descriptions is because of the trimming procedure after finding a correct concept description as described in Section 5.2.2. For example, to learn the problem II for LSTW(1) ontology, the first correct concept description is $\text{FullProfessor} \sqcap \exists \text{headOf.} \top \sqcap \exists \text{degreeFrom.} \top \sqcap \exists \text{teacherOf.} \top \cdot \cdot \cdot$ where it will be trimmed to $\exists \text{headOf.} \top$ since the rest of the conjunctions are irrelevant. This trim procedure prunes the search space effectively, since there is no need to refine those irrelevant conjunctions. The algorithm does not refine $\exists \text{degreeFrom.} \top$ concept any more and this concept never will be added as an conjunct further in the refinement steps as well.

Thus, CL-EL is superior to DL-Learner ELTL for two main reasons. First CL-EL took less than 2 seconds to return all short correct concept conjunctions. Second, it always returned a shorter list of concept descriptions for each learning problem.
5.4.3 Scalability Test

To test the scalability of CL-EL in finding correct concept descriptions, we have conducted another experiments using LSTW data sets of increasing sizes (1, 10, 50, and 100 universities). We also increased the number of example sets over different LSTW(\(n\)) to observe the CL-EL’s behaviour. The Figure 5.5 shows the scalability test on increasing sizes of LSTW ABoxes. The learning problems I and II are as mentioned in 5.4.1. The number of positive examples for problem I and II are 495 and 15 respectively and the number of negative examples for problem I and II are 577 and 434 respectively. CL-EL returns the same number of correct concept descriptions for each problem for all LSTW(\(n\)).

In the next experiments, we increased the size of example sets for each learning problem and LSTW(\(n\)) ontology. The result is shown in Figure 5.6. An interesting result from this experiment is that this increase of example sets helps to find lesser number of solutions which is more desirable while the efficiency is not deteriorated. The reason that efficiency is not deteriorated is that our heuristic search in CL-EL works in a top-down manner. Therefore,
Chapter 5. Concept Learning for $\mathcal{ELH}_\bot$ ontology

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
Ontology & $|E^+_G|$ & $|E^-_G|$ & #S & Time(secs) \\
\hline
\multicolumn{5}{|c|}{Problem I} \\
\hline
LSTW(1) & 495 & 577 & 14 & 1.88 \\
& 495 & 577 & 14 & 16.46 \\
& 6312 & 494 & 13 & 20.19 \\
LSTW(50) & 495 & 577 & 14 & 118.72 \\
& 6312 & 494 & 13 & 127.19 \\
& 33459 & 2709 & 13 & 149.89 \\
LSTW(100) & 495 & 577 & 14 & 339.19 \\
& 6312 & 494 & 13 & 333.09 \\
& 33459 & 2709 & 13 & 307.43 \\
& 67354 & 5389 & 13 & 347.52 \\
\hline
\multicolumn{5}{|c|}{Problem II} \\
\hline
LSTW(1) & 15 & 434 & 33 & 1.25 \\
& 15 & 434 & 33 & 8.55 \\
& 189 & 5440 & 19 & 9.22 \\
LSTW(10) & 15 & 434 & 33 & 91.3 \\
& 189 & 5440 & 19 & 93.29 \\
& 999 & 28899 & 18 & 97.71 \\
LSTW(50) & 15 & 434 & 33 & 260.97 \\
& 189 & 5440 & 19 & 266.58 \\
& 999 & 28899 & 18 & 268.89 \\
& 2007 & 58186 & 18 & 270.20 \\
\hline
\end{tabular}
\caption{Scalability test on LSTW ontology by increasing example sets and ABox sizes.}
\end{table}

Figure 5.6: Scalability test on LSTW ontology by increasing example sets and ABox sizes.

it is expected the number of examples does not effect the efficiency. It can be found from Figure 5.6 that by increasing the size of example sets, the time taken to find solutions did not increase by the same magnitude. For example, in the Problem I over LSTW(100), the example sets increased by 67 percent but the time taken to finish this problem is worsen only by 0.02 percent. Additionally, in the Problem II over LSTW(100) ontology, the number of solutions decreased by 45 percent after increasing the example sets by 65 percent. However, for most of the ILP tools, which their methods are based on a bottom-up approach, the increase of example sets heavily effects the efficiency.
In summary, experimental results on scalability tests show that CL-EL handled ontologies with large ABoxes which could not be handled by DL-Learner ELTL. Moreover, CL-EL is efficient in finding a shorter list of correct concept descriptions that is at most 6 minutes for the largest ontology which has approximately 10 millions ABox assertions and the example sets include almost 73 thousands individuals.
Chapter 6

Conclusions

“What the caterpillar calls the end, the rest of the world calls a butterfly.”

–Lao Tzu

6.1 Summary of Results

Abductive and inductive reasoning are challenging and there is a rising need for
the current reasoners to be equipped with such reasoning tasks. A substantial
volume of research efforts have been dedicated to develop such methods over
description logics ontologies. However, existing approaches are often neither
scalable nor efficient even for lightweight DL ontologies. Motivated by such
needs, we have proposed new approaches on abductive and inductive reasoning
over $\mathcal{ELH}_\bot$ ontologies to facilitate such reasoning tasks.

6.1.1 Results on Query Abduction

In this thesis, the first aim was to have a scalable query abduction system
that considers more general queries than previous systems. To achieve this, we
slightly reformulated the query abduction problem. Our QAP includes general conjunctive queries as observations. Moreover, our QAP makes it possible to have a finite number of solutions for $\mathcal{ELH}_\bot$ ontologies by having a domain of individuals in QAP’s solutions.

For a practical algorithm, we proposed a query abduction system having observations formulated in BCQs and given ontologies in $\mathcal{ELH}_\bot$. Our approach involves transforming an $\mathcal{ELH}_\bot$ ontology to a datalog program and rewriting the observation. Ontology transformations and rewriting techniques have been intensively studied for query answering. In contrast, our work adapts such techniques from standard query answering to query abduction. Indeed, by adapting these techniques, we provide, to the best of our knowledge, the first sound and complete algorithm for query abduction that supports both general CQs and ontologies with genuine existential rules.

Furthermore, we introduced a novel method for module extraction in query abduction, by applying data summarisation techniques, and we developed a pruning strategy that largely enhances the efficiency of solution generation. Finally, the proposed approach of query abduction including all optimisations was developed as a query abduction system for $\mathcal{ELH}_\bot$ ontology named ABEL.

We implemented our new procedure of computing QAP solutions using Prolog and make use of the in-built list structure in Prolog. To compute (minimal) solutions to a QAP, we encode transformed ontology and query into Prolog rules. Then we use the list structure in Prolog to store the solutions generated during the resolution. Our encoding is different from previous approaches [30, 33], since we also encode existential rules to Prolog rules. This encoding was challenging since a naive encoding results in generating a large number of redundant facts that consume a large amount of time and memory.

For experimental results of the ABEL system, we presented an evaluation involving several QAPs. In the first phase, we compared the ABEL against
DuQAP system, where it turned out that ABEL is clearly more superior than DuQAP. ABEL-Naive yields promising results in those QAPs. Afterwards, we concluded the scalability evaluation by ABEL and ABEL-Sum over very large ontologies having 10 millions of ABox assertions. The overall success rates of ABEL was satisfactory over the test case ontologies. Moreover, experimental results on ABEL-Sum showed that order of magnitude improvements in performance are achieved on many cases where most of them could not be handled by ABEL system. Therefore, regarding scalability and accuracy, we believe to be the best implemented abductive system for lightweight DL ontologies.

The present study confirms previous findings and contributes additional evidence. This evidence suggests that proposed query abduction approach can be embedded into current DL reasoners to enhance the usability of ontologies. Finally, the results of this thesis support the idea that ontology approximation plays crucial role in query abduction over ontologies with existential rules.

### 6.1.2 Results on Concept Learning

An aim of this thesis was to propose a concept learning framework over lightweight DL ontologies which is scalable. We accomplished various tasks including both theoretically and in practice to develop such a scalable learning framework.

To create the search space for finding a correct concept description, we adapted a refinement operator for $\mathcal{ELH}_\bot$ ontologies. We proved that the refinement operator is downward, complete and locally finite. The unique characteristic of our operator is that it is specifically designed to generate the concept descriptions only in $\mathcal{ELH}_\bot$. It is also designed to create shorter concept descriptions in earlier stage of our approach.
We employed a new top-down approach to find correct concept descriptions in the search space. We also employed different heuristics embedded in the top-down approach to efficiently prune the search space. An improvement to the previous approaches is to initialise the search algorithm by a more fine-grained concept description rather than $\top$ concept. To that end, we adapted a data summarisation approach to summarise all positive examples and to define the initiative concept by collecting all of their types. This initiative, in most cases, is more fine-grained than $\top$ concept according to our experiments. To find such an initiative, no reasoning task was needed and it does not depend on TBox as well. A key strength of our approach is that it tends to have a compact list of correct concept descriptions where makes the task of a knowledge engineer effortless. This is achieved through a trimming procedure by pruning the search space efficiently to explore concept descriptions with shorter size.

To evaluate concept descriptions, an instance checker framework is proposed. In the framework, we first transformed the TBox to an approximated datalog program. Then, we employed datalog reasoner which are more scalable than DL reasoner. This instance checker is sound and complete for finding instances of any $\mathcal{ELH}_\bot$-concept. The present study should prove to be particularly valuable to handle $\mathcal{ELH}_\bot$ ontologies with millions of assertions for evaluating concept descriptions.

Eventually, we implemented our proposed algorithm on concept learning for $\mathcal{ELH}_\bot$ ontology called CL-EL. We conducted two experimental evaluations on CL-EL over different concept learning problems. The experimental results show that CL-EL is always faster than state of the art concept learning system for DL ontologies. Moreover, CL-EL founds less number of solutions where includes all short correct concept descriptions w.r.t. example sets. We highlighted in the experimental section that having a long list of solutions is not necessarily desirable for knowledge engineers. The second part of the experiments showed the scalability of CL-EL. It can handle an $\mathcal{ELH}_\bot$ ontology with
10 millions of ABox assertions, as well as, an example set of 73,000 in less than 6 minutes.

The present study makes several noteworthy contributions to concept learning over $\mathcal{ELH}_\bot$ ontologies. First, CL-EL is a scalable concept learning system for $\mathcal{ELH}_\bot$ ontologies. Second, the evidence from this study suggests that an initialisation of the search in a concept learning approach can reduce the number of concept description evaluation. Third, the findings of this research provide insights for using ontology approximation in any learning framework. Furthermore, the results of this study indicate that the proposed scalable approach on concept learning is capable of being employed in an ontology construction and enrichment frameworks. Finally, the contribution of this thesis has also been to confirm that a scalable instance checker is necessary for a learning framework.

### 6.2 Directions for Future Work

**Extending the query abduction approach:** In this work, a query abduction for general CQs over $\mathcal{ELH}_\bot$ ontologies is proposed. The study could be extended to DL-Lite, Horn DLs and more expressive DLs. It looks relatively more feasible to adapt our method to DL-Lite, since there is a combined approach [56] to query answering in DL-Lite. However, it is non-trivial to adapt our approach for more expressive DL since it has been less investigated a well-behaved ontology transformation and an observation rewriting for such ontologies except an attempt by Eiter et al. [78] for DL $\mathcal{SH}$.

**Alternative explanation generation method:** The query abduction algorithm we proposed in this thesis follows a resolution based approach to generate explanations through Prolog rules. A natural progression of this work is to investigate
the possibility of using the adapted ABox materialisation method to generate explanations by extra datalog rules. Such an investigation would result in a more efficient system since the current datalog reasoners are very scalable for lightweight DL ontologies.

*Improving the implemented algorithm of query abduction:* In this thesis, the adapted observation rewriting explores the structures of the CQs to some extent. Future research should therefore concentrate on a more fine-grained analysis of query structures and possibly specific structures of some ontology axioms that allow us to apply advanced heuristics in explanation construction. Furthermore, as our current implementation uses XSB as a black box, certain interventions with the Prolog system may allow for improvement to the efficiency of our prototype implementation.

*Learning more expressive concept descriptions:* With regards to future work on concept learning, extending the current approach to consider data type properties, cardinality restrictions, and nominals in the learning framework would be very interesting. While it is promising to learn data type properties with simple modification of current approach, for learning cardinality restriction and nominals, the open word assumption in DLs has to be reconsidered.

*Extending the concept learning approach:* In this work, we proposed a concept learning approach for $\mathcal{ELH}_1$ ontologies mainly to have a scalable learning system. Further research might investigate the possibility of applying our approach to more expressive DLs where theoretical results have to be reconsidered for the instance checking approach. Greater efforts are needed to ensure the soundness of a scalable instance checker using ontology approximation for ontologies with universal quantifiers, negation, and inverse roles. The work by
Zhou et al. [73] would be an interesting starting point.

Learning from noisy or unstructured data The scope of our concept learning approach was limited to learning a concept description from data presented in an ABox. It will be possible to take into account the noisiness of example sets as well as ABox assertions - a very challenging problem that has grabbed the attention of researchers in recent years. Another possible area of future research would be to extend learning concept descriptions from text files which can be classified as learning from unstructured data. There are recent promising attempts [79, 80] which are great help in this field.
Bibliography


[42] Jens Lehmann and Christoph Haase. Ideal downward refinement in the el description logic. In Luc De Raedt, editor, Proceedings of the 19th International Workshop on Inductive Logic Programming, volume 5989 of
**Bibliography**


