FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Compilation Technique for Interactive Ontology-mediated Data Exploration

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

European Master in Computational Logic

eingereicht von

Medina Andreșel
Matrikelnummer 1428616

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.techn. MSc Magdalena Ortiz de la Fuente
Mitwirkung: Univ.Ass. Dr.techn. MSc Mantas Šimkus

Wien, 22. Jänner 2016

Medina Andreșel	Magdalena Ortiz de la Fuente

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Compilation Technique for Interactive Ontology-mediated Data Exploration

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

European Master in Computational Logic

by

Medina Andresel
Registration Number 1428616

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Assistant Prof. Dr.techn. MSc Magdalena Ortiz de la Fuente
Assistance: Univ.Ass. Dr.techn. MSc Mantas Šimkus

Vienna, 22nd January, 2016

Medina Andresel	Magdalena Ortiz de la Fuente

Erklärung zur Verfassung der Arbeit

Medina Andreșel
Springergasse 27/11 1020 Wien

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

Wien, 22. Jänner 2016

Medina Andreșel

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Assistant Prof. Magdalena Ortiz for her continuous support on completing my master studies, for her patience, motivation, and immense knowledge. Her guidance helped me overcome difficult research-related situations and finish this master thesis. I could not have imagined having a better advisor and mentor. Her dedication and commitment are truly inspiring and guided me throughout the entire process of completing this thesis. My sincere gratitude also goes to my co-advisor Univ. Ass. Mantas Šimkus for all his useful advice, insightful ideas that always made a difference and nonetheless his great support during this research work. Their contribution to this thesis is invaluable.

I would like to thank the Joint Commission of the *European Master's Program in Computational Logic* for the opportunity to be a part of this master program and for the received financial support during my studies. I am very grateful to all the people from each partner university that are involved in the organization of this program, especially for their support regarding all the formalities and making the transition from one country to another one as smooth as possible.

Special thanks go to *Recursive Queries over Semantically Enriched Data Repositories* Austrian Science Funds (FWF) T515 for financially supporting throughout this research project.

Last but not least, I want to thank my family: my parents and my sister for all their encouragements and enormous support during my studies. To all my friends back home, in Vienna, Dresden, Bolzano, Lisbon, and Sydney - thank you all for making the last two years so memorable for me.

Abstract

Ontologies have been extensively advocated in the last decade as a means to enable a shared understanding of resources between different users and applications. One of their championed applications is *Ontology-based Data Access* (OBDA), where ontologies mediate access to data sources, providing a high-level conceptual view of the data and making background knowledge available for reasoning at query time. Much research efforts in the last years have been dedicated to the core problem in OBDA: answering a query while leveraging the knowledge given by an ontology, sometimes called *ontology mediated query answering* (OMQA). However, most work focuses on providing algorithms for answering queries one-at-a-time. Advanced systems that allow to analyze data by making modifications to queries and exploring their answers are taken for granted in the setting of standard relational databases, but in the setting of OMQA they still seem very far away from the primitive querying capabilities currently available.

In this thesis, we make a first step towards the interactive exploration of data in the ontology-mediated setting. We introduce a technique to construct an offline compilation that allows for efficiently answering different variations of a family of queries on the online phase, without the need to access the original data source. We consider ontologies formalized using *DL-Lite*, and focus on conjunctive queries (CQs) that are tree-shaped and have one answer variable. The compilation phase takes as input two queries that bound the information needs of the user from above and below, and then in the online phase, it allows to navigate between all the queries (of the mentioned restricted form) that fall between these two bounds. We also propose algorithms that construct relevant variations of a given query that may help the user analyze the data by, for example, reducing or increasing the number of answers in a minimal way, or identifying common properties of all objects that are in the answer to a given query. The experiments carried out with our prototype implementation reveal an overall good performance of the query-answering procedure and, most importantly, seem to be a promising first step towards flexible ontology-mediated data exploration.

Kurzfassung

Ontologien haben sich in den letzten Jahren zu einem wichtigen Mittel für das gemeinsame Verständnis von Ressourcen zwischen verschiedenen Nutzern und Anwendungen entwickelt. Eine ihrer wichtigsten Anwendungen ist *Ontologiebasierter Datenbangkzugriff* (engl. Ontology-based Data Access, OBDA), wobei Ontologien die Rolle von Schnittstellen zu Datenquellen einnehmen, welche eine abstrahierte Konzeptansicht der Daten darstellen und Hintergrundwissen für das automatische Schließen zum Abfragezeitpunkt verfügbar machen. Die meiste Forschung in diesem Bereich fokussiert sich auf die Bereitstellung von Algorithmen für die sequentielle Beantwortung einzelner Abfragen. Fortgeschrittene Systeme, welche die Analyse von Daten durch Modifikation von Anfragen und Untersuchung ihrer Antworten ermöglichen, werden im Bereich relationaler Datenbanken als selbstverständlich angesehen, im Kontext von OMQA sind diese jedoch noch weit von den derzeit verfügbaren Beantwortungsmöglichkeiten entfernt.

In dieser Arbeit machen wir einen ersten Schritt in Richtung interaktiver Exploration von Daten im Kontext von Ontologie-vermittelten Datenbankabfragen. Wir führen eine Technik zur Offline-Zusammenstellung ein, welche die effiziente Beantwortung verschiedener Variationen einer Familie von Abfragen zur Laufzeit erlaubt, ohne auf die ursprüngliche Datenquelle zurückgreifen zu müssen. Wir betrachten in *DL-Lite* formalisierte Ontologien und fokussieren uns auf konjunktive Anfragen (engl. conjunctive queries, CQ), die eine Baumstruktur und eine Antwortvariable besitzen. Die Zusammenstellungsphase hat als Eingabe zwei Abfragen, welche die Informationen des Nutzers von oben und unten beschränken, und es zur Abfragezeit erlaubt, zwischen allen Abfragen (der erwähnten Form) innerhalb dieser beiden Schranken zu navigieren. Wir schlagen auch Algorithmen vor, die relevante Variationen der gegebenen Anfrage konstruieren, die dem Nutzer helfen können die Daten zu analysieren, z.B. indem sie die Anzahl der Antworten auf minimale Weise erweitern oder reduzieren, oder gemeinsame Eigenschaften aller Objekte in der Antwort zu einer gegebenen Anfrage identifizieren. Die mit unserer Prototyp-Implementierung durchgeführten Versuche zeigen insgesamt eine gute Leistung in Bezug auf die Anfragenbeantwortungsprozedur, und scheinen vor allem ein vielversprechender erster Schritt in Richtung einer flexiblen Ontologie-vermittelten Datenexploration zu sein.

Contents

Abstract **ix**

Kurzfassung **xi**

Contents **xiii**

List of Figures **xiv**

List of Tables **xv**

List of Algorithms **xvii**

1 Introduction **1**

2 Problem Description **5**
2.1 Problem Description 5
2.2 Expected Results 6
2.3 State-of-the-art 7

3 Ontology-mediated Query Answering **9**
3.1 Description Logics 9
3.2 Query Answering in DL-Lite 13
3.3 Related Queries 14

4 Compiling the Answers for a Family of Related Queries **17**
4.1 Matching Witnesses 18
4.2 Query Answering based on Matching Witnesses 25
4.3 Correctness and complexity 26

5 Query Modifications **35**
5.1 Construction of Maximal Queries 35
5.2 Query Modifications 41

6 Implementation and Experiments **51**
6.1 Implementation Details 51

6.2 Experiments 52
6.3 Discussion 62

7 Conclusions and Further Research **65**

Bibliography **67**

List of Figures

3.1 Example TBox describing the domain of universities 12
3.2 Query that has its primal graph as tree. 15
3.3 Example of superquery and subquery for a given 1treeCQ 16

4.1 Example of an ABox $\mathcal{A}$ and a query $q(x)$ for which there are 2 incomparable matches that map x to individual P_1 18

5.1 Example of incomparable queries 36

6.1 Performance of QA and **x**-maximal query construction relative to the number of possible answers and matching witnesses 54
6.2 Analysis of QA and construction of **x**-max. queries performance over the data sets, and the possible answers dependence of query modifications for Batch 1 55
6.3 Example of obtained modifications for in-between query for Batch 1 55
6.4 Example of obtained modifications for in-between query of Batch 1 56
6.5 Performance of QA and **x**-maximal queries construction for Batch2 58
6.6 Analysis of QA and Max queries performance over the data sets and the possible answers dependence of query modifications for Batch 2 59
6.7 Example of query modifications for in-between queries of Batch 2 59
6.8 Performance of QA and **x**-maximal queries construction for Batch3 61
6.9 Analysis of QA and Max queries performance over the data sets and the possible answers dependence of query modifications for Batch 3 61
6.10 Example of obtained modifications for in-between query of Batch 3 62

List of Tables

3.1 Syntax and semantics for allowed constructs of concepts and roles 10
3.2 Syntax and semantics for *DL-Lite*$_{\mathcal{R}}$ allowed axioms and assertions 11

6.1 Datasets - Departments(D) distributed over Universities(U) 52
6.2 Datasets with #Matching witnesses(MWs) and #Possible answers (PA) for Batch1 53
6.3 Datasets with #Matching witnesses(MWs) and #Possible answers (PA) for Batch2 57
6.4 Datasets with #Matching witnesses(MWs) and #Possible answers (PA) for Batch3 60

List of Algorithms

4.1 ConstructAllMatchings 22
4.2 ConstructMatchWitness 22
4.3 ConstructMatchingCandidates 23
4.4 ConstructLabelString 23
4.5 GetAnswers 25
4.6 CheckRolesAndConcepts 26
4.7 IsAnswer 27

5.1 constructMaxQueries 37
5.2 combine 38
5.3 answersMaxQueries 38
5.4 neutralSpecialization 43
5.5 intersect 43
5.6 maxNeutralSpecialization 44
5.7 strictSpecialization 47
5.8 minStrictSpecialization 47
5.9 minGeneralization 49

CHAPTER 1

Introduction

Ontologies have been extensively advocated in the last decade as a means to enable a shared understanding of resources between different users and applications. In computer science, an ontology is a representation scheme describing a formal conceptualization of a domain of interest. Basically, an ontology defines a common vocabulary over a domain and simplifies information sharing. It includes machine-interpretable definitions of basic concepts in the domain and relations among them. Ontologies have gain a lot of interest in recent years considering that they offer several advantages such as: sharing common understanding of how the information is structured, analyzing and reusing the domain knowledge, they allow domain assumptions to be easily updated and they abstract away the actual data layer of applications. To sum up, ontologies have a very valuable role to play in making the *semantics* of resources available for understanding, sharing, and reasoning. For this reason, the nowadays known *Semantic Web* community has devoted special attention to developing standards for writing and sharing ontologies, most remarkably the *Ontology Web Languages* (OWL) [MGH+08], and a huge number of ontologies that capture different domains are now available.

One powerful application of ontologies that has received much attention in recent years is *Ontology-based Data Access* (OBDA), where data sources are connected to ontologies, and the ontologies mediate the access to the data. It has been argued that ontologies can help overcome some of the many challenges that today's information management requirements pose to traditional data management systems [PLC+08]. For example, they allow developers to have a unified conceptual view of possibly heterogeneous and distributed sources, and provide rich inference facilities over weakly structured and possibly incomplete data. In traditional OBDA, the data is assumed to be stored in relational databases, which are connected to the ontology using *mappings* (which are usually defined in SQL and can be seen as *views* that act as virtual relations over the ontology vocabulary). However, many other kinds of data sources, including web data, can be accessed using ontologies as mediators.

One of the central problems that arise in OBDA is evaluating a query in the presence of an ontology, over a given dataset that contains facts according to the ontology vocabulary (that is, without considering possible mappings from arbitrary data sources). This problem is sometimes called *ontology-mediated query answering*(OMQA).

In classical relational databases, query answering is done very efficiently. However, when adding an ontology on top of the data, query answering becomes a costly task. Moreover, the presence of the ontology and the implicit reasoning phase, that is intrinsic to query answering, can make the formulation of queries and the analysis of their answers very challenging.

One of the main gaps in OBDA is that the existing OMQA systems only support answering queries one-at-a-time. If given multiple variations of a query, for each one the query answering engine accesses the data for gathering the answers. Moreover, it is left to the users to find and formulate the query that precisely captures their information needs. This aspect represents the main motivation behind our efforts to provide support for efficiently answer slightly modified versions of a query. Considering the existing powerful tools in classical database management systems, such as *Online Analytical Processing* (OLAP) technique which allows interactive data analysis, we observe the need to compile relevant information in a meaningful way, in order to boost query answering over the extracted data fragment. Using an offline compilation, we are interested in providing efficient online full-query answering for *related queries*, and compute meaningful *query suggestions* for a given query.

For a better understanding, let's take an example. Suppose that an existing ontology together with the data allows the user to pose queries over an accommodation booking domain. The user is interested in finding 4 stars hotels in Vienna that are located in a central area and have available rooms. Provided that the data does not retrieve any answers for this request, we consider very useful to suggest a slightly modified query that will certainly retrieve some answers. In case that the query is retrieving answers and in fact all these hotels have more things in common, for example they all have Wi-fi and free breakfast, we find this extra information to be significant for the user and therefore we want to retrieve it. Another interesting situation is if the initial request has too many answers, then the user could be interested in knowing the different ways of specializing the query, in such way that the modified query still offers sufficient options that may allow different combinations of features which might be of interest for the user.

We are interested in providing algorithms for proposing to the user this kind of modifications, with instantaneous information of the effect they have on the query answers. In addition, another reason to tackle this particular problem is that, in literature there exists OMQA systems which focus on helping the user to build queries based on a given ontology. However they lack on providing support for data exploration. Our solution can be used as an extension for these query interfaces, provided that each such system constructs the type of queries we are considering. We are going to expand this particular aspect on the next chapter when presenting the state-of-the-art arguments of this thesis.

Structure of the Thesis

The thesis is structured as follows. Firstly, in Chapter 2 we give a more comprehensible description of the problem together with the expected results, the related work and the contributions of this thesis. Chapter 3 provides the preliminaries and introduces the formalization of the relation that helps characterize related queries. In Chapter 4 we introduce the formalization of the structure that will store the data compilation. A solution for computing the compilation and the compilation-based query answering procedure represent the main part of this chapter. The correctness proof of our solution and the complexity analysis for the proposed algorithms conclude this chapter.

In Chapter 5 we dive into the formalization of data exploration, we offer solutions in support of computing different types of query modifications. For each such modification we provide formalization, an algorithmic solution and we prove that the algorithm is sound and complete. The implementation of our solution and the experimental part are presented in Chapter 6, while the last part of the thesis, Chaper 7, describes the conclusions and further research.

CHAPTER 2

Problem Description

This chapter presents a more detailed description of the problem we are approaching alongside the expected results. An overview of the related work is presented in the state-of-the-art section.

2.1 Problem Description

In recent years there has been an intensive research in OMQA, in which the semantic knowledge provided by an ontology is exploited when querying data. As mentioned before, adding an ontology over the data provides several advantages such as simplification of query formulation, reasoning over the data provides more complete answers, easier integration of the data from other sources.

In general, the OMQA task is difficult, this aspect representing the downside of enriching data with domain knowledge. Most work on OMQA focuses on finding the answers to a *conjunctive query* (CQ) in the presence of ontologies formalized in *description logics* (DLs), which represent decidable fragments of classical first-order predicate logic. We are interested in analyzing query answering over tractable DLs. Thus, we consider *DL-Lite*, introduced in [CGL$^+$07].

The goal of this thesis is to develop techniques that allow interactive query answering for related queries, rather than answering only one given query. We are interested in exploring how part of the data projects onto a fragment of the ontology.

We focus on CQs that are *tree-shaped* and have exactly one answer variable. We assume that two initial queries are given, which we call the lower- and upper-bound query. Intuitively, the lower bound query simply says which kind of objects the user is interested in. It may be something very general, like a query that retrieves all persons, or all courses. The lower bound query is a query that gathers into a tree-shaped CQ all specific properties that all these objects may have, and that the user may be interested in. Then the family of related CQs is composed of all queries of this special form, that

ask for objects of the type given in the lower bound, that satisfy some combination of properties from the upper-bound.

Now, given such *upper* and *lower bound* queries of this special form, we will develop techniques for compiling all relevant information into a suitable data structure, in such a way that one can efficiently answer any in-between query. We will give in Chapter 3 a clearer picture of what an in-between query is. The property that defines related queries is as well formalized later on (see Section 3.3).

Computing the offline compilation will provide support in tackling further questions such as: given an in-between query, compute the *minimal query modification* such that the new obtained query is retrieving *strictly more* or *strictly less* answers. By modifying a query we mean constructing a new query that has some syntactical difference compared with the initial query such that the obtained query gives strictly different set of answers. In this case minimality is defined in terms of syntactical changes to be made for a given query.

Another interesting problem to tackle is how to compute *most specialized queries*, given an in-between query, such that the set of answers remains the same. Intuitively to specialize a query means to add more constraints.

We provide concrete formalization of each query modification in Chapter 5.

2.2 Expected Results

The main purpose of this thesis is to come up with a solution for exploring the answers of related queries. We expect a trade-off between constructing the compilation and the compilation-based interactive query answering solution. Precisely, we predict the offline compilation step to be expensive, however we aim at efficient online query answering.

We intend to provide correct algorithms and to test their behavior in practice. Moreover, our goal is to formalize the solution in such a way that can then be easily extended to deal with different DLs, such as $\mathcal{EL}$ [BBL05].

The most interesting result is to obtain meaningful query suggestions that will provide concrete support in exploring the answers of a given query. The suggestions represent minimal variations of a query that reduce or increase the number of answers, or help identify the common properties of the retrieved answers for a given query.

Provided that the problem we are approaching has concrete practical features, we intend to implement our solution into a prototype. We expect in the future to integrate the prototype into a system with a useful user interface that supports interactive query answering and provides the underpinnings for ontology-mediated data exploration.

2.3 State-of-the-art

Query answering in presence of ontologies has been intensively investigated, with multiple reasoners being created that offer support for different description logics. Among the existing systems some of them were tailored especially for query answering, such as *PAGOdA* [ZGN+15], *Ontop* [RKZ13], *Requiem* [PUHM09] or *Rapid* [CTS11].

The importance of this task has driven other reasoners that were not created exclusively for query answering, for example *Pellet* [SPG+07] and *HermiT* [GHM+14], to implement query answering related techniques.

We now give a short overview of the existing OMQA systems. Pellet is an OWL 2 DL reasoner which provides a SPARQL API and is capable of computing all certain answers for internalisable conjunctive queries using the rolling-up technique. HermiT is a complete OWL 2 reasoner which allows, among other reasoning tasks entailment checking and it can be also used for query answering. PAGOdA combines the datalog reasoner *RDFox* [MNP+14], which does the bulk of the computation, and therefore limits the calls to HermiT, which are proven to be more expensive. Other OMQA implemented systems in the literature such as *Ontop* [RKZ13], *Requiem* [PUHM09] or *Rapid* [CTS11] were specially created for query answering and are focusing on efficiency improvement.

For all these systems the usability might represent an issue, meaning that an user who is not familiarized with the ontology might find it difficult to write queries. New OMQA systems were developed, considering this issue, which are emphasizing the importance of a user-friendly interface. In particular, they are specially designed for users without prior ontology-related experience. The main functionality is to help users to express queries that capture their information needs. They provide theoretical underpinnings for query generation, since SPARQL is not targeted towards users that have no prior information about the ontology. These systems generate the same kind of queries we are considering in this thesis, hence our results might be of interest for an eventual support of interactive data exploration.

One of the first projects in this area is *Quelo* [FGTT11], which offers a natural language interface that starts composing the query from a simple phrase as "I am looking for something". Using a reasoning layer, it retrieves suggestions of possible constraints and allows the user to add, edit or remove these constraints. The resulting query is a tree-shaped conjunctive query. The query answering is done using existing reasoning engines.

Other projects with similar features are the Faceted Search Interface [ACGK+14] and the Optique *virtual query formulation system* (VQS) [SKZ+14]. Unlike Quelo, the Faceted approach uses operations as *tick* or *untick* possible constraints and does not use natural language for this purpose. The Faceted Search Interface also allows interactive query answering, although internally, since their method proposes tick options only if the new obtained version of the query has non empty answers and the retrieved answers set is strictly different.

Optique VQS is being developed under Optique project [1] and aims as well for an

[1] http://optique-project.eu/

easy-to-use graphical interface that will allow the user to compose complex queries.

In contrast to all above mentioned systems, our main goal is to create a solution that will allow efficient query answering and interactive exploration of the answers based on pre-compiled data. This problem has not been tackled by any other OMQA system that we know of, so this will be the main original aspect that the thesis will bring.

Regarding the main techniques used in OMQA, almost all the above named systems rely on query reformulations, or materializing the data by adding the inferred information, and respectively by generating for one query so-called *tree witnesses* which represent implied objects that are not explicitly part of the data. Our technique combines the last two since we generate on-the-fly the relevant inferred information without materializing the entire data in order to obtain the offline compilation. Moreover, our query answering technique uses only reasoning over the ontology, without the need to access the data once that the pre-compiled structure is available.

Among the few related work with respect query specializations is presented in [Nut11]. The authors consider arbitrary CQs and explore only the case of information overload by providing different methods to specialize a query for $\mathcal{EL}$-family. We are trying to consider the opposite task as well, since there can be cases in which the user would like to generalize the query for obtaining a larger set of answers. Our approach is computing minimal modifications in terms of the changes that need to be made on the structure of queries.

The next innovative aspect that the thesis brings is to compute most specialized version of a query that still retrieves the same set of answers. These query modifications basically help characterizing all the common properties that the answers of a query share over the ontology.

This theoretical approach of investigating relations among objects based on a pre-compilation of the data can provide further research questions and bring a fresh perspective in OMQA area.

CHAPTER 3

Ontology-mediated Query Answering

In this chapter we define the syntax and semantics of the DL ontologies we consider in this thesis. We also introduce the restricted form of queries that we focus on, and define a special *subquery* relation that will allow us to formalize the notion of 'related queries'.

3.1 Description Logics

Description Logics [BCM+03] are fragments of first-order logic specifically adapted towards the representation of structured knowledge. There are many description logics of varying expressiveness, suitable for modeling diverse domains of interest. They all describe the domain in terms of two basic elements:

- *concepts* - which correspond to classes and denote sets of objects
- *roles* - which correspond to relationships and denote binary relations on objects.

Domain knowledge is then captured by *ontologies*, which are a set of *logical axioms* that describe the way concepts and roles interrelate.

3.1.1 DL-Lite

In this work we focus on a family of DLs called *DL-Lite*, and in particular on *DL-Lite*$_{\mathcal{R}}$ [CGL+07]. The *DL-Lite* family was explicitly tailored to support tractable query answering over large data. *DL-Lite*$_{\mathcal{R}}$ is the language of choice for ontology-based data management and access, and it provides the logical underpinning for OWL 2 QL[1].

[1] http://www.w3.org/TR/owl2-profiles/#OWL_2_QL

Construct	Syntax	Example	Semantics
Atomic role	P	worksFor	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Inverse role	P^-	teacherOf$^-$	$\{(o, o') \mid (o', o) \in P^{\mathcal{I}}\}$
Role negation	$\neg R$	$\neg$memberOf	$(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus R^{\mathcal{I}}$
Atomic concept	A	Employee	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Existential restriction	$\exists R$	$\exists$advisor	$\{o \mid \exists o' \text{ s.t. } (o, o') \in R^{\mathcal{I}}\}$
Concept negation	$\neg B$	$\neg\exists$advisor$^-$	$\Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$
Top concept	$\top$		$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Individual	a	Prof1	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$

Table 3.1: Syntax and semantics for allowed constructs of concepts and roles

DL-Lite Syntax

We now define the syntax of the basic *DL-Lite*$_{core}$, and of *DL-Lite*$_{\mathcal{R}}$.

Definition 1. Let N_C, N_R and N_I be countably infinite sets of concept names, role names, and individuals. We define $\overline{N_R} = N_R \cup \{P^- \mid P \in N_R\}$ as the set of *(complex) roles*. For each $R \in \overline{N_R}$, the inverse of R is defined as $R^- := P^-$ if $R = P \in N_R$, and $R^- := P$ if $R := P^-$ for some $P \in N_R$.

An **ABox** is a finite set of *assertions* of the form $A(b)$, $\neg A(b)$ and $R(b, c)$, $\neg R(b, c)$, where $A \in N_C$, $R \in N_R$ and $b, c \in N_I$. We use $\mathbf{Ind}(\mathcal{A})$ to denote the **set of all individuals** appearing in an ABox $\mathcal{A}$.

A **TBox** is a finite set of *axioms*, whose form depends on the particular DL. For *DL-Lite*$_{core}$ the axioms are inclusion statements of the form $B \sqsubseteq C$, where:

$$B := A \mid \exists R \quad C := B \mid \neg B$$

and $A \in N_C$ is an **atomic concept**, $R \in \overline{N_R}$ is an **atomic role**. B defines a **basic concept**, R is **basic role**, and finally, C is a **general concept**.

DL-Lite$_{\mathcal{R}}$ extends *DL-Lite*$_{core}$ by additionally allowing axioms of the form $R \sqsubseteq E$ with R a basic role, and E a **general role**:

$$E := R \mid \neg R$$

A **knowledge base (KB)** $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ consists of an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$.

DL-Lite Semantics

The semantics of DL KBs is defined in terms of *interpretations* as follows.

Axiom	**Syntax**	**Example**	**Semantics**
Concept inclusion	$B \sqsubseteq C$	Employee $\sqsubseteq \exists$worksFor	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
Role inclusion	$R \sqsubseteq E$	worksFor $\sqsubseteq$ memberOf	$R^{\mathcal{I}} \subseteq E^{\mathcal{I}}$
Concept assertion	$A(a)$	Student(Stud1)	$a^{\mathcal{I}} \in A^{\mathcal{I}}$
Role assertion	$P(a,b)$	teacherOf(Prof1, GradCourse1)	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$
Negated concept assertion	$\neg A(a)$	$\neg$Professor(Stud1)	$a^{\mathcal{I}} \notin A^{\mathcal{I}}$
Negated role assertion	$\neg P(a,b)$	$\neg$worksFor(Stud1, Dep1)	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin P^{\mathcal{I}}$

Table 3.2: Syntax and semantics for *DL-Lite*$_{\mathcal{R}}$ allowed axioms and assertions

Definition 2. An *interpretation* has the form $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where the *domain* $\Delta^{\mathcal{I}}$ is a non-empty set, the *interpretation function* $\cdot^{\mathcal{I}}$ maps each $a \in N_I$ to an object $a^{\mathcal{I}}$, each $A \in N_C$ to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each $P \in N_R$ to a binary relation $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ is extended to general concepts and roles in the usual way, see the right column of Table 3.1. An interpretation $\mathcal{I}$ **satisfies** a TBox axiom or ABox assertion σ, written $\mathcal{I} \models \sigma$, if the corresponding condition in Table 3.2 holds.

An interpretation $\mathcal{I}$ **satisfies** a TBox $\mathcal{T}$, $\mathcal{I} \models \mathcal{T}$, iff $\mathcal{I}$ satisfies all axioms in $\mathcal{T}$. An interpretation $\mathcal{I}$ **satisfies** an ABox $\mathcal{A}$, $\mathcal{I} \models \mathcal{A}$, iff $\mathcal{I}$ satisfies all assertions in $\mathcal{A}$. An interpretation $\mathcal{I}$ is a **model** of a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, written $\mathcal{I} \models \langle \mathcal{T}, \mathcal{A} \rangle$, iff it satisfies all axioms in $\mathcal{T}$ and all assertions in $\mathcal{A}$. For a KB $\mathcal{K}$, we let **Mods**$(\mathcal{K})$ denote the set of all models of $\mathcal{K}$.

A KB is **consistent** iff it has at least one model. Let Γ be a TBox $\mathcal{T}$ or a KB $\langle \mathcal{T}, \mathcal{A} \rangle$, and let Γ' be an assertion $C(a)$ or $E(a,b)$, or an inclusion $B \sqsubseteq C$ or $R \sqsubseteq E$. We say that Γ entails Γ', and write $\Gamma \models \Gamma'$, if $\mathcal{I} \models \Gamma$ implies $\mathcal{I} \models \Gamma'$ for every interpretation $\mathcal{I}$. We use the notation $\Gamma_1 \sqsubseteq_{\mathcal{T}} \Gamma_2$ as a shortcut for $\mathcal{T} \models \Gamma_1 \sqsubseteq \Gamma_2$, where $\Gamma_1 \sqsubseteq \Gamma_2$ is a concept or role inclusion.

We now give an example of a *DL-Lite*$_{\mathcal{R}}$ KB.

Example 1. As a running example we use the university domain. The following self-explanatory axioms are based on LUBM ontology [GPH05], and in particular on the restricted version in *DL-Lite*$_{\mathcal{R}}$ syntax presented in [KRRM$^+$14][2].

We consider the TBox in Figure 3.1. We note that the axiom Employee $\sqsubseteq \exists$worksFor.Organization is not in the *DL-Lite*$_{\mathcal{R}}$ syntax, but we use it as a shortcut for the following two axioms, where worksForOrg is a role name not occurring elsewhere:

$$\exists \text{worksForOrg}^- \sqsubseteq \text{Organization}$$
$$\text{worksForOrg} \sqsubseteq \text{worksFor}$$

We also consider the following ABox:

[2] https://github.com/ontop/iswc2014-benchmark/tree/master/LUBM

$$\begin{aligned}
\text{AdministrativeStaff} &\sqsubseteq \text{Employee} & \text{Faculty} &\sqsubseteq \text{Employee}\\
\exists \text{teacherOf}^- &\sqsubseteq \text{Course} & \exists \text{teacherOf} &\sqsubseteq \text{Faculty}\\
\text{Professor} &\sqsubseteq \text{Faculty} & \text{FullProfessor} &\sqsubseteq \text{Professor}\\
\exists \text{publicationAuthor} &\sqsubseteq \text{Publication} & \exists \text{publicationAuthor}^- &\sqsubseteq \text{Person}\\
\text{publicationAuthor} &\equiv \text{authorPublication}^- & \text{Department} &\sqsubseteq \text{Organization}\\
\exists \text{advisor} &\sqsubseteq \text{Person} & \exists \text{advisor}^- &\sqsubseteq \text{Professor}\\
\exists \text{takesCourse} &\sqsubseteq \text{Student} & \exists \text{takesCourse}^- &\sqsubseteq \text{Course}\\
\text{Employee} &\sqsubseteq \exists \text{worksFor.Organization} & \text{worksFor} &\sqsubseteq \text{memberOf}
\end{aligned}$$

Figure 3.1: Example TBox describing the domain of universities

$$\begin{aligned}
\text{FullProfessor}(\mathsf{Prof1}) \qquad & \text{publicationAuthor}(\mathsf{Pub1}, \mathsf{Stud1})\\
\text{teacherOf}(\mathsf{Prof1}, \mathsf{GradCourse1}) \qquad & \text{publicationAuthor}(\mathsf{Pub1}, \mathsf{Prof1})\\
\text{takesCourse}(\mathsf{Stud1}, \mathsf{GradCourse1})
\end{aligned}$$

Intuitively, the models of this KB are all interpretations where all assertions and inclusions in the ABox and the TBox are satisfied. For instance, where there is a full professor that teaches a course that is attended by some student who has co-authored a publication with him. This KB entails implicit facts that hold in all models. For instance, that Pub1 is a publication or, more interestingly, that Prof1 is an employee and hence works for an organization.

3.1.2 Canonical Model

We recall the following well known construction of a canonical interpretation for a *DL-Lite*$_\mathcal{R}$ KB:

Definition 3 (**Canonical interpretation**). For a *DL-Lite*$_\mathcal{R}$ KB $\langle \mathcal{T}, \mathcal{A} \rangle$ we construct an interpretation $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ such as the domain $\Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$ consists of all words $aR_1 \dots R_n$ $(n \geq 0)$, where $a \in \mathbf{Ind}(\mathcal{A})$, $R_i \in \overline{N_R}$ and:

- if $n \geq 1$, then $\mathcal{T}, \mathcal{A} \models \exists R_1(a)$ and there does not exist $R_1(a, b) \in \mathcal{A}$, for some $b \in \mathbf{Ind}(\mathcal{A})$; moreover for each $S \sqsubseteq_\mathcal{T} R_1$, we have that $\mathcal{T}, \mathcal{A} \models \exists S(a)$ implies $R_1 \sqsubseteq_\mathcal{T} S$;
- for $1 \leq i < n$, $\mathcal{T}, \mathcal{A} \models \exists R_i^- \sqsubseteq \exists R_{i+1}$, $R_i^- \neq R_{i+1}$, and for each $S \sqsubseteq_\mathcal{T} R_{i+1}$ it holds that $\mathcal{T}, \mathcal{A} \models \exists R_i^- \sqsubseteq \exists S$ implies $R_{i+1} \sqsubseteq_\mathcal{T} S$.

The interpretation function, $\cdot^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$ is defined as follows:

- $a^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} = a$ for all $a \in \mathbf{Ind}(\mathcal{A})$
- $A^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} = \{a \mid \mathcal{T}, \mathcal{A} \vDash A(a)\} \cup \{aR_1 \dots R_n \mid n \geq 1 \text{ and } \exists R_n^- \sqsubseteq_{\mathcal{T}} A\}$
- $P^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} = \{(a,b) \mid R(a,b) \in \mathcal{A} \text{ and } R \sqsubseteq_{\mathcal{T}} P\} \cup \{(b,a) \mid R(a,b) \in \mathcal{A} \text{ and } R \sqsubseteq_{\mathcal{T}} P^-\} \cup \{(w_1,w_2) \mid w_2 = w_1 S \text{ and } S \sqsubseteq_{\mathcal{T}} R\} \cup \{(w_2,w_1) \mid w_2 = w_1 S \text{ and } S \sqsubseteq_{\mathcal{T}} R^-\}$

We call words in $\Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$ of the form $aR_1 \dots R_n$, with $n > 0$, **anonymous objects**, and let $\mathbf{Anon\_Obj} := \Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}} \setminus \mathbf{Ind}(\mathcal{A})$ to be the **set of anonymous objects** for a given *DL-Lite*$_{\mathcal{R}}$ KB.

The following result is standard:

Fact 1 (**Canonical model existence** [CGL$^+$07]). For any given consistent *DL-Lite*$_{\mathcal{R}}$ KB $\langle \mathcal{T}, \mathcal{A} \rangle$ the interpretation $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ (constructed according to Definition 3) exists and is a model of the KB, called **the canonical model**.

3.2 Query Answering in DL-Lite

Conjunctive queries (CQs) represent a simple query language but which is able to express "common queries" that are intensively used by many applications. More precisely, CQs are a special form of first-order queries that can capture *select-project-join* fragment of *relational algebra* and SQL.

Definition 4 (**Conjunctive query**). A **conjunctive query** (CQ), is a first-order formula, $q(\overrightarrow{x}) \text{ :- } \varphi(\overrightarrow{x}, \overrightarrow{y})$, where φ is constructed using $\wedge$ from atoms of the form $A(t)$, $A \in N_C$ and $R(t,t')$, $R \in \overline{N_R}$, where t, t' are *terms* (*individuals* or *variables* from $\overrightarrow{x}, \overrightarrow{y}$). Variables $\overrightarrow{x}$ are called *answer variables* and those in $\overrightarrow{y}$ *bound variables*. All bound variables are implicitly existentially quantified. We use $\mathbf{vars}(q)$ to denote the set of all variables of a given CQ q.

It is known that any relational database corresponds to a first-order (FO) interpretation and querying databases can be alternatively viewed as *mapping query variables to elements of the interpretation's domain*.

Definition 5 (**CQ match**). Given a CQ q and an interpretation $\mathcal{I}$, **a match** π for q w.r.t. $\mathcal{I}$ is a mapping of each term in q to an element of $\Delta^{\mathcal{I}}$ such that:

- for each concept atom $A(t) \in q$ $\pi(t) \in A^{\mathcal{I}}$ and
- for each role atom $R(t,t') \in q$ $(\pi(t), \pi(t')) \in R^{\mathcal{I}}$.

If such a match π exists, we say that $\mathcal{I}$ **satisfies** q, written $\mathcal{I} \vDash_{\pi} q$.

Definition 6 (**Set of Answers**). Let $\mathfrak{I}$ be a FO interpretation and q a FO query with answer variables $(x_1, \ldots, x_n)$. We define **the set of answers** for q over $\mathfrak{I}$ as follows:

$$\mathbf{ans}(q, \mathfrak{I}) := \{(e_1, \ldots, e_n) \mid \mathfrak{I} \vDash_\pi q \text{ and } \pi(x_i) = e_i, 1 \leq i \leq n\}.$$

Usually in OMQA the so-called *certain answers semantics* is adopted, in which the retrieved answers are those that hold in every model of the KB. The intuition is that since we do not know which of the KB's models provides the correct description of the knowledge domain, one can trust only those answers that can be obtained from *every* model of the KB.

Definition 7 (**Certain answers**). Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite* KB, and let q be a CQ, with $n \geq 1$ answer variables. We define $\mathbf{cert}(q, \mathcal{K})$ to be **the set of certain answers** of q over $\mathcal{K}$, defined as follows:

$$\{(a_1, \ldots, a_n) \in \mathbf{Ind}(\mathcal{A})^n \mid (a_1^\mathfrak{I}, \ldots, a_n^\mathfrak{I}) \in \mathbf{ans}(q, \mathfrak{I}) \text{ for every } \mathfrak{I} \in \mathbf{Mods}(\mathcal{K})\}$$

Given a KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$ and a CQ q, we write that $\mathcal{T}, \mathcal{A} \vDash q(\vec{a})$ iff $\vec{a} \in \mathbf{cert}(q, \mathcal{K})$.

The following two facts about *DL-Lite*$_\mathcal{R}$ are well known [CGL$^+$07]. First, if the negative assertions and inclusions in a given KB do not cause inconsistency, then they have no effect on query answers and can be disregarded.

Fact 2. Let $\mathcal{K}$, and let $\mathcal{K}'$ be the result of removing all negtive inclusions and assertions from $\mathcal{K}$. If $\mathcal{K}$ is consistent, then $\mathbf{cert}(q, \mathcal{K}) = \mathbf{cert}(q, \mathcal{K}')$.

Query answering over inconsistent KBs is trivial, as the answers are simply all tuples of constansts of suitable arity. Moreover, the consistency of a given *DL-Lite*$_\mathcal{R}$ KB can be efficiently tested using off-the-shelf algotihms. Hence, in what follows, we may assume that KBs are consistent and do not contain negative assertions or inclusions.

Second, the canonical model we have described above (see Fact 1) can be used to answer any query over a given KB $\mathcal{K}$.

Fact 3. Let $\mathcal{K}$ be a consistent *DL-Lite*$_\mathcal{R}$ KB and let $\mathfrak{I}^\mathcal{K}$ be its canonical model. Then $\mathbf{cert}(q, \mathcal{K}) = \mathbf{ans}(q, \mathfrak{I}^\mathcal{K})$, for every CQ q.

3.3 Related Queries

In this section we define the special type of queries that we consider in the thesis.

Definition 8. **(Tree-shaped CQ, 1treeCQ)** Let q be a CQ and $\mathbf{term}(q) = \{t \mid t$ is a term in some atom of $q\}$ be the *set of terms* of Q. We construct for q its **primal graph** as follows: $\mathbf{term}(q)$ is the set of vertices and $\{\langle t, t' \rangle \mid R(t, t') \in q\}$ is the set of edges. Moreover, we define a *labeling function* $\mathcal{L}$ which maps each vertex and each edge to a set of concept names, respectively role names, as follows: $\mathcal{L}(t) = \{A \mid A(t) \in q\}$, $\mathcal{L}(\langle t, t' \rangle) = \{R \mid R(t, t') \in q\}$.

We call q a **tree-shaped** CQ if its primal graph is a tree. A CQ q is called **1treeCQ** if it is tree-shaped and has *exactly one answer variable*.

Example 2. For example, the following query is tree shaped, and its primal graph is represented in Figure 3.2. Since it has one answer variable, it is also a 1treeCQ.

$$q(x) : -\text{Professor}(x), \text{worksFor}(y_1), \text{Department}(y_1), \text{teacherOf}(x, y_2), \text{Course}(y_2),$$
$$\text{authorPublication}(x, y_3), \text{Publication}(y_3), \text{publicationAuthor}(y_3, z_1), \text{Person}(z_1)$$

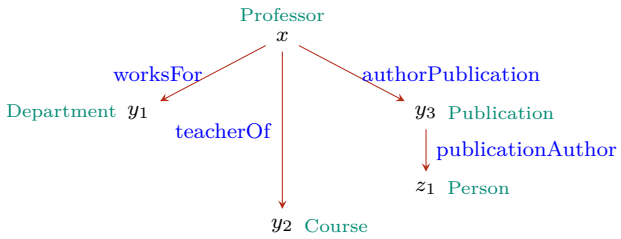


Figure 3.2: Query that has its primal graph as tree.

We now define a special subquery relation between 1treeCQs, that takes into account the tree-structure of the query, and the way concepts and roles imply each other in the TBox. Intuitively, a query is a subquery of another if it is a subtree rooted at the same node, and all the labels of the subquery are more general. Similarly, superqueries can have more nodes and edges, and have more specific labels.

Definition 9 (**Subquery**). Given a *DL-Lite*$_\mathcal{R}$ TBox $\mathcal{T}$ and two 1treeCQs q_1 and q_2, we call q_1 **subquery** of q_2 (w.r.t. $\mathcal{T}$), written $q_1 \trianglelefteq_\mathcal{T} q_2$, iff $vars(q_1) \subseteq vars(q_2)$ and

- for each atom $R_1(y_1, y_2) \in q_1$ there exists $R_2(y_1, y_2)$ or $R_2^-(y_2, y_1) \in q_2$ such that $R_2 \sqsubseteq_\mathcal{T} R_1$ and
- for each atom $C_1(y) \in q_1$ there exists $C_2(y) \in q_2$ such that $C_2 \sqsubseteq_\mathcal{T} C_1$.

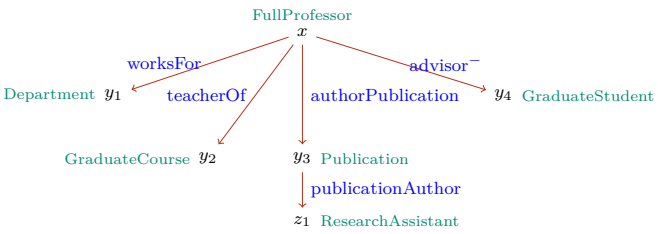
Symmetrically, we call q_2 **superquery** of q_1 (w.r.t. $\mathcal{T}$).

Example 3 (**Superquery/Subquery Example**). Let q be the query from Figure 3.2 and TBox $\mathcal{T}$ be such in Example 3.1. We add the following axioms:

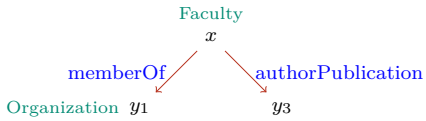
$$\text{GraduateStudent} \sqsubseteq \text{Person} \sqcap \exists \text{takesCourse.Course}$$
$$\text{ResearchAssistant} \sqsubseteq \text{Person} \sqcap \exists \text{worksFor.ResearchGroup}$$
$$\text{GraduateCourse} \sqsubseteq \text{Course}$$

An example of a superquery for q is represented in Subfigure 3.3a. As we can see a superquery must contain at least the same set of terms, while the corresponding node and edge labels, can be more restricted with respect to the TBox.

A subquery for q, illustrated in Subfigure 3.3b, can miss out labels or even edges, while the corresponding labels (for nodes and edges) can be more general with respect to the TBox (that is, super-concepts and super-roles).



(a) Superquery for q



(b) Subquery for q

Figure 3.3: Example of superquery and subquery for a given 1treeCQ

REMARK 1. Since "$\sqsubseteq_{\mathcal{T}}$" is transitive for any TBox $\mathcal{T}$, we obtain that "$\trianglelefteq_{\mathcal{T}}$" is also a transitive relation.

Definition 9 expresses the intuition for the characterization of related queries.

However, in order to pre-compile all information needed to correctly retrieve all answers for each such query, one needs to define boundaries in order to capture only a targeted fragment of the KB.

DEFINITION 10 (**1treeCQ Family**). For a given *DL-Lite*$_{\mathcal{R}}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$, let $\mathrm{q_L}$, $\mathrm{q_U}$ be two 1treeCQs such that $\mathrm{q_L} \trianglelefteq_{\mathcal{T}} \mathrm{q_U}$ holds. We call them **the lower bound query**, respectively **the upper bound query**.

Given a *DL-Lite*$_{\mathcal{R}}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$ and two 1treeCQ as $\mathrm{q_L}$, $\mathrm{q_U}$, any 1treeCQ q such that $\mathrm{q_L} \trianglelefteq_{\mathcal{T}} q \trianglelefteq_{\mathcal{T}} \mathrm{q_U}$ is called **in-between query**.

We relate the expression **1treeCQs family** to the set of 1treeCQs containing $\mathrm{q_L}$, $\mathrm{q_U}$ together with all the in-between queries.

Therefore, we will be using the expression of *1treeCQs family* whenever there exists two bound queries: a *lower-bound query*, that is a subquery of any other query, respectively an *upper-bound query* that is a superquery of any other query.

The following section focuses on the query answering task, setting the needed preliminaries for the formalization we are considering.

The following result will be useful:

COROLLARY 1. Let $\mathcal{K}$ be a given *DL-Lite*$_{\mathcal{R}}$ KB. For any two given 1treeCQs q_1, q_2 such that $q_1 \trianglelefteq_{\mathcal{T}} q_2$, the following holds:

$$\mathbf{ans}(q_2, \mathcal{I}) \subseteq \mathbf{ans}(q_1, \mathcal{I}), \text{ for every } \mathcal{I} \in \mathbf{Mods}(\mathcal{K}).$$

Proof (sketch). The proof is straight forward since any variable in q_1 must appear in q_2 and any query atom in q_1 is correlated to a more specific one in q_2 and therefore, for every model $\mathcal{I}$ of $\mathcal{K}$, every match for q_2 is also a match for q_1 under $\mathcal{I}$. □

CHAPTER 4

Compiling the Answers for a Family of Related Queries

The main goal of this chapter is to compile, from a given *DL-Lite*$_\mathcal{R}$ KB, a structure that contains all the information needed to answer a *family of related 1treeCQs*.

Since all the queries from a 1treeCQs family have the same answer variable in order to distinguish it we give the following notation:

NOTATION 1. In what follows, we assume a distinguished term $\mathbf{x}$ that is used to denote the common answer variable of all the queries in a 1treeCQ family.

Note that the upper bound query requires all properties that the user is interested in to hold simultaneously. Hence, when viewed as a CQ, it will often have no answers, as these properties will in general not hold simultaneously for any object. However, we are not interested in the answers to the upper bound query itself, but in the answers to all the queries that can be generated by taking different combinations of these properties.

As a remark, the lower- and upper-bound queries are not strictly needed but they are useful since they bound the information that the user wants to navigate. If not given, we can take some trivial lower- and upper-bound queries. For example take $q_L(\mathbf{x}) : -\top$ and q_U as containing all possible atoms that can be constructed using the TBox syntax. In this case we will compile answers for all 1treeCQs that can be constructed over the TBox vocabulary.

We want afterwards to use the offline compilation for retrieving the answers of any in-between 1treeCQ, without needing to consult the ABox again. To this aim, we observe that if an individual is an answer to some in-between query, then it must be an answer to q_L (see Corollary 1). Hence we use the answers of q_L as *possible answers* for the in-between queries, and for each of these individuals a, we identify the maximal queries in the 1treeCQs family for which a is an answer.

We note that there may be multiple such queries that are *incomparable* w.r.t. the KB. As an example, let's take the TBox described in 3.1 together with the following axioms:

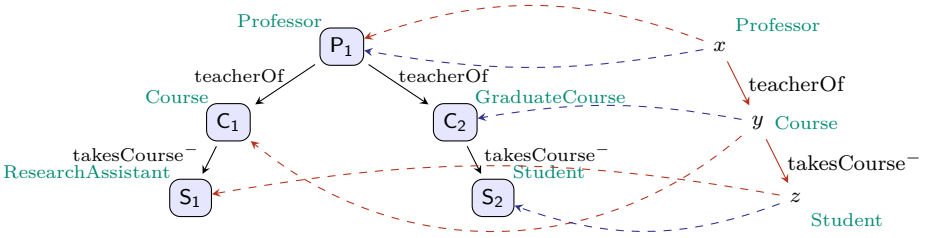


Figure 4.1: Example of an ABox $\mathcal{A}$ and a query $q(x)$ for which there are 2 incomparable matches that map x to individual P_1

$$\text{GraduateCourse} \sqsubseteq \mathit{Course} \qquad\qquad \text{ResearchAssistant} \sqsubseteq \text{Student}$$

Figure 4.1 illustrates an ABox represented as a graph, where each individual is a labeled node with all its satisfying concept names as labels, and the labeled edges represent the role assertions from the ABox. As showed in the figure, for query

$$q(x) : -\text{Professor}(x), \text{teacherOf}(x, y), \text{Course}(y), \text{takesCourse}^-, \text{Student}(z)$$

there exists two incomparable matches, w.r.t. the mentioned TBox, both mapping the answer variable x to individual P_1.

Remark 2. In case of KB inconsistency query answering is trivial and since consistency for a given *DL-Lite*$_\mathcal{R}$ KB can be tested in advance, we will not consider negative assertions in the ABox.

4.1 Matching Witnesses

In this subsection we introduce the notion of *matching witness*, which stores information about individuals and the maximal subqueries of $\mathrm{q_U}$ that they are an answer for. Given that we have a starting point from where to map into the canonical model, precisely the possible answers, one can build all the relevant matches for the 1treeCQs family by computing each *partial match* defined on terms $t_1, t_2 \in \mathbf{term}(\mathrm{q_U})$ for which there exists $R(t_1, t_2) \in \mathrm{q_U}$. More specifically, matching witnesses store how a given partial match from a term t in the upper-bound query to an object - an individual or an anonymous object - in the canonical model of $\mathcal{K}$ propagates upon the possible matches of its children.

The following definitions and notations come at hand for providing a formalization of such pre-compilation that will allow to answer any query in the 1treeCQ family.

Intuitively, we want to generate on the fly the objects in $\mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$ that are used in query answering for the 1treeCQs family, and we want to do that in an effective way by identifying the specific roles in the role hierarchy of some role atom in

q_U. We also need to store roles and concepts satisfied by the identified object but only those which are relevant for the 1treeCQs family, to avoid storing redundant information.

Note that the construction of $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ already uses this role inclusion specificity when generating the anonymous part. Thus, we introduce the following useful definitions:

Definition 11 (**Most specialized concepts and roles**). Given a *DL-Lite*$_{\mathcal{R}}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$ and any $a \in \Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$, we define:

$MS_{concept}(a, \mathcal{K}) := \{A \mid \mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash A(a)$ and for each A' s.t. $A' \sqsubseteq_{\mathcal{T}} A$, with $A' \neq A$, we have that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \nvDash A'(a)\}$ to be **the set of all most specialized concepts** satisfied by a w.r.t. $\mathcal{K}$

Respectively, for any $a, b \in \Delta^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$, we define:

$MS_{role}(a, b, \mathcal{K}) := \{R \mid \mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash R(a, b)$ and for each R' s.t. $R' \sqsubseteq_{\mathcal{T}} R$, with $R' \neq R$, we have that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \nvDash R'(a, b)\}$ to be **the set of all most specialized roles** satisfied by (a, b) w.r.t. $\mathcal{K}$.

Now, for each pair t_1, t_2 of query terms that occur together in some atom $R(t_1, t_2) \in q_U$, and some $v_1 \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$, we define a set of matching candidates. Intuitively, this set contains each v_2 such that if v_1 is a match for t_1 in $\mathcal{I}^{\mathcal{T},\mathcal{A}}$, then v_2 could be a match for t_2. That is, for some $R(t_1, t_2)$ in the query we have $(v_1, v_2) \in R^{\mathcal{I}^{\mathcal{T},\mathcal{A}}}$.

Recall that, by construction of $\mathcal{I}^{\mathcal{T},\mathcal{A}}$, this is possible if either both are individuals, or at least one of them is an anonymous object and a child of the other. The different possibilities are reflected in the items of the definition below.

Remark 3. We point out that in the canonical model (see Definition 3) we have $a^{\mathcal{I}} = a$, hence $\pi(a) = a$ for every query match π. Therefore in the following definitions, whenever a term of some query atom is an individual, it can only be mapped to itself.

Definition 12 (**Matching candidates**). Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_{\mathcal{R}}$ KB and let q_U be the upper bound CQ. Let $\{t_1, t_2\} \subseteq \mathbf{term}(q_U)$ be such that there is some atom $R(t_1, t_2)$ of q_U, and let $v_1 \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$.

The set of **matching candidates** for t_2 relative to $t_1 \mapsto v_1$ is

I. if $t_2 \in \mathbf{Ind}(\mathcal{A})$ then

- $\mathbb{MC}^{t_1 \mapsto v_1}(t_2) := \{t_2\}$, if there exists $R \in MS_{role}(v_1, t_2, \mathcal{K})$ s.t.$R' \sqsubseteq_{\mathcal{T}} R$ or $R \sqsubseteq_{\mathcal{T}} R'$ for some $R'(t_1, t_2) \in q_U$
- $\mathbb{MC}^{t_1 \mapsto v_1}(t_2) := \emptyset$, otherwise.

II. if $t_2 \in \mathbf{vars}(Q_U)$ then

$$\mathbb{MC}^{t_1 \mapsto v_1}(t_2) := cand_{\mathcal{A}} \cup cand_{vR} \cup cand_{wRS} \cup cand_{w}$$

where:

1. if $v_1 \in \mathbf{Ind}(\mathcal{A})$ then
   i. mapping t_2 in the ABox
   $cand_{\mathcal{A}} := \{v_2 \mid \mathcal{T}, \mathcal{A} \vDash R(v_1, v_2)$ where $R' \sqsubseteq_{\mathcal{T}} R$, for some $R'(t_1, t_2) \in \mathrm{q_U}\}$
   ii. mapping t_2 in the anonymous part
   $cand_{vR} := \{v_1R \mid \mathcal{T}, \mathcal{A} \vDash \exists R(v_1)$ where $R \in MS_{role}(v_1, v_1R, \mathcal{K})$ s.t. $R' \sqsubseteq_{\mathcal{T}} R$ or $R \sqsubseteq_{\mathcal{T}} R'$, for some $R'(t_1, t_2) \in \mathrm{q_U}\}$
2. if $v_1 := wR \in \mathbf{Anon\_Obj}$ then
   i. mapping t_2 further in the anonymous part
   $cand_{wRS} := \{wRS \mid \mathcal{T} \vDash \exists R^- \sqsubseteq \exists S$, where $S \in MS_{role}(wR, wRS, \mathcal{K})$ s.t. $R' \sqsubseteq_{\mathcal{T}} S$ or $S \sqsubseteq_{\mathcal{T}} R'$, for some $R'(t_1, t_2) \in \mathrm{q_U}\}$
   ii. mapping t_2 to the predecessor of v_1
   $cand_w := \{w \mid \mathcal{T} \vDash R^- \sqsubseteq S$, where $R' \sqsubseteq_{\mathcal{T}} S$, for some $R'(t_1, t_2) \in \mathrm{q_U}\}$

By constructing the matching candidates we compute partial matches that are relevant for some in-between query.

We now intend to compute the most specific concept and role names that such a partial match satisfies, which are important for the 1treeCQs family. For this purpose we give the following definition.

Definition 13 (**Labels String**). For a given a *DL-Lite*$_{\mathcal{R}}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$, $\mathrm{q_U}$ upper bound query and a partial match π, we define **the associated labels string** as follows:

I. if π is of the form $[t \mapsto v]$, where $t \in \mathbf{term}(\mathrm{q_U})$ and $v \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$ then

$$labels(t \mapsto v) := \langle \{C \mid C \in MS_{concept}(v, \mathcal{K}) \text{ s.t. } C' \sqsubseteq_{\mathcal{T}} C \text{ or } C \sqsubseteq_{\mathcal{T}} C', \text{ for some } C'(t) \in \mathrm{q_U}\} \rangle$$

II. if π is of the form $[t_1 \mapsto a, t_2 \mapsto b]$, where $t_1, t_2 \in \mathbf{term}(\mathrm{q_U})$, $a, b \in Ind(\mathcal{A}) \cup \mathbf{Anon\_Obj}$, such that $b \in \mathbb{MC}^{t_1 \mapsto a}(t_2)$ then

$$labels(t_1 \mapsto a, t_2 \mapsto b) := \langle \mathbb{R}_{Labels}, \mathbb{C}_{Labels} \rangle$$

where

$\mathbb{C}_{Labels} := \{C \mid C \in MS_{concept}(b, \mathcal{K})$ s.t. $C' \sqsubseteq_{\mathcal{T}} C$ or $C \sqsubseteq_{\mathcal{T}} C'$, for some $C'(t_2) \in \mathrm{q_U}\}$

is the set of satisfiable concept names, and $\mathbb{R}_{Labels}$ is constructed as follows:

A. if $a, b \in \mathbf{Ind}(\mathcal{A})$ or $a := bR$ then
   $\mathbb{R}_{Labels} := \{R \mid R \in MS_{role}(a, b, \mathcal{K})$ s.t. $R' \sqsubseteq_{\mathcal{T}} R$ or $R \sqsubseteq_{\mathcal{T}} R'$, for some $R'(t_1, t_2) \in \mathrm{q_U}\}$

B. if $b := aR$ then

$$\mathbb{R}_{Labels} := \{R\}$$

In the matching candidates and associated labels string, provided that we want to compute partial matches in any subquery of $\mathrm{q_U}$, but in the same time we want to avoid overloading the matching witnesses, we make use of the MS_{role} and $MS_{concept}$ definitions. However, since these definitions are not at all correlated to the upper-bound query, we need to make sure that we store only those most specialized roles and concepts which are relevant for the 1treeCQs family.

For example, given a partial match $t_1 \mapsto a$, $a \in \mathbf{Ind}(\mathcal{A})$ and suppose that there exists aR such that $MS_{role}(a, aR) = \{R\}$. Then aR is of relevance for the 1treeCQs family iff $R \sqsubseteq_{\mathcal{T}} S$ and there exists some $R'(t_1, t_2) \in \mathrm{q_U}$ such that $R' \sqsubseteq_{\mathcal{T}} S$. The same reasoning is used to identify those most specific concept names which are relevant for the 1treeCQs family.

Definition 14 (**Matching Witness**). For a given *DL-Lite*$_{\mathcal{R}}$ KB, lower bound 1treeCQ $\mathrm{q_L}$ and upper bound 1treeCQ $\mathrm{q_U}$, both rooted in $\mathbf{x}$, we define the **root matching witness**

$$w_{root} := \langle [labels(\mathbf{x} \mapsto a_1)a_1, \ldots labels(\mathbf{x} \mapsto a_n)a_n] \rangle,$$

where $\{a_1, \ldots a_n\} = \mathbf{ans}(\mathrm{q_L}, \mathcal{K})$.
Moreover, for a given term $t \in \mathbf{term}(\mathrm{q_U})$ and an object $a \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$ we construct a **matching witness** for t and a as follows:

$$w_a^t := \langle values_{t_1}, \ldots, values_{t_k} \rangle,$$

where $\{t_1, \ldots, t_k\} = \{t' \mid \text{ there exists } R(t, t') \in \mathrm{q_U}\}$, and $values_{t_i}$, $1 \leq i \leq k$, is constructed as follows:

1. $values_{t_i} := [\varepsilon]$ (ε is the empty string), if $\mathbb{MC}^{t \mapsto a}(t_i) = \emptyset$
2. $values_{t_i} := [\phi_1 b_1, \ldots, \phi_m b_m]$, where $b_j \in \mathbb{MC}^{t \mapsto a}(t_i)$, $1 \leq j \leq m$, and $\phi_j := labels(t \mapsto a, t_i \mapsto b_j)$.

Algorithm 4.1: ConstructAllMatchings

Input: $\mathrm{q_L}$,$\mathrm{q_U}$ 1treeCQs, *DL-Lite*$_\mathcal{R}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$
Output: $\mathbb{W}$ set of matching witnesses

$\mathbb{W} := \{\}$;
$values_{\mathbf{x}} := \varepsilon$;
foreach $a \in ans(\mathrm{q_L}, \mathcal{K})$ **do**
  $\phi_a := \{C \mid C \in MS_{concept}(a, \mathcal{K})$ s.t. $C' \sqsubseteq_\mathcal{T} C$ or $C \sqsubseteq_\mathcal{T} C'$, for some $C'(\mathbf{x}) \in \mathrm{q_U}\}$;
  `/* Append string "`$\phi_a a$`" to` $values_{\mathbf{x}}$ `*/`
  $values_{\mathbf{x}} := values_{\mathbf{x}} + \phi_a a_k +$ ",";
  `ConstructMatchWitness(`a_k,$\mathrm{q_U}$,$\mathcal{K}$,$\mathbb{W}$`)`
end
$w_{root} := \langle [values_{\mathbf{x}}] \rangle$;
add w_{root} **to** $\mathbb{W}$;

Algorithm 4.2: ConstructMatchWitness

Input: $a \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$, query $\mathrm{q_U} \downarrow_t$,where $t \in \mathbf{term}(\mathrm{q_U})$, *DL-Lite*$_\mathcal{R}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$, $\mathbb{W}$ set of matching witnesses
Output: matching witness w_a^t

$w_a^t := \langle\ \rangle$;
$child(t) := \{t' \mid$ there exists $R(t, t') \in \mathrm{q_U}\}$;
foreach $t' \in child(t)$ **do**
  $values_{t'} := \varepsilon$;
  $values\_set := \{\}$;
  $\mathbb{MC}^{t \mapsto a}(t') :=$ `ConstructMatchingCandidates(`t,a,t',$\mathrm{q_U}$,$\mathcal{K}$`)` ;
  foreach $v \in \mathbb{MC}^{t \mapsto a}(t')$ **do**
    $\phi_v :=$ `ConstructLabelString(`t,a,t',v,$\mathrm{q_U}$,$\mathcal{K}$`)` ;
    $values_{t'} := values_{t'} + \phi_v v +$ ", " ;
    $values\_set := values\_set \cup \{v\}$;
  end
  $w_a^t := w_a^t + [values_{t'}] +$ ",";
  foreach $v \in values\_set$ **do**
    `ConstructMatchWitness(`v,$\mathrm{q_U} \downarrow_{t'}$,$\mathcal{K}$`)`
  end
end
add w_a^t *to* $\mathbb{W}$;

Algorithm 4.3: ConstructMatchingCandidates

Input: q_U 1treeCQ, *DL-Lite*$_\mathcal{R}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$, $t_1, t_2 \in \mathbf{term}(q_U)$ and $a \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$

Output: the set of matching candidates for t_2, when $t1 \mapsto a$

$\mathbb{MC} := \{\}$;
if $t_2 \in \mathbf{Ind}(\mathcal{A})$ **then**
  if *there exists* $R \in MS_{role}(a, t_2, \mathcal{K})$ *s.t.* $R' \sqsubseteq_\mathcal{T} R$, *for some* $R'(t_1, t_2) \in q_U$ **then**
    return $\{t_2\}$;
  end
end
if $a \in \mathbf{Ind}(\mathcal{A})$ **then**
  /* mapping t_2 in the ABox */
  $\mathbb{MC} := \mathbb{MC} \cup \{b \mid \mathcal{T}, \mathcal{A} \vDash R(a, b)$, where $R' \sqsubseteq_\mathcal{T} R$, for some $R'(t_1, t_2) \in Q_U\}$;
  /* mapping t_2 in the anonymous part */
  $\mathbb{MC} := \mathbb{MC} \cup \{aR \mid \mathcal{T}, \mathcal{A} \vDash \exists R(a)$, where $R \in MS_{role}(a, aR, \mathcal{K})$ s.t. $R' \sqsubseteq_\mathcal{T} R$ or $R \sqsubseteq_\mathcal{T} R'$, for some $R'(t_1, t_2) \in q_U\}$
else
  /* a is an anonymous object */
  let $a := wR$, $wR \in \mathbf{Anon\_Obj}$;
  /* mapping t_2 further in the anonymous part */
  $\mathbb{MC} := \mathbb{MC} \cup \{wRS \mid \mathcal{T} \vDash \exists R^- \sqsubseteq \exists S$, where $S \in MS_{role}(wR, wRS, \mathcal{K})$ s.t. $R' \sqsubseteq_\mathcal{T} S$ or $S \sqsubseteq_\mathcal{T} R'$ for some $R'(t_1, t_2) \in q_U\}$;
  /* mapping y to the predecessor of a */
  $\mathbb{MC} := \mathbb{MC} \cup \{w \mid \mathcal{T} \vDash R^- \sqsubseteq S$, where $R' \sqsubseteq_\mathcal{T} S$ for some $R'(t_1, t_2) \in q_U\}$;
end
return $\mathbb{MC}$

Algorithm 4.4: ConstructLabelString

Input: q_U 1treeCQ, *DL-Lite*$_\mathcal{R}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$, $t_1, t_2 \in \mathbf{term}(q_U)$ and $a, b \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$

Output: the associated labels string

$\mathbb{C}_{Labels} := \{C \mid C \in MS_{concept}(a, \mathcal{K})$ s.t. $C' \sqsubseteq_\mathcal{T} C$ or $C' \sqsubseteq_\mathcal{T} C$, for some $C'(t_2) \in q_U\}$;
if $a, b \in \mathbf{Ind}(\mathcal{A})$ *or* a *is of form* bR **then**
  $\mathbb{R}_{Labels} := \{R \mid R \in MS_{role}(a, b, \mathcal{K})$ s.t. $R' \sqsubseteq_\mathcal{T} R$ or $R \sqsubseteq_\mathcal{T} R'$, for some $R'(t_1, t_2) \in q_U\}$;
end
if b *is of form* aR **then**
  $\mathbb{R}_{Labels} := \{R\}$;
end
return $\langle \mathbb{R}_{Labels}, \mathbb{C}_{Labels} \rangle$;

4.1.1 Computing Matching Witnesses

In this section we are going to present an algorithm that is computing the set of all matching witnesses $\mathbb{W}$, for a given problem instance: a *DL-Lite*$_{\mathcal{R}}$ KB and two 1treeCQs as $\mathrm{q_L}$ and $\mathrm{q_U}$. We provide in the following section, a solution for query-answering any in-between query based on the pre-computed matching witnesses.

NOTATION 2. Firstly we present some notations which will be used from this section onwards:

- A restriction of a 1treeCQ q to a subset of terms $\{t_1, \ldots, t_n\} \subset term(q)$, written as $q\,|_{\{t_1,\ldots,t_n\}}$, is a subquery of q formed by discarding all atoms which contain some term $t \notin \{t_1, \ldots, t_n\}$.
- For a given in-between query q and a term $t \in \mathbf{vars}(q)$, let $q \downarrow_t$ be a query obtained by discarding all nodes in the primal graph which are ancestors of t (respectively the corresponding query atoms in q), obtaining the sub-tree rooted in t.

Algorithm 4.1 computes the set of all matching witnesses $\mathbb{W}$ by iterating over the answers of $\mathrm{q_L}$ and for each answer a constructs $labels(\mathbf{x} \mapsto a)$. Algorithm 4.2 recursively computes for all children the matching candidates, using Algorithm 4.3, and the associated labels string constructed by Algorithm 4.4.

Algorithm 4.5: GetAnswers

Input: *DL-Lite*$_{\mathcal{R}}$ KB $\mathcal{K} := \langle \mathcal{T}, \mathcal{A} \rangle$, q in-between query, $\mathbb{W}$ set of all matching witnesses

Output: **ans**$(q, \mathcal{K})$

```
1  ans := {} ;
2  foreach w_a^x ∈ 𝕎 do
       /* x is the answer variable                                   */
3      is_answer := true ;
4      checkedAllChlidren := false ;
5      if CheckRolesAndConcepts(q|_{x}, labels(x ↦ a)) == false then
6          is_answer := false ;
7      end
8      while is_answer == true and checkedAllChildren == false do
9          child_x := {t | there exists R(x,t) ∈ q} /* children of x in q   */
10         foreach t ∈ child_x do
11             is_answer := IsAnswer(w_a^x, t, q ↓_t, 𝕎) ;
12             if is_answer == false then
13                 break foreach /* a is not an answer                 */
14             end
15             if t is last element in child_x then
16                 checkedAllNeigh := true ;
17             end
18         end
19     end
20     if is_answer == true then
21         ans := ans ∪ {a} ;
22     end
23 end
24 return ans ;
```

4.2 Query Answering based on Matching Witnesses

In this section we are going to provide an algorithmic solution for query answering any in-between query, for a given problem instance. Provided that we have already all information needed for this task stored in the matching witnesses, let $\mathbb{W}$ denote the set of all matching witnesses that is constructed beforehand by the Algorithm 4.1.

The following definition formalizes the conditions for which an individual or an anonymous object is part of a match for a given in-between query that is not necessarily rooted in $\mathbf{x}$. Given the tree-structure of any query in the 1treeCQs family, one can make use of recursion to check for a satisfiable match.

Definition 15 (**Satisfaction**). Let q be a 1treeCQ rooted in t, where $t \in \mathbf{term}(\mathrm{q_U})$,

Algorithm 4.6: CheckRolesAndConcepts

Input: Q in-between query, $\phi_v := \langle \mathbb{R}_{Labels}, \mathbb{C}_{Labels} \rangle$ associated label string
Output: **true** if ϕ_v satisfies concept and roles atoms in q, **false** otherwise

```
1  sat_labels := true ;
2  foreach R(t1, t2) ∈ q do
3  |  if φ_v does not contain any role name S s.t. S ⊑_T R then
4  |  |  sat_labels := false ;
5  |  |  break foreach ;
6  |  end
7  end
8  foreach C(t) ∈ q do
9  |  if φ_v does not contain any concept name A s.t. A ⊑_T C then
10 |  |  sat_labels := false ;
11 |  |  break foreach ;
12 |  end
13 end
14 return sat_labels
```

such that $q \trianglelefteq_{\mathcal{T}} \mathrm{q_U} \downarrow_t$. An object $b \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$ **satisfies** q **(w.r.t.** $\mathbb{W}$**)** iff:

1. if $t = \mathbf{x}$ then $b \in \mathbf{Ind}(\mathcal{A})$, and for every $C'(\mathbf{x}) \in q$ there exists $C \in labels(\mathbf{x} \mapsto b)$ such that $C \sqsubseteq_{\mathcal{T}} C'$
2. if t is a leaf in the tree-structure of q then $w_b^t := \langle\rangle \in \mathbb{W}$
3. for each t' child of t in q there exists v ($\in \mathbf{Ind}(\mathcal{A} \cup \mathbf{Anon\_Obj})$ in $values_{t'}$ of w_b^t such that $labels(t \mapsto b, t' \mapsto v) := \langle \mathbb{R}_{Labels}, \mathbb{C}_{Labels} \rangle$ satisfies:
   - for each $R'(t, t') \in q$ there exists $R \in \mathbb{R}_{Labels}$ such that $R \sqsubseteq_{\mathcal{T}} R'$
   - for each $C'(t') \in q$ there exists $C \in \mathbb{C}_{Labels}$ such that $C \sqsubseteq_{\mathcal{T}} C'$

   and v satisfies $q \downarrow_{t'}$ (w.r.t. $\mathbb{W}$).

Algorithm 4.7 encodes the satisfaction definition while Algorithm 4.5 gathers all the answers by checking if the associated labels strings satisfy conditions of item 3 in the above definition. Considering that the root matching witness collects all possible candidates and that each possible candidate is an answer for $\mathrm{q_L}$, if it satisfies the in-between query w.r.t. $\mathbb{W}$, implies there exists a match that maps $\mathbf{x}$ to that individual.

4.3 Correctness and complexity

The following theorem defines the correctness of Algorithm 4.5 that is indeed retrieving all answers for a given problem instance and an in-between query.

Algorithm 4.7: IsAnswer

Input: q_t 1treeCQ, t root of q_t, $w_v^{t_p}$ matching witness for parent of t, $\mathbb{W}$ set of all matching witnesses

Output: **true**, if there exists $v \in values_t$ in $w_v^{t_p}$ s.t. v satisfies q_t w.r.t. $\mathbb{W}$, **false** otherwise

```
1  if value_t == [ε] in w_v^{t_p} then
2  |  return false ;
3  end
4  is_answer := true ;
5  for v_t ∈ values_t do
6  |  if CheckRolesAndConcepts(Q|_{t_p,t}, labels(t ↦ v_t)) == false then
7  |  |  continue;
8  |  else
9  |  |  if t is a leaf and w_{v_t}^t ∈ 𝕎 then
   |  |  |  /* we matched the whole sub-tree                    */
10 |  |  |  return true ;
11 |  |  end
12 |  |  checkedAllChildren := false ;
13 |  |  while is_answer == true and checkedAllChildren == false do
14 |  |  |  child_t := {t' | there exists R(t,t') ∈ q}/* children of t in q_t
   |  |  |     */
15 |  |  |  foreach t' ∈ child_t do
16 |  |  |  |  is_answer := IsAnswer(w_{v_t}^t, t', Q↓_{t'}, 𝕎) ;
17 |  |  |  |  if is_answer == false then
18 |  |  |  |  |  break foreach /* v_p is not a match          */
19 |  |  |  |  end
20 |  |  |  |  if t' is last element of child_t then
21 |  |  |  |  |  checkedAllChildren := true ;
22 |  |  |  |  end
23 |  |  |  end
24 |  |  end
25 |  |  if is_answer == true and checkedAllChildren == true then
26 |  |  |  return true;
27 |  |  end
28 |  end
29 end
30 return is_answer;
```

THEOREM 1 (**Correctness of Constructing Matching Witnesses**). Given a *DL-Lite*$_\mathcal{R}$ KB $\mathcal{K}$, $\mathrm{q_L}, \mathrm{q_U}$ 1treeCQs, let $\mathbb{W}$ be the set of matching witnesses constructed by Algorithm 4.1 for $\mathcal{K}, \mathrm{q_L}$ and $\mathrm{q_U}$. For each in-between 1treeCQ q, $\mathrm{q_L} \trianglelefteq_\mathcal{T} q \trianglelefteq_\mathcal{T} \mathrm{q_U}$, the following holds:

$$\mathbf{cert}(q, \mathcal{K}) = \{a \mid a \in \mathbf{Ind}(\mathcal{A}) \text{ s.t. } a \text{ satisfies } q \text{ (w.r.t.}\mathbb{W})\}.$$

Proof. Let q be an arbitrary in-between query as above, and let

$$\mathbf{ans}_\mathbb{W}(q, \mathcal{K}) = \{a \mid a \in \mathbf{Ind}(\mathcal{A}) \text{ s.t. } a \text{ satisfies } q \text{ (w.r.t.}\mathbb{W})\}.$$

"$\subseteq$" **To prove that:** for q $\mathbf{cert}(q, \mathcal{K}) \subseteq \mathbf{ans}_\mathbb{W}(q, \mathcal{K})$
Let $a \in \mathbf{cert}(q, \mathcal{K})$ arbitrarily chosen. Then,

(1) $\mathcal{T}, \mathcal{A} \vDash q(a)$ which implies that there exists a match π s.t. $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\pi q$ and $\pi(\mathbf{x}) = a$.

To prove: a satisfies q w.r.t. $\mathbb{W}$.

(2) Since $q \trianglelefteq_\mathcal{T} \mathrm{q_U}$ we get that for each query atom $R(t_1, t_2) \in q$ there exists $R'(t_1, t_2) \in \mathrm{q_U}$ s.t. $R' \sqsubseteq_\mathcal{T} R$, and for each query atom $A(t) \in q$ there exists $A'(t) \in \mathrm{q_U}$ s.t. $A' \sqsubseteq_\mathcal{T} A$.

CLAIM 1. For each pair of terms $t, t' \in q$ s.t. there exists some $R(t, t') \in q$ the following holds:

$$\pi(t') \in \mathbb{MC}^{t \mapsto \pi(t)}(t')$$

Proof. In order to prove Claim 1, we need to follow the definition of matching candidates and to reason by cases.

1. if $t' \in \mathbf{Ind}(\mathcal{A})$ then in this case $\mathbb{MC}^{t \mapsto \pi(t)}(t') = \{t' \mid \mathcal{T}, A \vDash R(\pi(t), t')$ s.t. $R' \sqsubseteq_\mathcal{T} R$ or $R \sqsubseteq_\mathcal{T} R'$ for some $R'(t, t') \in \mathrm{q_U}\}$.
   Given the construction of $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ and **(1)** we get that $\pi(t') = t'$ and for each $R_q(t, t') \in q$ we have $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash R_q(\pi(t), t')$. Moreover, it follows from **(2)** that for some $R'(t, t') \in \mathrm{q_U}$ we have $R' \sqsubseteq_\mathcal{T} R_q$. Thus $\pi(t') \in \mathbb{MC}^{t \mapsto \pi(t)}(t')$.

2. if $\pi(t') := \pi(t)R$ then from construction of $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ it must be that $R \in MS_{role}(\pi(t), \pi(t'), \mathcal{K})$ and if $\pi(t) \in \mathbf{Ind}(\mathcal{A})$ then $\mathcal{T} \vDash \exists R(\pi(t))$, otherwise if $\pi(t) := wS \in \mathbf{Anon\_Obj}$ then $\mathcal{T} \vDash \exists S^- \sqsubseteq \exists R$. Moreover, also from construction of $\mathcal{I}^{\mathcal{T},\mathcal{A}}$ it follows that $R \in MS_{role}(\pi(t), \pi(t'), \mathcal{K})$. Given **(1)** we obtain $R \sqsubseteq_\mathcal{T} R_q$, for each $R_q(t, t') \in q$. Finally, using **(2)** we get that there exists some $R'(t, t') \in \mathrm{q_U}$ s.t. $R' \sqsubseteq_\mathcal{T} R \sqsubseteq_\mathcal{T} R_q$. Now, using Definition 12, it follows that $\pi(t') \in \mathbb{MC}^{t \mapsto \pi(t)}(t')$.

3. if $\pi(t) := \pi(t')R$ then using **(1)** follows that $\mathcal{T} \vDash R \sqsubseteq R_q$ for each $R_q(t, t') \in q$. From **(2)** and Definition 12 it follows that $\pi(t') \in \mathbb{MC}^{t \mapsto \pi(t)}(t')$.

4. if $\pi(t), \pi(t') \in \mathbf{Ind}(\mathcal{A})$ then from **(1)**, **(2)** and Definition 12 follows that $\pi(t') \in \mathbb{MC}^{t \mapsto \pi(t)}(t')$.

□

CLAIM 2. For each pair of terms $t, t' \in q$ s.t. there exists some $R(t,t') \in q$, $labels(t \mapsto \pi(t), t' \mapsto \pi(t')) := \langle \mathbb{R}_{Labels}, \mathbb{C}_{Labels} \rangle$ satisfy the following conditions:

a) for each $R_q(t,t') \in q$ there exists $R \in \mathbb{R}_{Labels}$ s.t. $R \sqsubseteq_{\mathcal{T}} R_q$ and

b) for each $A_q(t') \in q$ there exists $A \in \mathbb{C}_{Labels}$ s.t. $A \sqsubseteq_{\mathcal{T}} A_q$.

Proof. Claim 2 follows immediately from Definition 13 since all cases imply that whenever $R \in \mathbb{R}_{Labels}$ we have $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash R(\pi(t), \pi(t'))$ and $R \in MS_{role}(\pi(t), \pi(t'), \mathcal{K})$ (even the cases when $\pi(t') \in \mathbf{Anon\_Obj}_{\mathrm{q_U}}$ since $\pi(t') \in \mathbb{MC}^{t \mapsto \pi(t)}(t')$) s.t. for some $R'(t,t') \in \mathrm{q_U}$, $R' \sqsubseteq_{\mathcal{T}} R$ holds. From **(2)** we get that there exists some $R_q(t,t') \in q$ s.t. $R' \sqsubseteq_{\mathcal{T}} R_q$. Using definition of MS_{role} (see Definition 11) and **(1)** we can conclude that condition **a)** holds. Analogously, we can conclude that condition **b)** holds.

□

CLAIM 3. For each $t \in \mathbf{term}(q)$, $w^t_{\pi(t)} \in \mathbb{W}$.

Proof. For proving Claim 3 we will make use of the construction of matching witnesses algorithm. Algorithm 4.1 traverses the entire tree-structure of $\mathrm{q_U}$ starting from root $\mathbf{x}$. Since $\pi(\mathbf{x}) \in \mathbf{ans}(\mathrm{q_L}, \mathcal{K})$, then $w^{\mathbf{x}}_{\pi(\mathbf{x})} \in \mathbb{W}$. Next, Algorithm 4.2 iterates over each child t of $\mathbf{x}$ and using Claim 1 we get that $\pi(t) \in values_t$ (of $w^{\mathbf{x}}_{\pi(\mathbf{x})}$). Hence $w^t_{\pi(t)} \in \mathbb{W}$ and so on until the last level in (the tree-structure of) $\mathrm{q_U}$ is reached. □

CLAIM 4. For each $t \in \mathbf{term}(q)$, $\pi(t)$ satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$.

Proof. From **(2)** and the definition of $q \downarrow_t$, we conclude that $q \downarrow_t \trianglelefteq_{\mathcal{T}} \mathrm{q_U} \downarrow_t$. Proof by induction on the structure of Definition 15.

Induction base (IB): t is a leaf in the tree-structure of $q \downarrow_t$. Then either $t = \mathbf{x}$ or $t \neq \mathbf{x}$, from Claim 3 we get that $w^t_{\pi(t)} = \langle\rangle \in \mathbb{W}$.

Induction hypothesis (IH): For each t' child of t in (the tree-structure of) $q \downarrow_t$, $\pi(t')$ satisfies $q \downarrow_{t'}$ w.r.t. $\mathbb{W}$.

Induction step (IS): to prove that $\pi(t)$ satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$.

a) If $t = \mathbf{x}$ then given **(1)** and definition of $labels(t \mapsto \pi(t))$ implies that there exists $C \in labels(t \mapsto \pi(t))$, specifically $C \in MS_{concept}(\pi(t), \mathcal{K})$, s.t. $C \sqsubseteq_{\mathcal{T}} C_q$ for each $C_q(t) \in Q$. Moreover, using **(2)**, we get that there exists some $C' \in \mathrm{q_U}$ s.t. $C' \sqsubseteq_{\mathcal{T}} C \sqsubseteq_{\mathcal{T}} C_q$. Hence, using now **IH**, we get that $\pi(t)$ satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$.

b) if $t \neq \mathbf{x}$ then from Claim 3 follows that $w^t_{\pi(t)} \in \mathbb{W}$ and using Claim 1 we get that $\pi(t') \in \mathbb{MC}^{t \mapsto \pi(t)}(t')$, thus $\pi(t') \in values_t$ (of $w^t_{\pi(t)}$). Moreover, from Claim 2 it follows that $labels(t \mapsto \pi(t), t' \mapsto \pi(t'))$ satisfies conditions in Definition 15. Lastly, from **IH** follows that $\pi(t)$ satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$.

Using Claim 4 for $\mathbf{x}$ we conclude that $\pi(\mathbf{x}) := a$ satisfies q w.r.t. $\mathbb{W}$. Since a arbitrary chosen, we can now conclude that $\mathbf{cert}(q, \mathcal{K}) \subseteq \mathbf{ans}_{\mathbb{W}}(q, \mathcal{K})$.

□

"$\supseteq$" **To prove that:** $\mathbf{cert}(q, \mathcal{K}) \supseteq \mathbf{ans}_{\mathbb{W}}(q, \mathcal{K})$**.**
Let $a \in \mathbf{ans}_{\mathbb{W}}(q, \mathcal{K})$ arbitrarily chosen.

To prove that $a \in \mathbf{cert}(q, \mathcal{K})$ holds.

(3) a satisfies q w.r.t. $\mathbb{W}$ implies that:

i. $a \in values_{\mathbf{x}}$ s.t. for each $C_q(\mathbf{x}) \in q$ there exists $C \in labels(\mathbf{x} \mapsto a)$ s.t. $C \sqsubseteq_{\mathcal{T}} C_q$

ii. if $\mathbf{x}$ is a leaf then $w_a^{\mathbf{x}} := \langle\rangle \in \mathbb{W}$

iii. for each t child of $\mathbf{x}$ there exists $v_t \in values_t$ (of $w_a^{\mathbf{x}}$) s.t. for each $R_q(\mathbf{x}, t) \in q$, resp. $C_q(t) \in q$ there exists $R \in \mathbb{R}_{Labels}$, resp. $C \in \mathbb{C}_{Labels}$ (in $labels(\mathbf{x} \mapsto a, y \mapsto v_y)$) s.t. $R \sqsubseteq_{\mathcal{T}} R_q$, resp. $C \sqsubseteq_{\mathcal{T}} C_q$, and v_t satisfies $q \downarrow_t$ w.r.t. W.

Notation 3. For each pair $t, t' \in \mathbf{term}(q)$ s.t. there exists $R_q(t, t') \in q$, let $v_t, v_{t'} \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$ be such that v_t satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$ and $v_{t'}$ is the value in $values_{t'}$ of $w_{v_t}^t$ for which the labels conditions hold and $v_{t'}$ satisfies $q \downarrow_{t'}$. Whenever v_t and $v_{t'}$ are in the above described situation, we write $v_t \rightsquigarrow v_{t'}$.

Definition 16. For the ease of notation, we define a function *level* on terms in q which retrieves the term's level number in the tree-structure of q, with $level(\mathbf{x}) = 0$.

Claim 5. For each $t, t' \in \mathbf{term}(q)$ s.t. there exists some $R_q(t, t') \in q$, if $v_t \rightsquigarrow v_{t'}$ then $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\pi q\,|_{\{t,t'\}}$, where $\pi := [t \mapsto v_t, t' \mapsto v_{t'}]$.

Proof. For proving Claim 5 we distinguish two cases:

a) if $t = \mathbf{x}$ then $v_t \in \mathbf{ans}(\mathrm{q_L}, \mathcal{K})$. In this case $labels(t \mapsto v_t) := \{C \mid C \in MS_{concept}(v_t, \mathcal{K})$ s.t. for some $C'(t) \in \mathrm{q_U}, C' \sqsubseteq_{\mathcal{T}} C$ or $C \sqsubseteq_{\mathcal{T}} C'\}$. Using the definition of $MS_{concept}(v_t, \mathcal{K})$ we get that for each $C \in labels(t \mapsto v_t)$, $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash C(v_t)$ and since for each $C_q(t) \in q$ there exists $C \in labels(t \mapsto v_t)$ s.t. $C \sqsubseteq_{\mathcal{T}} C_q$ (because v_t satisfies $q \downarrow_t$), we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash C_q(v_t)$, thus $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\pi q\,|_{\{t\}}$, with $\pi := [t \mapsto v_t]$.

b) if $t \neq \mathbf{x}$ then Let t_p be the parent of t in the tree-structure of q. From **(3)** we get that there exists $v_{t_p} \rightsquigarrow v_t$. Also, since for each $C_q(t) \in q$ there exists some $C \in labels(t_p \mapsto v_{t_p}, t \mapsto v_t)$ s.t. $C \sqsubseteq_{\mathcal{T}} C_q$. From Definition 13 we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash C(v_t)$, since $C \in MS_{concept}(v_t, \mathcal{K})$. Hence $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash C_q(v_t)$ for each $C_q(t) \in q$. Thus $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\pi q\,|_{\{t\}}$, with $\pi := [t_p \mapsto v_{t_p}, t \mapsto v_t]$.

Let t' be a child of t. Now, v_t satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$ (from $v_{t_p} \rightsquigarrow v_t$) thus there must exists some $v_t' \in values_{t'}$ of $w_{v_t}^t$ s.t. for each $R_q(t, t') \in q$ there exists $R \in labels(t \mapsto v_t, t' \mapsto v_{t'})$ s.t. $R \sqsubseteq_{\mathcal{T}} R_q$, respectively for each $C_q(t') \in q$ there

exists $C \in labels(t \mapsto v_t, t' \mapsto v_{t'})$ s.t. $C \sqsubseteq_{\mathcal{T}} C_q$ and moreover, $v_{t'}$ satisfies $q \downarrow_{t'}$ w.r.t. $\mathbb{W}$. Hence $v_t \rightsquigarrow v_{t'}$.

Once again, since each $R \in \mathbb{R}_{Labels}$ and each $C \in \mathbb{C}_{Labels}$ (of $labels(t \mapsto v_t, t' \mapsto v_{t'})$) are contained in $MS_{role}(v_t, v_{t'}, \mathcal{K})$, respectively $MS_{concept}(v_{t'}, \mathcal{K})$ (even the case when $v_{t'} \in$ **Anon_Obj** because $v_{t'} \in \mathbb{MC}^{t \mapsto v_t}(t')$). This implies that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash R_q(v_t, v_{t'})$ and $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash C_q(v_{t'})$, for each $R_q(t, t') \in q$, respectively each $C_q(t') \in q$. Hence, we get that whenever $v_t \rightsquigarrow v_{t'}$, $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\pi q\,|_{\{t,t'\}}$ with $\pi := [t \mapsto v_t, t' \mapsto v_{t'}]$ holds.

□

Definition 17. Let γ be a mapping for each term $t \in \mathbf{term}(q)$ to a value in $\mathbf{Ind}(\mathcal{A}) \cup$ **Anon_Obj** constructed as follows:

a) if $t = \mathbf{x}$ then $\gamma(t) = a$

b) if there exists t_p parent of t in (the tree-structure of) q then if $\gamma(t_p) \rightsquigarrow v_t$ where $v_t \in values_t$ (of $w^{t_p}_{v_{t_p}}$) then $\gamma(t) = v_t$

Remark 4. From **(3)** we conclude that γ is defined on each term $t \in \mathbf{term}(q)$.

Claim 6. For each term $t \in \mathbf{term}(q)$, $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q \downarrow_t$.

Proof. Let t be an arbitrarily chosen term of q and n be the number of levels in q.We prove by induction on $level(t)$.

Induction base (IB): $level(t) = n - 1$ thus t is a leaf in the tree-structure of q, therefore $w^t_{\gamma(t)} := \langle\rangle \in \mathbb{W}$.

a) if $t = \mathbf{x}$ then $\gamma(t) = a$. Since $\gamma(t)$ satisfies $q \downarrow_t$ then for each $C_q(t) \in q$ there exists $C \in labels(t \mapsto \gamma(t))$ s.t. $C \sqsubseteq_{\mathcal{T}} C_q$. Since $C \in MS_{concept}(v_t, \mathcal{K})$ we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash C(\gamma(t))$, therefore $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash C_q(\gamma(t))$. Thus $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q \downarrow_t$.

b) if $t \neq \mathbf{x}$ then there exists t_p parent of t (in the tree-structure of) q. From Remark 4 we get that there exists $\gamma(t_p)$ s.t. $\gamma(t_p)$ satisfies $q \downarrow_{t_p}$ w.r.t. $\mathbb{W}$. Therefore there exists $\gamma(t) \in values_t$ (of $w^{t_p}_{\gamma t_p}$) s.t. $\gamma(t_p) \rightsquigarrow \gamma(t)$. Using Claim 5 we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q \mid \{t_p, t\}$, hence $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q\,|_{\{t\}}$ and since t is a leaf we get $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q \downarrow_t$.

Induction hypothesis (IH): for each t' child of t in (the tree-structure of) q, $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q \downarrow_{t'}$.

Induction step (IS): Following the same reasoning as in **IB** either $t = \mathbf{x}$ or $t \neq \mathbf{x}$ we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q\,|_{\{t\}}$. Once more from Remark 4 we conclude that for each child t' of t in (the tree-structure of) q there exists $\gamma(t')$ s.t. $\gamma(t) \rightsquigarrow \gamma(t')$. Using Claim 5 we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q\,|_{\{t,t'\}}$. Now, using **IH** we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q \downarrow_t$. □

Using Claim 6 when $t = \mathbf{x}$ we get that $\mathcal{I}^{\mathcal{T},\mathcal{A}} \vDash_\gamma q$. Since $a \in \mathbf{ans}_{\mathbb{W}}(q, \mathcal{K})$ arbitrarily chosen and since γ is a match for q and it was constructed based on **(3)** we can now conclude that whenever $a \in \mathbf{ans}_{\mathbb{W}}(q, \mathcal{K})$ then $a \in \mathbf{cert}(q, \mathcal{K})$. Therefore $\mathbf{ans}_{\mathbb{W}}(q, \mathcal{K}) \subseteq \mathbf{cert}(q, K)$ holds.

Since both $\mathbf{ans}_{\mathbb{W}}(q, \mathcal{K}) \subseteq \mathbf{cert}(q, K)$ and $\mathbf{ans}_{\mathbb{W}}(q, \mathcal{K}) \supseteq \mathbf{cert}(q, K)$ hold we obtain that $\mathbf{ans}_{\mathbb{W}}(q, \mathcal{K}) = \mathbf{cert}(q, K)$.
q is an arbitrarily chosen in-between query, therefore we can now claim that Theorem 1 holds.

□

4.3.1 Complexity Analysis

In this section we provide the complexity of constructing the entire set of matching witnesses and the complexity of the query answering procedure that makes use of the matching witnesses. The results are not surprising since trying to store information for all matches in the canonical model of every in-between query intuitively sounds like an exponential task, given that there can be exponentially many in-between queries. This pre-compilation represents the trade-off for providing a tractable (in the size of the input) query answering procedure, since it is known that query answering in *DL-Lite*$_{\mathcal{R}}$ is NP-hard.

Complexity of Constructing the Matching Witnesses Set

It is easy to observe that given a *DL-Lite*$_{\mathcal{R}}$ KB and two 1treeCQs as $\mathrm{q_L}$ and $\mathrm{q_U}$, Algorithm 4.1 will create exponentially many matching witnesses. Specifically, by iterating over each level in the tree-structure of $\mathrm{q_U}$, for each match of the parent t_p, the child t can be matched in $|\mathcal{A}| + l + 1$ ways, where $|\mathcal{A}|$ is the size of the ABox, $l = |\{R(t_p, t) \in \mathrm{q_U}\}|$ representing the maximal number of anonymous objects that can be created for t, which is liniar in size of $\mathrm{q_U}$, and 1 represents the case when t can be matched to the predecessor of t_p. Let k be the number of children of t in $\mathrm{q_U}$. Given the construction of a matching witness, it follows that

$$size(w^t_{v_t}) \leq k * ((|\mathbb{R}_{Labels} + \mathbb{C}_{Labels}|) * (|\mathcal{A}| + l + 1))$$

where $|\mathbb{R}_{Labels} + \mathbb{C}_{Labels}|$ represents the size of the associated label string, which is liniar in $\mathcal{K}$. Thus, we obtain that any matching witness is polynomial in the size of $\mathcal{K}$ and $\mathrm{q_U}$.

Since the tree-structure of $\mathrm{q_U}$ has n levels, it follows that at level i, $1 \leq i \leq n$, the number of matching witnesses created so far is bounded by $(|\mathcal{A}| + l + 1)^i$ since there are already at most $(|\mathcal{A}| + l + 1)^{(i-1)}$ matches for the parent and for each one there can be $(|\mathcal{A}| + l + 1)$ matching candidates. Thus, there can be exponentially many matching witnesses in the depth of $\mathrm{q_U}$.

Complexity of Query Answering based on Matching Witnesses

Algorithm 4.5, for input $\mathbb{W}$ and in-between query q, basically iterates over all $a \in \mathbf{cert}(\mathrm{q_L}, \mathcal{K})$ and collects only those that satisfy q w.r.t. $\mathbb{W}$.

We analyze the complexity of the decision problem for all possible answers. Precisely, for each a, the algorithm takes from the w_{root} the associated labels string to check if the concepts constraints are satisfied and then calls Subfunction 4.7 to verify if for each child t of $\mathbf{x}$, in (the tree-structure of) q, there exists $v \in values_t$ (of $w_a^{\mathbf{x}}$) such that v satisfies $q \downarrow_t$ w.r.t $\mathbb{W}$. Algorithm 4.7 is a recursive procedure, which can be used as an alternating procedure, to determine satisfaction for each term $t \neq \mathbf{x}$. Hence, it is possible to make use of alternation, to correctly and easily determine the time complexity for the task at hand [CKS81].

Since for each term t in q, the algorithm needs to store locally only the matching witness of the parent, the relevant atoms from the query and the current value from $values_t$, it can be done using only LOGSPACE working memory, hence we conclude that Algorithm 4.5 runs in ALOGSPACE. From the results of [CKS81], it follows that ALOGSPACE = PTIME, therefore our query answering procedure runs in polynomial time, in the size of $\mathbb{W}$ and q.

However, since $\mathbb{W}$ is exponentially large, we obtain that the query answering procedure measured in the KB $\mathcal{K}$, $\mathrm{q_L}$, $\mathrm{q_U}$ and the in-between query q runs in EXPTIME. This result is not at all surprising, since query answering in *DL-Lite*$_{\mathcal{R}}$ is known to be intractable in combined complexity [ACKZ09].

CHAPTER 5

Query Modifications

In this chapter, we show how the information stored in a matching witness can be used to explore our data. Given a query, we identify interesting variations of it that could be relevant to a user. For example, we identify the minimal ways to modify the current query so that it provides more or less answers. We are also interested in identifying all the common properties that our set of answers has, that is, the possible additions to our query that, while making it syntactically more restrictive, would still retrieve the same set of answers. In the experiments in the next section, we will illustrate the usefulness of these query modifications on several examples.

5.1 Construction of Maximal Queries

Intuitively, the set of constructed matching witnesses contains information regarding how much of the upper-bound query we have matched in the canonical model, so that some $a \in \mathbf{Ind}(\mathcal{A})$ is an answer. Hence we can read from the witness, for each individual a that is a possible answer, the set of maximal queries in the family for which a is a match. We will see that these maximal queries provide the key information to suggest the user the desired query modifications.

Before computing these maximal queries, we remark that there can be multiple such maximal queries for the same individual a.

Definition 18 (**Incomparable queries**). Two 1treeCQs q_1, q_2 are said to be **incomparable** iff $q_1 \not\sqsubseteq_{\mathcal{T}} q_2$ and $q_2 \not\sqsubseteq_{\mathcal{T}} q_1$.

We are going to give an example of why there can exist maximal incomparable queries. Suppose that TBox $\mathcal{T}$ is such as in Example 3.1 together with the following axioms:

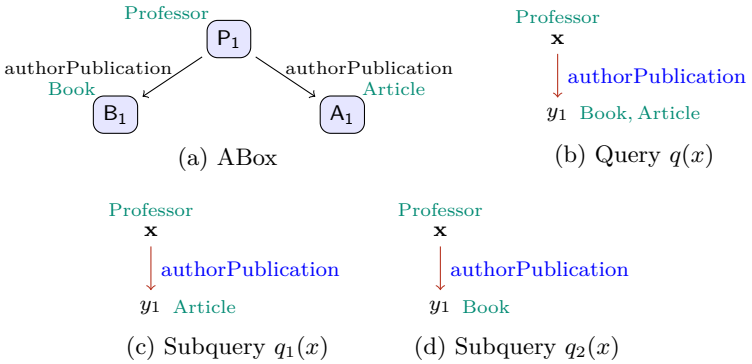
$$\text{Article} \sqsubseteq \text{Publication} \qquad \text{Book} \sqsubseteq \text{Publication}$$

Now, considering the ABox $\mathcal{A}$, represented in graph mode in Subfigure 5.1a, it is clear that query

$$q(\mathbf{x}) : -\text{Professor}, \text{authorPublication}(\mathbf{x}, y_1), \text{Book}(y_1), \text{Article}(y_1)$$

that has its primal graph as in Subfigure 5.1b, does not have a satisfiable match w.r.t. $\langle \mathcal{T}, \mathcal{A} \rangle$.

However, there exists subqueries of q, e.g. Subfigure 5.1c and Subfigure 5.1d for which there are satisfiable matches w.r.t. the KB.



(a) ABox (b) Query $q(x)$

(c) Subquery $q_1(x)$ (d) Subquery $q_2(x)$

Figure 5.1: Example of incomparable queries

Definition 19 (**t-Maximal Query**). Let q be a 1treeCQ rooted in t, where $t \in \mathbf{vars}(\mathrm{q_U})$, s.t. $q \trianglelefteq_{\mathcal{T}} \mathrm{q_U} \downarrow_t$. Query q is said to be **t-maximal** for $a \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$ iff:

(1) a satisfies $q \downarrow_t$ w.r.t. $\mathbb{W}$, and

(2) for each $q' \downarrow_t$ s.t. $q \downarrow_t \trianglelefteq_{\mathcal{T}} q' \downarrow_t \trianglelefteq_{\mathcal{T}} \mathrm{q_U} \downarrow_t$, a **does not satisfy** $q' \downarrow_t$ w.r.t. $\mathbb{W}$.

Suppose that $\mathrm{q_U}$ is the query represented in 5.1b. Then, queries 5.1c and 5.1d are both $\mathbf{x}$-maximal for $\mathsf{P_1}$ and they are incomparable considering the above presented KB.

In what follows, we assume fixed upper- and lower-bound queries $\mathrm{q_L}$ and $\mathrm{q_U}$, and use $\mathbb{W}$ to denote the set of all matching witnesses constructed in previous Chapter 4. We use $\mathbf{ans}(q, \mathbb{W})$ to denote the *answers for* q *over* $\mathbb{W}$, which are the individuals a such that a satisfies q w.r.t. $\mathbb{W}$ (see Theorem 1).

Now, we will provide a method of constructing these $\mathbf{x}$-maximal queries for some individual $a \in \mathbf{ans}(\mathrm{q_L}, \mathbb{W})$.

Algorithm 5.1: constructMaxQueries

Input: $b \in \mathbf{Ind}(\mathcal{A}) \cup \mathbf{Anon\_Obj}$, $t \in \mathbf{term}(\mathrm{q_U})$, $\mathbb{W}$
Output: the set of t-maximal queries for b

$q_t := \{C(t) \mid C \in labels(t \mapsto b)\}$;
$Q^t = \{\}$;
$S := \{\}$;
foreach t' *s.t.* $values_{t'} \neq [\varepsilon]$ *in* w_b^t **do**
  $Q^{t'} := \{\}$;
  foreach $v \in values_{t'}$ **do**
    $q_{t'}^v := \{R(t,t') \mid R \in labels(t \mapsto b, t' \mapsto v)\}$;
    foreach $q \in$ `constructMaxQueries(`v,t',$\mathbb{W}$`)` **do**
      $Q^{t'} := Q^{t'} \cup \{q_{t'}^v \cup q\}$;
    end
  end
  `/* discard the queries which are not maximal */`
  $incomp := false$;
  while $incomp == false$ **do**
    $incomp := true$;
    foreach $q_1, q_2 \in Q^{t'}$ **do**
      if $q_1 \trianglelefteq_{\mathcal{T}} q_2$ **then**
        $Q^{t'} := Q^{t'} \setminus \{q_1\}$;
        $incomp := false$;
      else
        if $q_2 \trianglelefteq_{\mathcal{T}} q_1$ **then**
          $Q^{t'} := Q^{t'} \setminus \{q_2\}$;
          $incomp := false$;
        end
      end
    end
  end
  foreach $q' \in Q^{t'}$ **do**
    $Q^t = Q^t \cup \{q_t \cup q'\}$;
  end
  $S := S \cup \{Q^t\}$;
  $Q^t = \{\}$;
end
if $S == \{\}$ **then**
  $Q^t = Q^t \cup q_t$;
  $S = S \cup Q^t$;
end
`/* combine all possible maximal queries of the children */`
`combine(`S, Q^t,$\{\}$,0`)` ;
return Q^t ;

Algorithm 5.2: combine

Input: $S = \{Q_1, \ldots, Q_n\}$ set of sets of queries, Q collects the combined queries, q query to be constructed, k the index of current set in S

```
1 if k == n then
2 |  Q := Q ∪ {q} ;
3 else
4 |  foreach query q' ∈ S_k do
5 |  |  q := q ∪ q' ;
6 |  |  combine(S,Q,q,k + 1)
7 |  end
8 end
```

Algorithm 5.3: answersMaxQueries

Input: $\mathbb{W}$ the set of matching witnesses

Output: $\mathbb{H}$ the set of pairs of the form $\langle a, Q^a \rangle$, where $a \in \mathbf{Ind}(\mathcal{A})$ s.t. $w_a^{\mathbf{x}} \in W$ and Q^a the set of $\mathbf{x}$-maximal queries for a

```
1 ℍ := { } ;
2 foreach a s.t. w_a^x ∈ 𝕎 do
3 |  Q^a := constructMaxQueries(a, x, 𝕎) ;
4 |  ℍ := ℍ ∪ {⟨a, Q^a⟩};
5 end
```

5.1.1 Constructing x-maximal Queries

Algorithm 5.3 iterates over each possible answer of the 1treeCQ family and creates for each such individual a all the $\mathbf{x}$-maximal queries. The recursive Algorithm 5.1 does the actual job, since given the match $\mathbf{x} \mapsto a$, it iterates over each child term t and over each matching candidate for t in $w_a^{\mathbf{x}}$ creating recursively all t-maximal queries.

Only the maximal queries are returned, since lines $17 - 30$ make sure to delete the queries formed that are subqueries of some other query. Lastly, since for each child we have constructed all the maximal queries, we need to combine all possibilities in order to obtain all $\mathbf{x}$-maximal queries for a.

Solution Correctness

THEOREM 2 (**t-maximal Queries Correctness**). Let a be such that $w_{\mathbf{x}}^a \in \mathbb{W}$, arbitrarily chosen. Then,

$$\{q \mid q \text{ is } \mathbf{x}\text{-maximal for } a\} = \mathit{constructMaxQueries}(a, \mathbf{x}, \mathbb{W})$$

Proof. "$\supseteq$" Let q be a query constructed by $\mathit{constructMaxQuery}(a, \mathbf{x}, \mathbb{W})$.

We prove by induction on the level of t in (the tree-structure of) q the following claim:

CLAIM 7. For each $t \in \mathbf{term}(q)$ and $v \in values_t$ such that $q \downarrow_t \in constructMaxQuery(v, t, \mathbb{W})$, $q \downarrow_t$ is t-maximal for v.

Proof. **Induction base (IB):** t is a leaf in (the tree-structure of) q. Then $w_v^t = \langle\rangle$. Given the construction of a matching witness, either t is a leaf in (the tree-structure of) $\mathrm{q_U}$, or for each child t' in $\mathrm{q_U}$, $\mathbb{MC}^{t \mapsto v} = \emptyset$.

That means, when t is mapped to v, neither of its children have a match. It follows that $q \downarrow_t \dot{\sqsubseteq}_{\mathcal{T}} \mathrm{q_U} \downarrow_t$ since in this case $q \downarrow_t = \{C(t) \mid C \in labels(t \mapsto v)\}$.

$labels(t \mapsto v)$ gathers all most specialized concept names for v w.r.t. $\mathrm{q_U}$, meaning that if there exists $C' \sqsubseteq_{\mathcal{T}} C$, for some $C \in labels(t \mapsto v)$ it follows that $\mathfrak{I}^{\mathcal{T},\mathcal{A}} \not\models C'(v)$. Hence we obtain that $q \downarrow_t$ is t-maximal for v.

Induction hypothesis (IH): For each t' child of t in (the tree-structure of) q, for each $v' \in values_{t'}$ of w_v^t such that $q \downarrow_{t'} \in constructMaxQueries(t', v', \mathbb{W})$, it holds that $q \downarrow_{t'}$ is t'-maximal for v'.

Induction step (IS): To prove that $q \downarrow_t$ is t-maximal for v.

Since Q^t is the returned set of queries of $constructMaxQueries(v, t, \mathbb{W})$ we get that $q \downarrow_t \in Q^t$. From **IH** it follows that for each t' such that $values_{t'} \neq [\varepsilon]$ in w_v^t there exists $v' \in values_{t'}$ such that $q \downarrow_{t'}$ is t'-maximal for v'. Thus $Q^{t'}$ (in Algorithm 5.1) will contain $q_{t'}^{v'} := \{R(t, t') \mid R \in labels(t \mapsto v, t' \mapsto v')\} \cup q \downarrow_{t'}$ (lines $8 - 10$).

If there exists other queries in $Q^{t'}$, lines $12 - 26$ makes sure that $Q^{t'}$ contains only incomparable such queries.

Since $q \downarrow_{t'}$ is t'-maximal and since $labels(t \mapsto v, t' \mapsto v')$ contains most specialized role names for v and v', we get that $q_{t'}^{v'}$ is not deleted from $Q^{t'}$ in this process and that $Q^{t'}$ will contain only incomparable queries. Therefore,

(1) for each t' such that $values_{t'} \neq [\varepsilon]$ and for each $v'' \neq v'$ we get that

- the constructed $q_{t'}^{v''} \dot{\sqsubseteq}_{\mathcal{T}} q_{t'}^{v'}$, or
- $q_{t'}^{v''}$ and $q_{t'}^{v'}$ are incomparable.

$q_{t'}^{v'}$ together with the other incomparable queries in $Q^{t'}$ are extended by adding atoms $\{C(t) \mid C \in labels(t \mapsto v)\}$ and it is collected firstly by Q^t (a set of queries), and then by S (a set of sets of queries), which collects the set of queries constructed for each t' (lines $27 - 30$). Now, taking into account **(1)** and the fact that associated labels string contain the most specialized concept names satisfied by v, we conclude that

(2) each such Q^t constructed for each t' will contain only incomparable queries.

Lastly, from the method *combine* we get that

$$q \downarrow_t = \{C(t) \mid C \in labels(t \mapsto v)\} \bigcup_{t' s.t. values_{t'} \neq [\varepsilon]} q_{t'}^{v'}$$

Using **(2)** and the construction of the associated labels string (see Definition 13) we obtain that $q \downarrow_t$ is t-maximal for v. □

We can use now Claim 7 for $\mathbf{x}$ and a, since $w_a^{\mathbf{x}} \in \mathbb{W}$, hence we obtain that q is $\mathbf{x}$-maximal for a and $\mathbf{x}$.

"$\subseteq$" Let q be an arbitrarily $\mathbf{x}$-maximal query for a.
We will prove that $q \in constructMaxQueries(a, \mathbf{x}, \mathbb{W})$.

Claim 8. For each $t \in \mathbf{term}(q)$ and for each v such that $q \downarrow_t$ is t-maximal for v, $q \downarrow_t \in constructMaxQueries(v, t, \mathbb{W})$.

We will provide a proof of the above defined claim by induction on the level of t in (the tree-structure of) q.

Proof. **Induction base (IB):** t is a leaf in q. Provided that $q \downarrow_t$ is t-maximal for v the following must hold: for each $C(t) \in q \downarrow_t$, $C \in MS_{concept}(v, \mathcal{K})$. Therefore, $C \in labels(t \mapsto v)$.

Given the construction of the matching witnesses and the fact that t is a leaf we get that $values_t = [\varepsilon]$. Lines $33 - 36$ in $constructMaxQueries(v, t, \mathbb{W})$ make sure that $q \downarrow_t \in constructMaxQueries(v, t, \mathbb{W})$ since in this case the method *combine* will return only q_t.

Induction hypothesis (IH): For each t' child of t in (the tree-structure of) q and for each v' such that $q \downarrow_{t'}$ is t'-maximal for v', it holds that

$$q \downarrow_{t'} \in constructMaxQueries(v', t', \mathbb{W})$$

Induction step (IS): To prove that $q \downarrow_t \in constructMaxQueries(v, t, \mathbb{W})$.

From **IH** for each t' child of t in (the tree-structure of) q and v' such that $q \downarrow_{t'}$ is t'-maximal for v', implies that v' satisfies $q \downarrow_{t'}$ w.r.t. $\mathbb{W}$ and since v satisfies $q \downarrow_t$ we get that $v' \in values_{t'} \in w_v^t$. From lines $4 - 32$ in $constructMaxQueries(v, t, \mathbb{W})$ and **IH** we get that

$$q_{t'}^{v'} := \{R(t, t') \mid R \in labels(t \mapsto v, t' \mapsto v')\} \cup q \downarrow_{t'}$$

Given that $q \downarrow_t$ contains $q \downarrow_{t'}$ and $q \downarrow_t$ is t-maximal for v, we can conclude that $q_{t'}^{v'}$ will not be deleted in the incomparable loop (lines $12 - 26$). Now using again the fact that $q \downarrow_t$ is t-maximal for v and $q \downarrow_{t'}$ is t'-maximal for v', we can conclude that for each $R(t, t') \in q$, $R \in MS_{role}(v, v', \mathcal{K})$, respectively for each $C(t) \in q$, $C \in MS_{concept}(v, \mathcal{K})$, therefore we obtain that each $q \downarrow_t|_{\{t,t'\}} \cup q \downarrow_{t'} \in Q^t$ in algorithm $constructMaxQueries(v, t, \mathbb{W})$. Lastly since S gathers, for each t', $\{C(t) \mid C \in labels(t \mapsto v)\} \cup q_{t'}^{v'}$, the method *combine* will create all possible combinations. Since

$$q \downarrow_t = \{C(t) \mid C \in labels(t \mapsto v)\} \bigcup_{t'} q_{t'}^{v'}$$

we can conclude that $q \downarrow_t \in constructMaxQueries(v, t, \mathbb{W})$. □

Applying now Claim 8 for $a, \mathbf{x}$ we obtain that $q \in constructMaxQueries(a, \mathbf{x}, \mathbb{W})$.
□

Complexity Analysis for x-maximal Queries Construction

In general for some individual a, over a *DL-Lite*$_{\mathcal{R}}$ KB there can be exponentially many such **x**-maximal queries since there can be exponentially many incomparable matches for a in the canonical model. The set of matching witnesses stores all these matches, while Algorithm 5.3 explores all these matches for constructing the maximal queries.

Moreover, the method combine (see Algorithm 5.2) for level i in $\mathrm{q_U}$ computes the n-fold Cartesian product over the maximal queries sets of the children on level $i+1$.

Therefore, the Algorithm 5.1 which constructs all **x**-maximal queries for some individual runs in EXPTIME.

5.2 Query Modifications

In this section we rely on the **x**-maximal queries constructed above, to compute modified versions of a query in a 1treeCQ family that can allow a user to flexibly explore the query answers.

Below we use a notion of intersection between queries that also takes into account the concept and role hierarchy, similarly as the subquery relation in Definition 9.

Definition 20 (**Intersection**). Given two 1treeCQs q_1, q_2 we define the **intersection of q_1 and q_2** w.r.t. $\mathcal{T}$ as follows:

$$\begin{aligned} q_1 \cap_{\mathcal{T}} q_2 = \{C(t) \mid\ & C(t) \in q_1 \text{ or } C(t) \in q_2, \text{ s.t. there exists } C_1(t) \in q_1, C_2(t) \in q_2 \text{ and} \\ & C_1 \sqsubseteq_{\mathcal{T}} C, C_2 \sqsubseteq_{\mathcal{T}} C \text{ hold } \} \cup \\ \{R(t,t') \mid\ & R(t,t') \in q_1 \text{ or } R(t,t') \in q_2, \text{ s.t. there exists } R_1(t,t') \in q_1, R_2(t,t') \in q_2 \text{ and} \\ & R_1 \sqsubseteq_{\mathcal{T}} R, R_2 \sqsubseteq_{\mathcal{T}} R \text{ hold } \} \end{aligned}$$

Intuitively, the intersection of two queries keeps those query atoms that are implied by some atom in each query.

Example 4. Let's consider the following two 1treeCQs created using the TBox $\mathcal{T}$ from Example 3.1.

$$q_1(x) : -\mathrm{Professor}(x), \mathrm{teacherOf}(x,y) \quad q_2(x) : -\mathrm{AssociatedProfessor}(x), \mathrm{worksFor}(x,y)$$

Now, suppose that AssociatedProfessor $\sqsubseteq_{\mathcal{T}}$ Professor holds for $\mathcal{T}$.
Then, $q_1 \cap_{\mathcal{T}} q_2 = \{\mathrm{Professor}(x)\}$

Additionally to this special notion of intersection, we use below the union and difference of queries, but these are defined as the standard set theoretic operations on their atoms.

Definition 21 (**Union, Difference**). Given two 1treeCQs q_1, q_2 we define the **union of q_1 and q_2** w.r.t. $\mathcal{T}$ as follows:

$$q_1 \cup_{\mathcal{T}} q_2 = \{C(t) \mid C(t) \in q_1(\text{ or } \in q_2)\} \cup \{R(t,t') \mid R(t,t') \in q_1(\text{ or } \in q_2)\}$$

Given two 1treeCQs q_1, q_2 we define the **difference between** q_1 **and** q_2 w.r.t. $\mathcal{T}$ as follows: $q_1 \setminus_{\mathcal{T}} q_2 = \{C(t) \mid C(t) \in q_1 \text{ and } C(t) \notin q_2\} \cup \{R(t,t') \mid R(t,t') \in q_1 \text{ and } R(t,t') \notin q_2\}$

Specializations and Generalizations

We use the term *query specialization* to refer to any superquery of a given q in a 1treeCQ family. Indeed, superqueries 'specialize' the query by requiring additional or more specific properties to hold. We further distinguish between two kinds of specializations: the ones that retrieve the same answers as the given q, and the ones that retrieve strictly less answers.

Definition 22 (**Neutral Specialization**). Let q_1 be a 1treeCQs. Any q_2 such that $q_1 \trianglelefteq_{\mathcal{T}} q_2$ and $\mathbf{ans}(q_2, \mathbb{W}) = \mathbf{ans}(q_1, \mathbb{W})$ is said to be a **neutral specialization** of q_1.

Remark 5. Trivially, any in-between 1treeCQ is a neutral specialization of itself.

Definition 23 (**Strict Specialization**). Let q_1 be a 1treeCQs. Any q_2 such that $q_1 \trianglelefteq_{\mathcal{T}} q_2$ and $\mathbf{ans}(q_2, \mathbb{W}) \subsetneq \mathbf{ans}(q_1, \mathbb{W})$, with $\mathbf{ans}(q_2, \mathbb{W}) \neq \emptyset$ is said to be a **strict specialization** of q_1.

Conversely, any subquery of a given 1treeCQ q is less constrained than q, and it can retrieve the same set of answers as q or strictly more answers. We are interested in those subqueries that retrieve strictly more answers.

Definition 24 (**Generalization**). Let q_1 be a 1treeCQs. Any q_2 such that $q_2 \trianglelefteq_{\mathcal{T}} q_1$ and $\mathbf{ans}(q_1, \mathbb{W}) \subsetneq \mathbf{ans}(q_2, \mathbb{W})$ is a **generalization** of q_1.

5.2.1 Maximal Neutral Specializations

The first query modification problem we consider is to construct the maximal neutral specializations of a query. Intuitively, the maximal neutral specialization includes all the additional constraints that we can add to the query without modifying its answers, and hence it makes explicit all the properties that are common to all the individuals in the query answer. It can be seen as a way to provide the user with the most complete information about the query answers.

Definition 25 (**Maximal Neutral Specialization**). Let q be an in-between 1treeCQ. A **maximal neutral specialization** for q w.r.t. $\mathbf{ans}(q, \mathbb{W})$ is a neutral specialization q' and for each superquery q'' such that $q' \trianglelefteq_{\mathcal{T}} q''$ we have $\mathbf{ans}(q'', \mathbb{W}) \neq \mathbf{ans}(q, \mathbb{W})$.

Algorithm 5.4 constructs a set of neutral specializations for a given in-between query, by computing the intersection of all possible combinations of the superqueries of the given 1treeCQ that are relevant in preserving the same set of answers. Not all the constructed specializations are maximal but, importantly, all the maximal ones are constructed.

Algorithm 5.4: neutralSpecialization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query
Output: Q the set of neutral specializations of q

```
1  S := {};
2  ans(q, 𝕎) := getAnswers(q,𝕎) ;
3  foreach a ∈ ans(q, 𝕎) do
4  |  Q^a := constructMaxQueries(a, x, 𝕎) ;
5  |  S := S ∪ {Q^a};
6  end
   /* the set of neutral specializations of q                    */
7  Q := ∅;
   /* compute intersection of each set in the Cartesian
      product of the sets in S                                   */
8  intersect(S,Q,q,∅,0);
9  return Q;
```

Algorithm 5.5: intersect

Input: $S = \{Q_1, \ldots, Q_n\}$ set of sets of queries, Q collects the neutral specializations, q in-between query, q_{inter} intersection query, k the index of current set in S

```
1  if k == n then
2  |  Q := Q ∪ {q_inter} ;
3  else
4  |  foreach query q' ∈ Q_k do
5  |  |  if q ⊑_T q' then
6  |  |  |  if q_inter == ∅ then
7  |  |  |  |  q_inter := q';
8  |  |  |  else
9  |  |  |  |  q_inter := q_inter ∩_T q' ;
10 |  |  |  end
11 |  |  |  intersect(S,Q,q,q_inter,k + 1)
12 |  |  end
13 |  end
14 end
```

Algorithm 5.6: maxNeutralSpecialization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query
Output: Q the set of maximal neutral specializations of q

```
1 Q := neutralSpecialization(q,𝕎);
2 foreach q1 ∈ Q do
3 |   foreach q2 ∈ Q do
4 |   |   if q1 ≠ q2 and q1 ⊑_T q2 then
5 |   |   |   Q := Q \ {q1};
6 |   |   end
7 |   end
8 end
9 return Q;
```

To obtain the maximal neutral specializations, it is then enough to eliminate the ones that are not maximal, as done in Algorithm 5.6.

We now prove that the algorithm is correct.

THEOREM 3. For any in-between query q the following holds:

$$\{q' \mid q' \text{ maximal neutral specialization of } q\} = maxNeutralSpecialization(q, \mathbb{W})$$

Proof. In order to prove theorem 3 we need to prove the following propositions:

(a)

$$\{q' \mid q' \text{ neutral specialization of } q\} \supseteq neutralSpecialization(q, \mathbb{W})$$

(b)

$$\{q' \mid q' \text{ maximal neutral specialization of } q\} \subseteq neutralSpecialization(q, \mathbb{W})$$

(c)

$$\{q' \mid q' \text{ maximal neutral specialization of } q\} = maxNeutralSpecialization(q, \mathbb{W})$$

"**(a)**" Let $q' \in neutralSpecialization(q, \mathbb{W})$ arbitrarily chosen. **To prove that:** q' is a neutral specialization of q. Let $\mathbf{ans}(q, \mathbb{W}) = \{a_1, \dots a_n\}$. $q' = \bigcap_{\mathcal{T}} q_{a_i}$, where $q \sqsubseteq_{\mathcal{T}} q_{a_i}$ and q_{a_i} is $\mathbf{x}$-maximal for a_i, $1 \leq i \leq n$.

CLAIM 9. For each q_1, q_2 such that $q \sqsubseteq_{\mathcal{T}} q_1$ and $q \sqsubseteq_{\mathcal{T}} q_2$ it holds that $q \sqsubseteq_{\mathcal{T}} q_1 \cap_{\mathcal{T}} q_2$.

Proof. From Definition 20 we get that $q_1 \cap_{\mathcal{T}} q_2$ contains only concept atoms such as $C(t) \in (q_1 \text{ or } q_2)$ and there exists $C_1(t) \in q_1$ and $C_2(t) \in q_2$ such that $C_1 \sqsubseteq_{\mathcal{T}} C$ and $C_2 \sqsubseteq_{\mathcal{T}} C$.

Now from the subquery definition we get that for each $C(t) \in q$ there must exist $C_1(t) \in q_1$ and $C_2(t) \in q_2$ such that $C_1 \sqsubseteq_{\mathcal{T}} C$ and $C_2 \sqsubseteq_{\mathcal{T}} C$. Hence, clearly the concept

axiom of the subquery definition holds in case of $q_1 \cap_{\mathcal{T}} q_2$. Using the same reasoning for role atoms we conclude that indeed $q \trianglelefteq_{\mathcal{T}} q_1 \cap_{\mathcal{T}} q_2$.

□

Using Claim 9 in case of $q' \in Q$ we obtain that $q \trianglelefteq_{\mathcal{T}} q'$.

To prove that ans$(q, \mathbb{W}) \subseteq$ **ans**$(q', \mathbb{W})$. Let $a_i \in ans(q, \mathbb{W})$ arbitrarily chosen. Thus there is q_{a_i} such that q_{a_i} is **x**-maximal for a_i, and $q \trianglelefteq_{\mathcal{T}} q_{a_i}$. Evidently $a_i \in$ **ans**(q_{a_i}). Since any other q_{a_j}, $j \neq i$, $1 \leq j \leq n$ is a superquery of q, we get that $a_i \in$ **ans**$(q_{a_j}, \mathbb{W})$. Now given the definition of the query intersection, we get that $a_i \in$ **ans**$(q_{a_i} \cap_{\mathcal{T}} q_{a_j})$, $1 \leq j \leq n$, $j \neq i$. Hence $a_i \in$ **ans**$(q', \mathbb{W})$ and since a_i was arbitrarily chosen we can conclude that:

(i) **ans**$(q, \mathbb{W}) \subseteq$ **ans**$(q', \mathbb{W})$.

(ii) Trivially, we obtain the other direction **ans**$(q', \mathbb{W}) \subseteq$ **ans**$(q, \mathbb{W})$ since $q \trianglelefteq_{\mathcal{T}} q'$.

Thus using **(i)** and **(ii)** we get that **ans**$(q', \mathbb{W}) =$ **ans**$(q, \mathbb{W})$, therefore q' is a neutral specialization of q.

q' is arbitrarily chosen, so we can now conclude that **(a)** holds indeed.

"**(b)**" Let q' be an arbitrarily chosen maximal specialization of q and let **ans**$(q, \mathbb{W}) = \{a_1, \ldots, a_n\}$.

Assume that $q' \notin neutralSpecialization(q, \mathbb{W})$. Let $\{a_1, \ldots a_n\} =$ **ans**$(q, \mathbb{W})$. Given that $neutralSpecialization(q, \mathbb{W})$ computes a set of queries such that:

$$Q = \{\bigcap_{\mathcal{T}} q_{a_i} \mid q \trianglelefteq_{\mathcal{T}} q_{a_i} \text{ and is } \mathbf{x}\text{-maximal for } a, 1 \leq i \leq n\}$$

Using **(a)** we get that any $q'' \in Q$ is a neutral specialization of q.

Particularly we are interested in queries $q'' \in Q$ that fulfill such condition for each $q* \in Q$, $q* \neq q''$ either:

(i) $q* \trianglelefteq_{\mathcal{T}} q''$ or

(ii) $q* \not\trianglelefteq_{\mathcal{T}} q''$ and $q'' \not\trianglelefteq_{\mathcal{T}} q*$ holds.

Let $q'' \in Q$ be such query. Therefore there exists $\{q_{a_1}, \ldots q_{a_n}\}$, where $q \trianglelefteq_{\mathcal{T}} q_{a_i}$ and q_{a_i} is **x**-maximal for a_i and **x**, $1 \leq i \leq n$, such that $q'' = \bigcap_{\mathcal{T}} q_{a_i}$.

Therefore, for each $q*$ such that $q_{a_i} \trianglelefteq_{\mathcal{T}} q*$ it holds that $a_i \notin$ **ans**$(q*, \mathbb{W})$.

Now, making use of the "$\cap_{\mathcal{T}}$" definition and a small abuse of the **x**-maximality definition, we can observe that:

(1) q'' is **x**-maximal for $\{a_1, \ldots, a_n\}$, meaning that for each $q*$ such that $\bigcap_{\mathcal{T}} q_{a_i} \trianglelefteq_{\mathcal{T}} q*$, it holds that $\{a_1, \ldots a_n\} \not\subseteq$ **ans**$(q*, \mathbb{W})$.

We distinguish now two cases for q':

(i) there exists such $q'' \in Q$ such that $q'' \trianglelefteq_{\mathcal{T}} q'$. But using relation **(1)** and the fact that q'' is a neutral specialization we get a contradiction with our initial assumption that q' is a maximal neutral specialization ↯

(ii) for each such $q'' \in Q$, q' and q'' are incomparable. This case cannot hold since using **(1)** and the fact that Algorithm $constructMaxQueries(q, \mathbb{W})$ is correct in retrieving all **x**-maximal queries for each a_i, $1 \leq i \leq n$, it must be the case that there exists such q'' for which $q' \trianglelefteq_{\mathcal{T}} q''$ holds.

Following cases **(i)** and **(ii)** we conclude that there exists $q'' \in Q$ such that $q' \trianglelefteq_{\mathcal{T}} q''$ and $q'' \trianglelefteq_{\mathcal{T}} q'$. Therefore assumption is false, $q' \in neutralSpecialization(q, \mathbb{W})$.

Moreover, since q' is arbitrarily chosen, we obtain that **(b)** holds.

"**(c)**" Algorithm 5.6 discards, from the neutral specializations, the ones that are subqueries of another one, obtaining only the maximal ones. Therefore, now using **(a)** and **(b)** we get that indeed algorithm $maxNeutralSpecialization(q, \mathbb{W})$ constructs all the maximal neutral specializations of q, hence proposition **(c)** holds. □

5.2.2 Minimal Strict Specialization

The second query modification problem we consider is to find the minimal strict specializations of a given query, that is, which are the minimal modifications to the query that will result in a smaller set of answers. In this section we provide an algorithm to construct all such minimal strict specializations.

Definition 26 (**Minimal Strict Specialization**). Let q be an in-between 1treeCQ. A **minimal strict specialization** for q is a strict specialization q' that satisfies the condition that for each q'' such that $q \trianglelefteq_{\mathcal{T}} q'' \trianglelefteq_{\mathcal{T}} q'$ and q'' is a neutral specialization of q.

Remark 6. It can be the case that q cannot be strictly specialized without loosing all answers. In this case we say that q's only strict modification is q_{U}.

The idea to retrieve all minimal strict specializations of a query, in case if there exists at least one, is to use once again the set of **x**-maximal queries for each $a \in \mathbf{ans}_{\mathbb{W}}(q, \mathcal{K})$.

Algorithm 5.7 computes strict specializations of an in-between 1treeCQ q by iteration over its answers and for each individual a, keeping in mind that each of the queries retrieved by algorithm $constructMaxQueries(a, q, \mathbb{W})$ are incomparable, we collect them as possible modifications of q.

Given the definition of minimal strict specialization, we make use of the maximal neutral specializations by identifying all the **x**-maximal queries that are proper superqueries of some maximal neutral specialization. By adding to the maximal neutral specialization any atom in the difference, we make sure that we will loose some answer while each such obtained query will have at least one answer.

Theorem 4. For any in-between 1treeCQ q the following holds:

Algorithm 5.7: strictSpecialization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query
Output: Q set of strict specializations of q

```
1  Q := ∅ ;
2  Q_max := ∅;
3  ans(q, 𝕎) := getAnswers(q,𝕎) ;
4  foreach a ∈ ans(q, 𝕎) do
5  |  Q_max := Q_max ∪ constructMaxQueries(a, x, 𝕎) ;
6  end
7  Q_neutral := maxNeutralSpecialization(q,𝕎) ;
8  foreach q1 ∈ Q_max do
9  |  foreach q2 ∈ Q_neutral do
10 |  |  if q2 ⊑_T q1 then
11 |  |  |  q_diff := q1 \_T q2;
12 |  |  |  foreach query atom A ∈ q_diff do
13 |  |  |  |  if q2 ∪ {A} is 1treeCQ then
14 |  |  |  |  |  Q := Q ∪ {q2 ∪ {A}} ;
15 |  |  |  |  end
16 |  |  |  end
17 |  |  end
18 |  end
19 end
20 return Q;
```

Algorithm 5.8: minStrictSpecialization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query, q_U upperbound query
Output: Q set of minimal strict specializations of q

```
1  Q := strictSpecialization(q,𝕎);
2  foreach q1 ∈ Q do
3  |  foreach q2 ∈ Q do
4  |  |  if q1 ≠ q2 and q2 ⊑_T q1 then
5  |  |  |  Q := Q \ {q1};
6  |  |  end
7  |  end
8  end
9  if Q == ∅ then
10 |  Q := Q ∪ {q_U} ;
11 end
12 return Q;
```

(a) $\{q' \mid q' \text{ is a strict specialization of } q\} \supseteq strictSpecialization(q, \mathbb{W})$

(b) $\{q' \mid q' \text{ is a minimal strict specialization of } q\} \subseteq minStrictSpecialization(q, \mathbb{W})$

Proof. "**(a)**" Since Algorithm 5.7 returns a set of queries that are obtained by adding atoms to some neutral specializations, clearly each such obtained query will loose some of the answers that the input in-between query has.

Therefore, **(a)** clearly holds.

"**(b)**" Let q be an in-between query arbitrarily chosen.

Case I. q does not have strict specializations without loosing all answers. Therefore, let $q' = \mathrm{q_U}$ be the only minimal strict specialization for q. Clearly, each of the **x**-maximal queries for $a \in \mathbf{ans}(q, \mathbb{W})$ is a maximal neutral specialization for q. Given that each of the queries returned by algorithm $maxNeutralSpecialization(q, \mathbb{W})$ is incomparable with the others, we get that lines $12 - 16$ in algorithm $strictSpecialization(q, \mathbb{W})$ will not compute since q_{diff} would be empty. Therefore, lines $9 - 10$ in algorithm $minStrictSpecialization(q, \mathbb{W})$ will apply in this case, hence $\mathrm{q_U}$ is returned.

Case II. q has strict specializations that retrieve answers. Let q' be a minimal strict specialization of q, arbitrarily chosen.

To prove that $q' \in minStrictSpecialization$. Since q' is a strict specialization, we get that there exists some $a \in \mathbf{ans}(q, \mathbb{W})$ such that $a \notin \mathbf{ans}(q', \mathbb{W})$.

Given that, for each q'' such that $q \trianglelefteq_{\mathcal{T}} q'' \trianglelefteq_{\mathcal{T}} q'$, q'' is a neutral specialization of q or $q'' = q$. Moreover, it is obvious that q' must contain exactly one extra atom compared to q'', otherwise q' is not minimal strict specialization or q'' is not maximal neutral specialization.

Let A be the query atom such that $A \notin q''$ and $A \in q'$. Since $q' \neq \mathrm{q_U}$ then $\mathbf{ans}(q', \mathbb{W}) \neq \emptyset$.

In each q_{max} **x**-maximal query for $a' \in \mathbf{ans}(q', \mathbb{W})$, there exists query atom $A' \in q_{max}$ such that $A' \sqsubseteq_{\mathcal{T}} A$, A' and A are defined on the same terms. But then q' is no longer minimal strict specialization of q, therefore we must conclude that $A' = A$. Hence, $q_{max} \in Q_{max}$ (Algorithm 5.7 lines $4 - 6$). Following lines $8 - 19$ for q_{max} and q'' we obtain that $A \in q_{diff}$. Therefore, from lines $12 - 16$ we get that q' will be returned by $strictSpecialization(q, \mathbb{W})$.

Since q' is minimal, it follows that lines $4 - 6$ in $minStrictSpecialization(q, \mathbb{W})$ do not apply in this case, therefore $q' \in minStrictSpecialization(q, \mathbb{W})$.

q' was arbitrarily chosen, hence we get that **(b)** holds.

□

5.2.3 Minimal Generalization

Finally, we consider the problem of finding the minimal generalizations of given query. That is, we want to relax the query in a minimal way to retrieve more answers.

Definition 27 (**Minimal Generalization**). Let q be an in-between 1treeCQ. A **minimal generalization or relaxation** for q is a generalization q' such that for each q'',

that is $q' \trianglelefteq_{\mathcal{T}} q'' \trianglelefteq_{\mathcal{T}} q$, it holds that:

$$\mathbf{ans}(q'', \mathbb{W}) = \mathbf{ans}(q, \mathbb{W})$$

Remark 7. It might be the case that an in-between 1treeCQ q cannot be generalized if $\mathbf{ans}(q, \mathbb{W}) = \mathbf{ans}(\mathrm{q_L}, \mathbb{W})$. Hence, whenever q is a neutral specialization of $\mathrm{q_L}$ it doesn't have any generalizations.

Algorithm 5.9: minGeneralization

Input: $\mathbb{W}$ the set of matching witnesses, q in-between query
Output: Q the set of minimal generalizations for q

```
1  Q := ∅;
2  ans(q, W) := getAnswers(q,W) ;
3  foreach a ∈ ans(q_L, W) and a ∉ ans(q, W) do
4  |  Q^a := constructMaxQueries(a, x, W) ;
5  |  foreach q' ∈ Q^a do
6  |  |  q_inter := ∅ ;
7  |  |  if q' ⊴_T q then
8  |  |  |  Q := Q ∪ q' ;
9  |  |  else
10 |  |  |  q_inter := q ∩_T q' ;
11 |  |  |  if q_inter ≠ ∅ then
12 |  |  |  |  Q := Q ∪ q_inter;
13 |  |  |  end
14 |  |  end
15 |  end
16 end
17 return Q;
```

Algorithm 5.9 computes generalizations of a given in-between 1treeCQ q by iterating over each non-answer individual's $\mathbf{x}$-maximal queries and checking whether there exists already a generalization, or computing the intersection with q to provide a possible generalization for q.

Theorem 5. For any in-between 1treeCQ q the following holds:

$$\{q' \mid q' \text{ minimal generalization of } q\} \subseteq \mathit{minGeneralization}(q, \mathbb{W})$$

Proof. Let q' be a minimal generalization of q, arbitrarily chosen.

To prove: $q' \in \mathit{minGeneralization}(q, \mathbb{W})$.

Assume that $q' \notin \mathit{minGeneralization}(q, \mathbb{W})$. It is easy to see that each query $q'' \in \mathit{minGeneralization}(q, \mathbb{W})$ represents a generalization of q. We are interested in identifying the minimal generalizations among those.

Since q' is a minimal generalization of q then there must exists some $a \in \mathbf{ans}(\mathrm{q_L}, \mathbb{W})$ such that $a \in \mathbf{ans}(q', \mathbb{W})$ and $a \notin \mathbf{ans}(q, \mathbb{W})$. Hence, lines 3-14 in Algorithm 5.9 apply for a. Then, due to the fact that Algorithm *constructMaxQueries* is correct and a is an answer for q', there must exist q'' that either:

- q'' is $\mathbf{x}$-maximal for a and $\mathbf{x}$ and $q'' \trianglelefteq_{\mathcal{T}} q$. This imply that for each $q*$ such that $q'' \trianglelefteq_{\mathcal{T}} q*$, a does not satisfy $q*$ w.r.t. $\mathbb{W}$. Hence, contradiction with the fact that $a \in \mathbf{ans}(q', \mathbb{W})$ and q' is minimal generalization of q ↯
- $q'' = q \cap_{\mathcal{T}} q_{max}$, where q_{max} is $\mathbf{x}$-maximal for a and $\mathbf{x}$. It is easy to see that "$\cap_{/T}$" preserves maximality since it takes only those role atoms and concept atoms that subsume something in both queries. Therefore, for each $q*$ such that $q'' \trianglelefteq_{\mathcal{T}} q* \trianglelefteq_{\mathcal{T}} q$ we get that $a \notin \mathbf{ans}(q*, \mathbb{W})$ which leads again to contradiction considering the facts that $a \in ans(q', \mathbb{W})$ and q' is minimal generalization of q that is not retrieved by algorithm $minGeneralization(q, \mathbb{W})$ ↯

Following the reasoning by cases, we obtain that our assumption is wrong so in fact $q' \in minGeneralization(q, \mathbb{W})$.

Since q' was arbitrarily chosen, we obtain that Theorem 5 holds for any in-between 1treeCQ q.

□

CHAPTER 6

Implementation and Experiments

The main purpose of this chapter is to get an idea of how our solution behaves in practice. Therefore, we implemented the algorithms presented in Chapter 4 and in Chapter 5 and ran some experiments. The obtained insights are presented in the following sections.

6.1 Implementation Details

The matching witness solution is implemented in Java (jdk1.8), using additional libraries. For loading the ontology we used the OWLAPI java library[1], a very useful library that is able to parse a KB in different formats and which has a reasoner interface that is implemented by many existing OBDA systems. For ABox reasoning we used the Ontop [RKZ13] since it provides complete answers under OWL 2 QL profile and it uses a datalog engine for retrieving the answers. However, Ontop was specially tailored for OBDA tasks and it does not provide full support when the data is not stored using relational databases. Therefore, we used HermiT reasoner[2] for TBox reasoning. HermiT is a more complex system that captures the entire OWL 2 profile but it is not optimal for OWL 2 QL, especially in case of ABox reasoning tasks. To boost the performance we used multi-threading for computing the matching witnesses as well as for query answering. However, we intend to modify our implementation in the future as we believe that some optimization can be made in order to obtain better performance.

Under this implementation, the matching witnesses, using the data sets presented in Table 6.1, took several minutes to compute for small and medium-sized data sets, on each 1treeCQs batch, while for the large-sized data sets the computation lasted several hours. One of the causes might be how Ontop evaluates 1treeCQs with more than 1 level, which are relatively simple. It computes in 2 seconds (for the smallest data set) up to 1 minute (for the largest data set) over the data sets presented in Table 6.1.

[1] http://owlapi.sourceforge.net/
[2] http://www.hermit-reasoner.com/

	#D, #U	Size (MB)
DS1	6D, 1U	3.3
DS2	11D, 1U	6
DS3	15D, 1U	8.05
DS4	21D, 2U	11.3
DS5	31D, 2U	16.8
DS6	34D, 2U	18.5
DS7	35D, 3U	18.9

Table 6.1: Datasets - Departments(D) distributed over Universities(U)

We also observed that the performance of query answering procedure, the construction of **x**-maximal queries and the algorithm for each query modification, are influenced by the performance of HermiT reasoner, which is not recommended when using multi-threading. We needed to limit the number of active threads in order to avoid an endless loop that is caused when accessing a specific method in the OWL API library (both reasoners use OWL API to store and access the ontology and the assertions). Therefore the implementation could not take advantage of the High Performance Computing (HPC) clusters that could have dramatically speed up reasoning for big ontologies.

However, once the compilation is created, we do store it on the physical memory and retrieving it back is done very fast.

In the following section we provide the details of the evaluation, with an emphasize on the interesting obtained query modifications and we analyze the dependence relations of the performance for the following algorithms: getAnswers (Algorithm 4.5), answersMaxQueries (Algorithm 5.1), maxNeutralSpecialization (Algorithm 5.6), minStrictSpecialization (Algorithm 5.8) and minGeneralization (Algorithm 5.9).

6.2 Experiments

Specifications. The entire evaluation was done on a server using 7 CPUs (each core running at 2.3GHz), 10 GB RAM, operating system Ubuntu Server amd64, java version Java SE Runtime Environment 7.

Ontology. As ontology we used LUBM [GPH05] that is a well known ontology suitable for testing, especially since it provides a data generator, and its domain is very easy to comprehend and relatively simple. To match our DL fragment, we used a modified version, over the OWL 2 QL profile, that was used in [KRRM$^+$14].

However, even when using different sizes ABoxes, see Table 6.1, the number of anonymous objects that can be constructed is quite restricted. We encountered difficulties on formulating 1treeCQs with depth larger than 3. Another issue we found is that concept

	#MWs	#PA
DS1	4915	435
DS2	8937	796
DS3	12180	1085
DS4	17109	1507
DS5	25328	2237
DS6	27977	2464

Table 6.2: Datasets with #Matching witnesses(MWs) and #Possible answers (PA) for Batch1

and role hierarchies are relatively small and concepts do not overlap on same relations, allowing only simple queries that can be formulated in order to make actual sense. Due to this aspect, we formulate upper-bound queries that do not have answers but provide possibilities for constructing large number of in-between queries. Nonetheless, the constructed in-between queries make sense with respect to the domain of the ontology and they retrieve answers.

Datasets. We used the LUBM generator[3] to compute the data sets. The concrete information about each used ABox is presented in Table 6.1.

Evaluation measurements. We evaluated 3 batches of 1treeCQs families that range over different concepts of the ontology. We measured the number of possible answers for the 1treeCQs family, the size of the matching witnesses set, the average time for answering an in-between query, average time to construct $\mathbf{x}$-maximal queries for a possible answer and for each query modification we measured the average computational time for an in-between query.

6.2.1 Batch1

For this batch, the lower bound query is quite general:

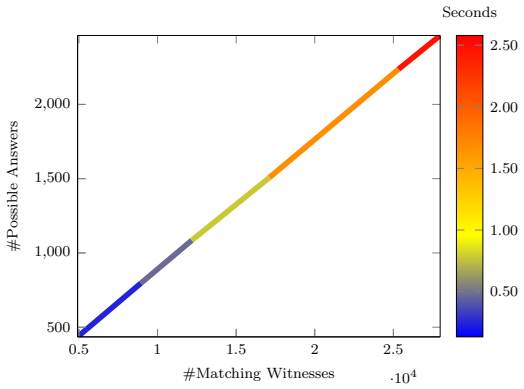
$$\mathrm{q_L}(\mathbf{x}) : -\mathrm{Employee}(\mathbf{x})$$

and, the upper bound is as follows:

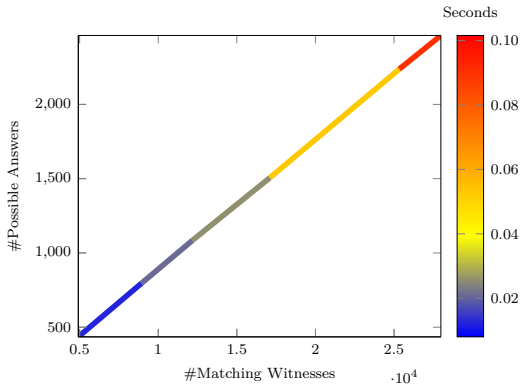
$$\mathrm{q_U}(\mathbf{x}) : -\mathrm{Dean}(\mathbf{x}), \mathrm{Professor}(\mathbf{x}), \mathrm{FullProfessor}(\mathbf{x}), \mathrm{teacherOf}(\mathbf{x}, y_1), \mathrm{Course}(y_1),$$
$$\mathrm{headOf}(\mathbf{x}, y_2), \mathrm{Department}(y_2), \mathrm{authorPublication}(\mathbf{x}, y_3), \mathrm{Publication}(y_3)$$

For this batch were generated 259 in-between queries of which 64 of the in-between queries have answers over the DS1-DS6 datasets, presented in Table 6.1.

[3] http://swat.cse.lehigh.edu/projects/lubm/



(a) Av. time for QA over Batch1



(b) Av. time for computing **x**-Maximal Queries over Batch1

Figure 6.1: Performance of QA and **x**-maximal query construction relative to the number of possible answers and matching witnesses

Table 6.2 shows the size of the matching witnesses set computed for the corresponding number of possible answers. Particularly, we measure the size of $\mathbb{W}$ after the execution of Algorithm 4.1.

Subfigure 6.1a shows that query answering for Batch1 is efficient for data sets that have at most $\simeq$ 1200 possible answers that produce $\simeq 1.4 \cdot 10^4$ matching witnesses. For data sets which have more than 1500 possible answers, and which produce more than $1.75 \cdot 10^4$ matching witnesses, algorithm getAnswers performs beyond the 1 second threshold.

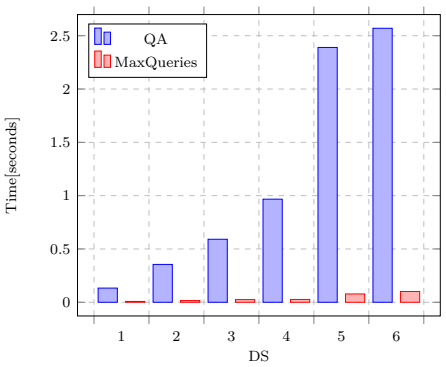
Analyzing Subfigure 6.1b we observe that for each data set, in spite of the relative large number of possible answers and the number of produced matching witnesses, the answersMaxQueries algorithm is very efficient, running for all ABoxes in less than 0.1 seconds. Thus we can infer that, for at least these particular KBs, the algorithm's performance does not dependent on the size of the possible answers set nor on the number of constructed matching witnesses.

The uniformity of the generated data is observed in Figure 6.1, since considering Table 6.2, the ratio #PA/#MWs is approximately the same for each data set.

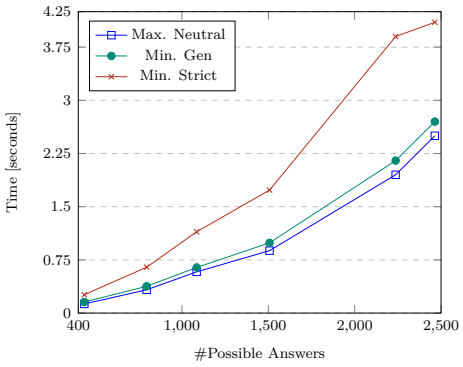
A look at the performance of query answering and **x**-maximal queries construction, for a specific data set, is presented in Subfigure 6.2a.

The query modifications performance for this batch is illustrated in Subfigure 6.2b. The fastest to compute for this batch are the maximal neutral specializations, whereas the minimal generalizations are slightly more costly. The most expensive to compute are the minimal strict specializations since the algorithm minStrictSpecialization explicitly computes the maximal neutral specializations for the input query.

We now illustrate a few of the concrete obtained modifications for this batch of 1treeCQs.

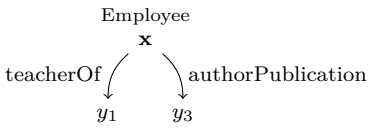


(a) QA and Max. queries performance per Data set for Batch1

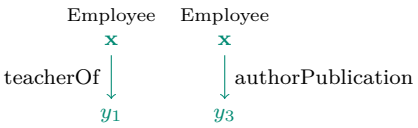


(b) Performance of query modifications for Batch1

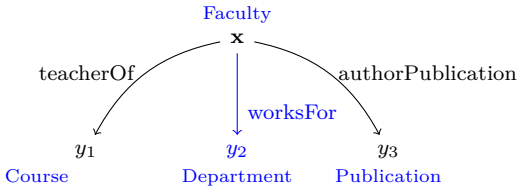
Figure 6.2: Analysis of QA and construction of **x**-max. queries performance over the data sets, and the possible answers dependence of query modifications for Batch 1



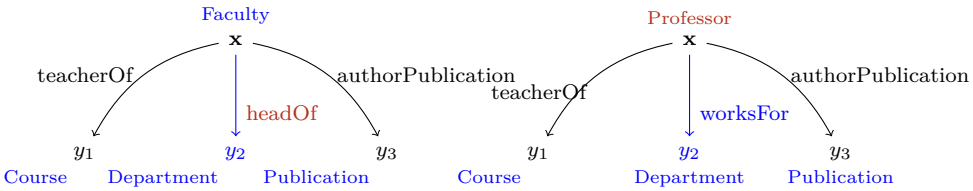
(a) In-between query



(b) Minimal Generalizations



(c) Maximal Neutral Specialization



(d) Minimal Strict Specializations

Figure 6.3: Example of obtained modifications for in-between query for Batch 1

From the ontology we know that:

$$\text{headOf} \sqsubseteq_{\mathcal{T}} \text{worksFor} \sqsubseteq_{\mathcal{T}} \text{memberOf} \text{ and } \text{Professor} \sqsubseteq_{\mathcal{T}} \text{Faculty}$$

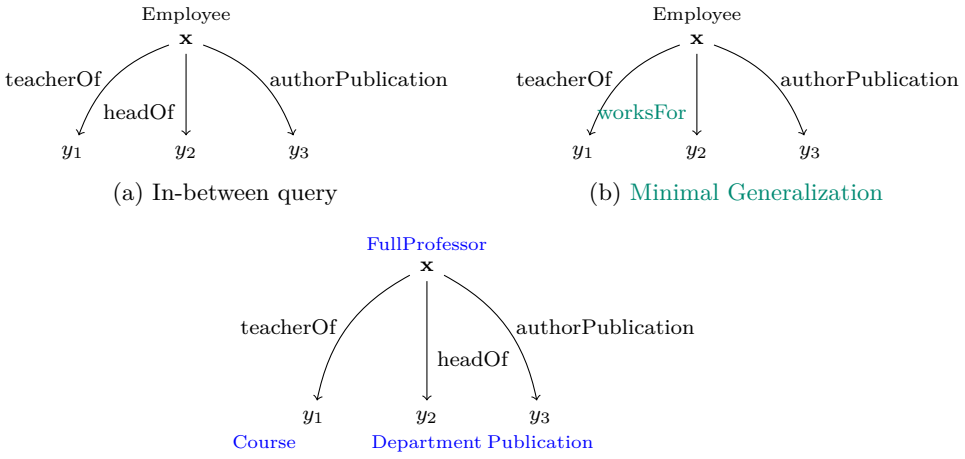
Therefore, the example of Figure 6.3 is very interesting since given the in-between query of Subfigure 6.3a, there are two incomparable minimal generalizations illustrated

in Subfigure 6.3b.

The unique maximal neutral specialization, represented in Subfigure 6.3c, adds more specific information that all answers of the in-between query have in common. Therefore, each Employee that teaches something and has published something is also, according to the existing data, a Faculty that worksFor a Department. Moreover we learn that each such individual teaches a Course, and is author of a Publication.

Considering the above axioms, if we want to specialize the query in a minimal way there are two options, according to Subfigure 6.3d.

Another interesting example, see Figure 6.4, is when all the answers of a given query, have in common an **x**-maximal query. Therefore, if we analyze the query of Subfigure 6.4c which is a maximal neutral specialization of query in Subfigure 6.4a, we obtain more concrete information with respect to the in-between query's answers. Basically each Employee that is headOf something is in fact a FullProfessor. What is interesting is that there is no axiom in the ontology which implies that each head of a department should be a full professor, so this information is inferred only from the data. In this specific case there are no strict specializations that would actually retrieve answers. For the same in-between query we obtained a minimal generalization, see Subfigure 6.4b, which relaxes headOf constraint and asks instead for employees that worksFor something.



(a) In-between query

(b) Minimal Generalization

(c) Maximal Neutral Specialization

Figure 6.4: Example of obtained modifications for in-between query of Batch 1

6.2.2 Batch2

The second round of experiments were conducted on the following lower- and upper-bound queries:

$$q_L(\mathbf{x}) : -\text{Organization}(\mathbf{x}), \text{member}(\mathbf{x}, y_1), \text{subOrganizationOf}(\mathbf{x}, y_2)$$

	#MWs	#PA
DS1	4108	6
DS2	7394	11
DS3	9988	15
DS4	14118	21
DS5	20917	31
DS6	22917	34
DS7	23523	35

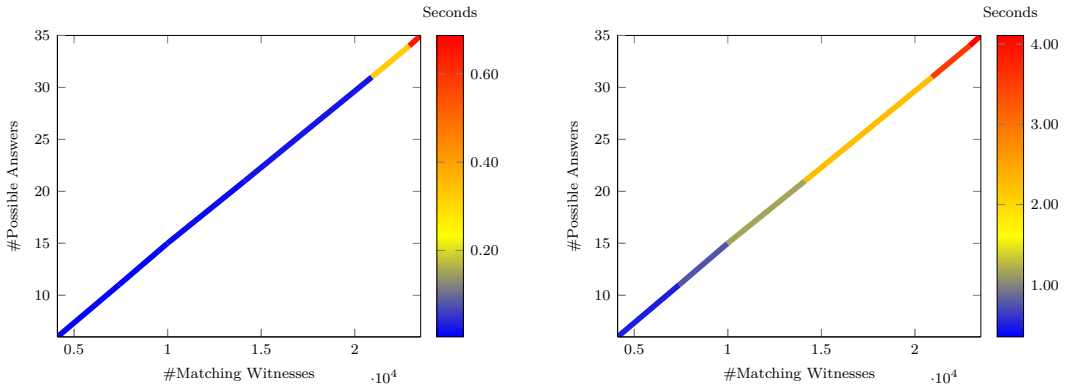
Table 6.3: Datasets with #Matching witnesses(MWs) and #Possible answers (PA) for Batch2

$$\begin{gathered}
\mathrm{q_U}(\mathbf{x}) : -\mathrm{Department}(\mathbf{x}), \mathrm{Institute}(\mathbf{x}), \mathrm{ResearchGroup}(\mathbf{x}), \mathrm{College}(\mathbf{x}), \\
\mathrm{subOrganizationOf}(\mathbf{x}, y_2), \mathrm{University}(y_2), \mathrm{member}(\mathbf{x}, y_1), \mathrm{FullProfessor}(y_1), \\
\mathrm{teacherOf}(y_1, z_1), \mathrm{orgPublication}(\mathbf{x}, y_3), \mathrm{publicationAuthor}(y_3, z_2), \\
\mathrm{ResearchAssistant}(z_2), \mathrm{worksFor}(z_3, u_1), \mathrm{ResearchGroup}(u_1), \\
\mathrm{takesCourse}(z_3, u_2), \mathrm{GraduateCourse}(u_2), \mathrm{Course}(u_2)
\end{gathered}$$

For this batch the 1treeCQs family contains 273 queries, among which only few of them retrieve answers. This is due to the fact that orgPublication is not liked to the data neither implicitly inferred from the ontology nor explicitly stated in the assertions of the ABoxes. The computation of matching witnesses is very influenced by the performance of Ontop and HermiT. For example gathering all the possible answers for all the batch, meaning the answers for $\mathrm{q_L}$, we use Ontop to get the instances by posing the associated SPARQL query. Since in this case $\mathrm{q_L}$ is a bit more complex than Batch1's $\mathrm{q_L}$, the evaluation time, which is dependent on the size of the data, ranges from 2.6 seconds for the small ABox, until 1 minute for the largest ABox.

This is a very interesting case regarding the number of possible answers and the number of created matching witnesses. As follows from Table 6.3, the possible answers are very restricted but they compute a very large number of matching witnesses. The reason is that each partial match for term y_1 in atom $\mathrm{member}(\mathbf{x}, y_1)$ is computed, meaning that for each organization and for each one of its members a matching candidate is created. However, the total number of organizations is very small.

This aspect has a very compelling effect on the performance of query answering procedure and $\mathbf{x}$-maximal queries computation as Figure 6.5 shows. Surprisingly, Subfigure 6.6a shows the reverse situation of the Batch1 performance. Query answering is done extremely fast over each data set. However, as seen in Subfigure 6.5a the number of



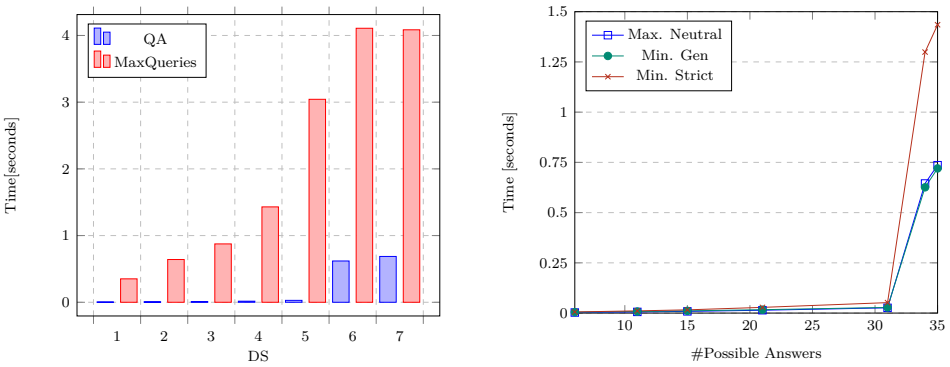
(a) Av. time for QA over Batch2

(b) Av. time for computing **x**-Maximal Queries over Batch2

Figure 6.5: Performance of QA and **x**-maximal queries construction for Batch2

possible answers is small for each data set. This is an interesting observation, keeping in mind that for Batch1, the number of possible answers is relatively large.

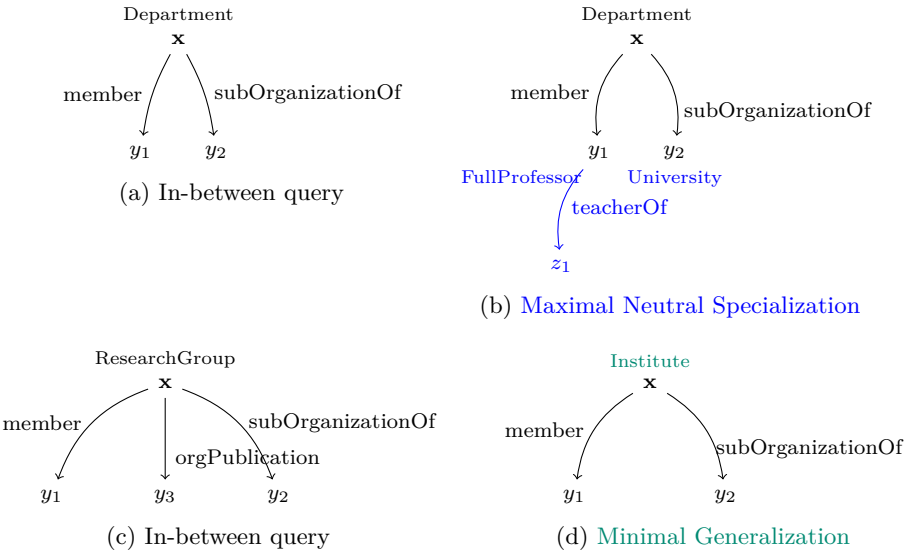
On the other hand, for larger data sets the average time to compute the **x**-maximal queries, as shown in Subfigure 6.6a, is almost 40 times higher than for Batch1, keeping in mind that the number of matching witnesses for Batch2 is slightly smaller than for Batch1. The results shown in Subfigure 6.5b are not explaining this alteration on performance compared to the previous batch. Intuitively, the cause is the size of a matching witness, since in case of Batch2, each organization has large number of members, which creates large set of matching candidates for variable y_1. Previously, in Batch1 the range of roles teacherOf, headOf or authorPublication contains considerably less individuals in the data.



(a) QA and Max. queries performance per Data set for Batch1

(b) Performance of query modifications for Batch2

Figure 6.6: Analysis of QA and Max queries performance over the data sets and the possible answers dependence of query modifications for Batch 2



(a) In-between query

(b) Maximal Neutral Specialization

(c) In-between query

(d) Minimal Generalization

Figure 6.7: Example of query modifications for in-between queries of Batch 2

The performance of query modifications for this batch is represented in Subfigure 6.6b. One can observe that in the case of small number of possible answers, the average computational time for any query modification is less than 1.5 seconds. Moreover, it looks like the hypothesis that the performance of each modification algorithm depends on the number of possible answers is rectified by these particular results. Since in this case we have small number of possible answers, the performance is significantly reduced compared to the results obtained over Batch1.

Provided that there are only 4 in-between queries in this batch which have answers

Table 6.4: Datasets with #Matching witnesses(MWs) and #Possible answers (PA) for Batch3

	#MWs	#PA
DS1	19880	3233
DS2	35537	5777
DS3	47873	7773

over the data sets, neither of these queries had minimal specializations without loosing all answers. Thus, Figure 6.7 shows the extra information from the KB that all the answers of the in-between query, pictured in Subfigure 6.7a, have in common over the Batch2 1treeCQs family. For query of Subfigure 6.7c, since there are no maximal neutral specializations, we get that it does not have answers. Thus, Subfigure 6.7d suggests to look up for Institute, given that each possible answer, accoring to the existing data in each ABox, is in fact a Department and the ontology contains the following axioms:

$$\text{Department} \sqsubseteq \text{Institute} \qquad \text{Institute} \sqsubseteq \text{Organization}$$

Moreover one needs to drop the orgPublication role atom since it does not have any matches in the data.

6.2.3 Batch3

The last batch represents the 1treeCQs family with the largest set of possible answers since the lower bound query is as follows:

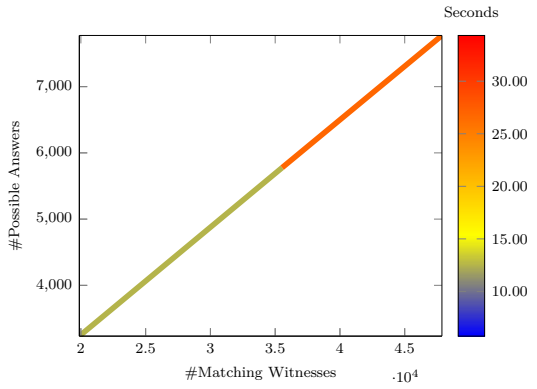
$$q_L(\mathbf{x}) : -\text{Student}(\mathbf{x})$$

The upper bound query fixed for this batch is:

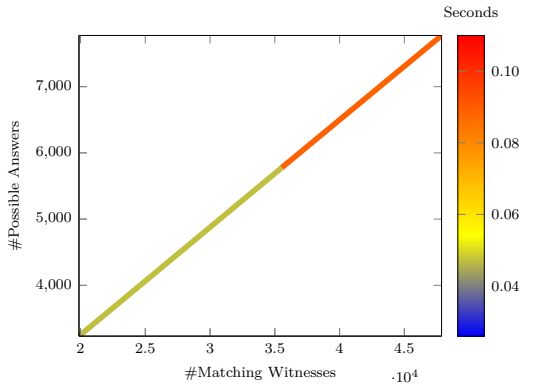
$$\begin{aligned} q_U(\mathbf{x}) : -&\text{GraduateStudent}(\mathbf{x}), \text{ResearchAssistant}(\mathbf{x}), \text{Student}(\mathbf{x}), \\ &\text{takesCourse}(\mathbf{x}, y_1), \text{Course}(y_1), \text{worksFor}(\mathbf{x}, y_3), \\ &\text{ResearchGroup}(y_3), \text{advisor}(\mathbf{x}, y_4), \text{AssistantProfessor}(y_4) \end{aligned}$$

For this batch, in particular, to compute the set of matching witnesses takes very long, and for the other data sets not presented in Table 6.4 the computation was time outed after few hours.

Also in this case it can be observed from Subfigure 6.8a that the performance of query answering procedure suffers when large number of possible answers are involved. However, once again the computation of $\mathbf{x}$-maximal queries is very efficient, see Subfigure 6.8b , in spite of the large number of possible answers, respectively of matching witnesses. For this batch again the size of a matching witness is relatively small, since each role atom in q_U ranges over a small part of the objects in the data.

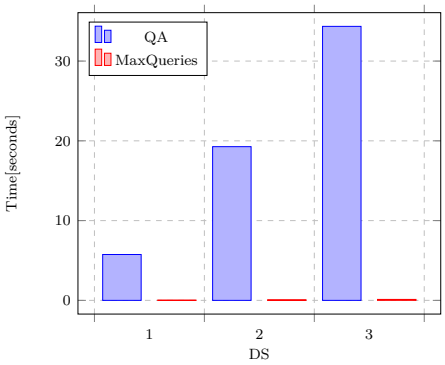


(a) Av. time for QA over Batch3

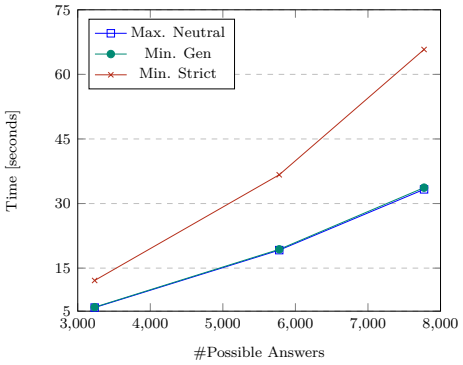


(b) Av. time for computing **x**-Maximal Queries over Batch3

Figure 6.8: Performance of QA and **x**-maximal queries construction for Batch3



(a) QA and Max. queries performance per Data set for Batch1



(b) Performance of query modifications for Batch3

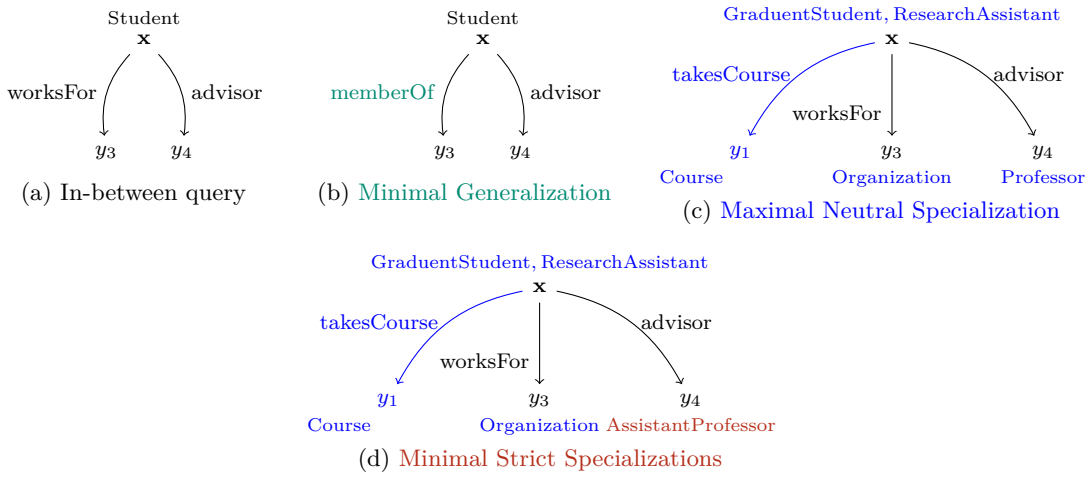
Figure 6.9: Analysis of QA and Max queries performance over the data sets and the possible answers dependence of query modifications for Batch 3

The striking difference between the performances of query answering procedure and **x**-maximal queries computation can be observed in the plot of Subfigure 6.9a.

Now, given that each type of query modification is making explicit use of the query answering procedure, the results of Subfigure 6.9b are quite predictable.

One of the most interesting modifications case retrieved in our evaluation is exemplified in Figure 6.10. For query pictured in Subfigure 6.10a, one can minimally relax the query, see Subfigure 6.10b, by changing worksFor label with a more general one, namely memberOf, since worksFor $\sqsubseteq_{\mathcal{T}}$ memberOf holds in the ontology.

Now, it looks like the in-between query can be maximally specialized in such way that the set of answers remains unchanged. Subfigure 6.10c offers much more information



(a) In-between query

(b) Minimal Generalization

(c) Maximal Neutral Specialization

(d) Minimal Strict Specializations

Figure 6.10: Example of obtained modifications for in-between query of Batch 3

regarding the answers since we obtained that each Student that worksFor something and has advisor is in fact a GraduateStudent and a ResearchAssistant that also takesCourse. If too many answers are retrieved, the query can be minimally specialized such that it will still have answers, see Subfigure 6.10d. In this case, concept atom Professor is changed with a sub-concept - AssistantProfessor.

6.3 Discussion

After analyzing the evaluation on each batch we reached to some interesting conclusions. The first one is that the performance of the solution depends very much on the implementation and especially on the reasoners used, therefore we believe that further improvements can be done.

The second important observation is that the performance of query answering procedure is strictly dependent on the number of possible candidates. Provided that each query modification algorithm is using the query answering procedure or it iterates over the possible answers, we can conclude that their performance is influenced by the size of the set of possible answers.

The third remark is a more intuitive one, given the surprising results related to the construction of **x**-maximal queries. The performance results do not show a connection between the number of possible answers or the number of matching candidates since for Batch1 and Batch3 the computation is very efficient in spite of the large number of possible answers or matching witnesses. We observed that if the size of a matching witness is large, since the algorithm branches over each matching candidate, the performance of algorithm constructMaxQueries is altered.

The most interesting part of our experiments are the computed query modifications which indeed offer useful information about targeted objects. Maximal neutral special-

izations help characterize the common properties that the answers of a query have in common, where in some cases the information is inferred strictly from the ABox. Minimal strict specialization provides solution for obtaining less answers, in a minimal way, such that the constructed query modification still retrieves answers. Minimal generalization modification provides the option to enlarge the set of answers, for a given query, and we obtained that in some cases there are indeed multiple incomparable choices to do so.

CHAPTER 7

Conclusions and Further Research

In this thesis we provided a compilation technique that can be used to retrieve and explore answers for queries within two bounds. Firstly, we defined the relation $\trianglelefteq_{\mathcal{T}}$ between two queries, with respect to the ontology, that together with the bound queries characterize a 1treeCQs family. We introduced the matching witness structure that is used to capture the part of the canonical model that is of interest for a family of 1treeCQs. The pre-computed set of matching witnesses represents the offline compilation tailored to support query answering of any 1treeCQ of the family and to compute the maximal matches on the upper-bound query for each possible answer. We provided an algorithm for answering queries over the compilation. We proved its correctness and analyzed its complexity.

Towards supporting the exploration of the possible answers of the 1treeCQs family, we defined $\mathbf{x}$-maximal queries and presented an algorithm for constructing all such queries for each possible answer. Using the computed maximal queries, we defined different types of query modifications for a given 1treeCQ: maximal neutral specialization, minimal strict specialization and minimal generalization. We implemented our algorithms using existing reasoners for dealing with ABox and TBox reasoning tasks and tested them based on experiments on 3 different batches of 1treeCQs.

There are still many questions that remain open. A natural next step is to explore how one can extend the solution to other DLs. It seems that this can be done at least for Horn-DLs, since a canonical model can be constructed. Moreover, the construction of the matching witnesses should not be hard to adapt to richer Horn-DLs. Another interesting but challenging question is how to extend the solution in such way that it considers queries that are not exactly tree-shaped. This would be indeed very valuable to achieve and we believe to be possible to adapt these techniques but it would be not trivial.

Regarding the actual solution, is there any optimization that can be made? One first look shows that collecting all possible matching candidates is costly and there are ways to reduce the size of a matching witness.

Another interesting direction for further work is to refocus on a different term as answer variable. For example using a compilation made for answering queries about students, we could also answer queries about courses. This seems very straight-forward considering that the provided solution is storing all the relevant information for this purpose.

Considering the implementation part, we are intending to provide a user interface. Moreover, as we mentioned before there are OMQA systems, such as Quelo [FGTT11] and SemFacet[ACGK$^+$14], which are constructing 1treeCQs based on the existing ontology. Therefore an integration of our solution might be interesting from a practical perspective.

The key contributions of this work are the novel perspective on query answering based on a pre-compilation of the KB, and the exploration of individuals in the ABox. The experiments carried out with our prototype implementation reveal an overall good performance of the query-answering procedure and, most importantly, seem to be a promising first step towards flexible ontology-mediated data exploration.

Bibliography

[ACGK+14] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. Faceted search over ontology-enhanced rdf data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 939–948, 2014.

[ACKZ09] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The dl-lite family and relations. *J. Artif. Int. Res.*, 36(1):1–69, September 2009.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the el envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 364–369, 2005.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[CGL+07] Diego Calvanese, G. De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. OF AUTOMATED REASONING*, page 2007, 2007.

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

[CTS11] Alexandros Chortaras, Despoina Trivela, and Giorgos B. Stamou. Optimized query rewriting for OWL 2 QL. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, pages 192–206, 2011.

[FGTT11] Enrico Franconi, Paolo Guagliardo, Sergio Tessaris, and Marco Trevisan. Quelo: an ontology-driven query interface, 2011.

[GHM+14] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An owl 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semant.*, 3(2-3):158–182, October 2005.

[KRRM+14] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *Proc. of International Semantic Web Conference (ISWC 2014)*, Lecture Notes in Computer Science. Springer, 2014.

[MGH+08] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. Owl 2 web ontology language: Profiles. World Wide Web Consortium, Working Draft WD-owl2-profiles-20081202, December 2008.

[MNP+14] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 129–137, 2014.

[Nut11] Premchand Nutakki. Specializing conjunctive queries in the el-family for better comprehension of result sets. Master's thesis, 2011.

[PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. In Stefano Spaccapietra, editor, *Journal on Data Semantics X*, volume 4900 of *Lecture Notes in Computer Science*, pages 133–173. Springer Berlin Heidelberg, 2008.

[PUHM09] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient query answering for owl 2. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 489–504, Berlin, Heidelberg, 2009. Springer-Verlag.

[RKZ13] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 558–573, 2013.

[SKZ+14] Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jimenez-Ruiz, Martin Giese, and Ian Horrocks. Optiquevqs: Visual query formulation for obda. In *Proceedings of the 27th International Workshop on Description*

Logics (DL 2014), volume 1193, pages 725–728, Vienna, Austria, 2014. CEUR-WS.org.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, June 2007.

[ZGN+15] Yujiao Zhou, Bernardo Cuenca Grau, Yavor Nenov, Mark Kaminski, and Ian Horrocks. Pagoda: Pay-as-you-go ontology query answering using a datalog reasoner. *J. Artif. Intell. Res. (JAIR)*, 54:309–367, 2015.