FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Explaining Query Answers in Lightweight Ontologies

The DL-Lite Case

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science (Computational Logic) (Msc)

im Rahmen des Studiums

Computational Logic (Erasmus Mundus)

eingereicht von

Giorgio Stefanoni
Matrikelnummer 1027562

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: o.Univ.-Prof. Dipl.-Ing. Dr. Thomas Eiter
Mitwirkung: Dr. Magdalena Ortiz
Dr. Mantas Šimkus

Wien, 21. Juli 2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Explaining Query Answers in Lightweight Ontologies

The DL-Lite Case

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computational Logic) (Msc)

in

Computational Logic (Erasmus Mundus)

by

Giorgio Stefanoni
Registration Number 1027562

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: o.Univ.-Prof. Dipl.-Ing. Dr. Thomas Eiter
Assistance: Dr. Magdalena Ortiz
Dr. Mantas Šimkus

Vienna, 21. Juli 2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Giorgio Stefanoni
Mailandstrasse 182, 39100 Bozen, IT

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

(Ort, Datum)	(Unterschrift Verfasser)

Abstract

Accessing data through ontologies expressed in Description Logics (DL) is important for the realization of the Semantic Web. *DL-Lite*$_{\mathcal{A}}$, member of the *DL-Lite* family of DLs, is particularly suitable for answering Conjunctive Queries over large data sets. In order to meet usability requirements, most logic-based applications provide explanation facilities for reasoning services. This holds also for *DL-Lite*$_{\mathcal{A}}$, where research focused on the explanation of both TBox reasoning and, more recently, query answering. This thesis consists of a complete study on the latter problem, hence, we study both explanations for the absence and the presence of a tuple in the answers.

Unfortunately, it happens that users are not always provided with the result they are expecting when querying an ontology. Then, it is important to provide them with an high-level motivation for the absence of a tuple in the results, i.e., find an explanation for a *negative answer*. In fact, in contrast with standard database technology, in the ontology setting this problem is not easily solvable by looking at the structure of the data stored in the data-layer and, for example, relaxing the conditions imposed in the query. The reason is that the reasoning involved for answering queries, provides an additional layer obfuscating the way the data is accessed by the system. We address the problem of *explaining negative answers for (conjunctive) query answering over DL-Lite*$_{\mathcal{A}}$ *ontologies*, by adopting abductive reasoning, that is, we look for additions to the ABox that force a given tuple to be in the result. As reasoning tasks, we consider existence and recognition of an explanation, and relevance and necessity of a certain assertion for an explanation. An important aspect in explanations is to provide the user with solutions that are simple to understand and free of redundancy, hence as small as possible. To address this requirement, we study various restrictions on solutions, in particular, we focus on subset minimal and cardinality minimal ones.

Besides explaining the absence of a tuple in a query answer, it is important to explain also why a given tuple is returned by the system. The presence of a tuple in the results of a query over a *DL-Lite*$_{\mathcal{A}}$ ontology does not only depend on the data but also on the constraints expressed at the conceptual level. For this reason, it is important to provide users with insights on how conceptual information has been used to return a certain tuple in the answers. This problem has been tackled already in the literature, where an high-level procedure *explaining positive answers to conjunctive queries over DL-Lite*$_{\mathcal{A}}$ *ontologies* is introduced. As now, the mentioned procedure has never been given a formal algorithmic counterpart and, therefore, it is not easily implementable in a real world system. For this reason, we close the gap between theory and practice by providing such an algorithmic solution to the problem. Also, we improve the given procedure by considering minimal explanations for positive answers. Further, this thesis contributes with a new use of this algorithm to solve the problem of explaining to domain users the inconsistency of a *DL-Lite*$_{\mathcal{A}}$ knowledge base.

Kurzfassung

Die Verwendung von Ontologien für die Vermittlung des Zugangs zu Daten ist für die Realisierung des Semantischen Web sicherlich von Wichtigkeit. *DL-Lite*$_{\mathcal{A}}$ ist Mitglied der *DL-Lite* Familie von Beschreibungslogiken und eignet sich besonders für den Zugriff auf grośe Datenbanken, insbesondere um konjunktive Abfragen auf dieselben durchzuführen.

Um den Ansprüchen einer einfachen Verwendung von Seiten der Benutzer entgegen zukommen, wurden erklärende Routinen innerhalb der Wissensbasiertes Systeme entwickelt, welche eine Hilfestellung für die verschiedenen Interpretationsdienste des Systems bieten. Dies gilt auch für *DL-Lite*$_{\mathcal{A}}$, wobei sich die Forschung auf die Erklärung der Interpretationsdienste auf Basis von TBox und, neuerdings, auf die Erklärung von Query answering konzentriert.

Unglücklicherweise kommt es vor, dass Benutzer bei Abfragen zu Ontologien nicht immer das von ihnen gewünschte Ergebnis geliefert bekommen. In diesem Fall ist es wichtig eine Begründung für die Abwesenheit eines Tupels in der Antwort des Systems zu liefern. Im Gegensatz zu relationalen Datenbanksystemen kann man bei Abfragen zu Ontologien dieses Problem nicht durch eine genaue Untersuchung der im System enthaltenen Daten, beispielsweise durch eine Änderung der Abfrage, lösen. Der Grund dafür ist, dass die logische Argumentation, mit der die Antworten für eine Abfrage zu Ontologien berechnet werden, die Art und Weise maskiert mit welcher die Abfrage auf die Daten zugreift. Diese Thesis behandelt die Problemstellung negative konjunktive Antworten auf *DL-Lite*$_{\mathcal{A}}$ Ontologien zu erklären, unter Verwendung einer abduktiven Argumentation, indem die Zusätze zur ABox gesucht werden welche ein bestimmtes Resultat der Abfrage verursachen. Wir betrachten in der Argumentation die Existenz und die Erkennung einer Erklärung, und die Relevanz und Notwendigkeit einer bestimmten Aussage für eine Erklärung. Ein wesentlicher Aspekt in den Erklärungen ist die Fähigkeit, Lösungen zu produzieren welche einfach zu verstehen sind, keine Redundanz aufweisen, und folglich so klein wie möglich sind. Um dieser Anforderung zu entsprechen untersuchen wir die Verwendung verschiedener Einschränkungen bei den Lösungen, insbesondere konzentrieren wir uns auf Erklärungen minimaler Untermengen und von minimaler Kardinalität.

Neben der Lieferung von Erklärungen für das Fehlen eines Tupels in den Antworten einer Abfrage, ist es auch wichtig die Gründe zu erläutern, die dazu führen dass ein Tupel Teil des Ergebnisses ist. In der Tat hängt die Präsenz eines Tupels in den Ergebnissen der Abfrage auf eine *DL-Lite*$_{\mathcal{A}}$ Ontologie nicht nur von den im System enthaltenen Daten ab, sondern auch von den Einschränkungen welche in der Ontologie auf konzeptioneller Ebene vorhanden sind. Aus diesem Grund ist es wichtig, den Benutzern zu erklären, wie die Aussagen auf der konzeptionellen Ebene verwendet wurden um die Antwort zu generieren. Dieses Problem wurde bereits von Borgida et. al in Angriff genommen, wobei diese ein Verfahren auf hoher Ebene einführen, um positive Antworten auf konjunktive Abfragen auf *DL-Lite*$_\mathcal{A}$ Ontologien zu erklären. Leider bietet diese Veröffentlichung keine algorithmische Lösung für das Problem, sondern nur eine Beschreibung des Verfahrens. Aus diesem Grund reduziert diese Thesis die Kluft zwischen Theorie und Praxis durch die Definition eines Algorithmus, der dieses Problem löst. Darüber hinaus tragen wir zur Lösung des Problems bei, den Benutzern die Inkonsistenz einer *DL-Lite*$_\mathcal{A}$ Ontologie zu erläutern, mittels Verwendung der Prozedur welche positive Antworten auf konjunktive Abfragen erklärt.

Contents

CHAPTER 1

Introduction

1.1 Overview

Conceptual analysis is a key aspect in both software development and knowledge engineering. In the past decade, ontologies have been advocated an important role in these fields. In philosophical terms, an *Ontology* is a formal specification of a shared conceptualization of a domain [18]. In 2004, the W3C[1] recommended a new family of knowledge representation languages for defining ontologies: the Web Ontology Language (OWL). The fundamental feature of these languages is their strong logical basis, since (most of) OWL is based on Description Logics [5]: a family of languages tailored for knowledge representation. The foundation in logics allows for the introduction of reasoning algorithms, which draw implicit conclusions from the explicit knowledge declared in ontologies. Subsequent efforts in increasing the expressivity of the language have led to the development of OWL2 [21], which introduces important modeling features and two lightweight profiles: OWL2 EL and OWL2 QL. The former is a subset of OWL2 suitable for applications requiring the use of large ontologies in terms of the defined vocabulary. The latter is an ontology language tailored for accessing large data sets through ontologies.

Accessing data through ontologies is an important aspect towards the realization of the Semantic Web [20]. Nowadays, organizations have to deal with a never-ending increase in the amount of information, while also coping with a diversification in the type of data (semi-structured and unstructured information) they may receive. Furthermore, an information system has also to be able to manage poor-quality information, for instance, incomplete data. Ontology-Based Data Access (OBDA) systems[2] aim at tackling these problems by mediating the access to data silos through an ontology [13]. This method allows to achieve logical-transparency in accessing data, that is, users are not aware on how the data is stored in the data layer, rather they are presented with a semantically-rich conceptual view of the stored information. Addi-

[1]`www.w3c.org`

[2]a.k.a. Ontology-Based Information Systems

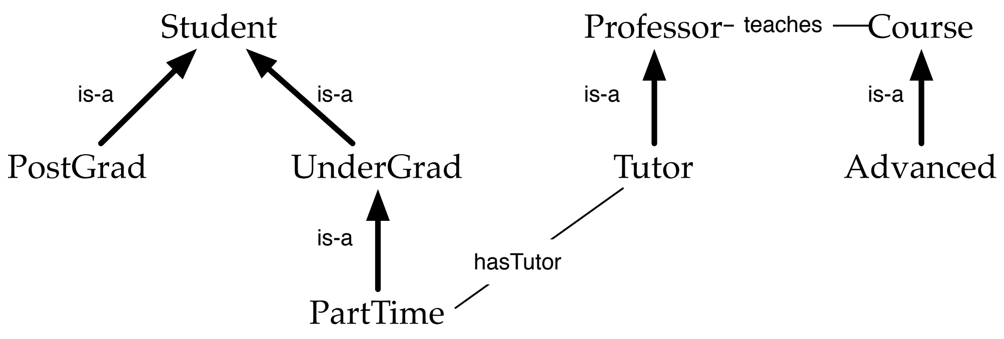


Figure 1.1: A graphic representation of a mock ontology $\mathcal{O}_{uni}$ describing a university domain.

tionally, data-management tasks are based on sound and complete reasoning algorithms, whose correctness can be proved formally.

The most important service users are interested in such a system is the ability of answering queries over an ontology. Answering queries over ontologies amounts at computing the *certain answers* to the query, that is, the tuples that are answers to the query in all the models of the ontology. Hence, unlike query evaluation over relational databases, computing the certain answers is a form of logical reasoning.

1.2 Problem Description

Query answering being a form of logical reasoning requires OBDA systems to be able to justify the computed certain answers. An explanation is needed both for the absence (*explaining negative query answers*) and for the presence of a tuple in the result (*explaining positive query answers*). In the following two subsections, we describe the two problems and we argue about their importance.

Explaining negative query answers

The logical reasoning adopted while answering queries over an ontology may cause a tuple, the user expects to be in results, not to be returned in the certain answers. The motivations of this absence cannot be found by simply looking at the way the user query accesses data. The reason is that the answers to queries over ontologies do not only depend on the data, but also on the conceptual level information provided by the ontology. The ontology is used to alter the way the user query accesses the data in order to take into account incompleteness in the information stored in the database. The logical reasoning involved in this process may be large in terms of inference schemes used and, therefore, almost impossible to be humanly understood by tracing the access to data.

In the relational database settings, there is the notion of database repair which allows to answer queries over inconsistent databases by suitably updating the relational instance [2, 3]. In case of query answering over ontologies, the fact that a tuple is considered to be part of the certain answer but it is not returned by the query answering system can be understood as a domain-level inconsistency in the ontology. That is, the data satisfies all logical constraints expressed in the ontology, but it is inconsistent from the domain perspective. Hence, it is important to provide users with the additions to the data that allows the given tuple to be returned in the certain answer. This problem is complementary to finding database repairs, since in that context, we search for the largest consistent subset of the given database over which we can evaluate queries safely.

The following example shows the use of such an explanation routine for negative answers over a mock ontology.

Example 1. Figure 1.1 shows a graphical representation of an ontology describing a university domain. In this fictitious university, there are two different kinds of students, $\mathit{PostGrad}$uates and $\mathit{UnderGrad}$uadetes. Moreover, undergraduate students may work while studying, in which case they are called $\mathit{PartTime}$ students. These students are helped by Tutors, who are particular professors. Each $\mathit{Professor}$ teaches at least one Course and viceversa. Additionally, the university offers some $\mathit{Advanced}$ courses.

The university administration uses the given ontology as a mean for accessing data. Unfortunately, the *University Information System* (UIS) has recently experienced a breakdown, which resulted in a data loss. Now, it contains only the following information:

$$\mathcal{A} = \{\mathit{teaches}(\mathit{craig}, \mathit{SWT}),\ \mathit{hasTutor}(\mathit{peter}, \mathit{craig})\}$$

i.e., craig teaches a course on Semantic Web Technologies (SWT) and tutors peter. Assume that the administration is interested in finding all those who both teach an advanced course and tutor a student. Then, the following query would be written:

$$q(x) \leftarrow \mathit{teaches}(x, y), \mathit{Advanced}(y), \mathit{hasTutor}(z, x).$$

The university administration may expect craig to be part of the result, but this is not the case. Intuitively, craig satisfies all the conditions imposed by the query, but the SWT course is not asserted to be $\mathit{Advanced}$. The following table shows some of the changes to the information system suggested by the explanation engine in order for craig to be in the result of the query.

Addition to the ontology	Motivation
$\mathit{Advanced}(\mathit{SWT})$	The course taught by craig is not asserted to be advanced.
$\mathit{teaches}(\mathit{craig}, \mathit{ALG}), \mathit{Advanced}(\mathit{ALG})$	The UIS lacks information about an advanced course taught by craig.
$\mathit{teaches}(\mathit{craig}, \mathit{TOC}), \mathit{Advanced}(\mathit{TOC})$, $\mathit{hasTutor}(\mathit{ben}, \mathit{craig})$	Add a new advanced course taught by craig and a new tutored student.

Hence, the explanation procedure provides users with suggestions on facts to be added to the information system, which help the user in two different ways. On the one hand, they give the user a clear insight on the portion of the ontology to be modified. On the other hand, it provides quick fixes for the query to run as expected.

In the example, one acceptable explanation for this lack in the query answers is the missing assertion of the course SWT to be advanced. However, another motivation may be given by the addition of an advanced course, which is asserted to be taught by $craig$. Finally, one can add both a new advanced course and a new student, which are related to $craig$. This latter solution is, however, the most expensive one in terms of facts to be added to the information system. In conclusion, the university administration can use domain knowledge to decide among the candidate solutions provided by the explanation engine.

Explaining positive query answers

Standard database management systems provide SQL debuggers, which may help database administrators in debugging SQL (recursive) stored procedures. These tools provide information on SQL call stacks and on the local/global views being used while returning a particular answer. Hence, these tools are only focused on how the data is accessed by means of SQL queries.

In query answering over ontologies, such a debugging procedure would not suffice in providing complete explanations on the reasons leading a tuple to be in the results. This is because, as explained before, the answers to queries do not only depend on the data, but also on the conceptual level information provided by the ontology. For this reason, it is important to have a routine able to unleash to domain users the conceptual level reasoning adopted in providing a particular answer to a query.

Providing such an explanation is vital in many fields. For instance, in critical decision-making systems a conceptual level explanation of an answer may facilitate business managers in taking risky decisions by increasing the trust in the answer itself. Also, the following example shows how such an explanation routine may be handy in managing data loss.

Example 2. Recalling the university mock ontology introduced in Example 1, suppose that the administration wants to understand how many professors are still listed in the information system and for this the following query is posed:

$$q(x) \leftarrow \mathit{Professor}(x).$$

which results in only $craig$ to be in the results. As the administration has checked, however, in the information system there is no direct information about professors, since the $\mathit{Professor}$ relation is empty. For this reason, they use the explanation routine in order to understand why $craig$ is considered to be a $\mathit{Professor}$. The routine returns the following motivation:

Rewritings	Ontology axiom	Motivation
$q(x) \leftarrow \mathit{Professor}(x)$	$\mathit{Tutor} \sqsubseteq \mathit{Professor}$	Tutors are Professors
$q(x) \leftarrow \mathit{Tutor}(x)$	$\exists hasTutor^{-} \sqsubseteq \mathit{Tutor}$	Tutors can be found in the range of hasTutor
$q(x) \leftarrow \mathit{hasTutor}(y, x)$	$\mathit{hasTutor}(peter, craig)$	$craig$ is in the range of hasTutor

That is, $craig$ is considered to be a professor because there is information in the system of him tutoring a student. Please note that this is not the only possible explanation. For instance, $craig$ being in the $teaches$ relation is another motivation for the answer.

In the next section, we will provide a summary on the state of the art in the context of explanations of query answers. In order to understand better the matter of the discourse, we will first introduce the relevant concepts and notations in Ontology-Based Data Access, and then focus more closely on the concept of explanation.

1.3 Related Work

Ontology-Based Data Access

OBDA systems mediate the access to data sources by means of an ontology [13]. The ontology provides a shared vocabulary that is used to describe the information contained in the data-layer. Also, this common vocabulary is tailored for formulating queries over the data layer. In fact, the most important service OBDA systems provide is answering queries expressed at the conceptual level. As now, all OBDA systems mediate the access to data by means of ontologies specified in one of the many Description Logics. The reason is that these systems have to deal with large amount of data and, hence, are sensitive to scalability issues.

Description Logics Description Logics (DL) are a family of formal languages specifically tailored for Knowledge Representation [5]. More precisely, a DL language is a decidable fragment of First Order Logic[3]. Due to this decidability result, it is possible to devise correct and terminating reasoning algorithms for DLs, which are used to explicit implicit knowledge. The different languages in this family are the result of the particular set of constructs available for modeling the domain. In general, the more constructs a DL has, the more complex it is to reason in that logic. This is the well known tradeoff between expressivity and complexity, which was first described by Brachman et. al in the late 80's [11].

A description logic is characterized by a concept language that is used to represent the domain of interest by means of *concept*s (sets of objects) and *role*s (binary relations between objects). However, DLs are not only suitable for describing classes of objects and relationships. In fact, it is possible to define Knowledge Bases (KB), which are used to model an aspect of the world, i.e., they are ontologies. A KB $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is constituted by a terminological component, the TBox $\mathcal{T}$, and by an assertional component, the ABox $\mathcal{A}$. The TBox is used to define the constraints that necessary have to hold in every valid instance of the ontology. The ABox asserts the facts that hold in every satisfying interpretation.

The fact that DLs are decidable fragments of FOL does not imply that query answering over DLs is decidable as well. As already mentioned before, answering a query over an ontology is a form of logical inference, since it amounts at computing the certain answers: the tuples that are answers to the query in all the models of the ontology (more formally: $\mathsf{cert}(q, \mathcal{O}) = \{\vec{c} \mid \vec{c} \in \mathsf{ans}(q, I), \text{for all models } \mathcal{I} \text{ of } \mathcal{O}\}$). Therefore, one cannot use Relational Calculus (a.k.a. Full SQL), a syntactic variation of First Order Logic, as query language over DLs, because this would lead to the undecidability of the problem [35, 36]. For this reason, the study of query answering over DLs has focused on less expressive, but decidable, query languages, such as Union of Conjunctive Queries (UCQs). This query language is the formal counterpart of union of select-project-join SQL queries.

Query Rewritability Unfortunately, answering UCQs over very expressive DLs is not scalable in terms of the size of the data to be accessed, i.e., w.r.t. the *data-complexity* of the prob-

[3]Some DLs have constructors that cannot be expressed in FOL (e.g. cyclic definitions). Nevertheless, it has been proved that such DLs are decidable as well.

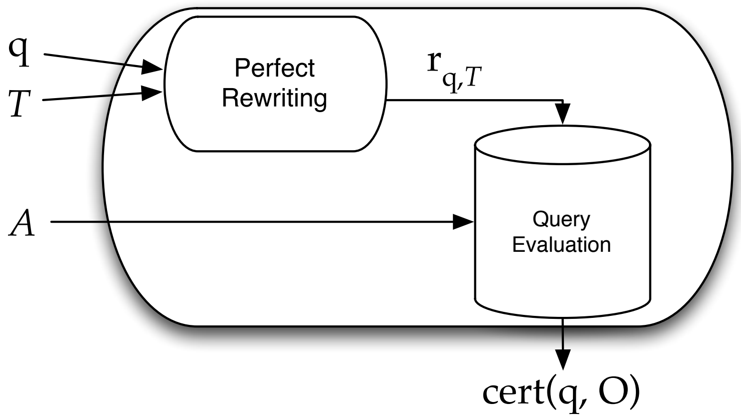


Figure 1.2: Computing query answers over ontologies by means of query rewriting. That is, computing the perfect reformulation of q over $\mathcal{T}$ and perform query evaluation over the ABox. This process is equivalent to computing the certain answers of q over $(\mathcal{T}, \mathcal{A})$.

lem [36]. For instance, the complexity of query answering over OWL2 ontologies is coNP-hard in data-complexity and 2-ExpTime-Complete in combined complexity [13, 26]. This result makes it practically impossible to use full OWL2 ontologies as scalable medium for accessing data.

In order to solve this issue, lightweight Description Logics have been introduced that are tailored for data access; among the others, we mention the *DL-Lite* family [4] and the $\mathcal{EL}$ family [24]. These two families are at the basis of the OWL2-QL and OWL2-EL profiles, respectively.

The introduction of specifically tailored ontology languages for data access does not solve the problem of managing large data sources. In fact, current ontology-based systems are not able to load large ABoxes containing GigaBytes or TeraBytes of data. The current idea behind OBDA systems is to delegate the storage of the data to external (relational) sources. In this way, one can reuse the ability of current DBMSs to deal with large datasets and, at the same time, cope with incompleteness and constraints over the data thanks to the reasoning involved at the conceptual level. Then, OBDA systems use the notion of query rewritability [13] (or combined query rewritability in case of $\mathcal{EL}$ [27]) to separate the contribution of the query and the conceptual schema from the contribution of the data. We say that (conjunctive) query answering over ontology language $\mathcal{L}$ is *$\mathcal{Q}$-rewritable* if for every ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ of $\mathcal{L}$ and for every query q, there exists a query $r_{q,\mathcal{T}}$ of $\mathcal{Q}$ such that evaluating $r_{q,\mathcal{T}}$ over the database instance $\mathcal{A}$ is equivalent to computing the certain answers to q over $\mathcal{O}$. Any such query $r_{q,\mathcal{T}}$ is called a *perfect reformulation* of q w.r.t. $\mathcal{T}$ (see Figure 1.2).

Intuitively, $r_{q,\mathcal{T}}$ is a query that extends q by taking into account the necessary constraints from the conceptual schema $\mathcal{T}$. Then, the certain answers of q over $\mathcal{O}$ are computed by simply evaluating the query rewriting $r_{q,\mathcal{T}}$ over the ABox. It is easy to see that $\mathcal{Q}$-rewritability is tightly related to the data complexity of computing the certain answers of the user query over the ontology. In fact, the data complexity of query answering corresponds precisely to the data complexity of evaluating $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$. Therefore, when it is possible to rewrite $\langle q, \mathcal{T} \rangle$ into an AC_0 query language, query evaluation can be delegated to a DBMS. In case the query can be rewritten into a coNP language, then query evaluation can be delegated to a disjunctive datalog database system.

Note that, if there is no constraint on the query language $\mathcal{Q}$ into which we want to rewrite to, the process described in Figure 1.2 is always applicable. However, if we want to rewrite into SQL, then query answering of UCQs over $\mathcal{T}$ must be AC_0 in data complexity. This has led to the investigation of the most expressive ontology language that enjoys FOL-rewritability. Both the *DL-Lite*$_\mathcal{A}$ and $\mathcal{EL}_{\perp}^{dr}$ enjoy (combined) FOL-rewritability, i.e., query answering can be delegated to a relational database by using the machinery of query rewriting. But only *DL-Lite*$_\mathcal{A}$ does not require any preliminary modification to the ABox, hence, it is the only one really achieving logical transparency. This is the reason why we decided to tackle the problem of explaining query answers over *DL-Lite*$_\mathcal{A}$ ontologies.

Having now presented the notion of query answering over ontologies, let us now introduce the notion of explanations both from a philosophical point of view and from a logical one.

Explanations for query answers

The notion of explanations stems from philosophy, where Carl Hampel and Paul Oppenheim have highlighted their properties and structure in the Deductive-Nonmological Model (D-N Model) [19,37]. An explanation is constituted by an *explanandum*, which is a sentence describing the phenomena to be explained and by an *explanans*, which is a class of sentences unfolding the meaning of the phenomena. For the explanans to be a correct explanation it must meet two requirements. First of all, the formality requirement, that is, the explanans must contain at least one *law of nature*. Since we focus on explanations in a logical environment, every valid formula in the ontology is considered to be a law of nature as well as any theorem of the logic. Finally, the explanans must meet the soundness requirement, i.e., the explanation should have the form of a sound deductive argument, where the explanandum follows from the explanans.

In order to meet usability requirements set by domain users, most logic-based applications provide explanation algorithms for reasoning services. This holds also for DLs, where research focused on the explanation of TBox reasoning [10,28,31,32]. Additionally, the use of logical reasoning in computing the answers to queries over ontologies has forced the study of a relatively new problem: the explanation of query answers.

Explanation for positive query answers The problem of explaining positive answers to Conjunctive Queries over *DL-Lite*$_\mathcal{A}$ ontologies has been studied by Borgida et al. in [9]. In this paper, they introduce both a logical calculus for explaining TBox reasoning and a description of a procedure for computing the reasons for a tuple $\vec{t}$ to be in the result sets of a CQ q. Unfortunately, this explanation procedure has never been formalized and, hence, an algorithmic solution to the problem is needed in order to facilitate the implementation of the procedure in real-world systems (see Section 5.1).

Explanation for negative query answers The same paper [9] advocates the importance of tackling the problem of computing explanations for negative query answers in the context of *DL-Lite*$_{\mathcal{A}}$ query answering. That is, provide users with the evidences and motivations that led a given tuple not to be in the certain answers to a query. This problem stems from the database community, where Chapman and Jagadish solved it in the context of databases extended with provenance information [14]. Later, this approach has been extended by Huang et al. in [22]. Unfortunately, these approaches are not general enough to be applied in the context of query answering over ontologies, where data-sets may not have any lineage information. To the best of our knowledge, in the literature there is no formalization of the problem in the DL context.

1.4 Methods and Results

After having looked at the state of the art in the explanation of query answering in the DL setting, we now present the methods we have used in tackling the described problems and the results we achieved.

Methodology

This thesis aims at tackling two fairly different kinds of problems. First of all, the problem of explaining negative query answers requires the formalization of a theoretical framework and complexity results to be established. Differently, solving the explanation problem for positive answers consists in finding an algorithmic solution to a procedure already presented in the literature.

As regards the explanation of negative query answers, we first have to devise a mathematical characterization of the problem, since it was not tackled before in the DL setting. We adopt *abductive ABox reasoning*, that is, we consider which additions need to be made to the ABox to force the given tuple to be in the result. An important aspect in explanations is to provide the user with solutions that are simple to understand and free of redundancy, hence as small as possible. To address this requirement, we study various restrictions on solutions, in particular, we focus on subset minimal and cardinality minimal ones. As reasoning tasks, we consider as suggested in [16]: *(i)* existence of an explanation, *(ii)* recognition of a given ABox as being an explanation, and *(iii)* relevance and *(iv)* necessity of an ABox assertion, i.e., whether it occurs in some or all explanations. After motivating such problems and formalizing them, we provide algorithms to solve them and a precise characterization of their computational complexity for *DL-Lite*$_{\mathcal{A}}$. The complexity of the various decision problems is delineated as follows. First, an effective algorithm solving the given problem is formalized and its worst-case asymptotic complexity is investigated. At this stage, it is important to formally demonstrate the correctness of the procedure. From this estimated complexity and by the algorithm's correctness, we can derive a (possibly non-optimal) upper-bound in complexity of the problem. Next, the standard technique for proving hardness of problems is used, that is, providing a reduction from a suitably hard problem. The reduction is then proved to be correct and that it can be effectively computed in polynomial time (or logarithmic space, when needed). This thesis aims at providing completeness results, i.e., each problem (if possible) has to be shown to be both, a member of a certain complexity class and to be hard for the same class. By a preliminary analysis over the considered reasoning problems, it seems that deciding the existence of an explanation is a keystone among all reasoning tasks, perhaps it can be used to solve other problems. For this reason, it is planned to start our complexity analysis from this particular problem. Afterwards, the complexity of the other reasoning tasks will be analyzed and proper connection with the existence problem will be highlighted. Finally, as rule of thumb, we decided to first concentrate on the complexity of problems under no minimality criterion. The reason is that the complexity analysis should be considerably easier and could provide a good starting point for solving the harder problems.

With respect to the problem of explaining positive answers, the idea is to provide a corresponding effective algorithm to the high-level procedure described in the literature [9]. This task consists in finding suitable data-structures into which to encode the information used in the high-level procedure, and, in identifying solutions to general-purpose problems (such as, search problems over graphs) that can be reused in the algorithmic solution. In this way, it is simpler to define the asymptotic complexity of the introduced algorithm. Furthermore, a closer look at the positive explanation procedure has highlighted some connections to the explanation problem for inconsistency in *DL-Lite*$_{\mathcal{A}}$ ontologies. In fact, it seems possible to solve the latter by using the algorithm computing explanations for positive answers. Therefore, in the thesis, this connection is studied and, if possible, a new procedure will be provided. In such a case, the differences between the new and the already introduced procedure solving the explanation problem for ontology unsatisfiability [9] will be discussed.

Results

This thesis provides a complete analysis at the problem of explaining query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies, by studying explanations both for positive and negative query answers.

First of all, this thesis addresses the problem of explaining negative answers over *DL-Lite*$_{\mathcal{A}}$ ontologies. For this, we introduce a mathematical framework modeling the problem based on abductive reasoning. Also, this thesis provides a detailed study on the complexity of the considered reasoning problems with respect to various minimality criteria (see Table 4.1), among the results we mention:

- checking the existence of an explanation for a negative query answer is PTIME-complete, which is proved through a double reduction from/to the PTIME-complete satisfiability problem for *DL-Lite*$_{\mathcal{A}}$ ontologies without the Unique Name Assumption;

- recognizing a set of assertions to be an explanation is NP-complete, whereas checking whether such a set is a minimal explanation w.r.t. the chosen minimality criterion is DP-complete. The NP-hardness is proved by a reduction from CQ-OPT (i.e., the problem of checking equivalences between two conjunctive queries), while DP-hardness is demonstrated by reducing the problem of deciding whether for two graphs G, G' it holds that G has an Hamiltonian Path and G' does not have one;

- deciding whether an assertion is cardinality-minimal necessary is $\mathsf{P}_{||}^{\mathsf{NP}}$-complete, while determining whether an assertions is subset-minimal relevant is Σ_2^{P}-complete. This latter hardness result is proved by a reduction from the problem named co-CERT3COL [34].

All the complexity results in the thesis are formally proved by providing both an algorithm justifying the upper-bound and a polynomial time reduction proving the hardness of the problem (or logarithmic space reduction in case we want to prove PTIME-hardness).

As we already mentioned, the problem of explaining positive answers has been tackled already by Borgida et. al in [9]. However, that paper does not provide an implementable algorithm, only a description of an high-level procedure. First of all, this thesis provides a new formalization of this explanation problem based on the derivability problem for $\textit{DL-Lite}_{\mathcal{A}}$ ABox reasoning. That is, the problem of finding formal proofs for ABox assertions that are entailed by a given ontology. Then, this thesis closes the gap between theory and practice by introducing an algorithmic solution to this newly formalized problem. This is achieved by defining a modified version of the standard algorithm computing perfect reformulations to conjunctive queries over $\textit{DL-Lite}_{\mathcal{A}}$ ontologies, which provides a data-structure that contains all the information needed to understand the reasons leading a tuple to be in the results. Differently from Borgida's procedure, our explanation algorithm is tailored for computing minimal explanations.

Finally, the thesis presents new algorithm solving the problem of explaining the reasons leading to an inconsistent $\textit{DL-Lite}_{\mathcal{A}}$ ontology. This procedure is based on the algorithm solving the explanation problem for positive answers and its peculiarity is that it highlights the contribution of both the TBox and the ABox in causing the unsatisfiability. Additionally, the algorithm provides domain users with the evidences in the data motivating the inconsistency in the ontology. The algorithm runs in polynomial time in the size of the TBox. During the design of this new procedure solving ontology unsatisfiability, the close study of explanation problems for TBox reasoning over $\textit{DL-Lite}_{\mathcal{A}}$ ontologies has brought up that the structural explanation routines for TBox reasoning provided in [9] were not optimal. In particular, we have designed a new set of structural procedures, which can explain TBox reasoning in polynomial space and time w.r.t. the size of the TBox, whereas explaining concept unsatisfiability according to [9] requires exponential time in the size of the TBox.

1.5 Structure of the Thesis

The rest of the work is organized as follows. The next chapter defines more in detail the notions of query answering and KB consistency for $\textit{DL-Lite}_{\mathcal{A}}$ ontologies. Chapter 3 defines formally the two problems of study, namely explaining the absence or the presence of a tuple in the results to a UCQ over a $\textit{DL-Lite}_{\mathcal{A}}$ ontology. Then, Chapter 4 provides complexity results for the relevant reasoning tasks over Query Abduction Problems and argues about the correctness of the introduced procedures. Chapter 5 tackles the problem of explaining positive query answers by introducing an algorithmic solution to the problem. Also, it presents a new solution to the problem of explaining KB inconsistency of $\textit{DL-Lite}_{\mathcal{A}}$ ontologies based on the algorithm computing explanations for positive query answers. Next, Chapter 6 provides a summary of the thesis and highlights some of the future steps that can be done to improve this work. Finally, the supplementary Chapter I introduces new algorithms for explaining *DL-Lite*$_{\mathcal{A}}$ TBox reasoning in polynomial time, while Appendix A introduces the Dijkstra's algorithm for finding shortest paths in a weighted graph.

CHAPTER 2

Preliminaries

2.1 *DL-Lite*$_{\mathcal{A}}$ Language

DL-Lite$_{\mathcal{A}}$ is an expressive member of the *DL-Lite* family of DLs [13]. Let $\mathbf{A}$, $\mathbf{P}$, $\mathbf{C}$ be, respectively, a countably infinite set of concept names, a countably infinite set of role names and a countably infinite set of constant (*individual*) names. Then in *DL-Lite*$_{\mathcal{A}}$, concept expressions C, denoting sets of objects, and role expressions R, denoting binary relations between objects, are formed as follows:

$$C \longrightarrow A \mid \exists R, \qquad\qquad R \longrightarrow P \mid P^{-}.$$

where A is a concept name from $\mathbf{A}$ and P a role name from $\mathbf{P}$[1]. In a *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, the TBox $\mathcal{T}$ consists of axioms of the form:

$$\begin{array}{lllllll} C_1 & \sqsubseteq & C_2, & \qquad R_1 & \sqsubseteq & R_2, & \\ C_1 & \sqsubseteq & \neg C_2, & \qquad R_1 & \sqsubseteq & \neg R_2, & \qquad (\mathsf{funct}\ R) \end{array}$$

where the first row consists of *positive inclusions* among concepts or roles, while the second row contains *negative inclusions* (*disjointness axioms*) among concepts or roles and *functionality assertions* on roles. Note that for computational reasons, roles asserted to be functional in the TBox $\mathcal{T}$ are not allowed to be further specialized by means of role inclusions in $\mathcal{T}$.

ABox assertions are expressions of the form:

$$C(a), \qquad R(a, b).$$

where C is a concept expression and R a role expression, while a and b are constants from $\mathbf{C}$. *DL-Lite*$_{\mathcal{A}}$ ABoxes consist of a restricted form of ABox assertions built only from concept names in $\mathbf{A}$ and role names in $\mathbf{P}$, i.e., expressions have the form $A(a)$ or $P(a, b)$.

[1]We ignore here the distinction between data values and objects, since it is immaterial for our results. As a consequence, we not consider value domains and attributes, which are present in *DL-Lite*$_{\mathcal{A}}$.

The semantics of *DL-Lite*$_{\mathcal{A}}$ is, as usual in DLs, based on first-order interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty interpretation domain. The interpretation function $\cdot^{\mathcal{I}}$ maps each individual name a to $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, each concept name A to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name P to $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Then, the semantics of expressions is determined as follows:

$$\begin{array}{rcl} (\exists R)^{\mathcal{I}} & = & \{o^{\mathcal{I}} \mid \exists o' \ \langle o^{\mathcal{I}}, o'^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}, \\ (P^{-})^{\mathcal{I}} & = & \{\langle o^{\mathcal{I}}, o'^{\mathcal{I}}\rangle \mid \langle o'^{\mathcal{I}}, o^{\mathcal{I}}\rangle \in P^{\mathcal{I}}\}. \end{array}$$

If the *Unique Name Assumption* (UNA) is adopted, then for every interpretation $\mathcal{I}$ and distinct individuals c_1, c_2, we have that $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$.

Also, the interpretation $\mathcal{I}$ *satisfies* $\alpha_1 \sqsubseteq \alpha_2$ if $\alpha_1^{\mathcal{I}} \subseteq \alpha_2^{\mathcal{I}}$, it satisfies $\alpha_1 \sqsubseteq \neg\alpha_2$ if $\alpha_1^{\mathcal{I}} \cap \alpha_2^{\mathcal{I}} = \emptyset$, and it satisfies $(\mathsf{funct}\ R)$ if $R^{\mathcal{I}}$ is a function (i.e., if $\langle o^{\mathcal{I}}, z_1^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$ and $\langle o^{\mathcal{I}}, z_2^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$, then $z_1^{\mathcal{I}} = z_2^{\mathcal{I}}$). Also, $\mathcal{I}$ satisfies $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and it satisfies $R(a, a')$ if $\langle a^{\mathcal{I}}, a'^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$, if it satisfies all the axioms in $\mathcal{T}$. Similarly, it is a model of the ABox $\mathcal{A}$, if it satisfies all the assertions in $\mathcal{A}$. Finally, $\mathcal{I}$ is a model of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$, if it satisfies both the TBox and the ABox.

An ontology $\mathcal{O}$ is called *satisfiable* (or *consistent*) if $\mathcal{O}$ admits at least a model, otherwise it is called *unsatisfiable* (*inconsistent*). A concept C (role R) is called satisfiable w.r.t. an ontology $\mathcal{O}$, if there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $C^{\mathcal{I}} \neq \emptyset$ (resp. $R^{\mathcal{I}} \neq \emptyset$). An ontology $\mathcal{O}$ *logically implies* an axiom (or assertion) α, written $\mathcal{O} \models \alpha$, if all models of $\mathcal{O}$ satisfy α.

Example 3. Let us now present a fragment of the *DL-Lite*$_{\mathcal{A}}$ characterization of the mock university ontology presented in Figure 1.1. The following *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}'$ models the conceptual relations among *Professor*s and *Course*s.

$$\begin{array}{lrcl} 1 & \mathit{Professor} & \sqsubseteq & \exists \mathit{teaches}, \\ 2 & \mathit{Course} & \sqsubseteq & \exists \mathit{teaches}^{-}, \\ 3 & \mathit{Advanced} & \sqsubseteq & \mathit{Course}, \\ 4 & \exists \mathit{teaches} & \sqsubseteq & \mathit{Professor}, \\ 5 & \exists \mathit{teaches}^{-} & \sqsubseteq & \mathit{Course}. \end{array}$$

That is, in this university domain each *Professor* has to teach at least one *Course* and each *Course* must be taught by at least one *Professor*. Also, this university offers special kind of *Course*s called *Advanced* courses. Finally, the domain of the *teaches* relation contains *Professor*s, while the range contains *Course*s.

Similarly, the following ABox $\mathcal{A}'$ asserts the known facts about *Professor*s teaching *Course*s.

$$\mathit{teaches}(\mathit{craig}, \mathit{SWT})$$

Then, $\mathcal{O}'_{uni} = \langle \mathcal{T}', \mathcal{A}'\rangle$ is the *DL-Lite*$_{\mathcal{A}}$ ontology, which characterizes the small fragment of the university domain that accounts for *Professor*s and *Course*s.

Before actually delving into the properties of this Description Logic, let us define the query language of study, namely Union of Conjunctive Queries.

2.2 Union of Conjunctive Queries

Let $\mathbf{V}$ be a countably infinite set of variables. Since we are interested in querying *DL-Lite*$_{\mathcal{A}}$ ontologies, in the following it is assumed that queries contain at most binary relations.

Expressions $A(t)$ and $P(t, t')$ are called *atoms*, where $t, t' \in \mathbf{V} \cup \mathbf{C}$. A *conjunctive query (CQ)* q is an expression

$$q(x_1, \ldots, x_n) \leftarrow a_1, \ldots, a_m.$$

where each a_i is an atom, with $1 \leq i \leq m$. Let $\mathbf{V}(q)$ denote the set of variables occurring in q, $\mathbf{C}(q)$ denote the set of constants in q, and let $at(q) = \bigcup_{1 \leq i \leq m}\{a_i\}$. The tuple $\langle x_1, \ldots, x_n \rangle$ is the tuple of *answer variables* of q and is denoted by $\mathbf{AV}(q)$. Then, $\mathbf{NV}(q) = \mathbf{V}(q) \setminus \mathbf{AV}(q)$ is the set of non-distinguished (or existential) variables of q.

Example 4. For instance, in the university example provided in the previous chapter:

$$q(x) \leftarrow teaches(x, y), Advanced(y), hasTutor(z, x)$$

is a CQ with three different atoms and a single answer variable x.

A *match* for q in an interpretation $\mathcal{I}$ is a mapping:

$$\pi : \mathbf{V}(q) \rightarrow \Delta^{\mathcal{I}}$$

such that π is the identity on constants, $\langle \pi(t), \pi(t') \rangle \in P^{\mathcal{I}}$ for all $P(t, t') \in at(q)$, and, $\langle \pi(t) \rangle \in A^{\mathcal{I}}$ for all $A(t) \in at(q)$. The *answer* to q over $\mathcal{I}$, denoted $\mathsf{ans}(q, \mathcal{I})$, is the set of all n-tuples $\langle d_1, \ldots, d_n \rangle \in \mathbf{C}^n$ such that $\langle d_1^{\mathcal{I}}, \ldots, d_n^{\mathcal{I}} \rangle = \langle \pi(x_1), \ldots, \pi(x_n) \rangle$ for some match π for q in $\mathcal{I}$.

A *union of conjunctive queries* (UCQ) is a set of conjunctive queries over the same answer variables $\vec{x}$. For a UCQ q, we let

$$\mathsf{ans}(q, \mathcal{I}) = \bigcup_{q' \in q} \mathsf{ans}(q', \mathcal{I})$$

The complexity of answering (U)CQs over relational databases is NP-Complete in combined complexity and AC$_0$ in data-complexity [1, 36]. Finally, the *certain answers* to a CQ or a UCQ q over an ontology $\mathcal{O}$ are defined as

$$\mathsf{cert}(q, \mathcal{O}) = \{\vec{c} \in \mathbf{C}^n \mid \vec{c} \in \mathsf{ans}(q, \mathcal{I}), \text{ for each model } \mathcal{I} \text{ of } \mathcal{O}\}.$$

2.3 *DL-Lite*$_{\mathcal{A}}$ Properties

DL-Lite$_{\mathcal{A}}$ ontologies admit a particular kind of models, called *canonical interpretations* [13]. Let $\mathcal{I}, \mathcal{J}$ be two FOL interpretations over the same set of predicate symbol $\mathcal{P}$, we call $h : \Delta^{\mathcal{I}} \mapsto \Delta^{\mathcal{J}}$ an homomorphism from $\mathcal{I}$ to $\mathcal{J}$ if for all predicate symbols $P \in \mathcal{P}$ of arity n and for all tuples of constants $\langle o_1, \ldots, o_n \rangle \in \Delta^{\mathcal{I}}$, we have that: if $\langle o_1, \ldots, o_n \rangle \in P^{\mathcal{I}}$, then $\langle h(o_1), \ldots, h(o_n) \rangle \in P^{\mathcal{J}}$.

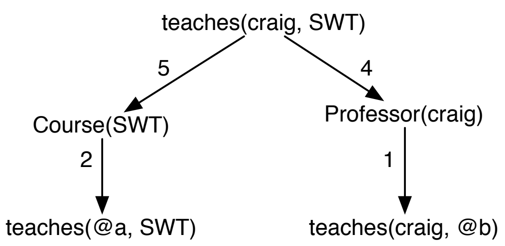


Figure 2.1: This figure shows the construction of the chase of $\mathcal{O}'_{uni}$ from Example 3. The edge labels indicate the TBox axioms being applied. Note that when we apply axiom 1 and 2 a new Skolem constant is introduced.

Definition 1 (Canonical Interpretation)**.** Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{A}$ ontology. A model $\mathcal{I}_\mathcal{O}$ of $\mathcal{O}$ is called a *canonical interpretation* if for every other model $\mathcal{I}$ of $\mathcal{O}$, there exists a homomorphism from $\mathcal{I}_\mathcal{O}$ to $\mathcal{I}$.

All these (possibly infinite) canonical interpretations $\mathcal{I}_\mathcal{O}$ are homomorphically equivalent between each other and, hence, one can assume to be dealing with a single *canonical model*. Furthermore, this canonical model can be defined in terms of a *chasing* procedure [13, 15], i.e., by constructing the chase of the ontology, denoted by $chase(\mathcal{O})$. Basically, $chase(\mathcal{O})$ is a possibly infinite set of ABox assertions constructed inductively from the ABox $\mathcal{A}$. At each step of the construction, a positive inclusion α from TBox $\mathcal{T}$ is applied to one of the assertions in the chase. The application of positive inclusions results in new assertions to be added. For instance, in Figure 2.1 assertion $Professor(craig)$ and axiom $Professor \sqsubseteq \exists teaches$ lead the assertion $teaches(craig, @b)$ to be added to the chase, where $@b$ is a fresh *Skolem constant*. The reason why the chasing procedure employs Skolem constants is that the canonical model is required to be as less restrictive as possible. The process terminates when no new assertions can be added to the chase. The details on the construction of the canonical model $can(\mathcal{O})$ from $chase(\mathcal{O})$ and the proof of correctness of this approach can be found in [13].

A particularly interesting result following from the notion of canonical model is the following:

Proposition 1 ([13])**.** *$can(\mathcal{O})$ is a model of $\mathcal{O}$ if and only if $\mathcal{O}$ is satisfiable.*

That is, every satisfiable *DL-Lite*$_\mathcal{A}$ ontology has a canonical model. On the other hand, if the ontology $\mathcal{O}$ is unsatisfiable, then constructing $can(\mathcal{O})$ from the ontology's chase would not provide a model of $\mathcal{O}$. Note that, it is not advisable to solve the problem of checking ontology satisfiability by constructing the canonical model, since, in general, $can(\mathcal{O})$ may be infinite. However, this result gives us a proper way to tackle the problem of ontology satisfiability as it is explained in the following.

KB Consistency

```
1 INPUT: $\textit{DL-Lite}_{\mathcal{A}}$ ontology $\mathcal{O}$
2 OUTPUT: true iff $\mathcal{O}$ is satisfiable
  1: $cln(\mathcal{T}) = computeNegClosure(\mathcal{T})$
  2: $DB(\mathcal{A}) = db(\mathcal{A})$
  3: $q_{unsat(\mathcal{T})} = \{\bot\}$
  4: for $\alpha \in cln(\mathcal{T})$ do
  5:     $q_{unsat(\mathcal{T})} = q_{unsat(\mathcal{T})} \cup \{\delta(\alpha)\}$
  6: end for
  7: if $\mathsf{ans}(q_{unsat}, DB(\mathcal{A})) = \emptyset$ then
  8:     return true
  9: end if
 10: return false
```

Algorithm 2.1: SATISFIABLE($\mathcal{O}$)

Consistency checking is an important reasoning problem over *DL-Lite*$_{\mathcal{A}}$ ontologies, the reason being that every other TBox reasoning task can be reduced to a KB consistency check [13]. Now, we will give the intuition on how it is possible to decide ontology satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies by evaluating a suitably constructed query over a database interpretation.

First of all, it is not difficult to see that every ontology that does not contain any disjointness nor functionality axiom is always satisfiable. Roughly speaking, the reason is that the way the canonical model is constructed deals with the positive inclusions occurring in the terminology, and, therefore $can(\mathcal{O})$ is always a model for such a positive ontology. Differently, if negative axioms are stated in the TBox, then one needs to take care of the interactions occurring between positive and negative inclusions. For this, we define the set $cln(\mathcal{T})$:

(1) all functionality assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;

(2) all negative inclusions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;

(3) if $B_1 \sqsubseteq B_2$ is in $\mathcal{T}$ and $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ are in $cln(\mathcal{T})$, then $B_1 \sqsubseteq \neg B_3$ is in $cln(\mathcal{T})$;

(4) if $R_1 \sqsubseteq R_2$ is in $\mathcal{T}$ and $\exists R_2 \sqsubseteq \neg B$ or $B \sqsubseteq \neg\exists R_2$ is in $cln(\mathcal{T})$, then also $\exists R_1 \sqsubseteq \neg B$ is in $cln(\mathcal{T})$;

(5) if $R_1 \sqsubseteq R_2$ is in $\mathcal{T}$ and $\exists R_2^- \sqsubseteq \neg B$ or $B \sqsubseteq \neg\exists R_2^-$ is in $cln(\mathcal{T})$, then also $\exists R_1^- \sqsubseteq \neg B$ is in $cln(\mathcal{T})$;

(6) if $R_1 \sqsubseteq R_2$ is in $\mathcal{T}$ and $R_2 \sqsubseteq \neg R_3$ or $R_3 \sqsubseteq \neg R_2$ is in $cln(\mathcal{T})$, then also $R_1 \sqsubseteq \neg R_3$ is in $cln(\mathcal{T})$;

(7) if one of the assertions $\exists R \sqsubseteq \neg\exists R$, $\exists R^- \sqsubseteq \neg\exists R^-$, or $R \sqsubseteq \neg R$, then all three assertions are in $cln(\mathcal{T})$.

$cln(\mathcal{T})$ denotes the negative closure of the assertions occurring in the TBox. It can be proved that, for every *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$ and negative inclusion or functionality assertion α the following holds: $\mathcal{T} \models \alpha$ if and only if $cln(\mathcal{T}) \models \alpha$ [13]. From this it follows that one can resort to $cln(\mathcal{T})$ to check whether negative inclusions are entailed by the TBox.

Now, it is possible to define the database interpretation $DB(\mathcal{A})$ over which we evaluate the query testing satisfiability. Subsequently, we present the relation between $DB(\mathcal{A})$ and $cln(\mathcal{T})$ on the one hand, and the satisfiability problem for *DL-Lite$_\mathcal{A}$* ontologies, on the other hand.

Definition 2. Let $DB(\mathcal{A})$ be the following interpretation:

(i) $\Delta^{DB(\mathcal{A})}$ is the set of individuals occurring in $\mathcal{A}$,

(ii) $A^{DB(\mathcal{A})} = \{o \in \Delta^{DB(\mathcal{A})} \mid A(o) \in \mathcal{A}\}$, for each atomic concept A, and

(iii) $P^{DB(\mathcal{A})} = \{\langle o, o' \rangle \in \Delta^{DB(\mathcal{A})} \mid P(o, o') \in \mathcal{A}\}$, for each atomic role P.

Proposition 2 ([13]). *DL-Lite$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.*

Given this result, it is simple to check whether $DB(\mathcal{A})$ is a model of the closure of the TBox. This can be done by evaluating a UCQ with inequalities over $DB(\mathcal{A})$ itself:

$$\begin{aligned}
&\delta((\mathsf{funct}\ P)) = \exists x, y_1, y_2.\ P(x, y_1), P(x, y_2), y_1 \neq y_2 \\
&\delta((\mathsf{funct}\ P^-)) = \exists x, y_1, y_2.\ P(y_1, x), P(y_2, x), y_1 \neq y_2 \\
&\delta(B_1 \sqsubseteq \neg B_2) = \exists x.\ \gamma_1(B_1, x), \gamma_2(B_2, x) \\
&\delta(Q_1 \sqsubseteq \neg Q_2) = \exists x, y.\ \rho(Q_1, x, y), \rho(Q_2, x, y)
\end{aligned}$$

where γ_i and ρ are defined as follows:

$$\gamma_i(B, x) = \begin{cases} A(x) & \text{if } B = A, \\ \exists y_i.\ P(x, y_i) & \text{if } B = \exists P, \\ \exists y_i.\ P(y_i, x) & \text{if } B = \exists P^-, \end{cases} \qquad \rho(Q, x, y) = \begin{cases} P(x, y) & \text{if } Q = P, \\ P(y, x) & \text{if } Q = P^-. \end{cases}$$

Algorithm Satisfiable in Figure 2.1 computes $DB(\mathcal{A})$ and the closure of the TBox, given an ontology $\mathcal{O}$. Then, query q_{unsat} is iteratively computed, which intuitively asks whether $DB(\mathcal{A})$ contains any instance violating the constraints imposed by the TBox. If no such evidence can be found, then the ontology is satisfiable. Therefore, *DL-Lite$_\mathcal{A}$* under the Unique Name Assumption enjoys FOL-rewritability of ontology satisfiability, that is, it can be decided by evaluating a suitable FOL query over $DB(\mathcal{A})$.

By FOL-rewritability it follows that *DL-Lite$_\mathcal{A}$* consistency is AC$_0$ in data-complexity. Additionally, it can be proven that the algorithm runs in polynomial-time in the size of the TBox.

Query Answering

Since the canonical model of a *DL-Lite$_\mathcal{A}$* ontology $\mathcal{O}$ is homomorphically equivalent to any other valid interpretation of $\mathcal{O}$, it provides a way for computing the certain answers to a UCQ over the ontology:

Table 2.1: Result of applying positive inclusion α over query atom g.

Atom g	Positive inclusion α	$gr(g, \alpha)$
$A(x)$	$A_1 \sqsubseteq A$	$A_1(x)$
$A(x)$	$\exists P \sqsubseteq A$	$P(x, \_)$
$A(x)$	$\exists P^- \sqsubseteq A$	$P(\_, x)$
$P(x, \_)$	$A \sqsubseteq \exists P$	$A(x)$
$P(x, \_)$	$\exists P_1 \sqsubseteq \exists P$	$P_1(x, \_)$
$P(x, \_)$	$\exists P_1^- \sqsubseteq \exists P$	$P_1(\_, x)$
$P(\_, x)$	$A \sqsubseteq \exists P^-$	$A(x)$
$P(\_, x)$	$\exists P_1 \sqsubseteq \exists P^-$	$P_1(x, \_)$
$P(\_, x)$	$\exists P_1^- \sqsubseteq \exists P^-$	$P_1(\_, x)$
$P(x_1, x_2)$	$P_1 \sqsubseteq P$ or $P_1^- \sqsubseteq P^-$	$P_1(x_1, x_2)$
$P(x_1, x_2)$	$P_1 \sqsubseteq P^-$ or $P_1^- \sqsubseteq P$	$P_1(x_2, x_1)$

Proposition 3 ([13]). *Let $\mathcal{O}$ be a satisfiable ontology and let q be a UCQ over $\mathcal{O}$. Then:*

$$\mathsf{cert}(q, \mathcal{O}) = \mathsf{ans}(q, can(\mathcal{O}))$$

Unfortunately, the canonical model is generally infinite. For this reason, query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies adopts a technique based on query rewritings in order to compute the certain answers to user queries. Intuitively, the idea is to use a set of rewriting rules (see Table 2.1) to encapsulate conceptual level information in the query itself. At the end of this process, the resulting query, called the *perfect reformulation* of the query w.r.t. the TBox, is evaluated over the database interpretation $DB(\mathcal{A})$.

Algorithm *PerfectRef* in Figure 2.2 inductively computes the perfect reformulation of a UCQ q w.r.t. the constraints expressed in a *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$. Roughly speaking, positive inclusions occurring in the TBox are used as rewriting rules, applied from left to right, which encapsulate the relevant TBox axioms into the query itself. At each step of the construction, for every positive inclusion $\alpha \in \mathcal{T}$ applicable (according to the rewriting rules) to atom g of some CQ r_i in the perfect reformulation, the algorithm adds a new query to the reformulation resulting from substituting g with $gr(g, \alpha)$ in r_i. The process stops when there is no new rewriting to be added. Step (b) is a unification step, whose role is to allow the largest possible number of rewriting steps to be applied. In order to achieve this, the result of the unification of two atoms $reduce(q', g_1, g_2)$ is given in input to the anonymization function $anon$, which replaces each anonymous variable[2] in q' by _.

Example 5. Recalling the mock university ontology $\mathcal{O}_{uni} = \langle \mathcal{T}, \mathcal{A} \rangle$ from the previous chapter,

[2]A variable is said to be anonymous if it is a non-distinguished variable occurring only once in the query body.

```
1 INPUT: UCQ $q$, $DL\text{-}Lite_{\mathcal{A}}$ TBox $\mathcal{T}$
2 OUTPUT: the perfect reformulation $pr$.
 1: $pr = q$
 2: repeat
 3:   $pr' = pr$
 4:   for CQ $q' \in pr'$ do
 5:     for (a) PI $\alpha \in \mathcal{T}$ do
 6:       if $\alpha$ is applicable to $g$ then
 7:         $pr = pr \cup \{q'[g/gr(g, \alpha)]\}$
 8:       end if
 9:     end for
10:     for (b) each pair of atoms $g_1, g_2$ in $q'$ do
11:       if $g_1$ and $g_2$ unify then
12:         $pr = pr \cup \{anon(reduce(q', g_1, g_2))\}$
13:       end if
14:     end for
15:   end for
16: until $pr' = pr$
17: return $pr$
```

Algorithm 2.2: function PERFECTREF$(q, \mathcal{T})$

the query $q(x) \leftarrow Student(x)$ has the following perfect reformulation $r = PerfectRef(q, \mathcal{T})$:

$$\begin{array}{l} r(x) \leftarrow Student(x); \\ r(x) \leftarrow PostGrad(x); \\ r(x) \leftarrow UnderGrad(x); \\ r(x) \leftarrow PartTime(x); \\ r(x) \leftarrow hasTutor(x, \_). \end{array}$$

where, for instance, the rewriting $r(x) \leftarrow hasTutor(x, \_)$ is generated from the rewriting asking for $PartTime$ students, since in the TBox we have that $\exists hasTutor \sqsubseteq PartTime$.

The certain answers to a UCQ over $\mathcal{O}$ are then computed as follows. First of all, the procedure needs to check the consistency of the KB. In fact, if $\mathcal{O}$ is unsatisfiable, then every tuple $\vec{c}$ belongs to the certain answers of any query q. Next, the procedure computes the perfect reformulation of q and evaluates it over $DB(\mathcal{A})$. The result of the evaluation is exactly the set of certain answers to q over $\mathcal{O}$. Hence:

Proposition 4. *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_{\mathcal{A}}$ ontology and let q be a UCQ over $\mathcal{O}$. Further, let the FOL query r = PerfectRef$(q, \mathcal{T})$ be the perfect reformulation of q w.r.t. $\mathcal{T}$ as in [13]. Then,*

$$\mathsf{cert}(q, \mathcal{O}) = \bigcup_{r_i \in r} \mathsf{ans}(r_i, DB(\mathcal{A})).$$

By FOL-rewritability, it follows that the data-complexity of the problem is AC_0, while the problem is in NP with respect to combined complexity. The interested reader can find the proofs of correctness and termination in [13].

CHAPTER 3

Explaining Query Answers

This chapter formalizes the problem of explaining answers to Union of Conjunctive Queries over *DL-Lite*$_{\mathcal{A}}$ ontologies. An explanation may be required both for the absence and for the presence of an answer in the result. Therefore, we first formalize the former problem as an abduction task. Then, the latter problem is characterized as a derivability problem in the context of *DL-Lite*$_{\mathcal{A}}$ ABox reasoning, that is, the search for proofs for the derivation of ABox assertions from a *DL-Lite*$_{\mathcal{A}}$ ontology.

3.1 Query Abduction Problem

Unfortunately, it happens that users are not always provided with the result they are expecting when querying an ontology. That is, a user may have the feeling that a certain tuple should be part of the result set of the query she posed, but the system is not able to return such an answer to the query. In this case, we say that the given tuple is a negative answer to the query over the ontology.

In contrast with standard database technology, in the ontology setting this problem is not easily solvable by looking at the structure of the data stored in the data-layer and, for example, slightly modifying the constraints of the query. The reason is that the reasoning involved for answering queries, provides an additional layer obfuscating the way the data is accessed by the system. For this reason, there is the need for algorithms and techniques aiming at computing explanations for negative answers to user queries.

In the following, we will first present the relevant concepts in Abductive Reasoning, which is a form of reasoning that is closely related to the problem of explaining negative answers. Then, we will provide a formalization to the problem of explaining negative answers as an abduction task.

Abductive Reasoning

Searching for an explanation for a negative query answer over an ontology is a form of abduction, a method of reasoning introduced by C.S. Peirce at the end of the 19th century. Abduction allows to infer an explanation A given an observation B, thus it is a form of backward reasoning. In the setting of query answering, the negative answer is the observation and the explanation A is the set of ABox assertions to be added to the ontology in order for the tuple to be returned in the results.

The general problem of abduction has been tackled from various perspectives, among the others we mention: logic-based abduction [16] and abduction by Set Covering [33]. Since we consider abduction in the context of Description Logics, in the following we will consider only the former.

In classical logic, abductive reasoning is a form of *non sequitur* argument, in which a conclusion B does not follow from the premises Γ ($\Gamma \not\models B$), even though B is assumed to hold true. The aim is to find a set of formulas A subset of a given set of hypotheses H, such that $\Gamma \cup A \models B$. There are several conditions that one can consider to narrow down the problem by considering only certain kinds of explanations. For example, given a knowledge base Γ and set of formulas A, B:

Consistency $\Gamma \cup A \not\models \bot$, that is, extending the theory with the explanation should preserve consistency,

Minimality A is *minimal* explanation for B, for a given minimality criteria.

Given an abduction problem, there are four basic reasoning problems one is interested to solve according to Eiter et al. [16], namely:

1. Does there exists a solution, i.e., does there exists A such that $\Gamma \cup A \models B$?
2. Does a formula ψ contribute to some acceptable solution to the problem (relevance), that is, exists an explanation A containing formula ψ?
3. Does a formula ψ occur in all acceptable solutions to the problem?
4. Is a set A of formulae a solution to the problem?

Surprisingly, abduction has not been studied widely in the context of description logics, even though, there have been researchers advocating the importance of such a line of research and the various problems have been formally characterized [7, 17]. Recently, the problem of ABox Abduction has been studied for $\mathcal{ALC}$ knowledge bases [23]. This is the problem of finding which additions need to be made to the ABox to force a set of ABox assertions to be logically entailed by the ontology. Klarman et al. provide two sound and complete calculi based on Semantic Tableaux and Resolution with Set of Support, respectively.

$$\begin{array}{rclrclrcl}
PostGrad & \sqsubseteq & Student, & Tutor & \sqsubseteq & Professor, & Advanced & \sqsubseteq & Course, \\
UnderGrad & \sqsubseteq & Student, & Professor & \sqsubseteq & \exists teaches, & \exists teaches & \sqsubseteq & Professor, \\
UnderGrad & \sqsubseteq & \neg PostGrad, & \exists hasTutor & \sqsubseteq & PartTime, & \exists teaches^- & \sqsubseteq & Course. \\
PartTime & \sqsubseteq & UnderGrad, & \exists hasTutor^- & \sqsubseteq & Tutor, & & &
\end{array}$$

$$teaches(craig, SWT), \quad hasTutor(peter, craig).$$

Figure 3.1: The complete characterization of the university domain as a *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O}_{uni}$.

Problem Formalization

We first note that the problem of ABox Abduction is not comparable with the problem of explaining negative answers to UCQs for two main reasons. First of all, the observations provided in the ABox Abduction problem are ABox assertions, which do not contain variables. The presence of variables forces one to consider different possible mappings of the variables while computing explanations, which introduces a new source of complexity. Secondly, Union of CQs contain a form of disjunction that has no counterpart in the ABox Abduction problem. This has led to a new formalization of the problem of finding an explanation to a negative query answer:

Definition 3 (Query Abduction Problem)**.** Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, q a UCQ, and $\vec{c}$ a tuple of constants. We call $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$ a *Query Abduction Problem* (QAP). A *solution* to $\mathcal{P}$ (or an *explanation for* $\mathcal{P}$) is any ABox $\mathcal{U}$ such that the ontology $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A} \cup \mathcal{U} \rangle$ is consistent and $\vec{c} \in \mathsf{cert}(q, \mathcal{O}')$. The set of all explanations to $\mathcal{P}$ is denoted $\mathsf{expl}(\mathcal{P})$.

If $\vec{c} \notin \mathsf{cert}(q, \mathcal{O})$, then we call $\vec{c}$ a *negative answer* to q over $\mathcal{O}$. Note that a query over the ontology can have a negative answer only if the ontology is satisfiable. On the other hand, if the ontology is unsatisfiable, then the QAP $\mathcal{P}$ does not admit any solution.

In the following, we will examine various restrictions to $\mathsf{expl}(\mathcal{P})$ to reduce redundancy in explanations. This is achieved by the introduction of a preference relation among explanations. This relation is reflexive and transitive, i.e., we have a pre-order among solutions.

Definition 4. Assume a QAP $\mathcal{P}$. Let $\preceq$ denote a pre-order on the set $\mathsf{expl}(\mathcal{P})$ of solutions. We write $\mathcal{U} \prec \mathcal{U}'$ if $\mathcal{U} \preceq \mathcal{U}'$ and $\mathcal{U}' \npreceq \mathcal{U}$. The preferred explanations $\mathsf{expl}_{\preceq}(\mathcal{P})$ of a QAP $\mathcal{P}$ under the pre-order $\preceq$ are defined as follows: $\mathsf{expl}_{\preceq}(\mathcal{P}) = \{\mathcal{U} \in \mathsf{expl}(\mathcal{P}) \mid$ there is no $\mathcal{U}' \in \mathsf{expl}(\mathcal{P})$ s.t. $\mathcal{U}' \prec \mathcal{U}\}$, i.e., $\mathsf{expl}_{\preceq}(\mathcal{P})$ contains all the $\preceq$*-explanations* that are *minimal* under $\preceq$.

Two different preference orders are considered in this thesis, namely the *subset-minimality order* denoted by $\subseteq$ and the *minimum explanation size order* denoted by $\leq$. The latter order is defined by $\mathcal{U} \leq \mathcal{U}'$ iff $|\mathcal{U}| \leq |\mathcal{U}'|$. Observe that $\mathsf{expl}_{\leq}(\mathcal{P}) \subseteq \mathsf{expl}_{\subseteq}(\mathcal{P})$.

Let us now provide an example of a Query Abduction Problem and its solutions by formally characterizing the problem described in Example 1.

Example 6. Let $\mathcal{O}_{uni}$ be the *DL-Lite*$_{\mathcal{A}}$ ontology from Figure 3.1, q be the following conjunctive query:

$$q(x) \leftarrow teaches(x, y), Advanced(y), hasTutor(z, x)$$

and $\vec{c} = \langle craig \rangle$. We define QAP $\mathcal{P}$ as $\langle \mathcal{O}_{uni}, q, \vec{c} \rangle$. Intuitively, $\langle craig \rangle$ is a negative answer to q over the ontology, since $craig$ satisfies all the conditions in the query but the SWT course taught by him is not asserted to be $Advanced$. We note that:

$$\{teaches(craig, TOC), Advanced(TOC), hasTutor(john, craig)\}$$

is an explanation for $\mathcal{P}$, $\{teaches(craig, ALG), Advanced(ALG)\}$ is a $\subseteq$-minimal solution, while $\{Advanced(SWT)\}$ is a $\leq$-minimal solution.

After having formalized the problem of explaining negative answers over *DL-Lite*$_{\mathcal{A}}$ ontologies, we turn our attention to the characterization of the opposite problem, namely explaining the presence of a tuple in the results.

3.2 Query Derivability Problem

Differently from query evaluation in relational databases, the presence of a tuple in the certain answers to a query over an ontology does not only depend on the data and on how the data is accessed by means of user queries. This follows from the same exact argument given to justify the need for explanations for negative answers. That is, the logical reasoning adopted to answer queries over ontologies alters the way the user query accesses the data. Also, this reasoning process may require the use of a large number of unobvious inference steps. Therefore, the presence of a tuple depends not only on the query and the data, but also on conceptual-level knowledge and reasoning. For these reasons, domain users require explanations on the motivations leading a certain tuple to be returned in the answers in terms of the conceptual level reasoning involved.

Next, we will first present the proof theory for ABox reasoning over *DL-Lite*$_{\mathcal{A}}$ ontologies, which is used to explain the deduction of ABox assertions from *DL-Lite*$_{\mathcal{A}}$ ontologies. Then, the problem of explaining positive query answers will be characterized as a derivation problem over this proof theory.

Explanations for ABox Reasoning

It is widely accepted that explanations correspond to *formal proofs* [29]. Then, solving the problem of explaining *DL-Lite*$_{\mathcal{A}}$ ABox Reasoning amounts at finding formal proofs justifying the inference of new facts about individuals [9].

We now briefly define the *DL-Lite*$_{\mathcal{A}}$ proof theory for ABox Reasoning as in [9]. A *sequent* S is an expression of the form $\langle \mathcal{T}, \mathcal{A} \rangle \vdash \alpha$, where $\langle \mathcal{T}, \mathcal{A} \rangle$ is a possibly empty *DL-Lite*$_{\mathcal{A}}$ ontology and α is either a TBox axiom or an ABox assertion of *DL-Lite*$_{\mathcal{A}}$. Roughly speaking, $\langle \mathcal{T}, \mathcal{A} \rangle \vdash \alpha$ means that α can be derived from the ontology $\langle \mathcal{T}, \mathcal{A} \rangle$. An *inference* is an expression of the form:

$$\frac{S_1}{S} \quad \text{or} \quad \frac{S_1 \quad S_2}{S}$$

where S, S_1 and S_2 are sequents and S is called the lower sequent while S_1, S_2 are called upper sequents. Such an inference rule means that if S_1 (resp. S_1 S_2) is asserted, then we can infer S. In the context of explaining ABox Reasoning over *DL-Lite*$_{\mathcal{A}}$ ontologies, we are interested in inferences that allow one to derive new facts about existing individuals. According to [9], these are the inference rules of interest:

$$\frac{\langle \mathcal{T}, \mathcal{A} \rangle \vdash B_1 \sqsubseteq B_2 \quad \langle \mathcal{T}, \mathcal{A} \rangle \vdash B_1(a)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash B_2(a)} \textit{ SubConcept}$$

$$\frac{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P_1 \sqsubseteq P_2 \quad \langle \mathcal{T}, \mathcal{A} \rangle \vdash P_1(a,b)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P_2(a,b)} \textit{ SubRole}$$

$$\frac{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P_1 \sqsubseteq P_2^- \quad \langle \mathcal{T}, \mathcal{A} \rangle \vdash P_1(a,b)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P_2(b,a)} \textit{ SubRole-Inv}$$

$$\frac{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P(a,b)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash \exists P(a)} \textit{ Dom-Intro}$$

$$\frac{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P(a,b)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash \exists P^-(b)} \textit{ Rng-Intro}$$

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, then we define $S(\mathcal{O})$ to be a function transforming $\mathcal{O}$ into a set of sequents, which works as follows:

- for each TBox axiom $\alpha \in \mathcal{T}$, $S(\mathcal{O})$ contains $\langle \mathcal{T}, \mathcal{A} \rangle \vdash \alpha$,
- for each ABox assertion $\beta \in \mathcal{A}$, $S(\mathcal{O})$ contains $\langle \mathcal{T}, \mathcal{A} \rangle \vdash \beta$.

Then, an ABox assertion β is said to be *derivable* from $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, written $\mathcal{O} \vdash \beta$, if there exists a tree of sequents D (called the *derivation*) satisfying the following conditions:

1. the top-most sequents are sequents from $S(\mathcal{O})$,
2. every sequent in D except the lowest one is one upper sequent of an inference whose lower sequent is also in D,
3. the lowest sequent in the proof is $\mathcal{O} \vdash \beta$.

We denote with $\mathsf{length}(D)$ the number of inference rules used in the derivation D. Since derivations for ABox assertions over ontologies solve an explanation problem, it is important to provide users with *minimal derivations*. A derivation D for sequent $S = \mathcal{O} \vdash \beta$ is called *minimal* if there is no other derivation D' for S, such that $\mathsf{length}(D') < \mathsf{length}(D)$.

One can prove that $\mathcal{O} \models \beta$ if and only if $\mathcal{O} \vdash \beta$ and we incite the interested reader to refer to [9] for further details.

Example 7. Let us now provide an example for the derivation of $\textit{Professor}(\textit{craig})$ from the university ontology $\mathcal{O}_{uni}$ from Figure 3.1. Let D be the derivation in Figure 3.2. It is easy to see that all top-most sequents of D are occurring in $S(\mathcal{O}_{uni})$. Also, D satisfies the two structural conditions on derivations by correctly applying inference rules throughout the tree. Then, it easily follows that D is a derivation for $\mathcal{O}_{uni} \vdash \textit{Professor}(\textit{craig})$.

Next, we will show how it is possible to solve the problem of explaining positive query answers by using the notion of derivation for ABox reasoning.

$$\dfrac{\mathcal{O}_{uni} \vdash \mathit{Tutor} \sqsubseteq \mathit{Professor} \qquad \dfrac{\mathcal{O}_{uni} \vdash \exists \mathit{hasTutor}^- \sqsubseteq \mathit{Tutor} \qquad \dfrac{\mathcal{O}_{uni} \vdash \mathit{hasTutor}(\mathit{peter}, \mathit{craig})}{\mathcal{O}_{uni} \vdash \exists \mathit{hasTutor}^-(\mathit{craig})}\,\textit{Rng-Intro}}{\mathcal{O}_{uni} \vdash \mathit{Tutor}(\mathit{craig})}\,\textit{SubConcept}}{\mathcal{O}_{uni} \vdash \mathit{Professor}(\mathit{craig})}\,\textit{SubConcept}$$

Figure 3.2: Derivation for $\mathcal{O}_{uni} \vdash \mathit{Professor}(\mathit{craig})$.

Problem Formalization

The intuition for solving the explanation problem for query answers over *DL-Lite*$_{\mathcal{A}}$ ontologies is as follows. Let $\vec{c}$ be an answer to UCQ $q(\vec{x})$ over the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. Since $\vec{c} \in \mathsf{cert}(q, \mathcal{O})$, it follows that there exists a match π for some $q_i \in q$ over the canonical model of $\mathcal{O}$ with $\pi(\vec{x}) = \vec{c}$. Now, consider the set of ABox assertions U resulting from instantiating all the atoms in $q_i(\vec{c})$ according to π. Then, the explanation problem reduces to finding a derivation for each assertion $U_i \in U$ w.r.t. $\mathcal{O}$, since the query answer can be seen as a set of ABox assertions. In fact, the various derivations provide sufficient evidence on the conceptual-level reasoning adopted for deriving the assertions and, in this way, a justification for the query answer.

Before actually formalizing the problem, let us first define the function called $\mathit{assertize}$, which given the body of a conjunctive query q and a match π for q over some interpretation, it returns the ABox resulting from instantiating all the variables in $at(q)$ according to π. Then, the problem of explaining positive query answers over *DL-Lite*$_{\mathcal{A}}$ ontologies can be defined as follows:

Definition 5 (Query Derivation Problem)**.** Let $\mathcal{O}$ be a *DL-Lite*$_{\mathcal{A}}$ ontology, q a UCQ and $\vec{c}$ a tuple of constants. We call $\mathcal{D} = \langle \mathcal{O}, q, \vec{c} \rangle$ a *Query Derivation Problem.* A *solution* to $\mathcal{D}$ (or an *explanation* for $\mathcal{D}$) is any tuple $\langle \pi, \langle D_1, \ldots, D_n \rangle \rangle$, such that:

- π is a match for some $q_i \in q(\vec{c})$ over the canonical model of $\mathcal{O}$,
- $\langle D_1, \ldots, D_n \rangle$ is a tuple of derivations, where each D_j is a derivation from $\mathcal{O}$ for ABox assertion $\phi_j(\vec{t}) \in \mathit{assertize}(at(q_i(\vec{c})), \pi)$.

If $\vec{c} \in \mathsf{cert}(q, \mathcal{O})$, then we call $\vec{c}$ a *positive answer* for q over $\mathcal{O}$. Clearly, if $\vec{c}$ is not a positive answer, then $\mathcal{D}$ does not have any solution. Finally, note that if the ontology in input is unsatisfiable, then $\mathcal{D}$ has no solutions as well, since there is no canonical model for $\mathcal{O}$.

QDPs formalize an explanation problem and, for this reason, we are interested in finding non-redudant solutions, i.e., *minimal solutions*. A solution $\langle \pi, \langle D_1, \ldots D_n \rangle \rangle$ to QDP $\mathcal{D}$ is called *minimal* if there is no other solution $\langle \pi', \langle D'_1, \ldots D'_m \rangle \rangle$ to $\mathcal{D}$, such that:

$$\mathsf{length}(D'_1) + \ldots \mathsf{length}(D'_m) < \mathsf{length}(D_1) + \ldots \mathsf{length}(D_n).$$

That is, minimal solutions require a small number of short derivations.

Example 8. We now formalize the explanation problem presented in Example 2. Let $\mathcal{O}_{uni}$ be the university ontology in Figure 3.1, $q(x)$ be the following CQ:

$$q(x) \leftarrow \mathit{Professor}(x).$$

and $\vec{c} = \langle craig \rangle$. We define the QAD $\mathcal{D}$ to be $\langle \mathcal{O}_{uni}, q, \vec{c} \rangle$, which intuitively asks to justify why $craig$ is considered to be a professor. It is easy to see that the only suitable match π for $q(\vec{c})$ is the empty match. By the chasing procedure described in Figure 2.1, it easily follows that $can(\mathcal{O}) \models \mathit{Professor}(craig)$ and hence π is a valid match for $q(\vec{c})$ over $can(\mathcal{O})$. Finally, we have that D in Example 7 is a derivation for $\mathit{Professor}(craig)$. Hence, it follows that $\langle \pi, \langle P \rangle \rangle$ is a solution to QAD $\mathcal{D}$.

In conclusion, this chapter formalizes the two problems related to the explanation of query answers. The next two chapters of this thesis deal with different aspects related to Query Abduction Problems and Query Derivation Problems. On the one hand, Chapter 4 introduces various reasoning problems over QAPs and it provides a deep analysis on their computational complexity in the context of *DL-Lite*$_{\mathcal{A}}$. On the other hand, Chapter 5 deals with the problem of computing solutions to QDPs by providing a formal algorithm solving the problem based on a high-level procedure proposed by Borgida et. al in [9].

CHAPTER 4

Complexity of Reasoning over Query Abduction Problems

In the previous chapter, we introduced the notion of Query Abduction Problem, a formalization of the explanation problem for negative answer over *DL-Lite*$_{\mathcal{A}}$ ontologies based on Abductive Reasoning. This chapter deals with the analysis of the computational complexity of reasoning over QAPs. For this we define four decision problems related to (minimal) explanations, which are parametric w.r.t. the chosen preference order $\preceq$, that is, given a QAP $\mathcal{P}$:

- $\preceq$-EXISTENCE: Does there exist a $\preceq$-explanation to $\mathcal{P}$?
- $\preceq$-NECESSITY: Does an assertion α occur in all $\preceq$-explanations to $\mathcal{P}$?
- $\preceq$-RELEVANCE: Does an assertion α occur in some $\preceq$-explanations to $\mathcal{P}$?
- $\preceq$-RECOGNITION: Is a set $\mathcal{U}$ of ABox assertions a $\preceq$-explanation to $\mathcal{P}$?

In the following, whenever there is no preference order in place (i.e., $\preceq$ is the identity) we omit to write $\preceq$ in front of the problem's names.

4.1 Outline of Complexity Results

In the next sections, the complexity of the introduced problems is inspected in the light of the different preference relations. Table 4.1 provides an overview of the complexity bounds of the various problems. Recall that the class Σ_2^{P} is a member of the Polynomial Hierarchy [30]. It contains all decision problems that can be solved in non-deterministic polynomial time using an NP oracle. Moreover, the class $\mathsf{P}_{\|}^{\mathsf{NP}}$ contains all the decision problems that can be computed in polynomial time with an NP oracle, where all the polynomially many oracle calls must be first prepared and then issued in parallel. Finally, the class DP [30] contains all those problems that, considered as languages, can be characterized as the intersection of a language in NP and one

Table 4.1: Summary of main complexity results (completeness)

$\preceq$	$\preceq$-EXISTENCE	$\preceq$-NECESSITY	$\preceq$-RELEVANCE	$\preceq$-RECOGNITION
none	PTIME (4.2)	PTIME (4.3)	PTIME (4.4)	NP (4.5)
$\leq$	PTIME	$\mathsf{P}^{\mathsf{NP}}_{\parallel}$ (4.3)	$\mathsf{P}^{\mathsf{NP}}_{\parallel}$ (4.4)	DP (4.5)
$\subseteq$	PTIME	PTIME (4.3)	Σ^{P}_2 (4.4)	DP (4.5)

in CONP. Note that: PTIME $\subseteq$ NP $\subseteq$ DP $\subseteq$ $\mathsf{P}^{\mathsf{NP}}_{\parallel}$ $\subseteq$ Σ^{P}_2 is believed to be a strict hierarchy of inclusions and here we make such an assumption.

The results can be explained as follows. In the next section, we will show how to reduce EXISTENCE to the PTIME-complete satisfiability problem for *DL-Lite*$_{\mathcal{A}}$ without the UNA [4]. This result can then be used to characterize the complexity of RELEVANCE, NECESSITY, and $\subseteq$-NECESSITY. $\leq$-RELEVANCE and $\leq$-NECESSITY are harder. The reason is that in order to solve these problems one has to compute first the minimum size of a solution and, then, inspect all the solutions of that size. Additionally, there is another increase in complexity when dealing with $\subseteq$-RELEVANCE. The intuition is that there are an exponential number of candidate solutions to examine and for each of them one has to check that none of its subsets is itself a solution, which requires a CONP computation. The results for $\preceq$-RECOGNITION can be explained as follows. The intuition for the NP bound for RECOGNITION is that one needs simply to check consistency and perform query answering to decide the problem. In case a preference order is in place, one has to check minimality as well and, hence, combine a NP computation with a CONP one. From which follows that the problem is in DP.

4.2 Complexity of $\preceq$-EXISTENCE

It is important to know whether a Query Abduction Problem has a solution. For instance, the QAP consisting of the query $q(x) \leftarrow \mathit{UnderGrad}(x), \mathit{PostGrad}(x)$ over the university mock ontology (see Figure 3.1) and the negative answer $\vec{c} = \langle \mathit{peter} \rangle$ does not have any solution. In fact, the two concepts mentioned in the query are disjoint and, hence, any solution would lead to the unsatisfiability of the ontology. Therefore, the non-existence of a solution signals an issue either at the level of the query or at the conceptual level.

The problem $\preceq$-EXISTENCE is defined as follows:

Definition 6 ($\preceq$-EXISTENCE)**.** Given a QAP $\mathcal{P}$, check whether $\mathsf{expl}_{\preceq}(\mathcal{P}) \neq \emptyset$.

Note first that the existence of an explanation for $\mathcal{P}$ implies the existence of an explanation under the $\subseteq$ and $\leq$ orderings. Thus, we only consider EXISTENCE. Our first task is to show that we can restrict ourselves on explanations built from the original signature of the input QAP.

Proposition 5. *If $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$ has a solution, then $\mathcal{P}$ has a solution $\mathcal{U}'$ with concepts and roles only from $\mathcal{O}$ and at most $max_{q_i \in q}|q_i|$ fresh ABox individuals.*

Proof. Assume an arbitrary solution $\mathcal{U}$ to $\mathcal{P}$. Given the consistency of $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A} \cup \mathcal{U} \rangle$, it follows that there exists a model $\mathcal{I}$ of $\mathcal{O}'$ under the UNA. W.l.o.g. we assume that $\Delta^{\mathcal{I}} = \mathbf{C}$ with $c^{\mathcal{I}} = c$ for each $c \in \mathbf{C}$. Additionally, the interpretation $\mathcal{I}$ admits a match π for some $q_i(\vec{x}) \in q$, such that $\pi(\vec{x}) = \vec{c}$. Let $\mathcal{U}' = \{A(o) \mid A(t) \in at(q_i) \text{ and } \pi(t) = o\} \cup \{R(o, o') \mid R(t, t') \in at(q_i) \text{ and } \pi(t) = o \text{ and } \pi(t') = o'\}$. Observe that $\mathcal{U}'$ has no more individuals than q_i has terms. It remains to see that $\mathcal{U}'$ is a solution. Clearly, the original match π witnesses also $\vec{c} \in \mathsf{ans}(q_i, \mathcal{A} \cup \mathcal{U}')$ in every model of $\mathcal{O}'' = \langle \mathcal{T}, \mathcal{A} \cup \mathcal{U}' \rangle$. It remains to see that $\mathcal{O}''$ is consistent. But this follows from the fact that $\mathcal{I}$ is a model of $\mathcal{O}'$ and that the atoms in $\mathcal{U}'$ hold in $\mathcal{I}$. □

The above restriction allows us to consider *canonical explanations*, i.e., explanations resulting from suitable instantiations of the bodies of CQs $q_i \in q$. Keeping in mind that CQs, seen as FOL formulae, are always satisfiable, an explanation does not exist only if the structure of the query is not compliant with the constraints expressed in the ontology. That is, for all the interpretations $\mathcal{J}$ of q with $\mathsf{ans}(q, \mathcal{J}) \neq \emptyset$, there is no model $\mathcal{I}$ of $\mathcal{O}$, such that $\mathcal{I} \cup \mathcal{J} \models \mathcal{O}$. To check whether a UCQ is compliant with the ontological constraints, a naïve method is to iteratively go through all the CQs in q and instantiate them in the ABox. If for none of the CQs we obtain a consistent ontology, then the query violates some of the constraints imposed at the conceptual level. We now show that the EXISTENCE problem is equivalent to the problem of checking consistency of *DL-Lite*$_{\mathcal{A}}$ ontologies without the Unique Name Assumption.

Proposition 6. *For DL-Lite*$_{\mathcal{A}}$ *ontologies,* EXISTENCE *is* PTIME*-complete.*

Proof. (MEMBERSHIP) Note that $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$, with q a UCQ, has a solution iff $\mathcal{P}_{q'} = \langle \mathcal{O}, q', \vec{c} \rangle$ has a solution for some $q' \in q$. Hence, it suffices to show the upper bound for CQs. To this end, we provide a reduction from EXISTENCE to consistency in *DL-Lite*$_{\mathcal{A}}$ without UNA, which in turn is PTIME-complete [4].

Assume a QAP $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$, where q is a CQ and $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. We argue that $\mathcal{P}$ has a solution iff $\mathcal{O}' = \langle \mathcal{T} \cup \mathcal{T}', \mathcal{A} \cup \mathcal{U}_q \cup \mathcal{A}' \rangle$ is consistent, where $\mathcal{O}'$ is an ontology obtained from $\mathcal{O}$ and $q(\vec{c})$ as follows. The ABox $\mathcal{U}_q$ is obtained from $at(q(\vec{c}))$ by replacing each variable x with a fresh individual name a_x. The ABox $\mathcal{A}'$ consists of assertions $A_o(o)$ for all constants o occurring either in $\mathcal{A}$ or in $q(\vec{c})$, where each A_o is a fresh concept name. The TBox $\mathcal{T}'$ consists of axioms $A_o \sqsubseteq \neg A_{o'}$ for all pairs $o \neq o'$ of constants occurring in $\mathcal{A}'$.

We now show that $\mathcal{P}$ has a solution iff $\mathcal{O}'$ is consistent.

"$\Rightarrow$" Assume that $\mathcal{P}$ has a solution $\mathcal{U}$. Then, due to consistency of $\mathcal{O}'' = (\mathcal{T}, \mathcal{A} \cup \mathcal{U})$, there is a model $\mathcal{I}$ of $\mathcal{O}''$ under the UNA. Additionally, $\mathcal{I}$ admits a match π for $q(\vec{c})$. Let $\mathcal{I}'$ be the extension of $\mathcal{I}$ that additionally interprets (i) constants in $\mathcal{U}_q$ as $a_x^{\mathcal{I}'} = \pi(x)$ for all variables x in q, and (ii) $A_o^{\mathcal{I}'} = \{o^{\mathcal{I}}\}$ for all freshly introduced A_o. It remains to show that $\mathcal{I}'$ is a model of $\mathcal{O}'$. Observe that since $\mathcal{I}$ is under the UNA, we have that $A_o^{\mathcal{I}'} \cap A_{o'}^{\mathcal{I}'} = \emptyset$ for all constant pairs $o \neq o'$. Thus $\mathcal{I}'$ satisfies all the disjointness axioms $A_o \sqsubseteq \neg A_{o'}$ in $\mathcal{T}'$. The assertions in $\mathcal{A}'$ are satisfied due to *(ii)*, while the assertions in $\mathcal{U}_q$ due to *(i)* above.

"$\Leftarrow$" Assume $\mathcal{O}'$ to be consistent. We want to show that QAP $\mathcal{P}$ admits a solution $\mathcal{U}$. Given the consistency of $\mathcal{O}'$, we can assume a model $\mathcal{I}'$ of the ontology without the UNA. W.l.o.g., we require $\Delta^{\mathcal{I}'} = \mathbf{C}$. By the validity of $\mathcal{I}'$ w.r.t. $\mathcal{O}'$, (a) it can be concluded that for all pairs of constants $o \neq o'$ in $\mathcal{A} \cup \mathbf{C}(q(\vec{c}))$, $A_o^{\mathcal{I}'} \cap A_{o'}^{\mathcal{I}'} = \emptyset$ and, hence, $o^{\mathcal{I}'} \neq o'^{\mathcal{I}'}$. Then, let the solution

$\mathcal{U}$ to $\mathcal{P}$ be defined as:

$$\mathcal{U} = \{A(c^{\mathcal{I}'}) \mid A(c) \in \mathcal{U}_q\} \cup \{P(c^{\mathcal{I}'}, c'^{\mathcal{I}'}) \mid P(c, c') \in \mathcal{U}_q\}$$

First, we show that $\mathcal{O}'' = \langle \mathcal{T}, \mathcal{A} \cup \mathcal{U} \rangle$ has a model $\mathcal{I}$ under the UNA. Let $\mathcal{I} = \langle \mathbf{C}, \cdot^{\mathcal{I}} \rangle$ be defined as follows:

- for all constants c in $\Delta^{\mathcal{I}}$, $c^{\mathcal{I}} = c$;
- for all concept names A in $\mathcal{O}''$, $A^{\mathcal{I}} = A^{\mathcal{I}'}$;
- for all role names P in $\mathcal{O}''$, $P^{\mathcal{I}} = P^{\mathcal{I}'}$.

Since $\mathcal{I}'$ satisfies $\langle \mathcal{T}, \mathcal{A} \rangle$ and by (a), it easily follows that the same holds for $\mathcal{I}$. Additionally, $\mathcal{I}$ satisfies the ABox $\mathcal{U}$ as well. The reason is that $\mathcal{I}' \models \mathcal{U}_q$ and ABox $\mathcal{U}$ is defined by interpreting $\mathcal{U}_q$ with respect to $\mathcal{I}'$. Then, we are left to prove the existence of a match π for $q(\vec{c})$ over all models $\mathcal{I}_{\mathcal{O}''}$ of $\mathcal{O}''$. Let π be $\pi(x) = (a_x)^{\mathcal{I}'}$, for all variables $x \in \mathbf{V}(q(\vec{c}))$. It is easy to see that, π maps $q(\vec{c})$ over $\mathcal{U}$, hence π is a valid match for any model of $\mathcal{O}''$.

(HARDNESS) Let us reduce consistency in *DL-Lite*$_{\mathcal{A}}$ without UNA to EXISTENCE. Given $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we create QAP $\mathcal{P} = \langle \mathcal{O}', q(), \langle\rangle \rangle$ as follows. We encode the ABox $\mathcal{A}$ in the CQ q by replacing each constant $a \in \mathcal{A}$ by a distinct variable name x_a in q. The ontology $\mathcal{O}'$ consists only of the TBox $\mathcal{T}$. We now show that $\mathcal{O}$ is satisfiable without the UNA iff $\mathcal{P}$ has a solution.

"$\Rightarrow$" Let $\mathcal{O}$ be a satisfiable ontology, then by the previous membership proof it follows that $\mathcal{P}$ has a solution.

"$\Leftarrow$" Assume that $\mathcal{P}$ has a solution $\mathcal{U}$, then we have that $\mathcal{O}' = \langle \mathcal{T}, \mathcal{U} \rangle$ is satisfiable under the UNA. Let $\mathcal{I}'$ be a model of $\mathcal{O}'$ and assume w.l.o.g. that $\Delta^{\mathcal{I}'} = \mathbf{C}$ with $c^{\mathcal{I}'} = c$ for all $c \in \mathbf{C}$. Since $\mathcal{U}$ is a solution, there exists a match π for $q(\vec{c})$ over $\mathcal{I}'$. We construct from $\mathcal{I}'$ an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ to $\mathcal{O}$ without the UNA, with $\Delta^{\mathcal{I}} = \Delta^{\mathcal{I}'}$:

- for all constants $a \in \mathcal{A}$, we let $a^{\mathcal{I}} = \pi(x_a)$. For all other constants c, we set $c^{\mathcal{I}} = c$;
- for all concept names A in $\mathcal{O}$, $A^{\mathcal{I}} = A^{\mathcal{I}'}$;
- for all role names P in $\mathcal{O}$, $P^{\mathcal{I}} = P^{\mathcal{I}'}$.

Since the interpretation $\mathcal{I}$ interprets $\mathcal{A}$ as $\mathcal{U}$, the satisfiability without the UNA of $\mathcal{O}$ easily follows from the satisfiability of $\langle \mathcal{T}, \mathcal{U} \rangle$ under the UNA. $\square$

4.3 Complexity of $\preceq$-NECESSITY

The existence of an explanation is most of the times not sufficiently informative to the user. In fact, given a negative answer to a user query, it is important to delineate the fundamental reasons that led to the absence of the expected result. That is, users would like to know which assertions occur in all the possible solutions to a QAP $\mathcal{P}$:

Definition 7 ($\preceq$-NECESSITY)**.** Given a QAP $\mathcal{P}$ and an ABox assertion α, decide whether for all $\mathcal{U} \in \mathsf{expl}_{\preceq}(\mathcal{P})$ it is the case that $\alpha \in \mathcal{U}$.

For instance, in the example introduced in Section 3.1, the assertion $Advanced(SWT)$ is $\leq$-necessary. Indeed, there is only one explanation of size 1, since all other solutions require to introduce one new advanced course taught by craig. Hence, the given assertion occurs in all the $\leq$-minimal solutions. Please also note that in case the QAP $\mathcal{P}$ admits no solution, every assertion α is necessary to $\mathcal{P}$. Let us now detail the complexity of this problem for the various minimality criteria.

Proposition 7. *For DL-Lite$_\mathcal{A}$ ontologies,* NECESSITY *and* $\subseteq$-NECESSITY *are* PTIME*-complete.*

Proof. We assume a QAP $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$ with $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, an assertion $\phi(\vec{t})$ and consider NECESSITY first. W.l.o.g, we restrict the study only to the case in which q is a CQ, the more general result for UCQ follows as corollary.

(MEMBERSHIP) We now provide a PTIME algorithm that decides NECESSITY for an ABox assertion $\phi(\vec{t})$ w.r.t. a QAP $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$. First, the algorithm checks whether $\mathcal{O}$ is satisfiable and that $\mathcal{O} \models \phi(\vec{t})$. If both the two conditions are met, then one can easily conclude that $\phi(\vec{t})$ is not necessary for $\mathcal{P}$. Otherwise, we construct a new ontology $\mathcal{O}' = \langle \mathcal{T}', \mathcal{A}' \rangle$ defined by setting $\mathcal{A}' = \mathcal{A} \cup \{\bar{\phi}(\vec{t})\}$ and $\mathcal{T}' = \mathcal{T} \cup \{\bar{\phi} \sqsubseteq \neg\phi\}$, where $\bar{\phi}$ is a fresh concept/role name. By construction, the models of $\mathcal{O}'$ are only those models of $\mathcal{O}$ in which $\phi(\vec{t})$ does not hold. Then, the algorithm considers $\mathcal{P}' = \langle \mathcal{O}', q, \vec{c} \rangle$ and checks that this new QAP does not have any solution. Assume that $\mathcal{P}'$ has a solution $\mathcal{U}$. Then, it is not difficult to see that $\mathcal{U}$ does not contain $\phi(\vec{t})$ and that also $\mathcal{U}$ is a solution to $\mathcal{P}$, which would lead $\phi(\vec{t})$ not to be necessary for $\mathcal{P}$.

(HARDNESS) The lower-bound can be proved through a logspace reduction from EXISTENCE to non-NECESSITY, that is, deciding whether there exists a solution to a QAP that does not contain the given assertion. The hardness for NECESSITY follows as corollary.

We build $\langle \mathcal{P}, \alpha \rangle$ such that $\mathcal{P}$ has a solution iff $\langle \mathcal{P}, \alpha \rangle$ is a negative instance to NECESSITY. Let $\alpha = A(o)$, for some fresh concept A and constant o not occurring in $\mathcal{P}$.

"$\Rightarrow$" Now, assume $\mathcal{P}$ has a solution $\mathcal{U}$. By Proposition 5, we know that there exists a solution $\mathcal{U}'$ to $\mathcal{P}$ containing only concept and role names from $\mathcal{O}$. Hence, $A(o) \notin \mathcal{U}'$, since concept name A is not in the ontology. Therefore, one can conclude that $A(o)$ is not a necessary assertion to $\mathcal{P}$. The other direction of the proof is straightforward. □

For $\subseteq$-NECESSITY, observe that $\phi(\vec{t})$ occurs in all explanation for $\mathcal{P}$ iff $\phi(\vec{t})$ occurs in all $\subseteq$-minimal explanations for $\mathcal{P}$. Thus, $\subseteq$-NECESSITY can be decided in polynomial time using our algorithm for NECESSITY.

□

Let us now consider the complexity of necessity under the *minimum size* ($\leq$) preference order.

Theorem 8. *For DL-Lite$_\mathcal{A}$ ontologies,* $\leq$-NECESSITY *is* $\mathsf{P}_{\parallel}^{\mathsf{NP}}$*-Complete.*

Proof. (MEMBERSHIP) Let's assume a QAP $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$ and an assertion α. By the use of canonical explanations, we know that if an explanation for $\mathcal{P}$ exists, then the size of a $\leq$-solution to $\mathcal{P}$ is bounded by the size m of the largest CQ in q. Observe that $\langle \mathcal{P}, \alpha \rangle$ is a negative instance of $\leq$-NECESSITY iff there is an $0 \leq i \leq m$ such that *(a)* $\mathcal{P}$ has an explanation $\mathcal{U}$ with $|\mathcal{U}| = i$ and $\alpha \notin \mathcal{U}$, and *(b)* $\mathcal{U}$ is $\leq$-minimal. Thus, we use an auxiliary problem SIZE-OUT, which is to decide given a tuple $\langle \mathcal{P}', \alpha', n' \rangle$, where $\mathcal{P}'$ is a QAP, α' is an assertion, and n' is an integer, whether there exists an explanation $\mathcal{U}'$ for $\mathcal{P}'$ such that $|\mathcal{U}'| = n'$ and $\alpha' \notin \mathcal{U}'$. Furthermore, the problem NO-SMALLER is to decide given a tuple $\langle \mathcal{P}', n' \rangle$ of a QAP and an integer whether there is no explanation $\mathcal{U}'$ for $\mathcal{P}'$ such that $|\mathcal{U}'| < n'$. Observe that SIZE-OUT is in NP, while NO-SMALLER is in CONP. Take the tuple $S = \langle A_0, B_0, \ldots, A_m, B_m \rangle$, where *(a)* $A_i = \langle \mathcal{P}, \alpha, i \rangle$, for all $0 \leq i \leq m$, and *(b)* $B_i = \langle \mathcal{P}, i \rangle$, for all $0 \leq i \leq m$. Due to the above observation, α occurs in all $\leq$-minimal explanations $\mathcal{U}$ for $\mathcal{P}$ iff for all $0 \leq i \leq m$, one of the following holds: *(i)* A_i is a negative instance of SIZE-OUT, or *(ii)* B_i is a negative instance of NO-SMALLER. Note that S can be built in polynomial time in the size of the input, while the positivity of the instances in S can be decided by making $2m$ parallel calls to an NP oracle. Thus we obtain membership in $\mathsf{P}_{||}^{\mathsf{NP}}$.

(HARDNESS) The hardness is proved by reducing a suitable hard problem to $\leq$-DISPENSABILITY, the complement of $\leq$-NECESSITY. That is, given a QAP $\mathcal{P}$ and an assertion α, decide whether there exists a $\leq$-explanation $\mathcal{U}$ for $\mathcal{P}$ with $\alpha \notin \mathcal{U}$. The hardness of $\leq$-NECESSITY follows as corollary from the following result.

Lemma 9. *For DL-Lite$_{\mathcal{A}}$ ontologies,* $\leq$-DISPENSABILITY *is* $\mathsf{P}_{||}^{\mathsf{NP}}$*-hard.*

Proof. We provide a reduction from EVEN-CQEVAL. Let X be a sequence of m pairs (q_i, D_i), where q_i is a Boolean CQ and D_i is a relational instance (under the logic-programming perspective [1]). A pair (q_i, D_i) is said to be true if $\mathsf{ans}(q_i, D_i) \neq \emptyset$. Otherwise, it said to be false. X is a yes instance of the problem if there is an even number of true pairs in X. This problem can be proved to be $\mathsf{P}_{||}^{\mathsf{NP}}$-hard[1]. Now, the aim is to create a QAP $\mathcal{P}$ and an ABox assertion $\phi(\vec{t})$ such that, X is a yes-instance if and only if $\phi(\vec{t})$ is $\leq$-dispensable to $\mathcal{P}$. W.l.o.g. ([12]), we can assume X to have an odd number of pairs in input, such that (q_1, D_1) and (q_2, D_2) are true, whereas (q_m, D_m) is false. Moreover, we assume that (q_{i+1}, D_{i+1}) is false only if (q_i, D_i) is false. Finally, we suppose that predicates are at most binary and that all the pairs are defined on different signatures.

Given the assumptions made, the problem results to be equivalent to finding the first index i in the sequence with (q_i, D_i) false. It just remains to check whether i is odd, in which case X contains an even number of true pairs, specifically the first $i - 1$ pairs.

We create a QAP $\mathcal{P} = (\mathcal{O}, q(), \langle\rangle)$ as follows. Queries q_i in X are encoded into a single tree-shaped query q by adding a new variable x and a role J that connects all the terms occurring in q_i to x. Let $\mathsf{term}(q_i)$ denote the set of terms in q_i. Then, $q()$ is defined as follows:

$$q() \leftarrow \bigwedge_{1 \leq i \leq m} [q_i, \bigwedge_{t \in \mathsf{term}(q_i)} J(x,t), R(t,x_i), OK(x_i), B(t)].$$

where x is the *join variable* over which the body of the m different queries are J-connected. Moreover, every term t occurring in some q_i is R-connected to a fresh variable x_i, which is required to be a member of fresh concept OK. Finally, each term in q_i is required to be a member of fresh concept B.

[1]From [12] we know that given boolean formulas $\{\psi_1, \ldots, \psi_m\}$, deciding whether there are an even number of valid formulas is hard for $\mathsf{P}_{||}^{\mathsf{NP}}$. Since we know that Satisfiability reduces to CQ-Evaluation, the claim follows.

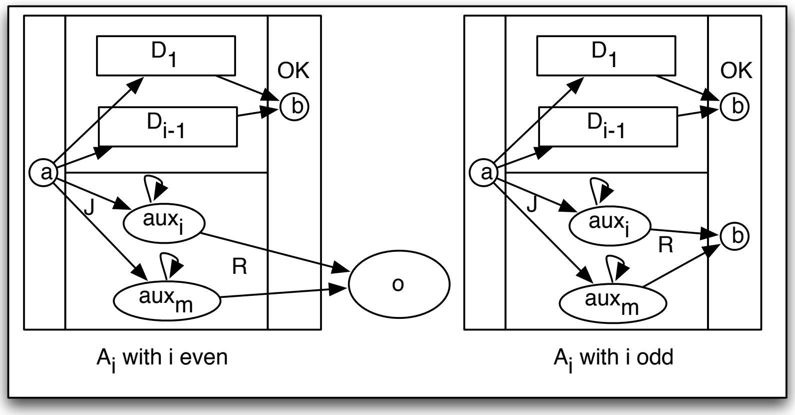


Figure 4.1: A representation of component A_i of $\mathcal{A}$. If i is even, then backdoors are R-connected to aux. Otherwise, they are R-connected to a constant b_i

Let us now consider the ontology $\mathcal{O}$. ABox $\mathcal{A}$ consists of $\{A_3, \ldots, A_m\}$ different components, where each component A_i represents a particular configuration of X. That is, each A_i models a different possible position of the first false pair in X. More specifically, component A_i represents the case in which (q_i, D_i) is the first false pair. Hence, A_i contains database instances $\{D_1, \ldots, D_{i-1}\}$ and it replaces the other database instances by $m - i + 1$ different backdoors over which queries can vacuously match (see Figure 4.1). By assumption, it is known that the first two tuples in X are always true. For this reason, we consider only $\{A_3, \ldots, A_m\}$ different configurations.

Let a_i, b_i, b'_i be fresh constants, then component A_i is defined as follows:

- for all database instances D_j $(1 \leq j < i)$ in X, we add the following assertions in A_i:

$$D_j, \bigwedge_{c \in \mathsf{C}(D_j)} B(c), J(a_i, c), R(c, b_i), OK(b_i)$$

  where $\mathsf{C}(D_j)$ denotes the set of constants occurring in D_j. That is, we encode D_j in A_i and we further require that for every constant c occurring in D_j the following holds: *a*) c is asserted to be a member of B, *b*) c is J-connected to join constant a_i and *c*) c is R-connected to an instance asserted to be of type OK.

- for all D_k with $i \leq k \leq m$, for all concept names A and role names Q occurring in q_k, we add:

$$J(a_i, aux^i_k), A(aux^i_k), Q(aux^i_k, aux^i_k)$$

  for some fresh constant aux^i_k. Additionally, if i is even, we add $R(aux^i_k, o)$ to A_i, otherwise, we add $R(aux^i_k, b'_i)$. That is, we encode $m - i + 1$ different backdoors over which queries $\{q_i, \ldots, q_m\}$ can potentially match. Also, backdoors are J-connected to join individual a_i and R-connected to either o or b'_i depending on the value of i.

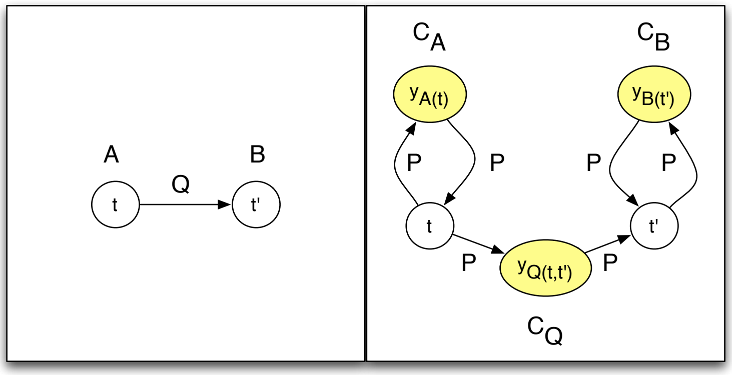


Figure 4.2: On the left you can see the original query $q(x)$, on the right the resulting query $tr_q(q)$ after the application of the transformation function.

By definition, the backdoors do not contain assertions over the B predicate, hence q cannot match directly over $\mathcal{A}$. Therefore, matching q over A_i requires an explanation containing assertions $B(aux_j^i)$, for $i \leq j \leq m$. Moreover, if i is even then every $\leq$-explanation must contain $OK(o)$, since auxiliary objects are R-connected to that instance, which is not a member of OK as required by $q()$. Otherwise, if i is odd, then auxiliary constants are R-related to b'_i, which is as well not a member of OK. Finally, the assertion we declare to be $\leq$-dispensable is $OK(o)$.

The intuition behind this reduction is as follows. If the minimal solution is found by matching q over A_i with i odd, then the $\leq$-explanation has to contain $OK(b'_i)$ and no assertion on o. If this is the case, then X is a positive instance of the problem, since the first $i-1$ pairs are true. Otherwise, if the minimal update is found by matching q over A_i with i even, then the $\leq$-solution asserts o to be a member of OK and the problem in input is false, since there are an odd number of true pairs in X.

Penalizing Database Updates However, for this reduction to work, explanations that modify database instances encoded in the ABox have to be penalized. The reason is that there are cases in which X is not a positive instance of the problem, but $OK(o)$ is $\leq$-dispensable. In these cases, one can argue that the solution has been found by altering a database instance D_j encoded in the ABox. That is, the query maps over A_i with i odd, but the $\leq$-explanation modifies the structure of some database instance D_j in A_i. In the following, we rule out this case by penalizing database updates through the introduction of two transformation functions tr_q and tr_{db} over queries and database instances respectively, and by using negative inclusions at the TBox level. The functions tr_q and tr_{db} work in a similar way. Let S be either the body of a query or an ABox and, interpret S as a graph. Then, the basic idea is to extend the structure of S by requiring that concept and role labels occur only on new vertexes in the graph. That is, for every edge (t, t') labeled with Q, the function introduces a new node $c_{Q(t,t')}$ that is suitably connected to t, t' using fresh edge P. Then, $c_{Q(t,t')}$ is labeled with fresh name C_Q, which intuitively says that there is an edge between t, t' of type Q. Hence, this transformation extends the graph by introducing new vertexes and labels, but it maintains its structure. The same kind of transformation is employed for concept labels on vertexes (see Figure 4.2). The two functions are defined as follows:

$tr_q(q_i)$: Given an atom $A(t)$ (or $P(t,t')$), we denote with $y_{A(t)}$ (or $y_{Q(t,t')}$) a fresh variable.

- if $A(t) \in at(q_i)$, then $\{P(t, y_{A(t)}), C_A(y_{A(t)}), P(y_{A(t)}, t)\} \subseteq tr(q_i)$;
- if $Q(t,t') \in at(q_i)$, then $\{P(t, y_{Q(t,t')}), C_Q(y_{Q(t,t')}), P(y_{Q(t,t')}, t')\} \subseteq tr(q_i)$;
- nothing else is in $tr_q(q_i)$.

$tr_{db}(D)$: Given an assertion $A(t)$ (or $P(t,t')$), we denote with $c_{A(t)}$ (or $c_{Q(t,t')}$) a fresh constant.

- if $A(t) \in D$, then $\{P(t, c_{A(t)}), C_A(c_{A(t)}), P(c_{A(t)}, t)\} \subseteq tr(D)$;
- if $Q(t,t') \in D$, then $\{P(t, c_{Q(t,t')}), C_Q(c_{Q(t,t')}), P(c_{Q(t,t')}, t')\} \subseteq tr(D)$;
- nothing else is in $tr_{db}(D)$.

Please note that it can be easily proven that (q_i, D_i) is true if and only if $(tr_q(q_i), tr_{db}(D_i))$ is true.

Finally, the TBox $\mathcal{T}$ consists of negative inclusions only. For every pair of distinct concept/role names N, N' occurring in the sequence X, $\mathcal{T}$ contains the following disjointness axiom: $C_N \sqsubseteq \neg C_{N'}$. These axioms ensure that no individual can get more than one concept or role label at the same time. From this it follows that if an explanation introduces a new assertion $A(t)$ on an instance D_j in the ABox, the same explanation over the transformed instance would require assertions $\{P(t, c_{A(t)}), C_A(c_{A(t)}), P(c_{A(t)}, t)\}$. That is, performing a single update over a transformed instances is three times more expensive than it is on the standard instance.

Consider $\mathcal{P}' = \langle \mathcal{O}', q', \langle\rangle\rangle$ with $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$. Intuitively, q' is constructed by replacing each q_i occurring in the definition of q with $tr(q_i)$. Similarly, we define $\mathcal{A}'$ by replacing each occurrence of D_j in $\mathcal{A}$ with $tr(D_j)$.

Now, we prove that X is a yes-instance to EVEN-CQEVAL if and only if $OK(o)$ is $\leq$-dispensable to $\mathcal{P}'$.

"$\Rightarrow$" Assume X to be a positive instance, then there exists $i \in \{3, \ldots, m\}$, such that (q_i, D_i) is false and i is odd. Let us assume that q' matches over component A_i. Then, it is easy to see that a solution has to contain exactly the following assertions: $\mathcal{U} = \{B(aux_i^i), \ldots, B(aux_m^i)\} \cup \{OK(b_i')\}$ with $|\mathcal{U}| = (m - i + 2)$, since by assumption query q_k matches over D_k, for $k \in \{1, \ldots, i-1\}$. $\mathcal{U}$ does not contain $OK(o)$, hence it is left to prove that $\mathcal{U}$ is a $\leq$-explanation. By construction, any smaller solution than $\mathcal{U}$ does not allow q' to match over A_i.

Let us assume by contradiction, that for $l < i$, matching q' over A_l leads to a smaller update $\mathcal{U}'$. By definition of A_l and $q', \mathcal{U}'$ has to contain $m - l + 1$ assertions on B and a further assertion on OK. Given that $l < i$ then $(m - l + 2) > (m - i + 2)$ and, hence, $|\mathcal{U}'| > |\mathcal{U}|$.

Similarly, it is proved by contradiction that for $u > i$, matching q' over A_u does not lead to a smaller solution $\mathcal{U}''$. By definition of A_u and $q', \mathcal{U}''$ has to contain $m - u + 1$ different assertions on B. Moreover, since by assumption q_k does not match over D_k, for $k \in \{i, \ldots, u-1\}$, then the explanation has to add $p \geq u - i$ different assertions on those database instances in order for the queries to match. But by the use of the transformation function, we know that any alteration to the database instance requires at least 3 assertions in the solution, from which follows that $p > u - i$. Hence $(m - u + 2 + p) > (m - i + 2)$, which contradicts our assumption.

Therefore, $\mathcal{U}$ is a $\leq$-explanation with $OK(o) \notin \mathcal{U}$.

"$\Leftarrow$" Assume $\mathcal{U}$ to be a $\leq$-explanation, such that $OK(o)$ is not in the solution. Then, by construction we know that $\mathcal{U}$ must have matched on one of the A_i components and the solution must have the following form: $\mathcal{U} = \{B(aux_i^i), \ldots, B(aux_m^i)\} \cup \{OK(b_i')\}$. Moreover, by the presence of $OK(b_i')$, one can conclude that i is odd, Now, using a similar argument as above, it is possible to prove that if $\mathcal{U}$ is minimal then the i-th pair is the first false one, leading X to be a positive instance.

Hence, we conclude that $\leq$-DISPENSABILITY is hard for $\mathsf{P}_{\|}^{\mathsf{NP}}$. $\square$

Given that $\mathsf{P}_{\|}^{\mathsf{NP}}$ is closed under complementation, it follows that $\leq$-NECESSITY is $\mathsf{P}_{\|}^{\mathsf{NP}}$-complete.

$\square$

4.4 Complexity of $\preceq$-RELEVANCE

A domain user faced with a negative answer to a query may ask herself whether, the absence of a certain ABox assertion α in the ontology is related with the lack of the tuple in the results. That is, she would like to know whether α occurs in some explanation to QAP $\mathcal{P}$:

Definition 8 ($\preceq$-RELEVANCE)**.** Given a QAP $\mathcal{P}$ and an assertion α, decide whether there exists $\mathcal{U} \in \mathsf{expl}_{\preceq}(\mathcal{P})$ such that $\alpha \in \mathcal{U}$.

In our running example introduced in Section 3.2, assertion $Advanced(MATH)$ is relevant because $\{teaches(craig, MATH), Advanced(MATH), hasTutor(john, craig)\}$ is an explanation. However, it is not $\leq$-relevant, since the only $\leq$-solution is $Advanced(SWT)$. Now, as done in the previous section, the complexity of the problem under the various orderings is analyzed.

Proposition 10. *For DL-Lite$_{\mathcal{A}}$ ontologies,* RELEVANCE *is* PTIME*-complete.*

Proof. We assume a QAP $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$ with $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and an assertion $\phi(\vec{t})$. As done before, we further require q to be a CQ, the general result for Union of CQs follows directly.

(MEMBERSHIP) We now provide a logspace reduction from RELEVANCE to EXISTENCE. We construct $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$, where $\mathcal{A}' = \mathcal{A} \cup \{\phi(\vec{t})\}$. Now, we show that $\mathcal{P}$ has an explanation $\mathcal{U}$ with $\phi(\vec{t}) \in \mathcal{U}$ iff there exists an explanation to $\mathcal{P}' = (\mathcal{O}', q, \vec{c})$.

"$\Rightarrow$" Assume $\phi(\vec{t})$ to be relevant for $\mathcal{P}$. Then, there exists a solution $\mathcal{U}$ to $\mathcal{P}$ with $\phi(\vec{t}) \in \mathcal{U}$. That is, $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A} \cup \mathcal{U} \rangle$ is satisfiable and $\vec{c} \in \mathsf{cert}(q, \mathcal{O}')$. The claim easily follows.

"$\Leftarrow$" Let $\mathcal{P}'$ have a solution $\mathcal{U}'$. Then, there exists a model $\mathcal{I}$ of $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \cup \mathcal{U}' \rangle$ that admits a match π for $q(\vec{c})$. W.l.o.g., we can suppose $\Delta^{\mathcal{I}} = \mathbf{C}$, with $c^{\mathcal{I}} = c$, for all $c \in \mathbf{C}$. Now, let

$$\mathcal{U}'' = \{A(\pi(t)) \mid A(t) \in at(q(\vec{c}))\} \cup \{P(\pi(t), \pi(t')) \mid P(t, t') \in at(q(\vec{c}))\}.$$

Clearly, $\mathcal{I} \models \phi(\vec{t})$ and, therefore, $\mathcal{U} = \mathcal{U}'' \cup \{\phi(\vec{t})\}$ is still a solution to $\mathcal{P}'$. Now, we show that $\mathcal{U}$ is a solution to $\mathcal{P}$. For this, we first argue that $\mathcal{O}'' = \langle \mathcal{T}, \mathcal{A} \cup \mathcal{U} \rangle$ is satisfiable. But this is not difficult to see, since $\mathcal{I}$ is also a model of $\mathcal{O}'$. It remains to see that $\vec{c} \in \mathsf{cert}(q, \mathcal{O}'')$. For this note that every model of $\mathcal{O}''$ has to satisfy $\mathcal{U}$ and that π matches $q(\vec{c})$ exactly on $\mathcal{U}'' \subset \mathcal{U}$. Therefore, $\phi(\vec{t})$ is relevant to $\mathcal{P}$.

(HARDNESS) Hardness can again be proved with a logspace reduction from EXISTENCE. We construct $\mathcal{P}' = \langle \mathcal{O}, q', \vec{c} \rangle$ as follows. Let $q' = q(\vec{x}) \cup q''(\vec{x})$, where q'' is:

$$q'' \leftarrow at(q), A(o).$$

That is, q'' extends q by adding a new atom over a fresh concept and a fresh individual.

We claim that: *$\mathcal{P}$ has a solution iff $A(o)$ is relevant to $\mathcal{P}'$.*

"$\Rightarrow$" Assume that $\mathcal{P}$ has a solution $\mathcal{U}$. This means that there exists a model $\mathcal{I}$ of $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{U} \rangle$. W.lo.g., we can assume $\Delta^{\mathcal{I}} = \mathbf{C}$ with $c^{\mathcal{I}} = c$, for all $c \in \mathbf{C}$. Moreover, we know that there exists a match π for $q(\vec{c})$ over $\mathcal{I}$. Now, let $\mathcal{I}'$ be the extension to $\mathcal{I}$ mapping $o^{\mathcal{I}'}$ to o and $A^{\mathcal{I}'} = \{o^{\mathcal{I}'}\}$. Then, by the freshness of A and o and by $\mathcal{I} \models \langle \mathcal{T}, \mathcal{A}, \cup \mathcal{U} \rangle$, it follows that $\mathcal{I}'$ is a model of $\mathcal{O}'' = \langle \mathcal{T}, \mathcal{A}, \cup \mathcal{U} \cup \{A(o)\} \rangle$. Additionally, all models $\mathcal{I}_{\mathcal{O}''}$ of $\mathcal{O}''$ have to satisfy $\mathcal{U} \cup \{A(o)\}$, hence π witnesses $\vec{c} \in \mathsf{cert}(q, \mathcal{O}'')$. Therefore, $\mathcal{U} \cup \{A(o)\}$ is a solution to $\mathcal{P}'$, from which follows that $A(o)$ is relevant to $\mathcal{P}'$.

The other direction of the proof is straightforward. □

This result is not surprising, in fact, when no minimality restriction is in place RELEVANCE is equivalent to check the existence of a solution $\mathcal{U}$ for QAP $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$ and to ensure that $\mathcal{O} \cup \mathcal{U} \cup \{\phi(\vec{t})\}$ is consistent. Let us now tackle the problem of $\subseteq$-RELEVANCE to a QAP.

Theorem 11. *For DL-Lite$_{\mathcal{A}}$ ontologies, $\subseteq$-RELEVANCE is Σ_2^{P}-Complete.*

Proof. (MEMBERSHIP) The membership in Σ_2^{P} is clear from Algorithm 4.1, which works as follows. An explanation $\mathcal{U}$ containing $\phi(\vec{t})$ is non-deterministically computed by guessing an instantiation of a subquery in *PerfectRef*$(q(\vec{c}), \mathcal{T})$, where $Anon$ is a set of fresh ABox individuals (see Proposition 5). Let HAS-SUBEXPL solve the problem of deciding whether a solution $\mathcal{U}$ has a subset which is itself an explanation. The problem can be easily proved to be in NP. Then, the algorithm checks the complement of HAS-SUBEXPL in order to assure that none of the subsets of $\mathcal{U}$ is itself an explanation, from which it follows that $\phi(\vec{t})$ is $\subseteq$-relevant. Checking

```
1 INPUT: QAP $\mathcal{P} = \langle q, \mathcal{O}, \vec{c} \rangle$ and ABox assertion $\phi(\vec{t})$
2 OUTPUT: yes iff $\phi(\vec{t})$ is relevant to $\mathcal{P}$
    1: Guess $q_i \in \{q_1, \ldots, q_n\} = q$
    2: Guess the derivation of one rewriting $r(\vec{c})$ in $PerfectRef(q_i(\vec{c}), \mathcal{T})$
    3: Guess a set of atoms $U \subseteq at(r)$
    4: Guess a mapping $\pi$ from $\mathbf{V}(q)$ to constants in $DB(\mathcal{A})$ and $Anon$
    5: Check that $(\mathcal{T}, \mathcal{A} \cup \mathcal{U}) \not\models \bot$, where $\mathcal{U}$ is the instantiation of $U$ through $\pi$.
    6: Check that $\phi(\vec{t}) \in \mathcal{U}$ and $\pi$ is a match for $r(\vec{c})$ over $DB(\mathcal{A} \cup \mathcal{U})$
    7: Check that HAS-SUBEXPL $(\mathcal{P}, \mathcal{U}) = no$.
```

Algorithm 4.1: Algorithm solving $\subseteq$-RELEVANCE

the complement of HAS-SUBEXPL requires the power of a coNP machine. For this reason, the algorithm is solvable in non-deterministic polynomial time by a TM with an NP oracle.

(HARDNESS) We prove Σ_2^{P}-hardness by a reduction from the Σ_2^{P}-complete problem co-CERT3COL [34] (see also [8]).

An instance of co-CERT3COL is given by a graph $G = (V, E)$ with vertices $V = \{0, \ldots, n-1\}$, $n \geq 1$, such that every edge is labeled with a disjunction of two literals over the Boolean variables $\{p_{(i,j)} \mid 0 \leq i, j < n\}$. G is a positive instance if there is a truth value assignment t to the Boolean variables such that the graph $t(G)$ obtained from G by including only those edges whose label evaluates to true under t is not 3-colorable.

Assume an instance G of co-CERT3COL. We show how to build in polynomial time a QAP $\mathcal{P}_G = \langle (\mathcal{T}_G, \mathcal{A}_G), q_G, \vec{c}_G \rangle$ and an ABox assertion α_G such that: G is a positive instance of CO-CERT3COL iff α_G is $\subseteq$-relevant for $\mathcal{P}_G$. We use an empty TBox and a Boolean query, thus $\mathcal{T}_G = \emptyset$ and $\vec{c}_G = \langle\rangle$. The query q_G is a UCQ $q_G = q_{e_1} \cup \cdots \cup q_{e_k} \cup q'$, where $\{e_1, \ldots, e_k\} = E$, each q_{e_i} is an atomic query $q_{e_i}() \leftarrow W_{e_i}(x, y)$, and q' is defined as follows. Assume $\mathcal{B} = \{t, f\}$ to be the set of truth values. The query q' has the following atoms for each edge $e = (i, j)$ in E:

(a) $B(x_i)$, $R_e(x_i, y_e)$, $R_e(y_e, x_j)$, $B(x_j)$, and

(b) $P(y_e, z_{p_i})$, $A_{p_i}(z_{p_i})$, $W_{p_i}(z_{p_i}, z'_{p_i})$, where $p_i \in \{p_1, p_2\}$ and p_1, p_2 are the first and the second proposition in the labeling of e, respectively.

The query q' simply incorporates G together with the disjunctions on the edges. Observe that if two edges have the same proposition in their label, then this will be reflected in q' by some shared variables z_{p_i}.

To build $\mathcal{A}_G$ we use individuals c_p and $c_{\neg p}$ for the truth value of proposition p. Intuitively, the truth value of p will be determined by either $A_p(c_p)$ or $A_p(c_{\neg p})$ being in the update. Assume a tuple $\vec{t} = \langle e, v_1, v_2, a, b \rangle$, where $e \in E$, $\{v_1, v_2\} \subseteq \mathcal{B}$, and a, b are individuals. Let p_1, p_2 be the first and the second propositions of e. For $i \in \{1, 2\}$ and $v_i = \mathbf{t}$, let $l_i = p_i$ if p_i is positive and $l_i = \neg p_i$ otherwise. Similarly, for $i \in \{1, 2\}$ and $v_i = \mathbf{f}$, let $l_i = \neg p_i$ if p_i is positive and $l_i = p_i$ otherwise. Then, the ABox $\mathcal{A}(\vec{t})$ consists of the assertions $R_e(a, d_T)$, $R_e(d_T, b)$, $P(d_T, c_{l_1})$ and $P(d_T, c_{l_2})$ depending on the Boolean values in input.

The ABox $\mathcal{A}_G$ is the union of the following ABoxes:

(A1) $\mathcal{A}(\langle e, v, v', a_i, a_j \rangle)$ for all $e \in E$, $v, v' \in \mathcal{B}$, $0 \leq i, j \leq 2$, and $i \neq j$;

(A2) $\mathcal{A}(\langle e, \mathbf{f}, \mathbf{f}, a_i, a_i \rangle)$ for all $e \in E$, and $0 \leq i \leq 2$;

(A3) $\mathcal{A}(\langle e, v, v', b, b \rangle)$ for all $e \in E$, $v, v' \in \mathcal{B}$;

(A4) The ABox $\{B(a_0), B(a_1), B(a_2)\}$;

(A5) The assertions $W_p(c_p, c_{\neg p})$ and $W_p(c_{\neg p}, c_p)$, for all propositions.

Let $\alpha_G = B(b)$. It is not too difficult to see that G is a positive instance of co-CERT3COL iff there exists an $\subseteq$-explanation $\mathcal{U}$ to $\mathcal{P}$ such that $\alpha_G \in \mathcal{U}$. Basically, definitions (A1)-(A3) encode a triangular structure T in which edges in G that evaluate to false according to a given truth assignment can be mapped on any edge of T, reflexive edges included. If an edge of G

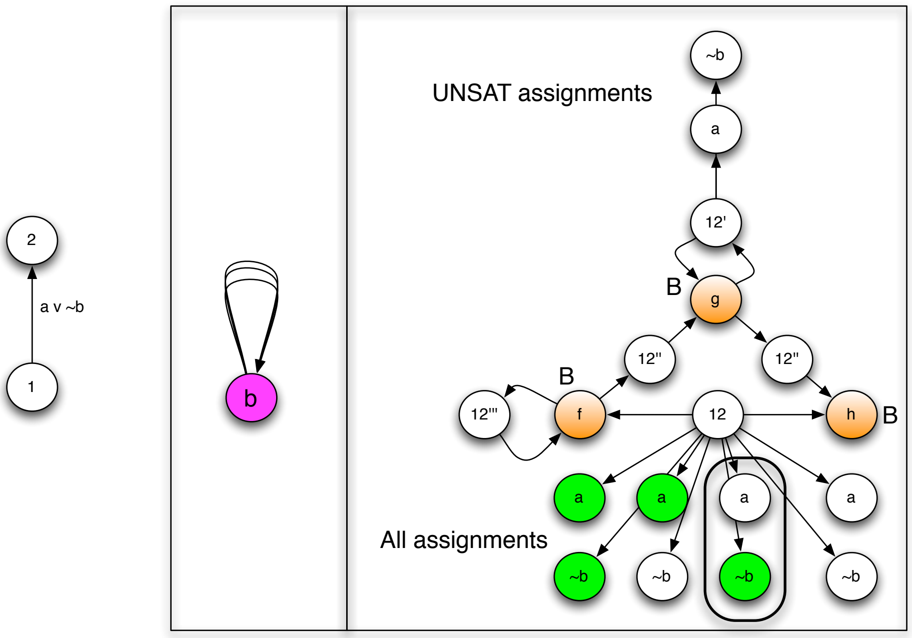


Figure 4.3: On the left you can see a very simple graph G, on the right a portion of the ABox $\mathcal{A}_G$. The structure over the purple node b represents a cycle over which every graph can be mapped. On the right side, there is a triangular structure over the orange nodes. The idea is that nodes that evaluate to true under a certain truth assignment can be mapped only on non-reflexive edges in the structure.

evaluates to true, then it must be mapped to one of the non-reflexive edges. This ensures that if G can be mapped to T under truth assignment t, then $t(G)$ is 3-colorable. Instead, definitions (A4)-(A5) define a cyclic structure C into which any graph G can be embedded. It has to be noted that the node b is not asserted to be a member of B, hence q_G cannot be mapped there directly with any truth assignment. Figure 4.3 shows an intuitive representation of the ABox encoding for a small graph G. In order to facilitate the understanding, we decided to represent the different truth assignments using distinct nodes in the figure. However, remember that in the internal encoding for every Boolean variable p there are only two different individuals in the ABox, namely c_p and $c_{\neg p}$.

Now, we prove that G is a positive instance of co-CERT3COL iff there exists an $\subseteq$-explanation $\mathcal{U}$ to $\mathcal{P}$ such that $\alpha_G \in \mathcal{U}$:

"$\Rightarrow$" Suppose there is a truth assignment t such that $t(G)$ is not 3-colorable. Let $\mathcal{U} = \{B(b)\} \cup \mathcal{U}_1$, where $\mathcal{U}_1 = \{A_p(c_p) \mid t(p) = t\} \cup \{A_p(c_{\neg p}) \mid t(p) = f\}$. It remains to argue that $\mathcal{U}$ is a $\subseteq$-explanation to $\mathcal{P}$. It is not hard to see that $\mathcal{U}$ is an explanation. Indeed q_G matches already in the ABox obtained by point (A3) (hint: since $B(b) \in \mathcal{U}$, we match q_G by mapping all variables of q_G to (the interpretation of) b). Suppose there is a smaller update $\mathcal{U}' \subset \mathcal{U}$. Observe that $\mathcal{U}_1 \subseteq \mathcal{U}'$. This is because for all propositions p, the symbol A_p does not occur in $\mathcal{A}_G$ but does occur in q_G. Then, $\mathcal{U} \setminus \{B(b)\}$ must be an update. If this is the case, then q_G can be matched in $\mathcal{A}_G$ without the ABox from (A3), i.e. in the triangle part. Then $t(G)$ is 3-colorable, which contradicts the assumption.

"$\Leftarrow$" Let $\mathcal{U}$ be a $\subseteq$-minimal explanation containing $B(b)$. Due to the presence of $q_{e_1} \cup \ldots \cup q_{e_k}$ in q_G and the assumption, the role W_p cannot occur in $\mathcal{U}$ for any proposition p. Since $\mathcal{U}$ is an explanation, by the definition of q' and (A5) we have that $A_p(c_p) \in \mathcal{U}$ or $A_p(c_{\neg p}) \in \mathcal{U}$ for all propositions p. Since for any proposition p we have that A_p occurs in q_G with one and only variable z_p, we know that exactly one of $A_p(c_p) \in \mathcal{U}$ and $A_p(c_{\neg p}) \in \mathcal{U}$ holds. Due to the atoms $W_p(z_p, z'_p)$ in q_G, we also have that individuals of the form c_p and $c_{\neg p}$ are the only ones that can get an A_p label. Consider the assignment t defined as follows: $t(e) = \mathsf{t}$ if $A_p(c_p) \in \mathcal{U}$, and $t(e) = \mathsf{f}$ if $A_p(c_{\neg p}) \in \mathcal{U}$. It is not difficult to argue that $t(G)$ is not 3-colorable and thus G is a positive instance of co-CERT3COL. Indeed, if $t(G)$ was 3-colorable, Q should be mappable into the triangle part obtained in (A1)-(A3). Then $\mathcal{U} \setminus \{B(b)\}$ would be a smaller update, which would mean a contradiction. □

Note that the above lower bound applies already for empty TBoxes. Next, we see that under the $\leq$ order the complexity of $\leq$-RELEVANCE is the same as the complexity of $\leq$-NECESSITY.

Proposition 12. *For DL-Lite$_{\mathcal{A}}$ ontologies,* $\leq$-RELEVANCE *is P$^{\mathsf{NP}}_{\|}$-Complete.*

Proof. (MEMBERSHIP) $\leq$-RELEVANCE can be tackled in a similar way as for $\leq$-NECESSITY. In fact, the algorithm described in Theorem 8 can be modified in order to solve relevance. Let SIZE-IN solve the following problem: given a tuple $\langle \mathcal{P}, \alpha, n \rangle$, where $\mathcal{P}$ is a QAP, α an assertion, and n an integer, decide whether there exists an explanation $\mathcal{U}$, with $|\mathcal{U}| = n$ and $\alpha \in \mathcal{U}$. Then, we change the positivity condition of the $\leq$-NECESSITY algorithm as follows: *α occurs in some $\leq$-minimal explanations $\mathcal{U}$ for $\mathcal{P}$ iff for some $0 \leq i \leq m$, it holds that: (i) A_i is a positive instance of* SIZE-IN*, and, (ii) B_i is a positive instance of* NO-SMALLER. That is, it is possible to find a solution $\mathcal{U}$ of size i containing the given assertion, such that there is no smaller ABox that is a solution. It is easy to see that SIZE-IN is solvable in NP, hence the whole algorithm is again in P$^{\mathsf{NP}}_{\|}$.

(HARDNESS) The hardness of this problem can be shown by the exact same reduction as for $\leq$-NECESSITY. The reason is that the reduction enforces the presence of only a single $\leq$-minimal solution, hence in this case relevance and necessity coincide. The only difference regards the way backdoors are connected to constant o in the ABox. We want to show that X is a positive instance to EVEN-CQEVAL iff $OK(o)$ is $\leq$-relevant to $\mathcal{P}$, i.e., we want to enforce that each time the query q matches over A_i with i odd, then a $\leq$-solution $\mathcal{U}$ contains $OK(o)$. This is done simply by inverting the way backdoors are defined. That is, when i is odd the backdoors are R-connected to individual constant o, otherwise they are connected to b'_i (see Figure 4.4). In this way, whenever q matches over A_i, with i odd, the minimal update will contain $OK(o)$. It is not difficult to see that with a similar argument as above one can prove the reduction to be correct.

□

4.5 Complexity of $\preceq$-RECOGNITION

Another important task is the recognition of explanations. That is:

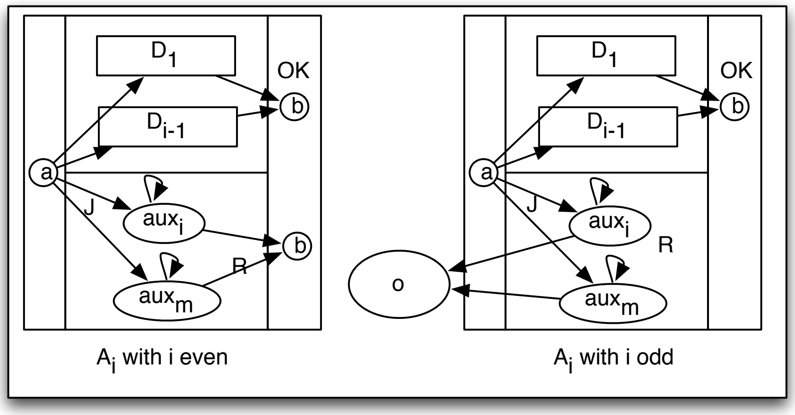


Figure 4.4: A schematic representation of the ABox, note that the way in which auxiliary objects are connected to $a\bar{u}x$ is inverted w.r.t. Fig. 4.1.

Definition 9 ($\preceq$-RECOGNITION)**.** Given an ABox $\mathcal{U}$, decide whether $\mathcal{U} \in \mathsf{expl}_{\preceq}(\mathcal{P})$.

In fact, in our running example, the university administration may be wondering whether asserting $craig$ to be a $Tutor$ can fix the negative answer. It is easy to see that this is not the case and, hence, the problem has to reside on another part of the ontology. Following the methodologt employed in the previous sections, let us systematically explore the complexity of the problems under the various minimality conditions.

Theorem 13. RECOGNITION *is NP-Complete.*

```
1 INPUT: QAP $\mathcal{P} = \langle q, \mathcal{O}, \vec{c}\rangle$ and ABox $\mathcal{U}$
2 OUTPUT: yes iff $\mathcal{U}$ is solution to $\mathcal{P}$
  1: Check that $(\mathcal{T}, \mathcal{A} \cup \mathcal{U}) \not\models \bot$
  2: Guess $i \in \{1, \ldots, n\}$
  3: Guess the derivation of one of the rewriting $r_j$ for $q_i(\vec{c})$ w.r.t. $\mathcal{T}$.
  4: Guess a match $\nu$ for $r_j$ over $DB(\mathcal{A} \cup \mathcal{U})$.
  5: Check that $\nu$ is a valid match for $r_j$ over $DB(\mathcal{A} \cup \mathcal{U})$.
```
Algorithm 4.2: Algorithm solving RECOGNITION.

Proof. (MEMBERSHIP) The membership of the problem in NP is clear from Algorithm 4.2, which is a simplification of Algorithm 4.1 explained in Theorem 11.

(HARDNESS) NP-hardness is shown by reducing CQ-OPT to RECOGNITION. That is, given a CQ q and a subquery q' of q, decide whether q' is equivalent to q. Before actually presenting the result, we introduce briefly the optimization problem for CQs. Let q be a CQ and w.l.o.g, assume that each relation in the query body is either unary or binary. Let q' be a subquery of q

and T, T' be the tableaux for q and q', respectively [1]. Then by the Homomorphism Theorem for tableaux, q' is equivalent to q iff there exists an homomorphism δ from T to T' [1]:

$$\delta : var(T) \to terms(T')$$

such that:

- $\delta(a) = a$, for every constants,
- $\delta(x) = x$, for every head variable,
- if $\vec{t}$ is a row of R in T, then $\delta(\vec{t})$ is a row of R in T', for every relation R.

The problem of checking whether a subquery is equivalent to a given CQ is known to be NP-complete [1].

The reduction works as follows. Let $q(\vec{x}), q'(\vec{x})$ be two conjunctive queries such that, q' is a subquery of q. Then, let $\mathcal{P} = (\mathcal{O}, q(\vec{x}), \vec{c})$ be a QAP, where $\mathcal{O} = (\emptyset, \emptyset)$ and $\vec{c}$ is the tuple of constants resulting from the replacement of each $x \in \mathbf{AV}(q)$ by its constant representative a_x. It is easy to see that $\vec{c} \notin cert(q, \mathcal{O})$, since both the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ are empty. Moreover, let U be the ABox resulting from replacing each variable x in the body of q' with their constant representative a_x.

Now, we want to show that q' is equivalent to q if and only if U is a solution to $\mathcal{P}$.

"$\Rightarrow$" Assume that q' is an equivalent subquery of q. Therefore by the Homomorphism Theorem for Tableaux, there exists an homomorphism δ mapping T' to T that respects the conditions specified above. Similarly, there exists also a tableaux homomorphism δ' mapping T to T'.

Given that the ontology is empty, the set of rewritings contains just $q(\vec{c})$ and $(\emptyset, U)$ is obviously satisfiable. Now, we have to demonstrate that we can find a match for $q(\vec{c})$ over $DB(U)$. Let us choose the match ν to be equivalent to the homomorphism resulting from the composition $\delta' \circ toConst$, where $toConst$ is a function mapping variables to their constant representative and the identity on constants. That is, whenever δ' maps an element to a variable x in T', $\delta' \circ toConst$ maps it to its constant representative a_x. By definition, δ' maps T into T', where T' can be seen as a different syntactical characterization of $DB(U)$. Hence, ν is a valid match for $q(\vec{c})$. From which follows that U is a solution to $\mathcal{P}$.

"$\Leftarrow$" Assume that U is an explanation for $\mathcal{P}$. Using the same argument as above, one can conclude that the set of rewritings of $q(\vec{c})$ contains only q itself. From U being an explanation it follows that $(\emptyset, U)$ is satisfiable and there exists a match ν from $q(\vec{c})$ to $DB(U)$. We need to show that there exists an homomorphism δ from T to T' (resp. the tableaux of q and q'). Let $toVar$ be a function from constants to variable, which given a variable's constant representative a_x returns the variable x itself. If the input is not a variable representative, then $toVar$ behaves as an identity mapping. Then, we define the homomorphism as:

$$\delta : \nu \circ toVar$$

That is, δ uses ν to map q over the database instance (that is, the explanation U) and then, uses the inverse relation $toVar$ to transform U into the body of q'. Therefore, δ is a tableaux homomorphism from T to T', since δ is a match and it respects the three conditions of Tableaux Homomorphisms. □

This proves that RECOGNITION is NP-complete. In particular, it proves that recognizing explanations is hard already for the case in which the ontology $\mathcal{O}$ is empty, that is $\mathcal{T} = \emptyset, \mathcal{A} = \emptyset$, and the user queries are restricted to CQs. Let us now focus on the recognition of $\leq$-explanations.

Theorem 14. *$\leq$-RECOGNITION is DP-Complete.*

Proof. (MEMBERSHIP) The membership of $\leq$-RECOGNITION to DP is shown by providing two languages $L_1 \in$ NP and $L_2 \in$ CONP, such that the set of all yes-instances of $\leq$-RECOGNITION is $L_1 \cap L_2$. This is easy: $L_1 = \{(\mathcal{P}, U) | U$ is an explanation to $\mathcal{P}\}$ and $L_2 = \{(\mathcal{P}, U) | \mathcal{P}$ does not have an explanation U' s.t. $|U'| \leq |U|\}$.

(HARDNESS) DP-hardness is proved by a reduction from the problem HP-NOTHP. That is, given two directed graphs $G = (V, E)$ and $G' = (V', E')$, $\langle G, G' \rangle$ is a positive instance to HP-NOTHP iff G has an Hamiltonian Path[2] and G' does not have one. The reduction is as follows. Let $\mathcal{P} = \langle \mathcal{O}, q, \vec{c} \rangle$ be a QAP, where the TBox is empty. Let the ABox be made of three different components:

1. $\mathcal{A}_G$ encodes the graph G and it is defined as follows:

$$\mathcal{A} = \{e(v_i, v_j) \mid (v_i, v_j) \in E\} \cup \{d(v_i, v_j), d(v_j, v_i) \mid v_i, v_j \in V.\ i \neq j\}$$

   where each $e(v_i, v_j)$ encodes an edge in the graph, whereas $d(v_i, v_j)$ encodes that nodes v_i and v_j are distinct.

2. $\mathcal{A}_{G'}$ encodes G' in a similar way as before using roles e' and d', but it adds the assertion $A(v'_i)$ to each individual $v'_i \in V'$ representing a vertex in G'.

3. Finally, $\mathcal{A}_C$ encodes a clique of size $|V'|$ over new individuals o_i with $1 \leq i \leq |V'|$ using e' and d' as roles.

The query $q()$ is composed by three different components as well: Q_1, Q_2 and Q_3. The first query Q_1 encodes an Hamiltonian Path of length $|V|$ over the first graph:

$$Q_1 \leftarrow \bigwedge_{i=1}^{|V|-1} e(x_i, x_{i+1}), e(x_{|V|}, x_1), \bigwedge_{v_i, v_j \in V. i \neq j} d(x_i, x_j), d(x_j, x_i)$$

Similarly, Q_2 also encodes the presence of an Hamiltonian Path over G'. Further, it requires that each vertex in the path is a member of concept A:

$$Q_2 \leftarrow \bigwedge_{i=1}^{|V'|-1} e'(y_i, y_{i+1}), e'(y_{|V'|}, y_1), \bigwedge_{v_i, v_j \in V. i \neq j} d'(y_i, y_j), d'(y_j, y_i), \bigwedge_{i=1}^{|V'|} A(y_i)$$

[2]A graph G is said to have an Hamiltonian Path if there exists a path in G visiting all the vertices in the graph exactly once.

Q_3 makes sure that in any case an explanation is needed by requiring a new individual a to be member of the fresh concept B:

$$Q_3 \leftarrow B(a)$$

Then, the Boolean CQ $q()$ is the result of the conjunction of Q_1, Q_2 and Q_3. Since we are dealing with a Boolean query, $\vec{c}$ denotes the empty tuple $\langle\rangle$. It is easy to see that $\vec{c} \notin cert(q, \mathcal{O})$, since $B(a)$ is not present in the ABox. Finally, U is the set of assertions $\{A(o_i) | 1 \leq i \leq |V'|\} \cup \{B(a)\}$.

Now, we want to prove that $\langle G, G' \rangle$ is a positive instance to HP-NOTHP if and only if U is a $\leq$-explanation for $\mathcal{P}$.

"$\Rightarrow$" Assume that G has an Hamiltonian Path but G' does not. First of all given that the TBox is empty, one can conclude that any possible solution maintains consistency of the ontology and, moreover, that one has only to consider $q()$ in the rewritings. Next, we need to show that U is actually an explanation for $\mathcal{P}$. By the definition of the ABox and the introduction of different roles modeling G and G', one can see that Q_1 can only be mapped to $\mathcal{A}_G$ and Q_2 to $\mathcal{A}_{G'} \cup \mathcal{A}_C$. Moreover, Q_2 is either mapped to $\mathcal{A}_G$ or to $\mathcal{A}_C$, since the two graphs are encoded on distinct individuals. From the assumption that G contains an Hamiltonian Path, it follows that there is a graph homomorphism from Q_1 to $\mathcal{A}_G$. Therefore, it is possible to find a match ν_1 for Q_1 over $\mathcal{A}$. Now, Q_2 can be mapped easily to $\mathcal{A}_C$, since $\mathcal{A}_C$ models a clique and the update U adds to $\mathcal{A}_C$ the required assertions on the membership of the vertexes of G' to A. Let ν_2 be such a match. Furthermore $\mathcal{A}_C$ and $\mathcal{A}_G$ are distinct, hence one can conclude that $\nu_1 \cup \nu_2$ is match for $Q_1 \wedge Q_2$. This match, which is the identity on constants, is valid also for Q_3, since U includes the required assertion $B(a)$. Finally, we need to show that U is a minimum size explanation. But this follows easily from the fact that G' does not have a hamiltonian path by assumption and, therefore, Q_2 cannot be mapped to $\mathcal{A}_{G'}$. Hence, one is forced to map Q_2 to $\mathcal{A}_C$, which requires all those assertions in the explanation.

"$\Leftarrow$" Assume that $U \in \mathsf{expl}_{\leq}(\mathcal{P})$. Then, one can conclude that there is a match ν for q. Let ν' be the restriction of ν to the variables of Q_1. It follows that ν' is a match for Q_1 over $\mathcal{A}_G$, since the atoms in the query refer only to this part of the ABox. From which it follows that the encoded graph G does have an Hamiltonian Path. Assume by contradiction that G' has an Hamiltonian Path. Then Q_2 can be mapped to $\mathcal{A}_{G'}$ and Q_3 to $\mathcal{A} \cup B(a)$. However, this would contradict the minimality of U. This is because $U' = \{B(a)\}$ would be an explanation for $q()$. Hence, G' cannot have an Hamiltonian Path.

This proves the claim that $\leq$-RECOGNITION is DP-complete. $\square$

The recognition problem maintains the same complexity even under the $\subseteq$ minimality criteria.

Proposition 15. $\subseteq$-RECOGNITION *is DP-Complete.*

Proof. (MEMBERSHIP) As done in the previous case, we define two languages $L_1 \in$ NP and $L_2 \in$ coNP, such that the set of all yes-instances of $\subseteq$-RECOGNITION is $L_1 \cap L_2$. Let L_1 be as before the set of all tuples $\langle \mathcal{P}, U \rangle$, where U is an explanation for $\mathcal{P}$. Then $L_2 = \{(\mathcal{P}, U) | \mathcal{P}$ does not have an explanation U' s.t. $U' \subseteq U\}$. Note that L_2 is exactly the complement of HAS-SUBEXPL introduced in Section 4.4. The claim easily follows.

(HARDNESS) The DP-hardness can be proved with exactly the same construction as in Theorem 14. In fact, it is sufficient to read $\subseteq$ in place of $\leq$ in order to have a valid reduction proving of hardness for $\subseteq$-RECOGNITION. $\square$

In conclusion, this chapter studies the computational complexity of four main reasoning tasks over Query Abduction Problems ($\preceq$-EXISTENCE, $\preceq$-NECESSITY, $\preceq$-RELEVANCE and $\preceq$-RECOGNITION) in the context of $DL\text{-}Lite_{\mathcal{A}}$ ontologies.

CHAPTER 5

Solving Query Derivation Problems

In Chapter 3, we introduced the notion of Query Derivability Problem, the formalization of the explanation problem for positive query answers to UCQ over *DL-Lite*$_{\mathcal{A}}$ ontologies. In this chapter, we address the problem of computing minimal solutions to QDPs. Given a QDP $\mathcal{D} = \langle \mathcal{O}, q, \vec{c} \rangle$, a solution to $\mathcal{D}$ is a tuple $\langle \pi, \langle D_1, \dots, D_n \rangle \rangle$, where π is a match for some $q_i \in q(\vec{c})$ on the canonical model of $\mathcal{O}$, and, $\langle D_1, \dots, D_n \rangle$ are derivations w.r.t. $\mathcal{O}$ for the ABox assertions resulting from instantiating the atoms in q_i w.r.t. the match π (see Section 3.2).

The problem of computing solutions to QDPs has been tackled already by Borgida et. al in [9], where an high-level procedure solving QDPs is provided. Unfortunately, the high-level procedure has never been given an algorithmic counterpart and, for this reason, it cannot be easily implemented in a query answering system over *DL-Lite*$_{\mathcal{A}}$ ontologies. Hence, this chapter aims at closing this gap between theory and practice by providing such a formalized algorithm computing explanations to positive query answers. Also, the above mentioned procedure is not tailored for searching minimal solutions, but our new algorithm takes minimality restriction into account.

Moreover, this chapter introduces a new procedure solving the explanation problem for unsatisfiable *DL-Lite*$_{\mathcal{A}}$ ontologies, which employs the new algorithm solving QDPs. The designed solution provides domain users with evidences both in the data and in the conceptual knowledge justifying the inconsistency in the knowledge base.

The rest of the chapter is organized as follows. First, Borgida's high-level procedure solving QDPs is introduced. Then, we present an implementable algorithm formalizing this procedure, which requires a modification to *DL-Lite*$_{\mathcal{A}}$'s standard query answering procedure. Finally, the application of this new explanation algorithm to the problem of explaining unsatisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies is studied.

5.1 An High-level Procedure Solving QDPs

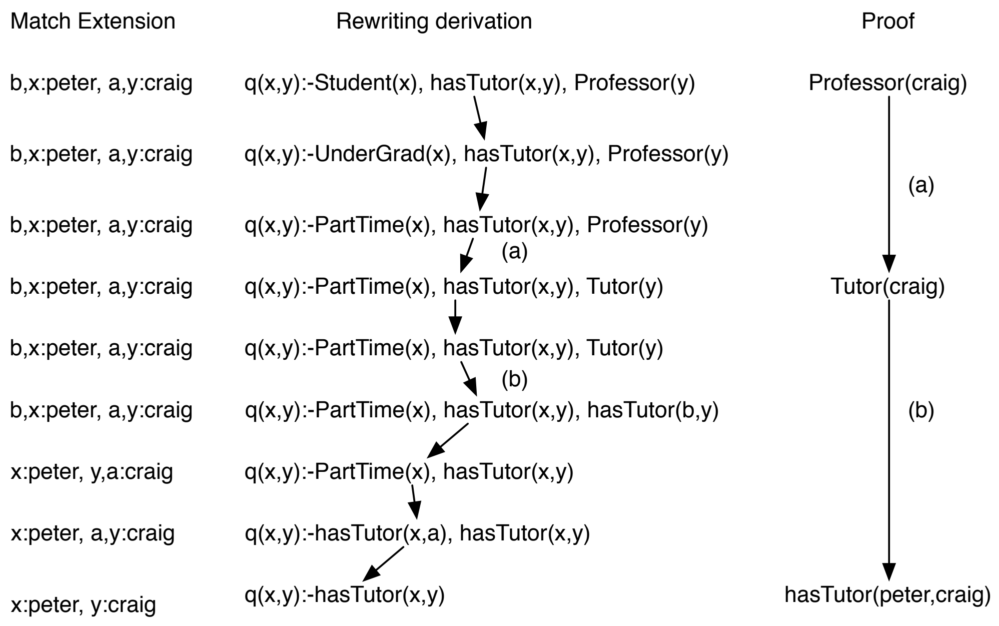
A solution to a QDP $\mathcal{D} = \langle \mathcal{O}, q, \vec{c} \rangle$ is composed by a match over the canonical model of $\mathcal{O}$ and by a series of derivations for ABox assertions w.r.t. the ontology. Let us first focus on the problem of finding such a match for a QDP. From theoretical results (see Proposition 3), we know that conjunctive query answering over a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}$ can be reduced to the problem of evaluating CQs over the canonical model $can(\mathcal{O})$. Therefore, in principle finding such a match for a CQ $q_i \in q$ is possible. Unfortunately, $can(\mathcal{O})$ is generally infinite and, for this reason, it cannot be used to effectively compute solutions to QDPs. Nevertheless, an important feature of the canonical model of a *DL-Lite*$_\mathcal{A}$ ontology is that a UCQ q can be answered by evaluating it on only a small finite portion of $can(\mathcal{O})$ [13]. Then, Borgida's procedure uses this fact in order to solve Query Derivation Problems. That is, it constructs the finite small portion of the canonical model and it justifies answer $\vec{c}$ over it, i.e., it finds an admissible match π for $q(\vec{c})$ over the constructed finite portion of $can(\mathcal{O})$.

We now present the technique introduced by Borgida et. al to construct such a finite portion of $can(\mathcal{O})$, which is based on the algorithm $\mathit{PerfectRef}(q, \mathcal{T})$ that computes the perfect reformulation of a query q w.r.t. an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ (see Section 2). The key observation about the perfect reformulation algorithm is the following: rewriting steps correspond to the inverses of the additions made to the canonical model of the knowledge base [9].

Assume $q(\vec{x})$ to be a UCQ over ontology $\mathcal{O}$ and, further, suppose that $\vec{c} \in \mathsf{cert}(q, \mathcal{O})$. From this follows that there exists a rewriting $r_i \in \mathit{PerfectRef}(q, \mathcal{T})$, such that $\vec{c} \in \mathsf{ans}(r_i, DB(\mathcal{A}))$. The fact that $r_i(\vec{c})$ evaluates positively over $DB(\mathcal{A})$ implies the existence of an admissible match π over the database instance, such that $\pi(\vec{x}) = \vec{c}$. Then, we compute the derivation of r_i from q, building a data-structure storing information on how rewritings have been generated from predecessor queries. In particular, in case of rewritings generated from the application of rewriting rules we store: the predecessor rewriting, the atom being substituted and the axiom used from the TBox. For unification steps, we store: the predecessor rewriting as well, the two atoms being unified and the most general unifier between them. Now, we traverse backwards the polynomially long trace of rewritings generating r_i until some $q_i \in q$ is reached. While backtracking, the match π is extended to be a match for the intervening queries by keeping track of unifications or by assigning to variables new Skolem constants. More specifically, let r_i be generated from r_{i-1}, if no variables has been eliminated from r_i to r_{i-1}, then the match is not extended. When a variable z has been eliminated in r_i because it has been unified with y in r_i, then we set $\pi(y) = \pi(z)$. Differently, if a variable z has been eliminated in r_{i-1} by replacing atom $R(x, z)$ with $A(x)$ in r_i, due to the inclusion $A \sqsubseteq \exists R$ in the TBox, then we set $\pi(z) = @c$, where $@c$ is a fresh Skolem constant. Please note that these extension procedure mimics the (inverse of the) chasing procedure employed while computing the canonical model of $\mathcal{O}$ (see Section 2.3)

At the end of this process, we are left with a match π for q over a finite portion of $can(\mathcal{O})$, such that $\pi(\vec{x}) = \vec{c}$. Additionally, note that the constructed data-structure together with the extended match provide sufficient information in order to construct derivation trees for the assertions resulting from the instantiation of the body of the user query w.r.t. the extended match π (see Figure 5.1). The reason is that the rewriting rules applied while generating rewritings (see Table 2.1) are syntactic variations of the inference rules used to find derivations for ABox assertions (see Section 3.2).

Then, the explanation procedure returns the extended match π together with the constructed data-structure. Please refer to [9] for the proof of correctness of this approach.



Match Extension
Rewriting derivation
Proof
b,x:peter, a,y:craig
q(x,y):-Student(x), hasTutor(x,y), Professor(y)
b,x:peter, a,y:craig
q(x,y):-UnderGrad(x), hasTutor(x,y), Professor(y)
b,x:peter, a,y:craig
q(x,y):-PartTime(x), hasTutor(x,y), Professor(y)
(a)
b,x:peter, a,y:craig
q(x,y):-PartTime(x), hasTutor(x,y), Tutor(y)
b,x:peter, a,y:craig
q(x,y):-PartTime(x), hasTutor(x,y), Tutor(y)
(b)
b,x:peter, a,y:craig
q(x,y):-PartTime(x), hasTutor(x,y), hasTutor(b,y)
x:peter, y,a:craig
q(x,y):-PartTime(x), hasTutor(x,y)
x:peter, a,y:craig
q(x,y):-hasTutor(x,a), hasTutor(x,y)
x:peter, y:craig
q(x,y):-hasTutor(x,y)
Professor(craig)
(a)
Tutor(craig)
(b)
hasTutor(peter,craig)

Figure 5.1: This figure shows how it possible to turn the rewriting data-structure into a proof for an ABox assertion. (a) denotes the application of TBox axiom $\mathit{Tutor} \sqsubseteq \mathit{Professor}$ and is an instance of the inference rule *SubConcept*. (b) denotes the application of $\exists \mathit{hasTutor}^- \sqsubseteq \mathit{Tutor}$ and represents an instance of the inference rule *Rng-Intro*.

Let us now introduce the algorithms that formalize the presented procedure.

5.2 Computing Minimal Solutions to QDPs

Let $\mathcal{D} = \langle \mathcal{O}, q(\vec{x}), \vec{c} \rangle$ be a Query Derivation Problem. We aim at providing a formal algorithm EXPLAINPOSITIVEANSWER$(\mathcal{D}, ds)$, where $\mathcal{D}$ is a QDP and ds is a data-structure containing generation information for the rewritings of $q(\vec{c})$. Then, the function returns a match π for $q_i \in q(\vec{c})$ over a finite fragment of $can(\mathcal{O})$ and the relevant portion d' of ds containing the derivation of the rewriting through which π was computed. We require that the tuple $\langle d', \pi \rangle$ returned by the algorithm generates a minimal solution to $\mathcal{D}$.

Data Structure

The implementation of such a procedure requires first the definition of a suitable data-structure over which to store information about the generation of rewritings. We decided to employ a directed graph $G = (V, E)$, where nodes are rewritings and edges characterize the derivation of a rewriting r_i from a predecessor query r_{i-1}. Then, we associate to each edge a label, which allows us to store the necessary information, such as TBox axioms or most general unifiers used to derive r_i from the predecessor query. The reason for choosing a graph structure over a linear structure (as suggested in [9]) is that a single rewriting may be generated through many different

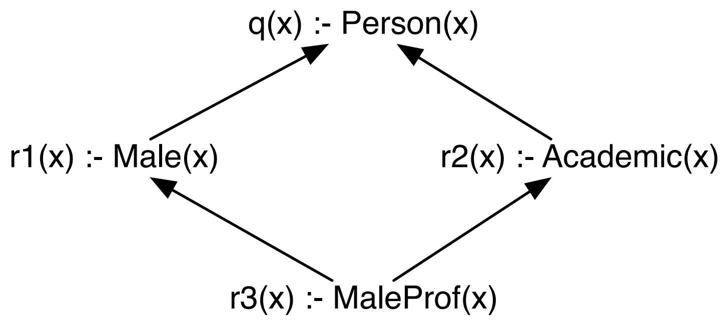


Figure 5.2: Assume a TBox $\mathcal{T} = \{MaleProf \sqsubseteq Male, MaleProf \sqsubseteq Academic, Male \sqsubseteq Person, Academic \sqsubseteq Person\}$ and a query $q(x) \leftarrow Person(x)$. Then, rewriting $r3$ can be generated in two different ways.

derivations (see Figure 5.2). Therefore, in order to find minimal proof trees (and hence minimal solutions), it is important to consider shortest derivations. The directed graph $G = (V, E)$ is defined as follows:

- $V = V_q \cup V_{rewr}$ contains the set of rewritings being computed and it is initialized with $V_q = q$ and $V_{rewr} = \emptyset$. Here we assume that conjunctive queries in q are numbered in increasing order from 1 to n.
- $E = E_{gr} \cup E_{unify}$, is the derivation relation, where an edge (r_i, r_{i+1}) in E_{gr} means that query r_{i+1} has been derived from r_i by a rewriting step, whereas an edge (r_i, r_{i+1}) in E_{unify} means that query r_{i+1} has been derived from query r_i via unification. Finally, let L be a labeling function over edges and $cost : E \mapsto \{0, 1\}$ be a cost function.

Now, we have to define a procedure that populates the defined data-structure. We have chosen to modify the standard perfect reformulation algorithm, rather than defining a new procedure to be ran after the answers of the queries were given to the user. The reason is that a separated procedure would have required the computation of the rewritings twice, while by suitably modifying the *PerfectRef* algorithm we don't have such an overhead, since the data-structure is populated during the computation.

Modified Reformulation Algorithm

Algorithm 5.1 shows the modified procedure for computing the perfect reformulation of a UCQ q over TBox $\mathcal{T}$. Roughly speaking, the new algorithm extends the standard procedure by suitably storing rewritings in the directed graph G rather than storing them in a set. Assume that rewriting r_i has been generated from r_{i-1} by the application of axiom α over atom $g \in at(r_{i-1})$. Then in steps [9-15], r_i is added to the vertexes of G (in V_{rewr}) and an edge $(r_{i-1}, r_i) \in E_{gr}$ is asserted in the graph. Additionally, the new edge (r_{i-1}, r_i) is labeled with (g, α) and the cost of traversing the edge is set to be 1. Differently, if r_i has been derived from r_{i-1} by the unification of two atoms $g_1, g_2 \in at(r_{i-1})$ through mgu μ, in steps [21-27] r_i is added to V_{rewr} and an edge $(r_{i-1}, r_i) \in E_{unify}$ is asserted in the graph. The new edge is labeled with (g_1, g_2, μ) and we

```
1 INPUT: UCQ $q$ and *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$
2 OUTPUT: graph $G$ of the rewritings of $q$ w.r.t. $\mathcal{T}$
 1: $V_{start} = q$
 2: $i = |V_{start}|$
 3: repeat
 4:    $(V', E') = G$
 5:    for $q' \in V'$ do
 6:       for atom $g \in q'$ do
 7:          for PI $\alpha \in \mathcal{T}$ do
 8:             if $\alpha$ is applicable to $g$ then
 9:                $i = i + 1$
10:                $q_i = q'[g/gr(g, \alpha)]$
11:                $e = (q', q_i)$
12:                $L(e) = (g, \alpha)$
13:                $cost(e) = 1$
14:                $V'_{rewr} = V'_{rewr} \cup q_i$
15:                $E'_{gr} = E'_{gr} \cup \{e\}$
16:             end if
17:          end for
18:       end for
19:       for atom $g_1, g_2 \in q'$ do
20:          if $g_1$ and $g_2$ unify then
21:             $i = i + 1$
22:             $q_i = anon(reduce(q', g_1, g_2))$
23:             $e = (q', q_i)$
24:             $L(e) = (g1, g2)$
25:             $cost(e) = 0$
26:             $V'_{rewr} = V'_{rewr} \cup q_i$
27:             $E'_{unify} = E'_{unify} \cup \{e\}$
28:          end if
29:       end for
30:    end for
31: until $G = (V', E')$
32: return $(extendAnswer(V'), E)$
```

Algorithm 5.1: function PERFECTREF'$(q, \mathcal{T})$

set $cost((r_{i-1}, r_i)) = 0$. The reason for the difference costs associated to $unify$- and gr-edges is that the former do not contribute in the reformulation with a new query, but they represent an equivalence preserving transformation of already derived queries. Therefore, when searching for shortest paths among rewritings, $unify$-edges should not count as rewriting steps.

Finally, let $G = (V, E)$ be the (graph of the) generated perfect reformulation, we reuse a technique from [9] to extend the tuple of answer variables of the perfect reformulation to include information on the rewriting generating a specific tuple in the answers. Let $extendAnswer$ be a function that alters each $r_i \in V$ in the following way: the tuple of answer variables is extended to contain fresh variable tag, and, the body of r_i is extended by the introduction of atom $tag = i$.

$$r_i(\vec{x}) \leftarrow at(r_i). \quad r_i(\vec{x}, tag) \leftarrow at(r_i), tag = i.$$

It is not difficult to see that (by abusing of notation) $\langle c_1, \ldots, c_n, k\rangle \in \mathsf{cert}(extendAnswer(V), \mathcal{O})$ iff $\langle c_1, \ldots, c_n\rangle \in \mathsf{ans}(r_k, DB(\mathcal{A}))$.

Correctness of the algorithm The algorithm modifies $PerfectRef$ by storing the rewritings in a graph structure. Also, extending the tuple of answer variables to store information on generating rewritings does not alter the results, it only adds redundancies in the answers. In fact, one can always remove the newly introduced answer variable from the projection of the results and get precisely the same perfect reformulation as with $PerfectRef$.

Complexity of the algorithm The size of the perfect reformulation of a UCQ q is worst-case exponential in the size of q [13]. Hence, generating the graph of rewritings is both exponential in space and time with respect to the size of the input query. However, the algorithm runs in polynomial time and space w.r.t. the size of the TBox $\mathcal{T}$. The intuition is that the number of new queries one can introduce is bounded by the number of axioms in the TBox [13].

Explanation Algorithm

The new rewriting algorithm provides a data-structure maintaining information on how rewritings have been derived from the initial user query. The core explanation algorithm, then, uses this information to generate the finite portion of the canonical model to be used in the explanation of $q(\vec{c})$. The algorithm EXPLAINPOSITIVEANSWER$(\mathcal{D}, G)$ works as follows (see Algorithm 5.2).

Given the QAP $\mathcal{D} = \langle \mathcal{O}, q, \vec{c}\rangle$ and the graph structure G, it first computes the set of rewritings that generate the answer tuple $\vec{c}$. This is easy since the new reformulation procedure extends the result set by appending information on the rewriting generating each answer tuple. Let $computeGenerator$ be the function returning the set of rewritings generating answer $\vec{c}$ in QDP $\mathcal{D}$.

Let Q be the set of rewritings generating answer tuple $\vec{c}$. Since we are interested in *minimal explanations*, it is important to compute the solution based on the rewriting in Q that leads to a minimal solution to $\mathcal{D}$. For this reason, we associate to each $r_i \in Q$ a cost:

$$cost(r_i) = bodySize(r_i) + minPath(r_i, q)$$

where $minPath(r_i, q)$ gives the length of the shortest path S in G from r_i to a query $q_j \in V_{start}$ and $bodySize(r_i)$ weights the size of the query in terms of the number of atoms occurring in the body. The shortest path among queries in the graph is found by using the Dijkstra's algorithm for weighted graph (see Appendix A). Therefore, we reward rewritings with short derivations and a small number of atoms. Note that this is exactly the definition of minimality for solutions

```
1 INPUT: QDP $\mathcal{D} = \langle \mathcal{O}, q, \vec{c} \rangle$ and the graph of rewritings $G$ of $q$ w.r.t. $\mathcal{O}$
2 OUTPUT: $S$ shortest path from a rewriting generating $\vec{c}$ to $q_i \in q$ and a match for $q_i$ over $can(\mathcal{O})$.
 1: $Q = computeGenerator(\mathcal{D})$
 2: $r_i = computeMinimal(Q, G)$
 3: $\pi = computeMatch(r_i, \mathcal{D})$
 4: $\langle dist, previous \rangle = Dijkstra(G, r_i)$
 5: $min = \infty$
 6: for $q_i \in V_{start}$ do
 7:     if $dist[q_i] < min$ then
 8:         $min = q_i$
 9:     end if
10: end for
11: $S = shortestPath(r_i, min, \langle dist, previous \rangle)$ (ordered sequence)
12: $q = r_i$
13: while $q \notin V_{start}$ do
14:     $q' =$ pop from S
15:     if $vars(q') \neq vars(q)$ then
16:         $z = vars(q') \setminus vars(q)$
17:         if $(q', q) \in E_{gr}$ then
18:             $\pi = \pi \cup \{z \mapsto newSkolem\}$
19:         end if
20:         if $(q', q) \in E_{unify}$ then
21:             $\pi = \pi \cup \{z \mapsto joinedVariable(label(q', q))\}$
22:         end if
23:     end if
24:     $q = q'$
25: end while
26: return (S, $\pi$)
```

Algorithm 5.2: function EXPLAINPOSITIVEANSWER($\mathcal{D}, G$)

to QDPs. Hence, by transforming the output of the algorithm into a solution for the QDP, we are guaranteed to find a minimal solution to the problem.

Then, let r_i be the minimal rewriting for q. We now compute a match π for r_i over $DB(\mathcal{A})$. This can be done by extending the answer tuple of r_i to include non-distinguished variables as well and by evaluating again the query over the database instance. The result of this computation is a match π for the rewriting.

Now, the aim of the algorithm is to traverse backwards the shortest path S from r_i until an initial query $q_i \in q$ is reached. During this backtracking through the shortest path leading to an initial query, the match π for r_i over $DB(\mathcal{A})$ is extended by means of variable substitution to be a match for the intervening queries, by using the information stored in the labels of graph G on how queries have been generated and by using the technique described in Section 5.1. At the end of the backtracking, the algorithm has generated an extended match π for $q(\vec{c})$, i.e., the finite relevant portion of the canonical model over which we can evaluate q. Then, the algorithm returns the generated match together with shortest path S for deriving r_i. This information can then be used to generate proof trees and complex interactive explanation to be shown to domain users.

Correctness of the algorithm The correctness of EXPLAINPOSITIVEANSWER$(\mathcal{D}, G)$ follows from the high-level procedure explained in Section 5.1.

Complexity of the algorithm The procedure is based on the Dijkstra's algorithm for shortest paths, which is known to run in $O(|V|^2)$. In this case, the graph in input is the graph of rewritings. It follows that the algorithm runs in exponential space and time w.r.t. the size of the UCQ q in input. Furthermore, a trace from a generating rewriting to a user query is at most polynomial in the size of the TBox $\mathcal{T}$, therefore the algorithm runs in polynomial time and space w.r.t. $\mathcal{T}$.

Let us now show how KB inconsistency can be explained by making use of this explanation procedure.

5.3 Explaining KB Inconsistency

As explained in Section 2.3, the inconsistency of a *DL-Lite*$_{\mathcal{A}}$ ontology does not only depend on the assertions occurring in the ABox. In fact, it can be generated by the combination of both positive and negative inclusions asserted in the terminological component. For this reason, it is important to give users a clear understanding on how axioms in the TBox and assertions in the ABox generate an inconsistent Knowledge Base.

In [9], Borgida et. al provide an intuition on how to explain ontology unsatisfiability. Roughly speaking, the idea is to search for individuals in the ABox that are either asserted to, or can be deduced to, be member of an unsatisfiable concept or role. The procedure they employ to find minimal explanations for the unsatisfiability of a concept or role is based on Breadth-First search and, therefore, runs in exponential time in the size of the TBox[1]. All in all, the unsatisfiability of an ontology is explained by providing the individual asserted to be a member of an unsatisfiable concept/role and the motivations leading the concept/role to be inconsistent. We now provide a different solution to the problem, which is based on the algorithm computing positive explanations to query answers that runs in polynomial time and space in the size of the TBox.

For this recall that deciding KB satisfiability can be reduced to the problem of evaluating a Boolean query over a relational instance (see Section 2.3), i.e., the ontology is inconsistent if and only if a specifically constructed query returns a non-empty answer. The procedure works by the definition of a query q_{unsat}, which asks for evidences in the data that do not satisfy the constraints imposed at the TBox level. This query is defined by using the negative closure of the TBox, $cln(\mathcal{T})$, hence combinations of negative and positive axioms are already taken into account at the construction level. For this reason, the procedure does not employ any rewriting

[1]The supplementary Chapter I provides a structural algorithm for explaining concept/role inconsistency which outperforms the one suggested in [9], since it runs in polynomial time in the size of the TBox.

technique to define the query. For instance, in our mock ontology over the university domain (see Figure 3.1), q_{unsat} would contain $q_{unsat}() \leftarrow Postgrad(x), PartTime(x)$, because we know that $Postgrad$ and $Undergrad$ are disjoint and, additionally, $PartTime$ students are undergraduates (see the definition of $cln(\mathcal{T})$ in Section 2.3).

In the following, we provide a new way of checking consistency which is based on query rewritings and which returns evidences in the data justifying the inconsistency. Then, these computed evidences are given in input to our explanation procedure for positive query answers, which will highlight the TBox axioms used in finding the inconsistency.

Modified Satisfiability Algorithm

```
1 INPUT: DL-Lite_A ontology $\mathcal{O}$
2 OUTPUT: yes iff $\mathcal{O}$ is satisfiable. Otherwise, data-structures for explaining inconsistency.

 1: $q'_{unsat}, q''_{unsat}, q'''_{unsat} = \{\bot\}$
 2: $G = \langle V, E \rangle$
 3: for $\alpha \in NCI(\mathcal{T})$ do
 4:     $q'_{unsat} = q'_{unsat} \cup \{\delta'(\alpha)\}$
 5: end for
 6: $G = PerfectRef'(q'_{unsat}, \mathcal{T})$.
 7: if $\mathsf{ans}(V, DB(\mathcal{A})) \neq \emptyset$ then
 8:     return EXPLAINPOSITIVEANSWER($(\langle \mathcal{O}, q'_{unsat}, \vec{c} \rangle, G)$, where $\vec{c}$ is the first tuple in the answers.
 9: end if
10: for $\alpha \in NRI(\mathcal{T})$ do
11:     $q''_{unsat} = q''_{unsat} \cup \{\delta'(\alpha)\}$
12: end for
13: $(V, E) = PerfectRef'(q''_{unsat}, \mathcal{T})$.
14: if $\mathsf{ans}(V, DB(\mathcal{A})) \neq \emptyset$ then
15:     return EXPLAINPOSITIVEANSWER($(\langle \mathcal{O}, q''_{unsat}, \vec{c} \rangle, G)$, where $\vec{c}$ is the first tuple in the answers.
16: end if
17: for $\alpha \in funct(\mathcal{T})$ do
18:     $q'''_{unsat} = q'''_{unsat} \cup \{\delta'(\alpha)\}$
19: end for
20: if $\mathsf{ans}(q'''_{unsat}, DB(\mathcal{A})) \neq \emptyset$ then
21:     return tuple breaking functionality assertion and the functionality assertion.
22: end if
23: return satisfiable
```

Algorithm 5.3: function UNSATISFIABLE'($\mathcal{O}$)

Let us now present the changes introduced by the new algorithm checking satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies (see Algorithm 5.3). The basic idea is to define the query q_{unsat} not in terms of the negative closure of the TBox, but rather by employing the perfect reformulation algorithm. That is, we let the rewriting procedure find all the implied constraints from the negative and positive inclusions in the TBox. By using the new algorithm $\mathit{PerfectRef}'$, we also have that the derivation of new negative inclusions from the TBox can be justified to the user. Moreover, we extend the way the query is generated from the negative axioms occurring in the terminological component in order to return the evidences of the values in the ABox that caused a certain negative inclusion to be violated:

$$\begin{aligned}
\delta'(B_1 \sqsubseteq \neg B_2) &= q(x) \leftarrow \gamma_1(B_1, x), \gamma_2(B_2, x) \\
\delta'(Q_1 \sqsubseteq \neg Q_2) &= q(x, y) \leftarrow \rho(Q_1, x, y), \rho(Q_2, x, y) \\
\delta'((\text{funct P})) &= q(x, y, z) \leftarrow P(x, y), P(y, z), y \neq z \\
\delta'((\text{funct P}^-) &= q(x, y, z) \leftarrow P(x, z), P(y, z), x \neq y
\end{aligned}$$

(γ and ρ are defined as before). This addition, however, requires some care on how we create the final query to be evaluated, because queries do not have the same arity anymore and, hence, they cannot be part of the same Union of CQs. For this reason, we use three different queries in Algorithm 5.3 checking for violation of concept disjointness (NCI), role disjointness (NRI) and functionality assertions, respectively.

Intuitively, the algorithm works as follows. It first goes through all the negative concept inclusions explicitly asserted in the TBox and uses the function δ' to derive a UCQ asking for evidences not complying with these constraints. Then, by means of the procedure $\mathit{PerfectRef}'$, the algorithm derives the implied constraints by using the rewriting rules. For instance, let $\mathcal{O}_{uni}$ be the university mock ontology, which contains $\alpha = \mathit{UnderGrad} \sqsubseteq \neg \mathit{PostGrad}$ in the TBox. Then, the following query will be generated by the application of δ' to α:

$$q_1(x) \leftarrow \mathit{UnderGrad}(x), \mathit{PostGrad}(x);$$

The TBox further contains the axiom $\mathit{PartTime} \sqsubseteq \mathit{UnderGrad}$, therefore we know that the perfect reformulation of q_1 w.r.t. $\mathcal{O}_{uni}$ contains:

$$q_1'(x) \leftarrow \mathit{PartTime}(x), \mathit{PostGrad}(x);$$

Given the graph of rewritings returned by $\mathit{PerfectRef}'$, the resulting UCQ r is evaluated over $DB(\mathcal{A})$ and if the answers are not empty, then there is an evidence in the data, which does not comply with the constraints expressed in the TBox. At this point, the explanation routine for positive answers is ran to provide domain users with a complete justification for the presence of the (possibly) implied constraint, which gives, together with the evidence in the data, a complete understanding on the reasons why the ontology is unsatisfiable.

The same approach is repeated also for negative role inclusions, while for functionality assertions it is not necessary to compute the perfect reformulation. The reason is that functional roles cannot be specialized in *DL-Lite*$_{\mathcal{A}}$ ontologies (see Section 2.1) and, hence, no other constraints can be further entailed. Figure 5.3 shows an example of the explanations provided to justify an inconsistent ontology.

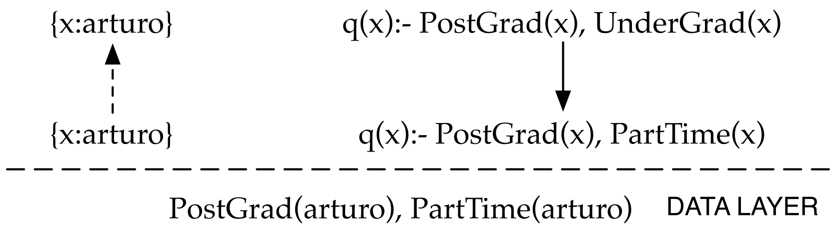

{x:arturo}
q(x):- PostGrad(x), UnderGrad(x)
{x:arturo}
q(x):- PostGrad(x), PartTime(x)
PostGrad(arturo), PartTime(arturo)
DATA LAYER

Figure 5.3: Recalling the mock ontology from Figure 3.1, we know that $UnderGrad$ is disjoint from $PostGrad$. Further assume that in the information system we have that $arturo$ is both a $PostGrad$uate and a $PartTime$ student. Then, the inconsistency explanation routine can, by means of query answer explanation, highlight the reasons leading to such an unsatisfiable ontology by making use of the structure of rewritings.

Correctness of the algorithm It is not difficult to see that this procedure is equivalent to the standard satisfiability procedure for *DL-Lite*$_\mathcal{A}$ (see Algorithm 2.1). The reason is that it is known that $PerfectRef$ (and $PerfectRef'$) is a correct procedure computing the perfect reformulation of a query q w.r.t. a TBox $\mathcal{T}$. Hence, one can compute the negative closure of a TBox $\mathcal{T}$, $cln(\mathcal{T})$, by encoding the explicit negative axioms occurring in $\mathcal{T}$ into a query q. Then, computing the perfect reformulation amounts to apply rewriting steps, i.e. positive inclusions, over the given negative axioms (encoded in the query). Therefore, a new rewriting in the perfect reformulation can be understood as a new implied negative axiom. In fact, one can see from the definition of $cln(\mathcal{T})$ that new negative inclusions can only be computed from the application of positive inclusions from $\mathcal{T}$. From the correctness and completeness of the rewriting procedure, one can argue that the new procedure computes the same implied constraints as $cln(\mathcal{T})$. Therefore, algorithm Unsatisfiable' decides satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies.

Complexity of the algorithm The number of rewritings generated by the procedure is polynomial in the size of the TBox. Each of this rewriting can be evaluated in AC$_0$ data complexity over the database. The query answers explanation procedure requires to store each single rewriting, which are polynomially many in terms of $\mathcal{T}$. Therefore, the whole algorithm runs in polynomial space and time w.r.t. $\mathcal{T}$.

This concludes this chapter, which introduced an algorithmic solution to the already described problem of explaining positive query answers. In addition, it presented a new approach in explaining KB inconsistency based on the explanation of positive query answers, which runs in polynomial time w.r.t. the size of the TBox.

CHAPTER 6

Conclusions and Future Work

This thesis tackled the problem of explaining query answers over lightweight ontologies. Lately, there has been a lot of emphasis on Ontology-Based Data Access systems, i.e., data management tools which mediate the access to data by means of a lightweight ontology. In this work, we focused on the ontology language *DL-Lite*$_{\mathcal{A}}$: an expressive member of the *DL-Lite* family of Description Logics at the basis of the OWL2-QL profile. In the context of explaining query answers, we decided to focus on two main problems: the explanation of the *absence* of a tuple in the answers and explaining the *presence* of a tuple in the results .

The problem of explaining the absence of results over DL ontologies was not yet given attention in the literature. For this reason we have provided a formalization of the problem of explaining negative query answers. A tuple is called a *negative answer*, if the user expects it to be part of $\mathsf{cert}(q, \mathcal{O})$ but the tuple is actually not. In our framework based on abductive reasoning, an explanation consists of an ABox that when added to the ontology leads the negative answer to be returned in the results of the query. We define various problems that help us in characterizing the complexity of this abduction problem, such as the existence of explanations and relevance/necessity of assertions. We further consider minimality criteria to be applied over explanations, such as subset-minimal and minimum explanation size preference orders. Within this framework, we provide a characterization of the computational complexity of the various problems for the DL *DL-Lite*$_{\mathcal{A}}$.

Differently, the explanation problem for positive answers over *DL-Lite*$_{\mathcal{A}}$ ontologies was already tackled in the literature, but not formally solved in terms of an implementable algorithm. With the aim of closing this gap, we provide an algorithm which could be implemented in a real-world system right out of the box. Furthermore, we employ this new algorithm to provide a new solution to the problem of explaining the inconsistency of a *DL-Lite*$_{\mathcal{A}}$ knowledge base.

Future work includes the implementation of the designed explanation algorithms over a real-world OBDA-enabled reasoner, such as QuOnto[1]. The implementation would also require a

[1]`http://www.dis.uniroma1.it/~quonto/`

study on Human-Computer Interaction in oder to devise the most suitable way for presenting explanations to users.

Specifically for explanation of negative answers, it would be interesting to investigate the application of this framework over different Description Logics. A perfect candidate seems to be the $\mathcal{EL}$ family of DLs, which relies on a similar notion of query rewriting. Also, we would like to investigate the problem in the case where the ontology signature and the explanation signature may be different. In most cases the ontology vocabulary extends the database vocabulary. For this reason, it may be the case that only a subset of the ontology signature may be used to devise explanations [6]. Additionally, it is important to characterize formally the complexity of computing solutions and provide algorithm solving the problem. Finally, it would be interesting to consider semantic-based minimality criteria. For instance, one could consider a solution to be minimal if it logically implies all other solutions to a given problem.

As regards the explanation of positive answers, in this thesis we focused only on computing explanations, but we did not consider reasoning tasks over this problem. Therefore, it is important in the future to consider reasoning tasks such as existence and recognition of solutions and analyze their complexity in the *DL-Lite*$_{\mathcal{A}}$ scenario.

SUPPLEMENT I

Explaining TBox Reasoning

In Chapter 5, we introduced a new procedure for the explanation of ontology unsatisfiability, which runs in polynomial time with respect to the given TBox. Borgida et. al in [9] have introduced a procedure for explaining concept and role unsatisfiability, which runs in exponential time w.r.t. the size of the TBox. However, it is known that for *DL-Lite*$_\mathcal{A}$ the problem of checking concept/role satisfiability can be reduced to the satisfiability problem for *DL-Lite*$_\mathcal{A}$ ontologies [13]. Therefore, the explanation problem for concept and role unsatisfiability can be solved in polynomial-time in the size of the TBox.

The introduction of optimal explanation algorithms for TBox reasoning is important also in the context of explanations of query answers. For instance, the absence of a solution to a Query Abduction Problem may be caused by the inconsistency of a concept. Hence, a complete explanation system should be able to motivate further the absence of a solution in terms of TBox reasoning. Therefore, the aim of this supplementary chapter is to provide new polynomial-time algorithms for the problem of explaining TBox reasoning. That is, we focus on explaining subsumption and disjointness among concepts and roles, and, the unsatisfiability of concepts and roles.

The new algorithms we propose are based on the Floyd-Warshall algorithm, which allows the design of explanation procedures that run in polynomial time and space w.r.t. the size of the TBox. This is a major improvement with respect to the explanation procedure for concept unsatisfiability introduced in [9], which runs in exponential time in the size of the TBox.

I.1 Floyd-Warshall Algorithm

The Floyd-Warshall Algorithm [25] is a graph analysis algorithm for finding shortest paths in a weighted graph. The algorithm is based on dynamic programming and a single execution finds the costs of the shortest paths between all pairs of vertices in the graph.

The algorithm works by comparing all the paths between each pair of nodes in the graph and stores the cost of the optimal path. Let $G = (V, E)$ be a weighted graph, where V is a set of nodes numbered from 1 to n and E the set of edges, then in worst case the algorithm takes $O(|V|^3)$ to compute all the shortest paths, i.e. it's polynomial in the number of vertexes.

The algorithm can be defined recursively by considering a function $shortPath(i, j, k)$ (where i, j, k are nodes) that returns the shortest possible path from i to j using only vertices 1 to k as intermediate points along the way. Now, given this function and in order to apply induction, our goal is to find the shortest path from each i to each j using only vertices 1 to $k + 1$. There are only two possibilities:

1. the true shortest path uses only nodes from 1 to k, or
2. there exists some path that goes from i to $k+1$ and then from $k+1$ to j, which is optimal.

Assume there exists a better path from i to j passing through $k + 1$, then this path will be the concatenation of the shortest path from i to $k + 1$ (using $\{1, \ldots, k\}$ nodes) and the shortest path from $k + 1$ to j (also using the same nodes). Therefore, we can define the function in a recursive fashion as follows:

$$shortPath(i, j, k) = min\{shortPath(i, j, k), shortPath(i, k, k-1) + shortPath(k, j, k-1)\}$$

where the base case ($k = 0$) is defined as follows:

$$shortPath(i, j, 0) = edgeCost(i, j)$$

The algorithm repeatedly computes the shortest paths by increasing k until $k = n$, i.e., the algorithm searches for all the shortest path passing through any node.

As it was mentioned before the algorithm is implemented in a linear programming manner as it is clear from Algorithm I.1. At each step in the algorithm, $path[i][j]$ is the shortest path from i to j using intermediate vertices $\{1, \ldots, k+1\}$. Each $path[i][j]$ is initialized to $edgeCost(i, j)$ or infinity if there is no edge between i and j.

```
1: **for** $k$ in $V$ **do**
2:   **for** $i$ in $V$ **do**
3:     **for** $j$ in $V$ **do**
4:       **if** $path[i][k] + path[k][j] < path[i][j]$ **then**
5:         $path[i][j] = path[i][k] + path[k][j]$
6:       **end if**
7:     **end for**
8:   **end for**
9: **end for**
```

Algorithm I.1: function FloydWarshall(), where $path$ is a 2-dimensional matrix.

Path reconstruction

The standard version of the algorithm stores only the cost of shortest path between two nodes, but no information about the path itself. The algorithm can be slightly modified in order to permit the reconstruction of the shortest path between two nodes. Information to reconstruct all paths can be stored in a single matrix $next$, where $next[i][j]$ stores the vertex one must travel through if one intends to take the shortest path from i to j. The $next$ matrix is updated along with the $path$ matrix such that at completion both tables are complete and accurate. The path from i to j is then the path from i to $next[i][j]$, followed by path from $next[i][j]$ to j. These two shorter paths are determined recursively. This modified algorithm has the same space and time complexity as the standard one.

I.2 Explanation Algorithms

TBox reasoning is probably the most well-known class of reasoning, which is used to uncover implicit knowledge that is present in the terminological axioms. All the following explanation algorithms are based on the Floyd-Warshall algorithm outlined in the previous section and hence are made of two phases:

- an *initialization phase* that has to be carried out each time the ontology is modified, which populates the two datastructures used in the graph algorithm,
- an *explanation phase*, in which the datastructures are used in order to explain a specific reasoning service (e.g. concept subsumption).

We now introduce explanation algorithms for important reasoning tasks, namely Role/Concept subsumption, Role/Concept Disjointness and Role/Concept unsatisfiability.

Role Subsumption

A role expression R_1 is said to be subsumed by R_2 given a TBox $\mathcal{T}$ if and only if for all models $\mathcal{I}$ we have that $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$. Due to the simplicity of the *DL-Lite*$_{\mathcal{A}}$ language in terms of expressivity, it is possible to devise a structural algorithm deciding role subsumption. According to [9], there are only four reasons why a role inclusion can be entailed:

1. R_1 is unsatisfiable, i.e., $R_1^{\mathcal{I}} = \emptyset$ and it trivially follows that $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$;
2. $\{R_1 \sqsubseteq R_2\} \subseteq \mathcal{T}$, that is, the subsumption is explicitly stated in the TBox;
3. R_1 and R_2 are syntactically the same concept expression and hence subsumption is valid because it is a reflexive relation;
4. there exists a role expression R_k such that $\{R_1 \sqsubseteq R_k\} \sqsubseteq \mathcal{T}$ and $\{R_k \sqsubseteq R_2\} \sqsubseteq \mathcal{T}$, recursively.

It follows that explaining role subsumption is equivalent to finding the shortest path between two vertices in the graph of role expressions, where the edge relation corresponds to the given subsumption axioms and reflexive edges are added for each vertex. The case in which the role expression R_1 is unsatisfiable is explained in terms of the reason leading to this unsatisfiability, because it is more informative to the user. Please note that these four rules correspond to the explanation rules: *Given*, *Transitivity* and *Reflexivity* which are introduced in the above mentioned paper.

```
1: for $R_2$ (Role expr) in $\mathcal{T}$ do
2:     for $R_1$ (Role expr) in $\mathcal{T}$ do
3:         for $R_3$ (Role expr) in $\mathcal{T}$ do
4:             if $ri[R_1][R_2] + ri[R_2][R_3] < ri[R_1][R_3]$ then
5:                 $ri[R_1][R_2] = ri[R_1][R_2] + ri[R_2][R_3]$
6:                 $nextRole[R_1][R_3] = R_2$
7:             end if
8:         end for
9:     end for
10: end for
```

Algorithm I.2: function INITIALIZERCI($\mathcal{T}$)

Then, the matrix of the Floyd-Warshall algorithm containing the cost of traversing the shortest path from any two nodes is initialized as follows:

$$\begin{aligned}\forall R_1, R_2 \in \mathcal{T}.\, ri[R_1][R_2] = \infty\\ \forall R \in \mathcal{T}.\, ri[R][R] = 0\\ \forall R_1, R_2 \in \mathcal{T}.\, \mathcal{T} \models R_1 \sqsubseteq \bot \rightarrow ri[R_1][R_2] = ri[R_1^-][R_2^-] = 0.5\\ \forall R_1, R_2 \in \mathcal{T}.\, \{R_1 \sqsubseteq R_2\} \subseteq \mathcal{T} \rightarrow ri[R_1][R_2] = ri[R_1^-][R_2^-] = 1\end{aligned}$$

where R_1 and R_2 are role expressions in the TBox. That is, if there is no subsumption between two different roles in the TBox, the cost is infinite. Reflexive subsumption relations are preferred over given ones and are assigned a null cost. Similarly for subsumption generated by an inconsistent subsumee, the cost is 0.5. For any given subsumption relation, the cost of traversing the relation is 1. The reason for these different costs is to always choose the explanation that is simpler, in the sense that we want to reward the explanations based on reflexive subsumptions and on inconsistent concepts. In this way, the algorithm will not try to find a more complex explanation, if it does not have to.

Algorithm I.2 uses the transitivity rule in order to compute the cost of all the subsumption relations among roles occurring in $\mathcal{T}$. Additionally, it computes the matrix $nextRole$ which allows to reconstruct the shortest path explaining the subsumption among two role expressions.

At this point, it is possible to define the explanation procedure $explainRCI$ (see Algorithm I.3). Given two roles R_1, R_2, the procedure first checks whether R_1 is inconsistent w.r.t. the TBox $\mathcal{T}$. If this is the case, then we call the routine explaining the unsatisfiability of R_1 w.r.t. $\mathcal{T}$. Otherwise, if $R_1 = R_2$, then we explain the subsumption with a reflexivity argument. Similarly, if $\{R_1 \sqsubseteq R_2\} \subseteq \mathcal{T}$, then the axiom in the TBox is shown to the user. Finally, we use the transitivity rule to explain subsumptions caused by the interactions of two or more axioms.

Correctness of the algorithm The correctness follows implicitly from the correctness of the structural procedure outlined by Borgida et. al in [9].

```
1: if $\mathcal{T} \models R_1 \sqsubseteq \bot$ then
2:     print $R_1 \sqsubseteq R_2$ because $explainUnsat(R_1)$
3: end if
4: if $R_1 = R_2$ then
5:     print Reflexivity $R_1 \sqsubseteq R_2$
6: end if
7: if $R_1 \sqsubseteq R_2 \in \mathcal{T}$ then
8:     print Given $R_1 \sqsubseteq R_2$
9: end if
10: $R_k = nextRole[R_1, R_2]$
11: print $explainRCI(R_1, R_k) + explainRCI(R_k, R_2) + \textit{ISA-Trans}(R_1 \sqsubseteq R_2)$
```

Algorithm I.3: function EXPLAINRCI(R_1, R_2)

Complexity of the algorithm The fact that the algorithm runs in polynomial time follows straightforwardly from the asymptotic complexity of the Floyd-Warshall algorithm, which is $O(|V|^3)$. In this case, the number of vertexes in the graph is bounded by the number of roles occurring in the TBox $\mathcal{T}$.

Concept Subsumption

A concept expression C is said to be subsumed by the expression D w.r.t. a TBox $\mathcal{T}$ if and only if for all interpretations $\mathcal{I}$ satisfying $\mathcal{T}$, we have that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

Explaining concept subsumption is a similar problem to the explanation of role inclusions. The only difference is that in explaining concepts inclusions we have to deal with an additional structural rule, which is the following one: if $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ then, $\exists Q_1 \sqsubseteq \exists Q_2$ and $\exists Q_1^- \sqsubseteq \exists Q_2^-$ are both entailed subsumptions.

The matrix of the Floyd-Warshall algorithm containing the cost of shortest paths is initialized as before:

$$\begin{aligned}
\forall C, D \in \mathcal{T}.\, ci[C][D] &= \infty \\
\forall C \in \mathcal{T}.\, ci[C][C] &= 0 \\
\forall C, D \in \mathcal{T}.\, \mathcal{T} \models C \sqsubseteq \bot \rightarrow ci[C][D] &= 0.5 \\
\forall C, D \in \mathcal{T}.\, \{C \sqsubseteq D\} \subseteq \mathcal{T} \rightarrow ci[C][D] &= 1
\end{aligned}$$

where C and D are concept expressions in the TBox. But the way the matrix is filled in with the missing costs is different. In fact, a part from the usual transitivity relation, we have to check whether there are shortest paths derived from role inclusions. For this reason, the second part of Algorithm I.4 traverses the role inclusion matrix searching for shortest paths and it applies the new structural rule. That is, we search for two roles R_1, R_2 with $\mathcal{T} \models R_1 \sqsubseteq R_2$, such that the cost associated to $\exists R_1 \sqsubseteq \exists R_2$ is higher than computing the explanation for $R_1 \sqsubseteq R_2$. The algorithm treats in a similar way the case regarding inverse roles.

Then, the explanation routine EXPLAINPCI (see Algorithm I.5) is extended to cope with this additional structural rule. Given the two concepts A and B and the two data-structures ci

```
for $C$ (Concept expr) in $\mathcal{T}$ do
  for $D$ (Concept expr) in $\mathcal{T}$ do
    for $E$ (Concept expr) in $\mathcal{T}$ do
      if $ci[D][C] + ci[C][E] < ci[D][E]$ then
        $ci[D][E] = ci[D][C] + ci[C][E]$
        $nextConcept[D][E] = C$
      end if
    end for
  end for
end for
for $Q_2$ (Role expr) in $\mathcal{T}$ do
  for $Q_1$ (Role expr) in $\mathcal{T}$ do
    if $ri[Q_1][Q_2] < ci[\exists Q_1][\exists Q_2]$ then
      $ci[\exists Q_1][\exists Q_2] = ri[Q_1][Q_2]$
      $nextConcept[\exists Q_1][\exists Q_2] = null$
    end if
  end for
end for
for $Q_2$ (Role expr) in $\mathcal{T}$ do
  for $Q_1$ (Role expr) in $\mathcal{T}$ do
    if $ri[Q_1][Q_2] < ci[\exists Q_1{}^-][\exists Q_2{}^-]$ then
      $ci[\exists Q_1{}^-][\exists Q_2{}^-] = ri[Q_1][Q_2]$
      $nextConcept[\exists Q_1{}^-][\exists Q_2{}^-] = null$
    end if
  end for
end for
```
Algorithm I.4: function INITIALIZEPCI($\mathcal{T}$)

```
if $\mathcal{T} \models A \sqsubseteq \bot$ then
  print $A \sqsubseteq B$ because $explainUnsat(A)$
end if
if $nextConcept[A][B] == \infty$ and $A = B$ then
  print Reflexivity $A \sqsubseteq A$
end if
if $nextConcept[A][B] == \infty$ and $A \sqsubseteq B \in \mathcal{T}$ then
  print Given $A \sqsubseteq B$
end if
$K = nextConcept[A, B]$
if $K == null$ and $A = \exists R_1, B = \exists R_2$ then
  return $explainRCI(R_1, R_2)$
end if
print $explainPCI(A, K) + explainPCI(K, B) + ISA\text{-}Trans(A \sqsubseteq B)$
```
Algorithm I.5: function EXPLAINPCI(A,B)

and $nextConcept$ produced in the initialization phase, the algorithm works as follows. It first checks that A is unsatisfiable, in which case it calls the explanation routine justifying concept inconsistency. Otherwise, the algorithm deals in a similar way with the reflexivity and given cases. Then, before applying transitivity, it checks whether $A = \exists R_1$ and $B = \exists R_2$ and that $nextConcept[A, B] = null$. If this is the case, then by construction of the data-structure we know that the shortest path justifying the subsumption is the one explaining $R_1 \sqsubseteq R_2$. Hence, we use EXPLAINRCI to compute an explanation for $R_1 \sqsubseteq R_2$. If $K \neq null$, then we apply the usual transitivity rule.

Correctness of the algorithm The algorithm mimics the structural procedure for computing concept subsumptions. The claim follows.

Complexity of the algorithm The algorithm runs in polynomial time w.r.t. the size of the TBox. As for the previous algorithm, the predominant cost is the run of the Floyd-Warshall algorithm, which runs in $O(|\mathcal{T}|^3)$.

Concept Disjointness

Let us now focus on the explanation of concept disjointness, two concept expressions are said to be disjoint w.r.t. to the TBox $\mathcal{T}$ ($\mathcal{T} \models A \sqsubseteq \neg B$) if for all models $\mathcal{I}$ of $\mathcal{T}$ we have that $A^{\mathcal{I}} \cap B^{\mathcal{I}} = \emptyset$. Obviously, if either A or B are unsatisfiable, it follows that they are disjoint as well.

In this case, the matrix containing the costs of the shortest paths between two disjoint concepts is initialized as follows:

$$\begin{aligned} \forall C, D \in \mathcal{T}.\, disj[C][D] = \infty \\ \forall C \forall D \in \mathcal{T}. \mathcal{T} \models C \sqsubseteq \bot \rightarrow disj[C][D] = 0.5 = disj[D][C] \\ \forall D \forall C \in \mathcal{T}. \mathcal{T} \models D \sqsubseteq \bot \rightarrow disj[C][D] = 0.5 = disj[D][C] \\ \forall C, D \in \mathcal{T}.\, \{C \sqsubseteq \neg D\} \subseteq \mathcal{T} \rightarrow disj[C][D] = 1 = disj[D][C] \end{aligned}$$

That is, disjoint relationships generated by inconsistent concepts are rewarded against explicit disjoint axioms. This is to avoid long explanations for a disjoint relationship, which is generated by a single inconsistent concept. Note that, since the disjointness relation is symmetric, at the implementation level only half matrix can be stored.

Algorithm I.6 shows how to compute the shortest *disjointness* path between two concepts. Implicit disjointness are computed using the inductive definition of the negative closure of the TBox (see Section 2.3). More specifically, we use rules (3)-(5) which are the only ones introducing concept disjointness axioms among concept expressions. Assume that we want to compute the cost of $D \sqsubseteq \neg E$. Then, we have to search for a concept C, such that *(a)* $D \sqsubseteq C$ and *(b)* $C \sqsubseteq \neg E$. The cost of $D \sqsubseteq \neg E$ is simply the sum of the cost of *(a)* and *(b)*. Note that, rules (4)-(5) are taken into account in the algorithm, since by the way matrix *ci* is computed, domain and range inclusions of roles are already considered. As usual, the algorithm maintains information on how to reconstruct the shortest path.

```
1: for $C$ (Concept expr) in $\mathcal{T}$ do
2:     for $D$ (Concept expr) in $\mathcal{T}$ do
3:         for $E$ (Concept expr) in $\mathcal{T}$ do
4:             if $ci[D][C] + disj[C][E] < disj[D][E]$ then
5:                 $disj[D][E] = ci[D][C] + disj[C][E]$
6:                 $disj[E][D] = disj[D][E]$
7:                 $nextDisj[D][E] = C = nextDisj[D][E]$
8:             end if
9:         end for
10:    end for
11: end for
```

Algorithm I.6: function INITIALIZEDISJOINTNESS($\mathcal{T}$)

Algorithm I.7 takes as input two concept expressions A and B such that $\mathcal{T} \models A \sqsubseteq \neg B$. As with concept subsumption, the procedure checks whether the disjointness is generated by an inconsistency, in such a case the disjointness is explained to the user in terms of the cause of the unsatisfiability of the concept. Otherwise, if the disjointness path is not complex, i.e., it is not involving any other vertexes apart from A and B, the *given* rule is used for the explanation. In case of a complex path, the explanation is recursively computed using the *disjointness structural rule*. This structural rule combines the explanation of a positive concept inclusion and the explanation of a disjointness among concepts. For this reason, in order to compute the correct subsumption explanation, the EXPLAINPCI algorithm is used. In this way, the user is provided with a comprehensive explanation of the disjointness, which intrinsically depends on an explanation of a concept subsumption.

```
1: if $A$ or $B$ is inconsistent then
2:     print $A \sqsubseteq \neg B$ because explainUnsat(A or B)
3:     return
4: end if
5: if $nextDisj[A][B] == \infty$ then
6:     print Given $A \sqsubseteq \neg B$
7: end if
8: $K = nextDisj[A, B]$
9: print $explainPCI(A, K) + explainDisj(K, B) + Disjointness(A \sqsubseteq \neg B)$
```

Algorithm I.7: function EXPLAINDISJ(A,B)

Correctness of the algorithm The algorithm computes implied disjointness axioms using structural rules derived from the definition of $cln(\mathcal{T})$. Also, algorithm EXPLAINPCI is used to guarantee a complete explanation of the reasons leading to the disjointness of two concepts.

Complexity of the algorithm The algorithm runs in polynomial time w.r.t. the size of the TBox. The argument is similar to the ones used before.

The explanation of role disjointness can be performed in a very similar way. In the following, we assume to have a matrix $roleDisj$, which is the equivalent to $disj$ for role expressions.

Concept and Role Unsatisfiability

A concept expression C (role expression R) is said to be unsatisfiable w.r.t. TBox $\mathcal{T}$ if and only if there is no model $\mathcal{I}$ of $\mathcal{T}$, such that $C^{\mathcal{I}} \neq \emptyset$ ($R^{\mathcal{I}} \neq \emptyset$). As it was highlighted in [9], there are three main reasons why a concept could result in being unsatisfiable:

1. the concept C is subsumed by two concepts that are disjoint;
2. the concept C is subsumed by a concept expression of the for $\exists R$ ($\exists R^-$) and R is unsatisfiable; or,
3. C is subsumed by an unsatisfiable concept.

Similarly, a role can only be unsatisfiable due to subsumption by another unsatisfiable role, subsumption by disjoint roles, or due to its current domain or range being unsatisfiable.

In oder to take these reasons into account, we define two different vectors: $cUnsat[X]$ and $rUnsat[X]$, which store the cost of the unsatisfiability of concept/role X. The vectors are initialized as follows:

$$\forall C.\ C \sqsubseteq \neg C \in \mathcal{T}\ cUnsat[C] = 0 \qquad \forall R.\ R \sqsubseteq \neg R \in \mathcal{T}\ rUnsat[R] = 0$$

```
1:  for $A, B$ such that $disj[A][B] \neq \infty$ do
2:      for $C$ such that $ci[C][A] \neq \infty$ and $ci[C][B] \neq \infty$ do
3:          if $ci[C][A] + ci[C][B] + disj[A][B] < cUnsat[C]$ then
4:              $cUnsat[C] = ci[C][A] + ci[C][B] + disj[A][B]$
5:              $nextCon[C] = (A, B)$
6:          end if
7:      end for
8:  end for
9:  for $A, B$ such that $ci[A][B] \neq \infty$ and $cUnsat[B] \neq \infty$ do
10:     if $ci[A][B] + cUnsat[B] < cUnsat[A]$ then
11:         $cUnsat[A] = ci[A][B] + cUnsat[B]$
12:         $nextCon[A] = (B)$
13:     end if
14: end for
15: for $A, \exists R$ such that $ci[A][\exists R] \neq \infty$ and $rUnsat[R] \neq \infty$ do
16:     if $ci[A][\exists R] + rUnsat[R] < cUnsat[A]$ then
17:         $cUnsat[A] = ci[A][\exists R] + rUnsat[R]$
18:         $nextCon[A] = (\exists R, R)$
19:     end if
20: end for
```

Algorithm I.8: function COMPUTECUNSAT($\mathcal{T}$)

Let us now detail how the cost of the unsatisfiability of a concept expression C is determined. Algorithm I.8 works upon the three reasons leading to unsatisfiability in order to compute the correct cost of unsatisfiability. It first searches for two disjoint concepts that both subsume C. Then, it checks whether there exists a subsuming concept D which is known to be unsatisfiable. Finally, it searches for a concept expression $\exists R$ that subsumes C for which R is unsatisfiable. The algorithm recursively computing the cost of role unsatisfiability works in a similar way (see Algorithm I.9). Please note that the content of the two vectors depend on one another. For this reason, the computations have to be ran in parallel and iteratively, until no new modification to the contents of the two vectors can be made. This process is, anyway, guaranteed to terminate in polynomial time, given that the number of implied inconsistent concepts and roles is polynomially bounded by the size of the TBox.

```
1: for $R', R''$ such that $roleDisj[R'][R''] \neq \infty$ do
2:   for $R$ such that $ri[R][R'] \neq \infty$ and $ri[R][R''] \neq \infty$ do
3:     if $ri[R][R'] + ri[R][R''] + disj[R'][R''] < rUnsat[R]$ then
4:       $rUnsat[R] = ri[R][R'] + ri[R][R''] + disj[R'][R'']$
5:       $nextRole[R] = (R', R'')$
6:     end if
7:   end for
8: end for
9: for $R, R'$ such that $ri[R][R'] \neq \infty$ and $rUnsat[R'] \neq \infty$ do
10:   if $ri[R][R'] + rUnsat[R'] < rUnsat[R]$ then
11:     $rUnsat[R] = ri[R][R'] + rUnsat[R']$
12:     $nextRole[R] = (R')$
13:   end if
14: end for
15: for $R$ such that $cUnsat[\exists R] \neq \infty$ do
16:   if $cUnsat[\exists R] < rUnsat[R]$ then
17:     $rUnsat[R] = cUnsat[\exists R]$
18:     $nextRole[R] = (\exists R)$
19:   end if
20: end for
```

Algorithm I.9: function COMPUTERUNSAT($\mathcal{T}$)

Finally, Algorithm I.10 and I.11 compute the explanation of the unsatisfiability of a concept or role by distinguishing the different cases and applying the different structural rules.

Correctness of the algorithm The algorithm works upon the three different structural rules that can be used to determine the inconsistency of a concept or role. Also, the fact that the content of the two vectors is computed in a inductive fashion, it guarantees that every possible unsatisfiability source is inspected and that we always compute minimal solutions.

```
if $A \sqsubseteq \bot \in \mathcal{T}$ then
    print Given $A \sqsubseteq \bot$
    return
end if
if $A \sqsubseteq \neg A \in \mathcal{T}$ then
    print Given $A \sqsubseteq \neg A$
    return
end if
if $nextCon[A] == (C, D)$ with $A, B$ concepts then
    print $explainPCI(A, C) + explainPCI(A, D) + explainDisj(C, D)$
end if
if $nextCon[A] == (C, R)$ with $R$ role then
    print $explainPCI(A, C) + explainRoleUnsat(R)$
end if
if $nextCon[A] == (B)$ then
    print $explainPCI(A, B) + explainConceptUnsat(B)$
end if
```

Algorithm I.10: function EXPLAINCONCEPTUNSAT(A)

```
if $R \sqsubseteq \bot \in \mathcal{T}$ then
    print Given $R \sqsubseteq \bot$
    return
end if
if $A \sqsubseteq \neg A \in \mathcal{T}$ then
    print Given $R \sqsubseteq \neg A$
    return
end if
if $nextRole[R] == (R', R'')$ with $R', R''$ concepts then
    print $explainRCI(R, R') + explainRCI(R, R'') + explainDisj(R', R'')$
end if
if $nextRole[R] == (\exists R)$ then
    print $explainConceptUnsat(\exists R)$
end if
if $nextRole[R] == (R')$ then
    print $explainRCI(A, R') + explainRoleUnsat(R')$
end if
```

Algorithm I.11: function EXPLAINROLEUNSAT(R)

Complexity of the algorithm The computation of the two vectors does not require more than a polynomial number of steps in the size of the TBox. The reason is that the number of concept and role that one can deduce to be unsatisfiable is polynomially bounded by the size of the TBox. Also, computing the cost of an unsatisfiable concept or role can be done in polynomial-time, since the computation is based on the Floyd-Warshall algorithm. Hence, the whole algorithm runs in polynomial time.

In this supplementary chapter, we have introduced new algorithms for the explanation of key reasoning tasks over *DL-Lite*$_{\mathcal{A}}$ TBoxes. The procedures are built upon the famous Floyd-Warshall algorithm for finding shortest paths over weighted graphs. This basis allows us to provide algorithms which are both polynomial in time and space with respect to the size of the TBox.

APPENDIX A

Dijkstra's Algorithm

1 INPUT: A query graph G, a source query.

2 OUTPUT: A binary tuple: the first component contains all the costs of all shortest path from s, the second component contains information on how to reconstruct the paths.

```
1:
2: for vertex v in G do
3:     dist[v] := infinity
4:     previous[v] := undefined
5: end for
6: dist[source] := 0
7: Q := the set of all nodes in Graph
8: while Q is not empty do
9:     u := vertex in Q with smallest dist[]
10:    if dist[u] = infinity then
11:        break
12:    end if
13:    remove u from Q
14:    for neighbor v of u where v has not yet been removed from Q do
15:        $alt := dist[u] + dist_between(u, v)$
16:        if alt < dist[v] then
17:            dist[v] := alt
18:            previous[v] := u
19:        end if
20:    end for
21: end while
22: return (dist[], previous[])
```

Algorithm A.1: function Dijkstra(G, s)

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arenas, L. Bertossi, and J. Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Flexible query answering systems: recent advances: proceedings of the Fourth International Conference on Flexible Query Answering Systems, FQAS'2000, October 25-28, 2000, Warsaw, Poland*, page 27. Springer Berlin Heidelberg, 2001.

[3] O. Arieli, M. Denecker, B. Van Nuffelen, and M. Bruynooghe. Database repair by signed formulae. *Foundations of Information and Knowledge Systems*, pages 14–30, 2004.

[4] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The DL-Lite Family and Relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009.

[5] F. Baader. *The Description Logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.

[6] F. Baader, M. Bienvenu, C. Lutz, and F. Wolter. Query and predicate emptiness in Description Logics. *Proc. of KR*, 10:324, 2010.

[7] M. Bada, C. Mungall, and L. Hunter. A call for an abductive reasoning feature in OWL-Reasoning tools toward ontology quality control. In *Proceedings of the 5th International Workshop OWL: Experiences and Directions*, volume 2008, 2008.

[8] P. Bonatti, C. Lutz, and F. Wolter. Description Logics with circumscription. *Proc. KR*, 6:400–41O, 2006.

[9] A. Borgida, D. Calvanese, and M. Rodriguez-Muro. Explanation in the DL-Lite Family of Description Logics. In *Proc. of the 7th Int. Conf. on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008)*, volume 5332 of *Lecture Notes in Computer Science*, pages 1440–1457. Springer, 2008.

[10] A. Borgida, E. Franconi, I. Horrocks, D.L. McGuinness, and P.F. Patel-Schneider. Explaining ALC subsumption. In *ECAI*, pages 209–213, 2000.

[11] R.J. Brachman and H. Levesque. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3(2):78–93, 1987.

[12] S.R. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991.

[13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and Databases: The DL-Lite Approach. In S. Tessaris and E. Franconi, editors, *Semantic Technologies for Informations Systems - 5th Int. Reasoning Web Summer School (RW 2009)*, volume 5689 of *Lecture Notes in Computer Science*, pages 255–356. Springer, 2009.

[14] A. Chapman and H.V. Jagadish. Why not? In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 523–534. ACM, 2009.

[15] A. Deutsch, A. Nash, and J. Remmel. The chase revisited. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 149–158. ACM, 2008.

[16] T. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the ACM (JACM)*, 42(1):42, 1995.

[17] C. Elsenbroich, O. Kutz, and U. Sattler. A case for abductive reasoning over ontologies. *Proc. OWL: Experiences and Directions, Athens, Georgia, USA*, 2006.

[18] T. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5-6):907–928, November 1995.

[19] C.G. Hempel and P. Oppenheim. Studies in the logic of explanation. *Philosophy of Science*, 15(2):135–175, 1948.

[20] I. Horrocks. Ontologies and the semantic web. *Communications of the ACM*, 51(12):58–67, 2008.

[21] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, 2006.

[22] J. Huang, T. Chen, A.H. Doan, and J.F. Naughton. On the provenance of non-answers to queries over extracted data. *Proceedings of the VLDB Endowment*, 1(1):736–747, 2008.

[23] S. Klarman, U. Endriss, and S. Schlobach. ABox Abduction in the Description Logic ALC. *Journal of Automated Reasoning*, 2010.

[24] A. Krisnadhi and C. Lutz. Data complexity in the EL family of Description Logics. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 333–347. 2007.

[25] E.L. Lawler. *Combinatorial optimization: networks and matroids.* Dover Pubns, 2001.

[26] C. Lutz. Inverse roles make conjunctive queries hard. In *Proceedings of the 20th International Workshop on Description Logics (DL 2007)*, volume 106, pages 150–153, 2007.

[27] C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the Description Logic EL using a relational database system. In *Proc. of IJCAI*, volume 9, 2009.

[28] D.L. McGuinness and A. Borgida. *Explaining subsumption in Description Logics*. 1994.

[29] D.L. McGuinness and P. Pinheiro da Silva. Explaining answers from the Semantic Web: The Inference Web approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):397–413, 2004.

[30] C.H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[31] R. Peñaloza and B. Sertkaya. Complexity of axiom pinpointing in the DL-Lite family of Description Logics. In *Proceeding of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 29–34. IOS Press, 2010.

[32] R. Peñaloza and B. Sertkaya. On the complexity of axiom pinpointing in the EL family of Description Logics. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, 2010.

[33] Y. Peng and J.A. Reggia. *Abductive inference models for diagnostic problem-solving. Symbolic Computation*. Springer-Verlag New York, Inc, 1990.

[34] I.A. Stewart. Complete problems involving boolean labelled structures and projection transactions. *Journal of Logic and Computation*, 1(6):861, 1991.

[35] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Journal of Math*, 58:345–363, 1938.

[36] M.Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

[37] J. Woodward. Scientific explanation. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2010 edition, 2010.