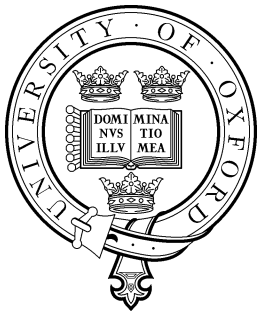
Tractable Query Answering for Description Logics via Query Rewriting



Héctor Manuel Pérez-Urbina
St. John's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2009

A mis furpas.

Abstract

We consider the problem of answering conjunctive queries over description logic knowledge bases via query rewriting. Given a conjunctive query Q and a TBox $\mathcal{T}$, we compute a new query Q' that incorporates the semantic consequences of $\mathcal{T}$ such that, for any ABox $\mathcal{A}$, evaluating Q over $\mathcal{T}$ and $\mathcal{A}$ can be done by evaluating the new query Q' over $\mathcal{A}$ alone. We present RQR—a novel resolution-based rewriting algorithm for the description logic $\mathcal{ELHIO}^{\neg}$ that generalizes and extends existing approaches. RQR not only handles a spectrum of logics ranging from DL-Lite$_{core}$ up to $\mathcal{ELHIO}^{\neg}$, but it is worst-case optimal with respect to data complexity for all of these logics; moreover, given the form of the rewritten queries, their evaluation can be delegated to off-the-shelf (deductive) database systems. We use RQR to derive the novel complexity results that conjunctive query answering for $\mathcal{ELHIO}^{\neg}$ and DL-Lite^{+} are, respectively, PTime and NLogSpace complete with respect to data complexity. In order to show the practicality of our approach, we present the results of an empirical evaluation. Our evaluation suggests that RQR, enhanced with various straightforward optimizations, can be successfully used in conjunction with a (deductive) database system in order to answer queries over knowledge bases in practice. Moreover, in spite of being a more general procedure, RQR will often produce significantly smaller rewritings than the standard query rewriting algorithm for the DL-Lite family of logics.

The Author

Héctor Manuel Pérez-Urbina received his Bachelor's degree in Computer Systems Engineering from the Universidad de las Américas, Puebla in Mexico in December 2004. Shortly after that, he spent a few months at the École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble in France as a research intern. He worked on ontology-based data access under the supervision of Dr. Genoveva Vargas-Solar and Prof. Christine Collet at the Logiciels, Systèmes, Réseaux Laboratory.

Héctor started his Ph.D. at the University of Manchester in January 2006, under the supervision of Prof. Ian Horrocks and Prof. Ulrike Sattler. He was then transferred to the University of Oxford in October 2007 working under the supervision of Prof. Ian Horrocks and Dr. Boris Motik. This document summarizes his research work in the fields of description logics and resolution-based theorem proving. The main topic of his research is the problem of answering conjunctive queries over description logic knowledge bases via query rewriting.

Acknowledgements

Unfortunately, mentioning all the people I am grateful to would take a prohibitively large amount of space. I, however, cannot lose the opportunity to express my gratitude to those without whom I would not be writing these words.

Ian, despite your busy schedule you always had time for a friendly chat. You are in many respects the type of person I would like to become someday: knowledgeable and respected but utterly humble and kind. I will miss your advice and guidance as much as the terrific squash and tennis rallies.

Boris, in spite of the difficulties we faced along the way, I am very happy to say that you have become a true friend. Thank you for your patience and for never losing faith in me. I honestly could not have done this without you.

Cristina, no matter how bad times were, I instantly felt better just by remembering your sweet smile. You were the extra strength I needed not to fall apart. Beyond any doubt, your love is the most valuable treasure I am taking from Oxford.

Furpa and murma, I cannot express my gratitude for your unconditional love and support. Thank you for always believing in me. Putting in practice everything you taught me has taken me to where I am now. We have sown together for many years, it is finally time to reap the rewards.

Finally, I want to express my deep gratitude to the people and government of Mexico who, through the Secretariat of Public Education and the National Council of Science and Technology, kindly provided me with generous scholarships without which I could not have fulfilled this dream.

Contents

List of Figures

List of Tables

Part I

Foundations

Chapter 1

Introduction

One of the major areas of present day research in the field of computer science is concerned with the automated processing of information. The demand for information management systems keeps growing as more and more data sources become available. One branch of the research in this area led to the development of the field of *knowledge representation and reasoning*. Research in this area is concerned with the design of well-founded formalisms that can be used to represent application domains and to *reason* about them in a systematic and automated way. In this thesis, we are mainly concerned with a family of such formalisms called Description Logics (DLs).

In this chapter, we first attempt to convey the motivation for the work contained in the thesis, including an outline of the problem to be addressed. We then state the objectives of the thesis and, finally, we summarized our contributions and present a chapter-by-chapter synopsis of the thesis contents.

1.1 Problem Description and Goals

DLs are a family of knowledge representation formalisms that can be used to represent application domains in terms of individuals, concepts (sets of individuals), and roles (binary relations between individuals). A DL *terminology* or TBox introduces the vocabulary of an application domain by stating *subsumption* relationships between concepts and between roles. Concepts and roles can be *atomic*, which means that they

are denoted by name, or *complex*, which means that they are built using *constructors* that specify necessary and sufficient conditions for concept and role membership, respectively. A particular DL is characterized by the set of concept and role constructors it provides. Apart from the TBox, a DL *Knowledge Base* (KB) usually has an *assertional component* or ABox, which specifies concept and role membership of individuals or pairs of individuals, respectively. DLs have a well-understood model theoretic semantics, which provides a formal foundation for the vague and imprecise semantics of earlier approaches, such as semantic networks [Qui67] or frames [Min81]. This formal semantics allows one to derive new conclusions from a particular KB unambiguously.

A *DL system* or *reasoner* allows users to *reason* about a given KB—that is, to *infer* implicit knowledge from the knowledge explicitly represented by the KB. Typical reasoning problems concerning TBoxes are to determine whether one concept *subsumes* another—that is, whether the set of individuals represented by the former concept necessarily contains the individuals represented by the latter; or whether a concept is *satisfiable*—that is, whether a concept can have any members or *instances* at all. Typical reasoning problems concerning KBs include deciding whether a given individual is an instance of a concept or whether a pair of individuals is an instance of a role. Most reasoning problems can be reduced to checking whether a KB is *satisfiable*—that is, deciding whether it is possible to satisfy all the constraints defined in the TBox and the ABox. Most current reasoners are based on *tableau algorithms* [SSS91] which can be used to decide the satisfiability problem for a given KB.

DLs have been successfully applied to numerous problems, such as the design of software information and documentation systems [DBSB90, HMvdSW04], the configuration of telecommunications systems [BIW94, MW98], and the generation of action plans in robotics [DGINR96], in various domains, such as biology [SDCS05], medicine [GZB06], geography [Goo05], geology [Ras06], astronomy [DRPM06], agri-

culture [SLL+04], and defense [LAF+05]. The performance of tableau-based reasoners, such as FaCT++ [TH06] and RACER [HM01], has usually been found to be adequate for applications mainly requiring terminological reasoning. However, new applications, such as metadata management in the semantic Web or information integration, require efficient query answering over large ABoxes. Unfortunately, as the number of ABox individuals increases, the performance of current tableau-based reasoners deteriorates quite rapidly and is often inadequate in practice.

In an attempt to provide efficient algorithms for reasoning with large ABoxes, Motik [Mot06] studied the relation between DLs and *deductive databases* [CGT90]. Motik proposed a resolution-based algorithm that reduces a $\mathcal{SHIQ}(D)$ KB to a disjunctive datalog program such that the KB and the resulting datalog program entail the same set of ground facts. Thus, standard DL reasoning problems can be solved by using mature deductive database techniques. Motik's reduction algorithm has been implemented in the system KAON2[1] [MS06]. Other similar approaches include the work of Krötzsch et al. [KRH08], who proposed a reduction algorithm for ELP KBs to datalog programs; and the work of Kazakov [Kaz06], who defined a range of resolution-based decision procedures for various logics of the $\mathcal{EL}$ family [BBL05].

The approaches of Motik, Krötzsch et al., and Kazakov have shown that resolution-based techniques can be effectively used for reasoning with DLs, which allows one to make use of a range of optimization techniques that have been developed in the fields of deductive databases and resolution theorem proving. However, in many applications—particularly those that are data-intensive—users are often interested in posing complex queries over KBs that in general cannot be answered by relying on the reasoning problems addressed by the aforementioned approaches. Many of such complex queries can be captured by the language of Conjunctive Queries (CQs)—a well–known query language that has proven to be extremely useful to query rela-

[1] http://kaon2.semanticweb.org/

tional databases [AHV95]. Unfortunately, as the complexity of reasoning for many very expressive DLs is worst-case exponential with respect to *data complexity*—that is, with respect to the size of the ABox—there is little hope for tractable CQ answering algorithms. Therefore, the development of nontrivial DLs for which CQ answering remains tractable with respect to data complexity is of paramount importance for the effective use of DLs in data-intensive applications, such as information integration or the semantic Web.

In an attempt to develop an efficient and scalable CQ answering technique for DLs, Calvanese et al. [CDL+07] developed the so-called DL-Lite family of DLs, for which CQ answering was shown to be in AC^0 with respect to data complexity. The solution to CQ answering proposed by Calvanese et al. is based on the notion of *query rewriting*: given a conjunctive query Q and a TBox $\mathcal{T}$, one computes a *Union of Conjunctive Queries* (UCQ) Q'—a so-called *rewriting* of Q with respect to $\mathcal{T}$—such that, for every ABox $\mathcal{A}$, the answers to Q over $\mathcal{T}$ and $\mathcal{A}$, and the answers to Q' over $\mathcal{A}$ coincide. Thus, the problem of answering Q over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ for a specific $\mathcal{A}$ can be reduced to evaluating the rewriting Q' over $\mathcal{A}$ only. The rewriting algorithm of Calvanese et al. has been implemented in reasoners such as QuOnto,[2] Owlgres,[3] and ONTOSEARCH2;[4] additionally, an extension of this algorithm for an n-ary DL has been implemented in the SoftFacts system.[5] Such an approach to query answering is interesting from a practical perspective because it allows one to use existing database systems to store $\mathcal{A}$ and to evaluate Q', thus taking advantage of the extensive body of research in data storage, indexing, and query evaluation for databases.

Shortly after Calvanese et al., Rosati [Ros07] proposed a CQ rewriting algorithm for the DL $\mathcal{EL}$ [BBL05] that was used to show that CQ answering for $\mathcal{EL}$ is PTIME-complete with respect to data complexity [Ros07, KL07, KR07]. Rosati's algorithm

[2] http://www.dis.uniroma1.it/~quonto/
[3] http://pellet.owldl.com/owlgres/
[4] http://www.aktors.org/technologies/ontosearch/
[5] http://gaia.isti.cnr.it/~straccia/software/SoftFacts/SoftFacts.html

can be used to transform a CQ Q and an $\mathcal{EL}$ TBox $\mathcal{T}$ into a *datalog program* Q' that is a rewriting of Q with respect to $\mathcal{T}$. As in the case of $\mathcal{EL}$ the rewriting Q' is no longer guaranteed to be a UCQ, it might be necessary to use a *deductive* database system—such as IRIS[6] or XSB[7]—for evaluating Q'.

The rewriting techniques of Calvanese et al. and Rosati show that it is possible to use existing highly optimized (deductive) database systems in order to answer CQs over KBs. However, the practicality of these rewriting approaches is not straightforward as the size of the rewriting Q' is worst-case exponential with respect to the size of Q and $\mathcal{T}$ even for the rather inexpressive DL DL-Lite$_R$ [CDL+07] (see Example 7.0.1 for a simple proof). As we shall see, empirical evaluations have shown that realistic TBoxes and queries can result in Q' being extremely large (e.g., containing tens of thousands of clauses). Thus, on the one hand, Q' may be costly to compute, and, on the other hand, evaluation by (deductive) database systems may be costly or even unfeasible. Trying to produce small rewritings is therefore of critical importance to the practical application of the query rewriting approach to CQ answering over KBs.

The main aim of this research is to investigate whether one can employ resolution-based techniques—similar to those used by Motik, Krötzsch et al., and Kazakov—in order to solve the problem of CQ answering for various DLs via query rewriting. In particular, our objective is to develop an efficient resolution-based CQ rewriting algorithm for expressive but yet tractable DLs that produces small rewritings in practice. Our hope is that such an algorithm can be effectively used in practice in order to answer queries over KBs with large ABoxes.

[6] http://www.iris-reasoner.org/
[7] http://xsb.sourceforge.net/

1.2 Contributions

The main contribution of our investigations is a resolution-based CQ rewriting algorithm for the DL $\mathcal{ELHIO}^{\neg}$. This DL is an expressive extension of $\mathcal{EL}$ [BBL05] that allows (limited) concept and role negation, role inclusions, inverse roles, and nominals—concepts that are to be interpreted as singletons (e.g. $\{\mathsf{Mexico}\}$). To the best of our knowledge, the problem of CQ rewriting had never been considered for DLs that allow nominals.

Our rewriting algorithm, called RQR (Resolution-based Query Rewriting), is a slight modification of a resolution-based *CQ answering algorithm for* $\mathcal{ELHIO}^{\neg}$ that employs a novel way of reasoning with equality. Notably, RQR exhibits "pay-as-you-go" behavior: if $\mathcal{T}$ is expressible in $\mathcal{ELHIO}^{\neg}$, then the computed rewriting $\mathsf{RQR}(Q, \mathcal{T})$ is a datalog query (a datalog program with a special query predicate); if $\mathcal{T}$ is expressible in DL-Lite^{+}, then $\mathsf{RQR}(Q, \mathcal{T})$ consists of a UCQ and a linear datalog program; finally, if $\mathcal{T}$ is expressible in DL-Lite$_R$, then $\mathsf{RQR}(Q, \mathcal{T})$ is a UCQ. Therefore, RQR not only handles a variety of DLs ranging from DL-Lite$_{core}$ up to $\mathcal{ELHIO}^{\neg}$, but it is worst-case optimal with respect to data complexity for all these DLs. RQR can thus be seen as a generalization and an extension of the rewriting algorithms of Calvanese et al. and Rosati.

We use RQR in order to derive the novel result that CQ answering for $\mathcal{ELHIO}^{\neg}$ is PTime-complete with respect to data complexity, which makes $\mathcal{ELHIO}^{\neg}$ one of the most expressive Horn logics for which CQ answering is tractable with respect to data complexity. Additionally, we show that CQ answering for DL-Lite^{+} is NLogSpace-complete with respect to data complexity, thus closing the open question of whether CQ answering for this DL could be solved in NLogSpace. Moreover, we present novel proofs for the known results that CQ answering for $\mathcal{ELHI}$, $\mathcal{ELH}$, and $\mathcal{EL}$ is PTime-complete with respect to data complexity [Ros07, KL07, KR07] and in AC^0 for DL-Lite$_R$ and DL-Lite$_{core}$ [CDL^{+}07].

In order to test the practicability of our rewriting approach, we implemented RQR in a system called REQUIEM (REsolution-based QUery rewrIter for Expressive Models) and conducted an empirical evaluation. Our evaluation consisted in using REQUIEM to rewrite typical queries over realistic TBoxes expressible in logics of the DL-Lite and $\mathcal{EL}$ families, as well as using REQUIEM in a real use case—identified by researchers of the Japanese National Institute of Information and Communications Technology—to answer CQs over a KB with a relatively large ABox. Our results suggest that RQR, enhanced with various straightforward optimizations, can be successfully used in conjunction with current (deductive) database technology in order to answer CQs over realistic KBs. Moreover, our evaluation suggests that, in case the TBox does not entail axioms of the form $\exists \mathsf{hasParent}.\mathsf{Human} \sqsubseteq \mathsf{Human}$, RQR can be used to obtain UCQs even for TBoxes that go beyond DL-Lite$_R$. This is a significant result because it means that it will often be possible to use an off-the-shelf database system for CQ answering even for DLs of the $\mathcal{EL}$ family. Finally, RQR will often produce significantly smaller rewritings than the standard query rewriting algorithm for the DL-Lite family developed by Calvanese et al.

1.3 Structure of the Thesis

Part I is mainly concerned with the motivation, objectives, and contributions of our work—which have been summarized above—and with the general theoretical background.

- In Chapter 2 we fix the notation, introduce necessary terminology, and recapitulate relevant definitions and results regarding logic programming and resolution.
- In Chapter 3 we provide a general introduction to DLs. We present a brief summary of their history and application areas; additionally, we present the syntax and semantics of $\mathcal{ELHIO}^{\neg}$ and other DLs considered in this thesis. We

also formally define various relevant reasoning problems, including the problems that we consider in this thesis: CQ answering and rewriting for DLs.

Part II is concerned with the theoretical contributions of our investigations.

- In Chapter 4 we present $\mathsf{RQR}^{\text{lite}}$—a simpler version of RQR scaled down to the DL DL-Lite$_R$. The description of $\mathsf{RQR}^{\text{lite}}$ conveys all the basic ideas of our approach without overloading the presentation with details. Additionally, we present a comparison of $\mathsf{RQR}^{\text{lite}}$ with CGLLR—the rewriting algorithm of Calvanese et al. —as well as a discussion of their relative strengths and weaknesses.

- In Chapter 5 we present RQR. We first describe a resolution-based CQ answering algorithm for $\mathcal{ELHIO}^{\neg}$ that makes use of a novel way of reasoning with equality. We then derive RQR from this algorithm and show that RQR is worst-case optimal with respect to data complexity for various sublanguages of $\mathcal{ELHIO}^{\neg}$. We conclude the chapter with a discussion on the size of the rewritings produced by RQR and present a data complexity analysis of our approach in the context of CQ answering.

- In Chapter 6 we recapitulate related work. We briefly present various approaches related to the problems of CQ answering and rewriting for DLs, and the use of resolution as the basis for decision procedures over KBs. For each approach we outline the differences and similarities with respect to our work.

Part III is concerned with the practical aspects of CQ rewriting.

- In Chapter 7,we describe how to use RQR with a (deductive) database system in order to answer CQs over a KB. Additionally, we present various optimization techniques aimed at reducing the size of the rewritings produced by RQR, followed by a description of REQUIEM—our implementation of RQR.

- In Chapter 8 we present the results of an empirical evaluation in which we used REQUIEM to rewrite typical queries over realistic TBoxes expressible in logics of the DL-Lite and $\mathcal{EL}$ families. Additionally, we present an analysis of REQUIEM's performance in the context of a real use case identified by researchers of the Japanese National Institute of Information and Communications Technology.

Finally, Part IV and its only Chapter 9 is concerned with the achievements and the significance of the results obtained in this thesis. Additionally, remaining open questions and directions for future research are discussed.

1.4 Publications Resulting from the Thesis

The results presented in this thesis have been published in a number of papers. Our first important step towards the development of RQR was presented in [PUMH08a], where we outlined a CQ rewriting algorithm for the DL DL-Lite$^+$. Shortly after, in [PUMH08b], we presented an extension of our previous algorithm that handles the DL $\mathcal{ELHI}$. We then presented the final and technically most challenging extension of our algorithm in [PUMH09b] where we first outlined RQR.

After the development of RQR we focused on the implementation of REQUIEM. A preliminary comparison of REQUIEM to an implementation of the algorithm of Calvanese et al. was presented in [PUMH09a]. Shortly after, we presented a more comprehensive comparison of both techniques in [PUHM09a]. Finally, a preliminary evaluation of REQUIEM regarding TBoxes that go beyond DL-Lite$_R$ was presented in [PUHM09b].

Chapter 2

Preliminary Definitions

In this chapter we introduce necessary terminology and recapitulate relevant definitions and results. We present relevant standard definitions of logic programming and resolution in Sections 2.1 and 2.2, respectively. This chapter is not intended to be of a tutorial nature; the interested reader is advised to consult the given references for a more detailed presentation.

2.1 Logic Programming

In this section we recapitulate some standard definitions of Logic Programming. The interested reader is referred to [Llo84] or [Lif96] for an extended discussion.

Let $\mathcal{P}$ be a finite or countable set of *predicate symbols*, $\mathcal{F}$ a finite or countable set of *function symbols*, $\mathcal{C}$ a finite or countable set of *constant symbols*, and $\mathcal{V}$ a countably infinite set of *variables*. Each predicate and function symbol is associated with a positive integer n called the *arity*. With $\Sigma(\mathcal{P}, \mathcal{F}, \mathcal{C}, \mathcal{V})$ or simply Σ if $\mathcal{P}$, $\mathcal{F}$, $\mathcal{C}$, and $\mathcal{V}$ are clear from the context, we denote the *first-order language* determined by $\mathcal{P}$, $\mathcal{F}$, $\mathcal{C}$, and $\mathcal{V}$.

The set of *terms* $\mathcal{T}(\Sigma)$ of Σ is the smallest set such that (i) $\mathcal{C} \cup \mathcal{V} \subseteq \mathcal{T}(\Sigma)$, and (ii) if $f \in \mathcal{F}$ has arity n, and $t_i \in \mathcal{T}(\Sigma)$ for $1 \leq i \leq n$, then $f(t_1, ..., t_n) \in \mathcal{T}(\Sigma)$. Terms of the form $f(t_1, ..., t_n)$ are called *functional terms*. We consider functional terms of arity 1 only. For the sake of brevity, we may write functional terms of the form

$f_1(...(f_n(t))...)$ as $f_1...f_n(t)$. The *depth of a term* t is defined as $\mathsf{depth}(t) = 0$ if t is a constant or a variable, and $\mathsf{depth}(f(s)) = 1 + \mathsf{depth}(s)$ if t is a functional term of the form $f(s)$. A term is *ground* if it contains no variables. The *Herbrand universe* U_Σ of a first-order language Σ is the set containing exactly every ground term in $\mathcal{T}(\Sigma)$.

The set of *atoms* $\mathcal{A}(\Sigma)$ is the smallest set such that, if $P \in \mathcal{P}$ has arity n, and $t_i \in \mathcal{T}(\Sigma)$ for $1 \leq i \leq n$, then $P(t_1, ..., t_n) \in \mathcal{A}(\Sigma)$. The *depth of an atom* $P(t_1, ..., t_n)$ is defined as $\mathsf{depth}(P(t_1, ..., t_n)) = \max(\mathsf{depth}(t_1), ..., \mathsf{depth}(t_n))$. An atom $P(t_1, ..., t_n)$ is *ground* if all terms t_i are ground. The *Herbrand base* B_Σ of Σ is the set containing exactly every ground atom in $\mathcal{A}(\Sigma)$.

A *substitution* σ is a function from $\mathcal{V}$ into $\mathcal{T}(\Sigma)$. We are typically interested in substitutions that are not identity on a finite number of variables. We denote these substitutions by listing all nonidentity mappings as $\{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$. An application of a substitution σ to a term t, written $t\sigma$, is defined recursively as follows: $a\sigma = a$ for a constant a, $x\sigma = \sigma(x)$ for a variable x, and $f(s)\sigma = f(s\sigma)$ for a functional term $f(s)$. An application of a substitution σ to an atom $R(t_1, ..., t_n)$, written $R(t_1, ..., t_n)\sigma$, is defined as $R(t_1\sigma, ..., t_n\sigma)$. A *composition* of substitutions δ and σ, written $\sigma\delta$, is defined as $t\sigma\delta = (t\sigma)\delta$ for a term t. A substitution σ is called a *variable renaming* if it contains only mappings of the form $x \mapsto y$ for variables x and y. A substitution σ is said to be *equivalent up to variable renaming* to a substitution δ if there is a variable renaming λ such that $\delta = \sigma\lambda$; in such a case, δ is also equivalent to σ up to variable renaming [BS01]. The notion of equivalence up to variable renaming is generalized to terms and atoms in the natural way. A substitution σ is *more general* than a substitution δ if there is a substitution λ such that $\delta = \sigma\lambda$. A substitution σ is a *unifier* of terms s and t if $s\sigma = t\sigma$. A unifier σ of s and t is called a *most general unifier* if it is more general than any other unifier of s and t. If a most general unifier σ of s and t exists, then it is unique up to variable

renaming [BS01], so we write $\sigma = \mathsf{MGU}(s, t)$. The notions of unifier and most general unifier are generalized to atoms in the natural way.

A *Horn clause* is an expression of the form $H \leftarrow B_1 \wedge ... \wedge B_n$, where H and each B_i are atoms. The atom H is called the *head*, and the set of atoms $\{B_i\}$ is called the *body*. With $\square$ we denote the *empty clause*—that is, a clause where the body is empty and H is the symbol $\perp$, which represents the logical constant 'false'. A Horn clause C of the form $H \leftarrow$, often written as H, is called a *fact*; furthermore, if H is a ground atom, then C is called a *ground fact*. With $\mathsf{varNo}(C)$ we denote the number of variables in a clause C. An atom A is *covering* for a clause C if A contains all the variables occurring in C. Variables occurring in the head are called *distinguished* variables and all other variables are called *nondistinguished*. Nondistinguished variables that occur in the body at most once are called *unbound* variables, all other variables are called *bound* variables. Unbound variables may be denoted with the symbol '_' as it is unnecessary to keep track of their relative position within the clause they occur in. A Horn clause C is *safe* if all distinguished variables occur in the body. We consider safe Horn clauses only. The *depth of a Horn clause* C is $\mathsf{depth}(C) = \max(\mathsf{depth}(H), \mathsf{depth}(B_1), ..., \mathsf{depth}(B_n))$.

A *logic program* LP is a finite set of Horn clauses. The *extensional database (EDB) predicates* of a logic program LP are those that do not occur in the head of any Horn clause in LP other than facts; all other predicates are called *intensional database (IDB) predicates*. A logic program LP is called a *datalog program* if no Horn clause C in LP contains functional terms. A datalog program D is said to be *linear* if each Horn clause C in D contains at most one IDB predicate in the body.

With each logic program LP we associate the first-order language $\mathcal{L}(LP)$ that is determined by the predicates, functions, constants, and variables occurring in LP. If LP does not contain constants, we define $\mathcal{L}(LP)$ over a nonempty set of constants. A *Herbrand interpretation* of a logic program LP is any subset I of its Herbrand base

$B_{\mathcal{L}(LP)}$. A Horn clause $C = H \leftarrow B_1 \wedge ... \wedge B_n$ is *satisfied* in a Herbrand interpretation I if and only if for every substitution σ that replaces variables by constants, whenever $B_i\sigma \in I$ for $1 \leq i \leq n$, then $H\sigma \in I$. A *Herbrand model* of LP is a Herbrand interpretation I that satisfies every Horn clause in LP. A logic program LP is *satisfiable* if it has at least one Herbrand model. A logic program LP *logically implies* a set of ground atoms $\{F_1, ..., F_n\}$, written $LP \models \{F_1, ..., F_n\}$, if for every Herbrand model I of LP and each F_i for $1 \leq i \leq n$, we have that $F_i \in I$. A logic program LP can be regarded as the clausal normal form of a first-order logic (FOL) formula $\epsilon(LP)$ [Fit96]; therefore, LP can be interpreted under standard FOL semantics. It is well known that this semantics is equivalent to the aforementioned semantics based on Herbrand models [Fit96].

A *Datalog Query* (DQ) Q is a tuple $\langle Q_P, Q_C \rangle$, where Q_P is a *predicate symbol* and Q_C is a datalog program. A DQ Q is a *Linear Datalog Query* (LDQ) if Q_C is a linear datalog program. A DQ Q is called a *Union of Conjunctive Queries* (UCQ) if Q_P is the only IDB predicate in Q_C, and Q_P does not occur in the body of any clause in Q_C. We may denote a UCQ simply with Q_C as Q_P is the head predicate of every Horn clause in Q_C. A DQ Q is a *Conjunctive Query* (CQ) if it is a UCQ and Q_C contains exactly one Horn clause. We may denote a CQ simply with the only Horn clause contained in Q_C. Given a CQ $Q = \langle Q_P, Q_C \rangle$ and a logic program LP, the *certain answers* $\mathsf{ans}(Q, LP)$ of Q with respect to LP is the set containing exactly every tuple $\vec{a}$ of constants occurring in LP such that $LP \cup Q_C \models Q_P(\vec{a})$.

2.2 Resolution with Free Selection

The *Resolution with Free Selection* (RFS) family of calculi is a set of resolution-based calculi that can be used to check whether a logic program LP is satisfiable [BG01]. Each RFS calculus is defined by a *selection function* S that assigns to each Horn clause C in LP a nonempty set of atoms such that either $S(C)$ is a singleton set containing

the head of C, or $S(C)$ is a subset of the body of C. The atoms in $S(C)$ are said to be *selected* by S in C. We denote the selected atoms in $S(C)$ of a given Horn clause C by underlying them. Each RFS calculus $\mathcal{R}$ consists of only the *resolution* inference rule, which is defined as

$$\frac{A \leftarrow B_1 \wedge ... \wedge \underline{B_i} \wedge ... \wedge B_n \qquad \underline{C} \leftarrow D_1 \wedge ... \wedge D_m}{A\sigma \leftarrow B_1\sigma \wedge ... \wedge B_{i-1}\sigma \wedge D_1\sigma \wedge ... \wedge D_m\sigma \wedge B_{i+1}\sigma \wedge ... \wedge B_n\sigma}$$

where B_i and C unify and $\sigma = \mathsf{MGU}(B_i, C)$. The two clauses above the inference line are called the *premises* and the clause below is called the *resolvent*. As selected atoms might not unify in case they share variables, without loss of generality we make the usual technical assumption that the premises do not have variables in common.

A set of Horn clauses LP is *saturated* by $\mathcal{R}$ if, for every two premises P_1 and P_2 in LP and every resolvent P_R of P_1 and P_2, the set LP contains a clause equivalent to P_R up to variable renaming. A *derivation* by $\mathcal{R}$ from a set of Horn clauses LP is a sequence of sets of Horn clauses $LP_0, LP_1, ...$ such that $LP_0 = LP$ and for each $i \geq 0$ we have that $LP_{i+1} = LP_i \cup \{C\}$, where C is the resolvent of an inference by $\mathcal{R}$ for a pair of premises in LP_i. The *limit* $LP_\mathcal{R}$ of a derivation from a set of Horn clauses LP by $\mathcal{R}$ is defined as $LP_\mathcal{R} = \bigcup LP_i$. A derivation with limit $LP_\mathcal{R}$ is *fair* if each clause C that can be derived from premises in $LP_\mathcal{R}$ is contained in some set LP_i. It is well known that the limit $LP_\mathcal{R}$ of a fair derivation by $\mathcal{R}$ from a set of Horn clauses LP is saturated by $\mathcal{R}$ [BG01]. A clause C is said to be *derivable* from LP by $\mathcal{R}$ if and only if $C \in LP_\mathcal{R}$. Each RFS calculus is sound and complete regardless of its selection function; that is, a set of Horn clauses LP is unsatisfiable if and only if $\square \in LP_\mathcal{R}$ [BG01].

A *tree* T is a prefix-closed subset of $\mathbb{N}^*$, where the root node is denoted by ϵ, and the i-th child of a node $t \in T$ is denoted by $t.i$. Let LP be a set of Horn clauses and let I be a Herbrand model of LP. A *derivation tree* Σ for a fact $F \in I$ is a triple $\langle T, \delta, \lambda \rangle$ where T is a finite tree, δ is a function that maps every node $t \in T$ to a fact

$\delta(t) \in I$ such that $\delta(\epsilon) = F$, and λ is a partial function that maps every nonleaf node $t \in T$ with n children to a clause $\lambda(t) \in LP$ such that $\delta(t)$ is obtained by resolving each $\delta(t.i)$ with the i-th body literal of $\lambda(t)$ simultaneously. If t is a leaf node, then $\delta(t) \in LP$ and $\lambda(t)$ is undefined.

Chapter 3

Introduction to Description Logics

Description Logics (DLs) are a family of knowledge representation formalisms that can be used to represent application domains in terms of individuals, concepts (sets of individuals), and roles (binary relations between individuals). Concepts and roles can be *atomic*, which means that they are denoted by name, or *complex*, which means that they are built using *constructors* that specify necessary and sufficient conditions for concept and role membership of individuals and pairs of individuals, respectively. For instance, the complex concept

$$\text{Woman} \sqcap \exists.\text{hasFriend}(\forall \text{hasParent}.(\text{Doctor} \sqcup \text{Engineer}))$$

represents the set of women that have a least one friend both of whose parents are doctors or engineers. A particular DL is characterized by the set of concept and role constructors it provides.

A DL *Knowledge Base* (KB) or *ontology* consists of a *terminological* component, called the TBox, and an *assertional* component called the ABox. The TBox consists of a set of *inclusion axioms* that introduces the terminology or vocabulary of an application domain by stating *subsumption* relationships between concepts and between roles. The ABox contains concept and role *membership assertions* about individuals and pairs of individuals, respectively. DLs have a well-understood model theoretic semantics that can be used to derive conclusions from a particular KB unambiguously.

Example 3.0.1. Consider a TBox $\mathcal{T}$ that captures knowledge regarding grandparents consisting of the following axioms:

$$\text{Grandparent} \sqcap \text{Female} \sqsubseteq \text{Grandmother}$$

$$\text{Grandparent} \sqcap \text{Male} \sqsubseteq \text{Grandfather}$$

$$\exists \text{hasChild.Parent} \sqsubseteq \text{Grandparent}$$

$$\exists \text{hasChild} \sqsubseteq \text{Parent}$$

$$\text{hasSon} \sqsubseteq \text{hasChild}$$

$$\text{hasDaughter} \sqsubseteq \text{hasChild}$$

The TBox $\mathcal{T}$ states that every female grandparent is a grandmother, that every male grandparent is a grandfather, that someone that has a child who is a parent is a grandparent, that someone who has a child is a parent, and that the sons and daughters of someone are his/her children.

Consider an ABox $\mathcal{A}$ consisting of the following membership assertions:

$$\text{Male}(\mathsf{Pepe})$$

$$\text{hasSon}(\mathsf{Pepe}, \mathsf{Hector})$$

$$\text{hasDaughter}(\mathsf{Hector}, \mathsf{Kari})$$

The ABox $\mathcal{A}$ states that the individual Pepe is an instance of the concept Male—that is, Pepe is male—that Hector is the son of Pepe, and that Kari is the daughter of Hector. △

A *DL system* or *reasoner* allows users to *reason* about a given KB—that is, to infer implicit knowledge from the knowledge explicitly represented by the KB. For instance, the explicit knowledge represented by the KB described in Example 3.0.1 implies that both Hector and Pepe are instances of Parent, and that Pepe is an instance of Grandparent and Grandfather. Typical reasoning problems concerning

TBoxes are to determine whether one concept *subsumes* another—that is, whether the set of individuals represented by the former concept necessarily contains the individuals represented by the latter according to a given TBox; or whether a concept is *satisfiable*—that is, whether a concept can have any instances at all according to a given TBox. Typical reasoning problems concerning KBs include deciding whether a given individual is an instance of a concept or whether a given pair of individuals is an instance of a role. Note that the reasoning problems previously mentioned are *decision problems*—that is, questions with a 'Yes' or 'No' answer. Other more complex reasoning problems can be defined in terms of these basic decision problems. For instance, the so-called *concept retrieval* problem consists of computing the set of individuals that are instances of a given concept with respect to a given KB.

In this thesis we are particularly interested in a reasoning problem called *Conjunctive Query (CQ) answering over KBs*. The language of CQs is a popular query language that has proven to be extremely useful in the field of databases [AHV95]. This language allows one to pose complex queries that go beyond the capabilities of concept (and role) retrieval [Gli07]. For instance, regarding the KB described in Example 3.0.1, we may ask for all ternary tuples of individuals such that the first one is the son of the second one, and the second one is the son of the third one.

In this chapter, we start by presenting a brief summary of the history of DLs in Section 3.1; then, in Section 3.2, we present various applications of DLs. The syntax and semantics of the DLs considered in this thesis are presented in Section 3.3; then, in Section 3.4, we formally describe various reasoning problems; and finally, we formally define the problem of CQ answering for DLs in Section 3.5.

3.1 Historical Background

The historical ancestors of DLs can be traced back to the period of 1965–1980, in which various approaches, such as semantic networks [Qui67] and frames [Min81],

were used for representing knowledge in a structured way. One of the main problems with such approaches was the lack of an unambiguous semantics [Woo75, Bra77, Hay77, Hay79]. One of the first attempts to overcome this problem was the so-called structured inheritance networks [Bra78], which lead to the development of the first DL system KL-ONE [BS85].

KL-ONE was only the first of many systems developed in the 80s, such as K-REP, KRYPTON, BACK, and LOOM [MDW91, BFL83, Pel91, Mac91]. These systems employed so-called structural subsumption algorithms [Neb90a]. Despite being relatively efficient (polynomial), such algorithms have the disadvantage of being complete only for very inexpressive DLs—that is, they can only guarantee to detect all subsumption/instance relationships for very inexpressive DLs. It was during this period that the first logic-based accounts of the formal semantics of the underlying representation formalisms were given [BFL83, BL85], which set the stage for the formal investigation of the computational complexity of reasoning in DLs.

Early results in the area of complexity of reasoning for DLs showed that even apparently small additions to the expressive power of DLs can cause intractability of the concept subsumption problem [BL85]. Subsumption was shown to be undecidable for the DL underlying KL-ONE [SS89]. Shortly after, it was shown that subsumption becomes intractable for DLs that allow the use of complex concept abbreviations in addition to concept conjunction and value restrictions [Neb90b]—features that were supported in all the DL systems available at that time. The system CLASSIC was designed taking into account these negative complexity results, which resulted in the expressive power of the underlying DL being carefully restricted [PSMBR91, BMPSB99].

The first half of the 90s saw the development of a new algorithmic paradigm for DLs—the so-called tableau based algorithms [SSS91, DLNN91a, HNSs90]. Such algorithms were developed to check the consistency of a KB for propositionally closed

DLs—that is, DLs able to capture all Boolean formulae—however, since in propositionally closed DLs the concept subsumption and the instance problem can be reduced to consistency, tableau algorithms can also be used to solve the aforementioned inference problems. Moreover, as opposed to their predecessors, these algorithms are complete for expressive DLs. Tableau algorithms were first implemented in the systems KRIS and CRACK [BH91, BFT95]. In particular, the highly optimized system KRIS showed that it is possible to obtain an acceptable behavior in practice, in spite of the corresponding reasoning problems being no longer tractable. It was also during this period that a thorough analysis of the complexity of reasoning was conducted for various DLs [DLNN91a, DLNN91b, DLN+92]. Additionally, it was observed that there is a very close relation between DLs and modal logics [Sch91].

During the second half of the 90s various reasoning algorithms were developed for very expressive DLs, either based on the tableau approach [HS99, HST99] or a translation into modal logics [GL94a, GL94b, Gia95, GL96]. Various highly optimized DL systems—such as FaCT, RACE, and DLP [Hor98a, Hor98b, HM99, Ps98]—demonstrated that is possible to have good practical behavior even on (some) large KBs. Additionally, during this period, the relationship to modal logics [GL94a, Sch95] and to decidable fragments of first-order logic [Bor96, PST97, GKV97, Gra98, Gra99] was studied in more detail, and applications in the area of databases—such as schema reasoning, query optimization, and integration of databases—were first investigated [BDNS98, CDL98, CDGL+98].

Currently, a great effort has been devoted to the development of efficient DL systems—such as RACER, FaCT++, KAON2, PELLET, and HermiT [HM01, TH06, MS06, SP06, MSH07]—that handle very expressive DLs. In addition, the interest in less expressive DLs has been revived with the goal of developing systems that can deal with KBs containing very large TBoxes and/or ABoxes [BBL05, CDL+05, ACL+05,

BLS06, CDL+07, LTW09]. Our work places itself within the current efforts being made to effectively handle KBs with relatively small TBoxes and large ABoxes.

3.2 Application Areas

DLs have been proven to be useful in a variety of applications, such as the design of software information and documentation systems [DBSB90, HMvdSW04], the configuration of telecommunications systems [BIW94, MW98], reasoning about actions and plan generation in robotics [DGINR96], and process planning and control in manufacturing [Ryc96]. DLs have been extensively used in the field of databases [BLR03]. Specific database application areas include schema design, query optimization and reasoning about views in object oriented databases [CLN98, BCDG01, BBLS94, BJNS94, Bre95, LR96], federated databases and cooperative information systems design [BIG94, TM93, GBBI96], database interoperability [GL94c], schema and information integration [CDGL+98, CDGR99], and CQ answering [CDL98, CGL99, HSTT00].

DLs have often been used to provide formal foundations for ontology languages such as OIL, DAML+OIL and OWL [HPSv03]. In particular, OWL was standardized by the World Wide Web Consortium (W3C) in 2004 [BvHH+04]. The ability to use DL systems in OWL applications was one of the motivations for basing the design of OWL on DLs. OWL is actually a family of three language variants or *species*: OWL Lite, OWL DL and OWL full. OWL lite, and OWL DL both have DL based semantics, while the semantics of OWL full is based on an extension of the RDF model theory [Hay04]. The standardization of OWL by the W3C has given DLs a prominent role in the development of the Semantic Web [BLHL01, HH01]. In September 2007 the W3C approved the OWL Working Group[1] whose main objective was to produce a W3C recommendation that refines and extends OWL with useful yet practical features.

[1] http://www.w3.org/2007/06/OWLCharter

As from April 2008 this revised version of OWL is known as OWL 2 [GHM+08]. The OWL 2 specification identifies several *profiles*—trimmed down versions of the language that trade some expressive power for efficiency of reasoning. OWL 2 provides three profiles: OWL 2 EL, OWL 2 QL, and OWL 2 RL. Each profile provides different expressive power and targets different application scenarios. In this thesis we are particularly interested on OWL 2 QL and OWL 2 EL.

The development of various OWL ontology design tools, such as TopBraid Composer,[2] OilEd [BHGS01], Protégé [KFNM04], and Swoop [KPH05], has contributed to the use of OWL in several fields including biology [SDCS05], medicine [GZB06], geography [Goo05], geology [Ras06], astronomy [DRPM06], agriculture [SLL+04], and defense [LAF+05]. OWL has been widely used in the life sciences to model various large ontologies, such as the Biological Pathways Exchange (BioPAX) ontology [RRL05], the GALEN ontology [RR06], the Foundational Model of Anatomy (FMA) [GZB06], the National Cancer Institute (NCI) thesaurus [HdCD+05], the SNOMED ontology [SC98], the Gene Ontology (GO), and various ontologies stemming from the Open Biomedical Ontologies (OBO) project [GH07, GHH+07].

3.3 Syntax and Semantics

We now present the syntax and semantics of the DLs we consider in this thesis. The interested reader is referred to [BCM+03] for an extended discussion. We focus on the DL $\mathcal{ELHIO}^{\neg}$ as other logics that we consider can be defined in terms of restrictions on this logic.

Definition 3.3.1. Let N_C, N_R, and N_I be countable, infinite, and pairwise disjoint sets of *atomic concepts*, *atomic roles*, and *constants*, respectively.

[2] http://www.topbraidcomposer.com/

$\mathcal{ELHIO}^{\neg}$ roles are built according to the following syntax rules, where R is called a *basic role*, E is called a *general role*, and $P \in N_R$:

$$R ::= P \mid P^{-} \qquad E ::= R \mid \neg R$$

A basic role of the form P^{-} is called the *inverse* role of P.

$\mathcal{ELHIO}^{\neg}$ concepts are built according to the following syntax rules, where B is called a *basic concept*, C is called a *general concept*, $a \in N_I$, $A \in N_C$, R is a basic role, and B_1 and B_2 are basic concepts:

$$B ::= A \mid \{a\} \mid \top \mid B_1 \sqcap B_2 \mid \exists R.B \qquad C ::= B \mid \neg B$$

An $\mathcal{ELHIO}^{\neg}$ *TBox* is a finite set of *axioms* of the form $B \sqsubseteq C$, called *concept inclusions*, or $R \sqsubseteq E$, called *role inclusions*, where B is a basic concept, C is a general concept, R is a basic role, and E is a general role. An axiom α is called a *negative inclusion* if it contains the symbol $\neg$; otherwise, it is called a *positive inclusion*. Given a TBox $\mathcal{T}$, with $\mathcal{T}_{\text{NI}}$ we denote the set of all negative inclusions of $\mathcal{T}$, and with $\mathcal{T}_{\text{PI}}$ we denote the set of all positive inclusions of $\mathcal{T}$. An *ABox* is a finite set of *membership assertions* of the form $A(a)$ or $P(a, b)$, where $A \in N_C$, $P \in N_R$, $a \in N_I$, and $b \in N_I$. An $\mathcal{ELHIO}^{\neg}$ *KB* $\mathcal{K}$ is a tuple $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is an $\mathcal{ELHIO}^{\neg}$ TBox and $\mathcal{A}$ is an ABox. △

Without loss of generality we only consider $\mathcal{ELHIO}^{\neg}$ TBoxes in *normal form*—that is, TBoxes that only contain axioms of the form $A_1 \sqsubseteq \{a\}$, $\{a\} \sqsubseteq A_1$, $A_1 \sqsubseteq (\neg)A_2$, $A_1 \sqcap A_2 \sqsubseteq A_3$, $A_1 \sqsubseteq \exists R_1$, $A_1 \sqsubseteq \exists R_1.A_2$, $\exists R_1 \sqsubseteq A_1$, $\exists R_1.A_1 \sqsubseteq A_2$, or $R_1 \sqsubseteq (\neg)R_2$, where A_1, A_2, and A_3 are in N_C; $a \in N_I$; and R_1 and R_2 are basic roles. Each $\mathcal{ELHIO}^{\neg}$ TBox $\mathcal{T}$ can be transformed into an equisatisfiable $\mathcal{ELHIO}^{\neg}$ TBox $\mathcal{T}'$ in normal form by systematically replacing complex concepts with atomic ones along the lines of [BBL05]. This process can produce axioms of the form $\top \sqsubseteq C$, which are then replaced by $A \sqsubseteq C$, $\{a\} \sqsubseteq A$, and $\exists R \sqsubseteq A$, for every $A \in N_C$, every individual

Table 3.1: Semantics of $\mathcal{ELHIO}^{\neg}$

Semantics of concepts and roles:		Semantics of assertions:	
$\top^{\mathcal{I}}$	$= \triangle^{\mathcal{I}}$		
$\{a\}^{\mathcal{I}}$	$= \{a^{\mathcal{I}}\}$		
$(B_1 \sqcap B_2)^{\mathcal{I}}$	$= B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}}$	$\mathcal{I} \models A(a)$	iff $a^{\mathcal{I}} \in A^{\mathcal{I}}$
$(\exists R)^{\mathcal{I}}$	$= \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}}\}$	$\mathcal{I} \models P(a, b)$	iff $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in P^{\mathcal{I}}$
$(\exists R.B)^{\mathcal{I}}$	$= \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in B^{\mathcal{I}}\}$	$\mathcal{I} \models B \sqsubseteq C$	iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
$(P^{-})^{\mathcal{I}}$	$= \{\langle x, y\rangle \mid \langle y, x\rangle \in P^{\mathcal{I}}\}$	$\mathcal{I} \models R \sqsubseteq E$	iff $R^{\mathcal{I}} \subseteq E^{\mathcal{I}}$
$(\neg B)^{\mathcal{I}}$	$= \triangle^{\mathcal{I}} \setminus B^{\mathcal{I}}$		
$(\neg R)^{\mathcal{I}}$	$= \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}} \setminus R^{\mathcal{I}}$		

a, and every basic role R occurring in the TBox. It is straightforward to see that this transformation preserves satisfiability of a KB.

The semantics of an $\mathcal{ELHIO}^{\neg}$ KB is typically defined by an *interpretation* that consists of a nonempty interpretation domain $\triangle^{\mathcal{I}}$ and a function that maps each constant to an element of $\triangle^{\mathcal{I}}$, each concept to a subset of $\triangle^{\mathcal{I}}$, and each role to a subset of $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$. We formally define the notion of interpretation as follows.

Definition 3.3.2. An *interpretation* $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty interpretation domain $\triangle^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$ that maps each atomic concept $A \in N_C$ to a subset $A^{\mathcal{I}}$ of $\triangle^{\mathcal{I}}$, each atomic role $P \in N_R$ to a subset $P^{\mathcal{I}}$ of $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$, and each constant $a \in N_I$ to an element $a^{\mathcal{I}}$ of $\triangle^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ is extended to complex concepts and roles as shown in the left column of Table 3.1. An interpretation $\mathcal{I}$ is a *model* of an inclusion or membership assertion α, written $\mathcal{I} \models \alpha$, if and only if $\mathcal{I}$ and α satisfy the conditions shown in the right column of Table 3.1.

Given an interpretation $\mathcal{I}$ and a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, we say that $\mathcal{I}$ is a model of $\mathcal{K}$, written $\mathcal{I} \models \mathcal{K}$, if and only if $\mathcal{I}$ satisfies every assertion in $\mathcal{T}$ and $\mathcal{A}$. Given a KB $\mathcal{K}$, we say that $\mathcal{K}$ *logically implies* a set of inclusion or membership assertions $\{\alpha_i\}$, written $\mathcal{K} \models \{\alpha_i\}$, if and only if for every model $\mathcal{I}$ of $\mathcal{K}$ and every α_i for $1 \leq i \leq n$, we have that $\mathcal{I} \models \alpha_i$. $\triangle$

Table 3.2: Translating an $\mathcal{ELHIO}^{\neg}$ KB into FOL

Translating Concepts and Roles into FOL	
$\pi_x(\top) = \top$	$\pi_y(\top) = \top$
$\pi_x(A) = A(x)$	$\pi_y(A) = A(y)$
$\pi_x(\{a\}) = x \approx a$	$\pi_y(\{a\}) = y \approx a$
$\pi_x(B_1 \sqcap B_2) = \pi_x(B_1) \wedge \pi_x(B_2)$	$\pi_y(B_1 \sqcap B_2) = \pi_y(B_1) \wedge \pi_y(B_2)$
$\pi_x(\exists R) = \exists y(\pi_{x,y}(R))$	$\pi_y(\exists R) = \exists x(\pi_{y,x}(R))$
$\pi_x(\exists R.B) = \exists y(\pi_{x,y}(R) \wedge \pi_y(B))$	$\pi_y(\exists R.B) = \exists x(\pi_{y,x}(R) \wedge \pi_x(B))$
$\pi_x(\neg B) = \neg\pi_x(B)$	$\pi_y(\neg B) = \neg\pi_y(B)$
$\pi_{x,y}(P) = P(x, y)$	$\pi_{y,x}(P) = P(y, x)$
$\pi_{x,y}(P^-) = P(y, x)$	$\pi_{y,x}(P^-) = P(x, y)$
$\pi_{x,y}(\neg R) = \neg\pi_{x,y}(R)$	$\pi_{y,x}(\neg R) = \neg\pi_{y,x}(R)$
Translating Axioms into FOL	
$\pi(B \sqsubseteq C) = \forall x(\pi_x(B) \rightarrow \pi_x(C))$	
$\pi(R \sqsubseteq E) = \forall x, y(\pi_{x,y}(R) \rightarrow \pi_{x,y}(E))$	
$\pi(A(a)) = \pi_x(A)\{x \mapsto a\}$	
$\pi(P(a, b)) = \pi_{x,y}(P)\{x \mapsto a, y \mapsto b\}$	
$\pi(\mathcal{K}) = \bigwedge_{\alpha \in \mathcal{K}} \pi(\alpha)$	

$\mathcal{ELHIO}^{\neg}$ can be regarded as a fragment of first-order logic (FOL) [Bor96]. We define the translation of $\mathcal{ELHIO}^{\neg}$ KBs into FOL as follows.

Definition 3.3.3. Let $\mathcal{K}$ be an $\mathcal{ELHIO}^{\neg}$ KB. We define the first-order logic formula $\pi(\mathcal{K})$ using the operator π defined in Table 3.2. △

An $\mathcal{ELHIO}^{\neg}$ KB $\mathcal{K}$ can be interpreted under standard FOL semantics by translating $\mathcal{K}$ into $\pi(\mathcal{K})$; it is well known that this semantics is equivalent to the direct model-theoretic semantics defined previously [Bor96, BCM+03].

The various other DLs that we consider in this thesis can be defined in terms of $\mathcal{ELHIO}^{\neg}$ as follows.

Definition 3.3.4. Given two DLs $\mathcal{L}_1$ and $\mathcal{L}_2$, we say that $\mathcal{L}_1$ is a *fragment* of $\mathcal{L}_2$ if each axiom of $\mathcal{L}_1$ is an axiom of $\mathcal{L}_2$. We define various fragments of $\mathcal{ELHIO}^{\neg}$ as follows:

- $\mathcal{ELHIO}$ is the fragment of $\mathcal{ELHIO}^{\neg}$ obtained by disallowing negative inclusions.

- $\mathcal{ELHI}$ is the fragment of $\mathcal{ELHIO}$ obtained by disallowing basic concepts of the form $\{a\}$.
- $\mathcal{ELH}$ is the fragment of $\mathcal{ELHI}$ obtained by disallowing inverse roles.
- $\mathcal{EL}$ is the fragment of $\mathcal{ELH}$ obtained by disallowing role inclusions.
- DL-Lite$^+$ is the fragment of $\mathcal{ELH}$ obtained by disallowing basic concepts of the form $B_1 \sqcap B_2$.
- DL-Lite$_R$ is the fragment of $\mathcal{ELHIO}^\neg$ obtained by disallowing basic concepts of the form $\{a\}$, $\top$, and $B_1 \sqcap B_2$, as well as axioms of the form $\exists R.B \sqsubseteq C$.
- DL-Lite$_{core}$ is a fragment of DL-Lite$_R$ obtained by disallowing role inclusions.

△

3.4 Standard Reasoning Problems

In this section we present common reasoning problems that most current DL reasoners consider. Before doing so, we formally define the following necessary notions:

Definition 3.4.1. Given a KB $\mathcal{K}$, we say that $\mathcal{K}$ *is satisfiable* if and only if $\mathcal{K}$ has at least one model.

Given a KB $\mathcal{K}$ and a concept C, we say that C *is satisfiable with respect to* $\mathcal{K}$ if and only if there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^\mathcal{I} \neq \emptyset$.

Given a KB $\mathcal{K}$ and concepts C and D, we say that C *is subsumed by* D *with respect to* $\mathcal{K}$ if and only if $\mathcal{K} \models C \sqsubseteq D$.

Given a KB $\mathcal{K}$, a concept C, and a constant a occurring in $\mathcal{K}$, we say that a *is an instance of* C *with respect to* $\mathcal{K}$ if and only if $\mathcal{K} \models C(a)$.

Given a KB $\mathcal{K}$, a role R, and constants a, b occurring in $\mathcal{K}$, we say that $\langle a, b\rangle$ *is an instance of* R *with respect to* $\mathcal{K}$ if and only if $\mathcal{K} \models R(a, b)$. △

With the previous definitions in place, we first present a set of basic reasoning problems in terms of which other more complex reasoning problems can be defined:

- *KB satisfiability* is the problem of deciding whether a KB $\mathcal{K}$ is satisfiable.
- *Concept satisfiability* is the problem of deciding whether a concept C is satisfiable with respect to a KB $\mathcal{K}$.
- *Concept subsumption* is the problem of deciding whether a concept C is subsumed by another concept D with respect to a KB $\mathcal{K}$.
- *Instance checking* is the problem of deciding whether a constant a is an instance of a concept C with respect to $\mathcal{K}$.
- *Role checking* is the problem of deciding whether a binary tuple $\langle a, b \rangle$ of constants is an instance of a role R with respect to a KB $\mathcal{K}$.

Note that all the aforementioned reasoning problems are *decision problems*—that is, questions with a 'Yes' or 'No' answer. Other more complex reasoning problems can be defined in terms of the previously-defined decision problems. We now present two such complex reasoning problems.

- *Concept retrieval* is the problem of computing the set containing exactly every instance of a concept C with respect to a KB $\mathcal{K}$.
- *Role retrieval* is the problem of computing the set containing exactly every instance of a role R with respect to a KB $\mathcal{K}$.

Reasoners can be used by applications in order to support users in building and querying KBs. For instance, a reasoner that provides support for concept and role retrieval can be used in order to answer certain limited queries, such as "What are the instances of concept A?". In many applications, however, users are typically interested in obtaining the answers of more complex queries over a KB, such as "What are the

triples of individuals such that the first is an instance of concept A and the second is related to the third via the role R?". In the next section, we shall define another reasoning problem—similar to the retrieval problems—that considers more expressive queries.

3.5 CQ Answering for DLs

In this section we define the problem of CQ answering for DLs. For each CQ of the form $Q_P(\vec{x}) \leftarrow B_1(\vec{t_1}) \wedge ... \wedge B_n(\vec{t_n})$ posed over a KB $\mathcal{K}$, we assume that (i) every body atom is unary or binary, (ii) the predicate of each unary body corresponds to some atomic concept of $\mathcal{K}$, (iii) the predicate of each binary body atom corresponds to an atomic role of $\mathcal{K}$, and (iv) Q_P does not occur in $\mathcal{K}$. For example, we could pose the following CQ—asking for triples of grandfathers with their children and grandchildren—over the KB described in Example 3.0.1:

$$Q_P(x, y, z) \leftarrow \text{Grandfather}(x) \wedge \text{hasChild}(x, y) \wedge \text{hasChild}(y, z) \tag{3.1}$$

Before formally defining the problem of CQ answering for DLs, we present an example in which we give informal intuitive explanations.

Example 3.5.1. Consider the following fragment of the KB introduced in Example 3.0.1. Let $\mathcal{T}$ consist of the following axioms:

$$\text{Grandparent} \sqcap \text{Male} \sqsubseteq \text{Grandfather} \tag{3.2}$$

$$\exists \text{hasChild.Parent} \sqsubseteq \text{Grandparent} \tag{3.3}$$

$$\exists \text{hasChild} \sqsubseteq \text{Parent} \tag{3.4}$$

Let $\mathcal{A}$ consist of the following membership assertions:

$$\text{Male}(\mathsf{Pepe}) \tag{3.5}$$

$$\text{hasChild}(\mathsf{Pepe}, \mathsf{Hector}) \tag{3.6}$$

$$\text{hasChild}(\mathsf{Hector}, \mathsf{Kari}) \tag{3.7}$$

Let Q be query (3.1). It can be readily verified that, given (3.7) and (3.4), $\mathcal{K}$ implies

$$\text{Parent}(\mathsf{Hector}). \tag{3.8}$$

Similarly, given (3.6), (3.8), and (3.3), $\mathcal{K}$ implies

$$\text{Grandparent}(\mathsf{Pepe}). \tag{3.9}$$

Finally, given (3.9), (3.5), and (3.2), $\mathcal{K}$ implies

$$\text{Grandfather}(\mathsf{Pepe}). \tag{3.10}$$

We have that $\mathcal{K} \models \{(3.10), (3.6), (3.7)\}$ which intuitively means that, according to $\mathcal{K}$, Pepe is a grandfather, Hector is his child, and Kari is the child of Hector. It is easy to verify that these individuals satisfy Q; therefore, we have that the triple $\langle \mathsf{Pepe}, \mathsf{Hector}, \mathsf{Kari} \rangle$ is an answer or *certain answer* to Q over $\mathcal{K}$. △

A set of clauses Q_C can be regarded as the clausal normal form of a FOL formula $\epsilon(Q_C)$ [Fit96]. Moreover, the DLs considered in this thesis can be regarded as fragments of FOL (see Definition 3.3.3). Therefore, a CQ $Q = \langle Q_P, Q_C \rangle$ and a KB $\mathcal{K}$ can be translated into the equisatisfiable FOL formulae $\epsilon(Q_C)$ and $\pi(\mathcal{K})$, respectively. Based on this fact, the certain answers of a CQ over a KB can be defined in terms of standard FOL entailment as follows.

Definition 3.5.1. Given a CQ $Q = \langle Q_P, Q_C \rangle$ and a KB $\mathcal{K}$, the set of certain answers is defined as

$$\mathsf{ans}(Q, \mathcal{K}) = \{\vec{a} \mid \pi(\mathcal{K}) \wedge \epsilon(Q_C) \models Q_P(\vec{a})\},$$

where $\vec{a}$ is a tuple of constants occurring in $\mathcal{K}$, and $\pi(\mathcal{K})$ and $\epsilon(Q_C)$ respectively correspond to the translation of $\mathcal{K}$ and Q_C to first-order logic formulae. △

With the notion of certain answers in place, we now present the problem of CQ answering over KBs as well as its associated decision problem—so-called *CQ entailment*.

- *Conjunctive query answering* is the problem of computing the set of certain answers $\mathsf{ans}(Q, \mathcal{K})$ of a CQ Q with respect to a KB $\mathcal{K}$.
- *Conjunctive query entailment* is the problem of deciding whether a tuple of constants $\vec{a}$ is in $\mathsf{ans}(Q, \mathcal{K})$ for a CQ Q and a KB $\mathcal{K}$.

3.5.1 Query Rewriting

As we shall show in Chapter 5, the problem of CQ answering over $\mathcal{ELHIO}^{\neg}$ KBs can be solved via *query rewriting*. Intuitively, given a CQ Q and a TBox $\mathcal{T}$, the idea is to compute a Datalog Query (DQ) Q'—a so-called *rewriting of Q with respect to $\mathcal{T}$*—such that answering Q over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ for any $\mathcal{A}$ can be reduced to answering Q' over $\mathcal{A}$ only. Such an approach to query answering is interesting from a practical perspective because it allows one to use existing (deductive) database systems to store $\mathcal{A}$ and to evaluate Q'.

Before formally defining the problem of CQ rewriting over KBs, we present informal intuitive explanations with an example.

Example 3.5.2. Consider the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ introduced in Example 3.5.1 and the following query Q

$$Q_P(x) \leftarrow \text{Grandfather}(x) \tag{3.11}$$

that asks for the set of grandfathers. Instead of using Q, $\mathcal{T}$, and $\mathcal{A}$ to answer the query (as we did in Example 3.5.1), we will use Q and $\mathcal{T}$ first to obtain a rewriting Q'.

According to (3.2) every male grandparent is a grandfather; therefore, the set of male grandparents is a subset of the answers to Q over $\mathcal{K}$. The query that asks for such a set is

$$Q_P(x) \leftarrow \text{Grandparent}(x) \wedge \text{Male}(x). \tag{3.12}$$

Now, according to (3.3), someone with a child who is a parent is a grandparent; therefore we can obtain the following query:

$$Q_P(x) \leftarrow \text{hasChild}(x, y) \wedge \text{Parent}(y) \wedge \text{Male}(x) \tag{3.13}$$

Finally, according to (3.4), someone with a child is a parent; therefore, we can obtain the following query:

$$Q_P(x) \leftarrow \text{hasChild}(x, y) \wedge \text{hasChild}(y, z) \wedge \text{Male}(x) \tag{3.14}$$

It can be shown that no other axiom of $\mathcal{T}$ can be used to generate another relevant query; therefore, the rewriting of Q with respect to $\mathcal{T}$ is the DQ $Q' = \langle Q_P, Q'_C \rangle$, where $Q'_C = \{(3.11), (3.12), (3.13), (3.14)\}$.

We can now answer Q over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ by answering Q' over $\mathcal{A}$ only. In this case, as explained in Example 3.5.1, we have that $\mathcal{K} \models \{\text{Grandfather}(\mathsf{Pepe})\}$ which implies that $\mathsf{ans}(Q, \mathcal{K}) = \{\mathsf{Pepe}\}$. It can be readily verified that $\mathsf{ans}(Q', \mathcal{A}) = \{\mathsf{Pepe}\}$ due to (3.14). △

We formally define the notion of rewriting of a CQ with respect to a TBox as follows.

Definition 3.5.2. A DQ Q' is a *rewriting* of a CQ Q with respect to a TBox $\mathcal{T}$ if for every ABox $\mathcal{A}$, $\mathsf{ans}(Q, \mathcal{T} \cup \mathcal{A}) = \mathsf{ans}(Q', \mathcal{A})$. △

With the notion of rewriting a CQ with respect to a TBox in place, we now present the problem of CQ rewriting for DLs.

- *Conjunctive query rewriting* is the problem of computing a rewriting of a CQ with respect to a TBox.

In the following chapters we shall describe RQR—an algorithm for rewriting CQs with respect to $\mathcal{ELHIO}^{\neg}$ TBoxes.

Part II

Query Answering via Resolution-Based Query Rewriting

Chapter 4

Our Approach at a Glance

The description of RQR—our CQ rewriting algorithm for $\mathcal{ELHIO}^{\neg}$—is fairly complex and technical. This is so mainly because $\mathcal{ELHIO}^{\neg}$ allows for nominals, which requires reasoning with equality. For instance, the $\mathcal{ELHIO}^{\neg}$ axiom $\mathsf{Pope} \sqsubseteq \{\mathsf{BenedictXVI}\}$ implies that all instances of the class Pope are equal to the individual BenedictXVI.

To make our general results easier to understand, in this chapter we present $\mathsf{RQR}^{\mathrm{lite}}$—a simpler version of RQR scaled down to the basic DL DL-Lite$_R$ [CDL+07]. The description of $\mathsf{RQR}^{\mathrm{lite}}$ conveys all the basic ideas of our approach without overloading the presentation with details. We shall refer back to the principles described in this chapter when we present RQR in Chapter 5.

DL-Lite$_R$ allows for expressing cyclic subsumption relations on concepts and roles, disjointness between concepts, role-typing, participation constraints (i.e., assertions stating that all instances of a concept participate to a specified role), and nonparticipation constraints. Although DL-Lite$_R$ is relatively simple, it is suitable for expressing a variety of representation languages widely adopted in different contexts, such as basic TBox languages, conceptual data models (e.g., Entity-Relationship [AHV95]), and object-oriented formalisms (e.g., basic UML class diagrams[1]).

DL-Lite$_R$ was specifically designed with the objective of enabling the use of relational databases for answering queries via query rewriting. In fact, Calvanese et

[1] http://www.omg.org/uml/

al.—the creators of DL-Lite$_R$—proposed a CQ rewriting algorithm, which we call CGLLR after its authors. As RQR$^{\text{lite}}$, CGLLR can be used to rewrite a CQ with respect to a DL-Lite$_R$ TBox. In Section 4.5, we briefly present CGLLR and discuss the differences and similarities with respect to RQR$^{\text{lite}}$. This discussion will be useful in order to understand the results of an empirical evaluation of both algorithms that we present in Chapter 8.

4.1 Description Logic DL-Lite$_R$

DL-Lite$_R$ has been defined in Section 3.3 as a fragment of $\mathcal{ELHIO}^{\neg}$ obtained by disallowing basic concepts of the form $\{a\}$, $\top$, and $B_1 \sqcap B_2$, as well as axioms of the form $\exists R.B \sqsubseteq C$. In this section we present the syntax of DL-Lite$_R$ independently of $\mathcal{ELHIO}^{\neg}$ in order to make the exposition easier to follow.

Definition 4.1.1. Let N_C, N_R, and N_I be countable, infinite, and pairwise disjoint sets of *atomic concepts*, *atomic roles*, and *constants*, respectively.

DL-Lite$_R$ roles are built according to the following syntax rules, where R is called a *basic role*, E is called a *general role*, and $P \in N_R$:

$$R ::= P \mid P^- \qquad E ::= R \mid \neg R$$

A basic role of the form P^- is called the *inverse* role of P.

DL-Lite$_R$ concepts are built according to the following syntax rules, where B is called an *antecedent concept*, C is called a *consequent concept*, D is called a *general concept*, $A \in N_C$, and R is a basic role:

$$B ::= A \mid \exists R \qquad C ::= B \mid \exists R.C \qquad D ::= C \mid \neg C$$

A DL-Lite$_R$ *TBox* is a finite set of *axioms* of the form $B \sqsubseteq D$, called *concept inclusions*, or $R \sqsubseteq E$, called *role inclusions*, where B is an antecedent concept, D is a general concept, R is a basic role, and E is a general role. △

Without loss of generality we can restrict our attention only to DL-Lite$_R$ TBoxes in *normal form* in which all axioms are of the form $A_1 \sqsubseteq (\neg)A_2$, $A_1 \sqsubseteq \exists R_1$, $A_1 \sqsubseteq \exists R_1.A_2$, $\exists R_1 \sqsubseteq A_1$, or $R_1 \sqsubseteq (\neg)R_2$, where A_1 and A_2 are in N_C, and R_1 and R_2 are basic roles. Each DL-Lite$_R$ TBox $\mathcal{T}$ can be transformed into an equisatisfiable DL-Lite$_R$ TBox $\mathcal{T}'$ in normal form by systematically replacing complex concepts with atomic ones along the lines of Baader et al. [BBL05].

4.2 Approach Overview

Our main objective is to devise a CQ rewriting algorithm for DL-Lite$_R$. Our rewriting algorithm should take as input a CQ Q and a DL-Lite$_R$ TBox $\mathcal{T}$, and it should produce as output a rewriting of Q with respect to $\mathcal{T}$ (see Section 3.5). This rewriting has been defined as a DQ; however, it follows from [CDL+07] that in the case of DL-Lite$_R$ rewritings can be expressed as UCQs. Therefore, we additionally impose the condition that our algorithm should be *optimal for DL-Lite$_R$* —that is, it should compute UCQs. We derive the rewriting algorithm in two phases: we first devise an algorithm for CQ *answering* and then we slightly modify it in order to derive the rewriting algorithm.

According to the definition of certain answers, it follows that a tuple of constants $\vec{a}$ is a certain answer to a CQ $Q = \langle Q_P, Q_C \rangle$ over a DL-Lite$_R$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ if and only if $Q_P(\vec{a})$ is a logical consequence of the FOL formula $\pi(\mathcal{K}) \wedge \epsilon(Q_C)$, where $\pi(\mathcal{K})$ and $\epsilon(Q_C)$ respectively correspond to the translation of $\mathcal{K}$ and Q_C to FOL (see Section 3.5). Note that if $\mathcal{K}$ is unsatisfiable, then $\mathsf{ans}(Q, \mathcal{K})$ is trivially the set of all possible n-ary tuples of constants of $\mathcal{K}$, where n is the arity of Q_P. We develop our approach under the assumption that the input KB is satisfiable; therefore, before posing queries over a DL-Lite$_R$ KB $\mathcal{K}$, it is important to verify that $\mathcal{K}$ is satisfiable.

A DL-Lite$_R$ KB may be unsatisfiable due to the presence of negative inclusions that can lead to contradictions. Consider the following example:

Example 4.2.1. Consider a DL-Lite$_R$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ such that $\mathcal{T}$ consists of the following axiom:

$$\text{Theist} \sqsubseteq \neg \text{Atheist} \tag{4.1}$$

Axiom (4.1) intuitively says that if an individual is an instance of Theist, then it is not an instance of Atheist. Let $\mathcal{A}$ contain the assertions Theist(John) and Atheist(John). Clearly, the axiom in $\mathcal{T}$ together with the assertions in $\mathcal{A}$ are contradictory; therefore, $\mathcal{K}$ can have no model, i.e., it is unsatisfiable. △

As shown in [CDL+07], it is possible to reduce the problem of KB satisfiability for DL-Lite$_R$ to the problem of CQ answering by transforming the negative inclusions of a given KB into CQs: one can reduce KB satisfiability to CQ answering by transforming every negative inclusion $\alpha \in \mathcal{T}_{\text{NI}}$ into a boolean CQ $\mathsf{sat}(\alpha)$ such that $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if for every $\alpha \in \mathcal{T}_{\text{NI}}$, the query $\mathsf{sat}(\alpha)$ is false over $\mathcal{K}' = \langle \mathcal{T}_{\text{PI}}, \mathcal{A} \rangle$. We formally define the transformation of negative inclusions into CQs as follows.

Definition 4.2.1. Let α be a DL-Lite$_R$ negative inclusion, let B_1 and B_2 be atomic concepts, and let R_1 and R_2 be atomic roles. Let $\mathsf{sat}(\alpha) = \langle Q_P, Q_C \rangle$, where

- if $\alpha = B_1 \sqsubseteq \neg B_2$, then $Q_C = \{Q_P() \leftarrow B_1(x) \wedge B_2(x)\}$
- if $\alpha = R_1 \sqsubseteq \neg R_2$ or $\alpha = R_1^- \sqsubseteq \neg R_2^-$, then $Q_C = \{Q_P() \leftarrow R_1(x, y) \wedge R_2(x, y)\}$
- if $\alpha = R_1 \sqsubseteq \neg R_2^-$ or $\alpha = R_1^- \sqsubseteq \neg R_2$, then $Q_C = \{Q_P() \leftarrow R_1(x, y) \wedge R_2(y, x)\}$

△

Lemma 4.2.1 (cf. [CDL+07])**.** *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_R$ KB. $\mathcal{K}$ is satisfiable if and only if for every $\alpha \in \mathcal{T}_{NI}$ we have that $\mathsf{ans}(\mathsf{sat}(\alpha), \mathcal{K}') = \emptyset$, where $\mathcal{K}' = \langle \mathcal{T}_{PI}, \mathcal{A} \rangle$.*

Example 4.2.2. Consider the DL-Lite$_R$ KB $\mathcal{K}$ defined in Example 4.2.1. We can check whether $\mathcal{K}$ is unsatisfiable by answering the boolean query

$$\mathsf{sat}((4.1)) = Q_P() \leftarrow \text{Theist}(x) \wedge \text{Atheist}(x)$$

over $\mathcal{K}' = \langle \mathcal{T}_{\text{PI}}, \mathcal{A} \rangle$. In this case, we have that $\mathsf{ans}(\mathsf{sat}(\alpha), \mathcal{K}') \neq \emptyset$ (because it contains the empty tuple); therefore, $\mathcal{K}$ is unsatisfiable. △

Since the language of CQs does not allow for negated atoms and DL-Lite$_R$ does not allow to model disjunctive knowledge, the set $\mathcal{T}_{\text{NI}}$ of negative inclusions can be simply regarded as a set of "constraints" that can only be used to check the consistency of $\mathcal{A}$ with respect to $\mathcal{T}$. An important consequence of this fact is that if the input KB is satisfiable, then the negative inclusions are not needed for CQ answering; that is, for every satisfiable DL-Lite$_R$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we have that $\mathsf{ans}(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = \mathsf{ans}(Q, \langle \mathcal{T}_{\text{PI}}, \mathcal{A} \rangle)$. This proposition has been proved by Calvanese et al. [CDL+07]. Intuitively, the proposition follows from the fact that negative inclusions cannot be used to derive positive facts; this is so given that negative inclusions can only be used to derive negative facts directly, no axiom in $\mathcal{T}$ contains negated atoms on the left-hand-side or union on the right-hand-side, and Q does not contain negated atoms in the body. As previously mentioned, we develop our answering algorithm under the assumption that $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable; therefore, in the rest of this chapter we assume that $\mathcal{T}$ does not contain negative inclusions.

By the definition of certain answers, it follows that $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if the logic program $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable, where $\Xi(\mathcal{K})$ is the set of clauses that corresponds to the clausal normal form [Fit96] of the FOL formula $\pi(\mathcal{K})$ (see Definition 3.3.3), and $\bot$ is the logical constant False. It is easy to see that $\Xi(\cdot)$ corresponds to the transformation shown in Table 4.1.

Our CQ answering algorithm is based on a procedure that decides whether $\vec{a}$ is in $\mathsf{ans}(Q, \mathcal{K})$ by checking whether $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable. In order to do so, we employ a resolution-based procedure that verifies whether the empty clause is derivable from $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ by $\mathcal{R}^{\text{lite}}$—an RFS calculus that we shall define later. Note, however, that such an approach allows one to decide only whether some tuple $\vec{a}$ is an answer to Q over $\mathcal{K}$. In order to compute the entire set

Table 4.1: Translating a DL-Lite$_R$ KB $\mathcal{K}$ into a set of clauses $\Xi(\mathcal{K})$

DL-Lite$_R$ clause(s)	DL-Lite$_R$ axiom
$A(a)$	$A(a)$
$P(a, b)$	$P(a, b)$
$A_2(x) \leftarrow A_1(x)$	$A_1 \sqsubseteq A_2$
$P(x, f(x)) \leftarrow A(x)$	$A \sqsubseteq \exists P$
$P(x, f(x)) \leftarrow A_1(x)$ $A_2(f(x)) \leftarrow A_1(x)$	$A_1 \sqsubseteq \exists P.A_2$
$P(f(x), x) \leftarrow A(x)$	$A \sqsubseteq \exists P^-$
$P(f(x), x) \leftarrow A_1(x)$ $A_2(f(x)) \leftarrow A_1(x)$	$A_1 \sqsubseteq \exists P^-.A_2$
$A(x) \leftarrow P(x, y)$	$\exists P \sqsubseteq A$
$A(x) \leftarrow P(y, x)$	$\exists P^- \sqsubseteq A$
$S(x, y) \leftarrow P(x, y)$	$P \sqsubseteq S, P^- \sqsubseteq S^-$
$S(x, y) \leftarrow P(y, x)$	$P^- \sqsubseteq S, P \sqsubseteq S^-$

Note 4.2.1. Each axiom of the form $A \sqsubseteq \exists R$ or $A_1 \sqsubseteq \exists R.A_2$ is uniquely associated with a distinct function symbol f.

$\mathsf{ans}(Q, \mathcal{K})$, we apply the so-called *answer literal technique* (cf. [Gre69]); that is, we show that

$$\mathsf{ans}(Q, \mathcal{K}) = \{Q_P(\vec{a}) \mid Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}\}.$$

Therefore, our answering algorithm amounts to saturating $\Xi(\mathcal{K}) \cup Q_C$ by $\mathcal{R}^{\text{lite}}$ and returning each tuple $\vec{a}$ such that $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$. Since every RFS calculus is sound and complete [BG01], the main challenge in proving that our algorithm is correct is to ensure that the saturation of $\Xi(\mathcal{K}) \cup Q_C$ by $\mathcal{R}^{\text{lite}}$ always terminates.

We present our answering algorithm in Section 4.3. Based on this algorithm, in Section 4.4 we present $\mathsf{RQR}^{\text{lite}}$—a CQ *rewriting* algorithm for DL-Lite$_R$. $\mathsf{RQR}^{\text{lite}}$ takes as input Q and $\mathcal{T}$ and computes a UCQ by saturating $\Xi(\mathcal{T}) \cup Q_C$ by $\mathcal{R}^{\text{lite}}$. In a nutshell, $\mathsf{RQR}^{\text{lite}}$ first computes $(\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$ and then it transforms it into a UCQ. As we shall see, the set $(\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$ is a logic program in general; therefore,

the main challenge in devising our rewriting algorithm is to ensure that such a logic program can always be translated into a UCQ that is a rewriting of Q with respect to $\mathcal{T}$.

4.3 CQ Answering for DL-Lite$_R$

Given a CQ $Q = \langle Q_P, Q_C \rangle$ and a DL-Lite$_R$ KB $\mathcal{K}$, our CQ answering algorithm returns every tuple $\vec{a}$ such that $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$, where $\mathcal{R}^{\text{lite}}$ is the RFS calculus defined in Definition 4.3.2 (given later). Before formally showing the correctness of our algorithm, we present an example where we give informal intuitive explanations.

Example 4.3.1. Consider a DL-Lite$_R$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ about monotheistic religions. Let $\mathcal{T}$ contain the following axioms:

$$\begin{aligned} \text{Catholic} &\sqsubseteq \text{Devotee} \\ \text{Devotee} &\sqsubseteq \exists \text{hasReligion} \\ \exists \text{hasReligion} &\sqsubseteq \text{Theist} \\ \text{Theist} &\sqsubseteq \exists \text{believesIn.Deity} \\ \text{praysTo} &\sqsubseteq \text{believesIn} \end{aligned}$$

The TBox $\mathcal{T}$ states that catholics are devotees, that a devotee has at least one religion, that someone who has religion is a theist, that a theist is someone who believes in a deity, and that anyone who prays to something believes in it.

Let $\mathcal{A}$ contain the following assertions:[2]

$$\underline{\text{Catholic}(\mathsf{BenedictXVI})} \tag{4.2}$$

$$\underline{\text{praysTo}(\mathsf{John}, \mathsf{FSM})} \tag{4.3}$$

$$\underline{\text{Deity}(\mathsf{FSM})} \tag{4.4}$$

[2]We underline the atoms selected by the selection function of $\mathcal{R}^{\text{lite}}$.

The ABox $\mathcal{A}$ states that BenedictXVI is catholic, that John prays to FSM (i.e., the Flying Spaghetti Monster), and that FSM is a deity.

Finally, consider the following query Q:

$$Q_P(x) \leftarrow \underline{\text{believesIn}(x, y)} \wedge \underline{\text{Deity}(y)} \tag{4.5}$$

We now show how our answering algorithm obtains the set $\mathsf{ans}(Q, \mathcal{K})$. We start by translating $\mathcal{T}$ into the set of clauses $\Xi(\mathcal{T})$. According to Table 4.1, the set $\Xi(\mathcal{T})$ contains the following clauses:

$$\text{Devotee}(x) \leftarrow \underline{\text{Catholic}(x)} \tag{4.6}$$

$$\underline{\text{hasReligion}(x, \text{religionOf}(x))} \leftarrow \text{Devotee}(x) \tag{4.7}$$

$$\text{Theist}(x) \leftarrow \underline{\text{hasReligion}(x, y)} \tag{4.8}$$

$$\underline{\text{believesIn}(x, \text{deityOf}(x))} \leftarrow \text{Theist}(x) \tag{4.9}$$

$$\underline{\text{Deity}(\text{deityOf}(x))} \leftarrow \text{Theist}(x) \tag{4.10}$$

$$\text{believesIn}(x, y) \leftarrow \underline{\text{praysTo}(x, y)} \tag{4.11}$$

After the logic program $\Xi(\mathcal{K}) \cup \{Q\}$ is computed, we saturate it using our RFS calculus $\mathcal{R}^{\text{lite}}$. Note that this calculus is parameterized in a particular way, which determines the selected atoms in each clause and, consequently, the inferred clauses. The saturation $(\Xi(\mathcal{K}) \cup \{Q\})_{\mathcal{R}^{\text{lite}}}$ contains the following clauses:[3]

$$[(4.7) + (4.8)] \quad \text{Theist}(x) \leftarrow \underline{\text{Devotee}(x)} \tag{4.12}$$

$$[(4.5) + (4.9)] \quad Q_P(x) \leftarrow \text{Theist}(x) \wedge \underline{\text{Deity}(\text{deityOf}(x))} \tag{4.13}$$

$$[(4.13) + (4.10)] \quad Q_P(x) \leftarrow \underline{\text{Theist}(x)} \tag{4.14}$$

$$[(4.2) + (4.6)] \quad \underline{\text{Devotee}(\mathsf{BenedictXVI})} \tag{4.15}$$

$$[(4.15) + (4.12)] \quad \underline{\text{Theist}(\mathsf{BenedictXVI})} \tag{4.16}$$

[3]The expression $[(x) + (y)]$ next to each clause states that the corresponding clause was obtained by resolving clauses (x) and (y).

$$[(4.16) + (4.14)] \quad \underline{Q_P(\mathsf{BenedictXVI})} \tag{4.17}$$

$$[(4.3) + (4.11)] \quad \underline{\text{believesIn}(\mathsf{John}, \mathsf{FSM})} \tag{4.18}$$

$$[(4.18) + (4.5)] \quad Q_P(\mathsf{John}) \leftarrow \underline{\text{Deity}(\mathsf{FSM})} \tag{4.19}$$

$$[(4.19) + (4.4)] \quad \underline{Q_P(\mathsf{John})} \tag{4.20}$$

It can be shown that (4.17) and (4.20) are the only clauses of the form $Q_P(\vec{a})$ contained in $(\Xi(\mathcal{K}) \cup \{Q\})_{\mathcal{R}^{\text{lite}}}$; so, our algorithm returns $\{\mathsf{BenedictXVI}, \mathsf{John}\}$. △

As mentioned in Section 4.2, it follows from the definition of certain answers that a tuple $\vec{a}$ is an answer of a CQ $Q = \langle Q_P, Q_C \rangle$ with respect to a KB $\mathcal{K}$ if and only if the logic program $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable. Therefore, to answer Q over $\mathcal{K}$, we need a decision procedure for checking satisfiability of the latter set of clauses. We derive this procedure using the principles for deciding a first-order fragment $\mathcal{L}$ by resolution outlined by Joyner [Joy76]. First, one selects a sound and complete clausal calculus $\mathcal{R}$; second, one identifies a set of clauses $\mathcal{N}$ such that (i) $\mathcal{N}$ is finite for a finite signature, and (ii) the translation of each formula $\alpha \in \mathcal{L}$ into clauses produces only clauses from $\mathcal{N}$; and third, one demonstrates that $\mathcal{N}$ is closed under $\mathcal{R}$—that is, one shows that applying an inference of $\mathcal{R}$ to clauses from $\mathcal{N}$ produces a clause in $\mathcal{N}$. This is sufficient to obtain a refutation decision procedure for $\mathcal{L}$: given any formula $\alpha \in \mathcal{L}$, a saturation by $\mathcal{R}$ of the clauses corresponding to α will, in the worst case, derive all clauses of $\mathcal{N}$. Therefore, provided we find a suitable clause set closed under a sound and complete clausal calculus, we can decide whether $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable—that is, whether $\vec{a}$ is an answer to Q over $\mathcal{K}$.

As noted before, this approach allows us to check only whether some tuple $\vec{a}$ is an answer to Q over $\mathcal{K}$. In order to compute the set $\mathsf{ans}(Q, \mathcal{K})$, we make use of the *answer literal* technique [Gre69]: instead of saturating $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$,

Table 4.2: DL-Lite$_R$ types of clause

Type	DL-Lite$_R$ clause
1	$Q_P(\vec{s}) \leftarrow \bigwedge D_i(\vec{t_i})$
2	$D(\vec{s}) \leftarrow [A(x)]$
3.1	$A(x) \leftarrow \underline{P(x,y)}$
3.2	$A(x) \leftarrow \underline{P(y,x)}$
3.3	$P(x,y) \leftarrow \underline{S(x,y)}$
3.4	$P(x,y) \leftarrow \underline{S(y,x)}$

Note 4.3.1. D (possibly with subscripts) denotes a unary or a binary predicate; x, and y denote variables; s and t (possibly with subscripts) denote terms; and $[A(x)]$ denotes a possible occurrence of the atom $A(x)$ in the clause.

we saturate $\Xi(\mathcal{K}) \cup Q_C$ by $\mathcal{R}^{\text{lite}}$. The following lemma shows that by doing so we shall compute all answers to Q over $\mathcal{K}$.

Lemma 4.3.1. *Let $\mathcal{K}$ be a DL-Lite$_R$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. We have that $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{lite}}$.*

Proof. By the definition of the certain answers, we have that $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable. We show that the latter is the case if and only if $Q_P(\vec{a})$ is contained in $(\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$. The ($\Leftarrow$) direction is trivial. For the ($\Rightarrow$) direction, note that $\Xi(\mathcal{K}) \cup Q_C$ does not contain clauses with $\bot$ in the head, so a saturation of $\Xi(\mathcal{K}) \cup Q_C$ by $\mathcal{R}^{\text{lite}}$ cannot derive the empty clause. Furthermore, the predicate Q_P does not occur in the body of any clause in $\Xi(\mathcal{K}) \cup Q_C$; thus, $\mathcal{R}^{\text{lite}}$ can derive the empty clause from $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$. Since any RFS calculus is sound and complete [BG01], the claim follows. □

We now apply Joyner's principles and the answer literal technique to show how to compute $\mathsf{ans}(Q, \mathcal{K})$ using resolution. We define the clause set $\mathcal{N}^{\text{lite}}$ as follows.

Definition 4.3.1. Let $\mathcal{K}$ be a DL-Lite$_R$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. The clause set

$\mathcal{N}^{\text{lite}}$ for $Q = \langle Q_P, Q_C \rangle$ and $\Xi(\mathcal{K})$ is the set containing every safe clause C constructed with the symbols in Q and $\Xi(\mathcal{K})$ such that:

- (i) C is of one of the types shown in Table 4.2.
- If C is of type 1, then
  - (ii) $\mathsf{varNo}(C) \leq \mathsf{varNo}(C')$ where C' is the only clause in Q_C,
  - (iii) $\mathsf{depth}(C) \leq \max(1, \mathsf{varNo}(C') - \mathsf{varNo}(C))$ where C' is the only clause in Q_C, and
  - (iv) if a variable x occurs in a functional term in C, then x occurs in every functional term in C.
- If C is of type 2, then
  - (v) $\mathsf{varNo}(C) \leq 1$, and
  - (vi) the head of C is of the form $A(a)$, $A(x)$, $A(f(x))$, $P(a, b)$, $P(x, f(x))$, or $P(f(x), x)$.

△

Clearly, $\mathcal{N}^{\text{lite}}$ is finite assuming that Q and $\Xi(\mathcal{K})$ are finite; moreover, it can be readily verified that $\Xi(\mathcal{K}) \cup Q_C \subseteq \mathcal{N}^{\text{lite}}$. In the rest of this section we define $\mathcal{R}^{\text{lite}}$ and we show that $\mathcal{N}^{\text{lite}}$ is closed under $\mathcal{R}^{\text{lite}}$. We formally define $\mathcal{R}^{\text{lite}}$ as follows.

Definition 4.3.2. With $\mathcal{R}^{\text{lite}}$ we denote the RFS calculus parameterized with the selection function S^{lite} defined as follows.

- If $C \in \mathcal{N}^{\text{lite}}$ is of type 1 or 2, then S^{lite} selects the head atom H if the body is empty or if H contains function symbols; otherwise, S^{lite} selects all deepest body atoms.

Table 4.3: Inferences of $\mathcal{R}^{\text{lite}}$ on $\mathcal{N}^{\text{lite}}$

1 + 2 = 1:

$$\frac{Q_P(\vec{s}) \leftarrow \underline{D(\vec{t})} \wedge \bigwedge D_i(\vec{u_i}) \quad \underline{D(\vec{v})} \leftarrow [A(x)]}{Q_P(\vec{s})\sigma \leftarrow [A(x)\sigma] \wedge \bigwedge D_i(\vec{u_i})\sigma}$$

(a) $\sigma = \mathsf{MGU}(D(\vec{t}), D(\vec{v}))$

2 + 2 = 2:

$$\frac{C(x) \leftarrow \underline{B(x)} \quad \underline{B(s)} \leftarrow [A(x)]}{C(s) \leftarrow [A(x)]}$$

3.1 + 2 = 2:

$$\frac{B(x) \leftarrow \underline{P(x,y)} \quad \underline{P(s,t)} \leftarrow [A(x)]}{B(s) \leftarrow [A(x)]}$$

3.2 + 2 = 2 is analogous.

3.3 + 2 = 2:

$$\frac{S(x,y) \leftarrow \underline{P(x,y)} \quad \underline{P(s,t)} \leftarrow [A(x)]}{S(s,t) \leftarrow [A(x)]}$$

3.4 + 2 = 2 is analogous.

Note 4.3.2. The notation $X + Y = Z$ denotes that resolving a clause of type X with a clause of type Y produces a clause of type Z.

- If $C \in \mathcal{N}^{\text{lite}}$ is of type 3.1–3.4, then S^{lite} selects the atom underlined in Table 4.2.

△

Since $\mathcal{R}^{\text{lite}}$ is sound and complete [BG01], in order to obtain a decision procedure we only need to show that for every set of clauses $P \subseteq \mathcal{N}^{\text{lite}}$, a saturation by $\mathcal{R}^{\text{lite}}$ of P terminates. We do this in Lemma 4.3.2 by showing that $\mathcal{N}^{\text{lite}}$ is closed under $\mathcal{R}^{\text{lite}}$ for each $\Xi(\mathcal{T})$ and Q.

Lemma 4.3.2. *Let $\mathcal{K}$ be a DL-Lite$_R$ KB, Q a CQ, and $\mathcal{N}^{lite}$ the clause set for $\Xi(\mathcal{K})$ and Q. For every two clauses C_1 and C_2 in $\mathcal{N}^{lite}$, and every resolvent C_r of C_1 and C_2 by $\mathcal{R}^{lite}$, we have that $C_r \in \mathcal{N}^{lite}$.*

Proof. The possible inferences are summarized in Table 4.3. As can be seen, if neither C_1 nor C_2 is of type 1, it is straightforward to check that $C_r \in \mathcal{N}^{\text{lite}}$; hence, we assume

that C_1 is of type 1. For resolution to be possible, C_2 must be of type 2 and the selection function S^{lite} must select the head $D(\vec{v})$ of C_2. By the definition of S^{lite}, if $D(\vec{v})$ is of the form $A(a)$ or $P(a, b)$, then the body of C_2 is empty. Therefore, in this case, the inference only reduces the number of body atoms of C_1, so C_r is of type 1. We now consider the case where $D(\vec{v})$ is of the form $P(x, f(x))$; the proof for the other forms of $D(\vec{v})$ is analogous. Let $D(\vec{t})$ be the atom selected in C_1 to resolve with C_2 to produce C_r. We assume that $D(\vec{t})$ is of the form $P(s, u)$. For unification to be possible, the term u must be a variable x_u or a functional term of the form $f(u')$.

- Let u be a variable x_u and $\sigma = \{x \mapsto s, x_u \mapsto f(s)\}$. Due to the occurs-check in unification, x_u cannot occur in s. The inference thus decreases the number of variables by one, so C_r satisfies condition (ii). Moreover, since C_1 satisfies condition (iv), x_u does not occur in any functional term in C_1 (because it does not occur in s). Hence, even though x_u is mapped to a functional term, we have that $\mathsf{depth}(C_r) \leq \mathsf{depth}(C_1) + 1$ and since the number of variables is decreased by one in C_r, condition (iii) is satisfied. Finally, since every occurrence of x_u is replaced with the same term, the resolvent C_r satisfies condition (iv) as well.

- Let u be a functional term $f(u')$. Clearly, either $s = u'$ and $\sigma = \{x \mapsto u'\}$; s is a variable x_s (not occurring in u') and $\sigma = \{x \mapsto u', x_s \mapsto u'\}$; or u' is a variable x'_u and $\sigma = \{x \mapsto s, x'_u \mapsto s\}$. In all cases, the inference does not increase the number of variables or function symbols; therefore, C_r satisfies conditions (ii) and (iii). Furthermore, the inference does not introduce new function symbols, so C_r satisfies condition (iv) as well.

□

Theorem 4.3.1 follows from Lemma 4.3.1 and Lemma 4.3.2.

Theorem 4.3.1. *For $\mathcal{K}$ a DL-Lite$_R$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ, the saturation of $\Xi(\mathcal{K}) \cup Q_C$ by $\mathcal{R}^{lite}$ terminates and $\mathsf{ans}(Q, \mathcal{K}) = \{\vec{a} \mid Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{lite}}\}$.*

4.4 CQ Rewriting for DL-Lite$_R$

In this section we present $\mathsf{RQR}^{\text{lite}}$—a CQ rewriting algorithm for DL-Lite$_R$. We derive such an algorithm from the answering algorithm presented in Section 4.3: we show that in order to answer Q with respect to $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, one can *first* saturate $\Xi(\mathcal{T}) \cup Q_C$ with $\mathcal{R}^{\text{lite}}$ to produce a rewriting of Q with respect to $\mathcal{T}$, and *then* evaluate this rewriting over $\mathcal{A}$. Our goal is to obtain a worst-case optimal rewriting for DL-Lite$_R$—that is, $\mathsf{RQR}^{\text{lite}}$ should produce UCQs.

In a nutshell, $\mathsf{RQR}^{\text{lite}}$ takes as input Q and $\mathcal{T}$, and proceeds in two steps: in the *saturation step*, the set $\Xi(\mathcal{T}) \cup Q_C$ is saturated using $\mathcal{R}^{\text{lite}}$ and the resulting logic program is transformed into a datalog program $\mathsf{ff}^{\text{lite}}(Q, \mathcal{T})$; and in the *unfolding step*, the datalog program is transformed into a UCQ $\mathsf{RQR}^{\text{lite}}(Q, \mathcal{T})$. By Theorem 4.3.1, we know that every saturation by $\mathcal{R}^{\text{lite}}$ terminates; hence, the main challenge in computing the rewriting lies in ensuring that $(\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$ can always be transformed into a UCQ that is a rewriting of Q with respect to $\mathcal{T}$. In the rest of this section, we first show how to convert $(\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$ into a datalog program and then we present an optimization step to obtain rewritings of optimal form.

4.4.1 Elimination of Function Symbols

A naive attempt to compute a rewriting of $Q = \langle Q_P, Q_C \rangle$ with respect to $\mathcal{T}$ consists of translating Q and $\mathcal{T}$ into the set of clauses $\Xi(\mathcal{T}) \cup Q_C$. In case $\Xi(\mathcal{T}) \cup Q_C$ does not contain function symbols, then the problem is trivially solved. This is, however, not the case in general even for the rather inexpressive DL-Lite$_R$. Typically, the set $\Xi(\mathcal{T}) \cup Q_C$ is a logic program—that is, it contains function symbols.

Functional terms in $\Xi(\mathcal{T})$ represent individuals that are known to exist according to $\mathcal{T}$ but whose names are unknown. Such individuals arise when $\mathcal{T}$ contains axioms with existential quantification on the right-hand side. Consider the following example.

Example 4.4.1. Consider a DL-Lite$_R$ TBox $\mathcal{T}$ consisting of the following axiom:

$$\text{Devotee} \sqsubseteq \exists \text{isDevoteeOf.Religion}$$

According to Table 4.1, we have that $\Xi(\mathcal{T})$ consists of the following clauses:

$$\text{isDevoteeOf}(x, f(x)) \leftarrow \text{Devotee}(x) \tag{4.21}$$

$$\text{Religion}(f(x)) \leftarrow \text{Devotee}(x) \tag{4.22}$$

The TBox $\mathcal{T}$ states that every devotee is a devotee of at least one religion. Hence, if we knew that, say, the individual Ian is an instance of Devotee, then it would follow that there is at least one instance of Religion (represented with the functional term $f(\mathsf{Ian})$ and whose name is unknown) to which Ian is related via the role isDevoteeOf. △

In order to transform $\Xi(\mathcal{T}) \cup Q_C$ into a DQ, we employ the RFS calculus $\mathcal{R}^{\text{lite}}$: we first compute the set $(\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$ and then discard all clauses containing function symbols. Intuitively, we can simply discard such clauses because by saturating $\Xi(\mathcal{T}) \cup Q_C$ we obtain new clauses that act as "inference shortcuts" rendering clauses containing function symbols unnecessary to derive new facts. We illustrate this point with an example.

Example 4.4.2. Consider a DL-Lite$_R$ TBox $\mathcal{T}$ containing the following axioms:

$$\text{Devotee} \sqsubseteq \exists \text{hasReligion}$$

$$\exists \text{hasReligion} \sqsubseteq \text{Theist}$$

The TBox $\mathcal{T}$ states that all devotees have at least one religion and that someone that has a religion is a theist. Clearly, we have that $\mathcal{T} \models \text{Devotee} \sqsubseteq \text{Theist}$.

According to Table 4.1, the set $\Xi(\mathcal{T})$ contains the following clauses:

$$\underline{\text{hasReligion}(x, f(x))} \leftarrow \text{Devotee}(x) \tag{4.23}$$

$$\text{Theist}(x) \leftarrow \underline{\text{hasReligion}(x, y)} \tag{4.24}$$

Resolving (4.23) with (4.24) using $\mathcal{R}^{\text{lite}}$ produces

$$\text{Theist}(x) \leftarrow \text{Devotee}(x). \tag{4.25}$$

As can be seen, the saturation of $\Xi(\mathcal{T})$ by $\mathcal{R}^{\text{lite}}$ produced a new *function-free* "shortcut" clause (4.25) that explicitly represents that $\mathcal{T} \models \text{Devotee} \sqsubseteq \text{Theist}$. △

According to the definition of the certain answers, we are only interested in tuples of *constants* (not of functional terms); moreover, as we shall show, every function-free clause (such as (4.25)) that can be used to derive new facts will be contained in the saturation of $\Xi(\mathcal{T}) \cup Q_C$. Therefore, we can safely discard every clause containing function symbols in $(\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$. The following definition summarizes the first step of $\mathsf{RQR}^{\text{lite}}$.

Definition 4.4.1. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_R$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. With $\mathsf{ff}^{\text{lite}}(Q, \mathcal{T})$ we denote the set that contains exactly every function-free clause contained in $(\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$. △

We now show that the DQ $\langle Q_P, \mathsf{ff}^{\text{lite}}(Q, \mathcal{T}) \rangle$ is a rewriting of $Q = \langle Q_P, Q_C \rangle$ with respect to $\mathcal{T}$, albeit not necessarily an optimal one. To this end, we first prove that in order to compute the answers to Q over a DL-Lite$_R$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we can always "postpone" the inferences involving ground facts of the form $A(a)$ or $P(a, b)$ in the saturation of $\Xi(\mathcal{K}) \cup Q_C$.

Lemma 4.4.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_R$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. For every clause C of type 1, if $C \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{lite}}$, then there is a clause C' of type 1 such that $C' \in (\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{lite}}$ and there is a derivation tree for C from $\{C'\} \cup \mathcal{G}$, where $\mathcal{G}$ is the set containing exactly every clause of the form $A(a)$ or $P(a, b)$ contained in $(\mathsf{ff}^{lite}(Q, \mathcal{T}) \cup \mathcal{A})_{\mathcal{R}^{lite}}$.*

Proof. We prove the claim by induction on the height of a derivation tree of C. If the derivation tree has height zero, then C' is the only clause contained in Q_C, and

the claim trivially follows. We assume that the claim holds for each clause derived by a derivation tree of height n, and consider a clause C derived by a derivation tree of height $n+1$. Let C_1 and C_2 be the clauses that are resolved to obtain C. According to Table 4.3, one of such clauses is of type 1 and the other is of type 2. Without loss of generality we assume that C_1 is of type 1 and C_2 is of type 2. Hence, by the induction hypothesis, there is a clause C_1' of type 1 such that $C_1' \in (\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$ and C_1 can be derived from $\{C_1'\} \cup \mathcal{G}$.

The selection function S^{lite} selects a deepest covering body atom in C_1 and the head $D(\vec{s})$ of C_2, which is of the form $A(a)$, $A(f(x))$, $R(a,b)$, $R(x,f(x))$, or $R(f(x),x)$.

If $D(\vec{s})$ is ground, it follows from the definition of $\mathcal{R}^{\text{lite}}$ that the body of C_2 is empty. Given the fact that S^{lite} selects the deepest atoms and that function symbols do not unify with constants, we have that every fact has function-free premises or is contained in $\mathcal{A}$. Therefore, we have that $C_2 \in \mathcal{G}$ and the claim follows for $C' = C_1'$.

We now consider the case where $D(\vec{s})$ is not ground. We show the claim for the case where $D(\vec{s})$ is of the form $R(x,f(x))$; the proof for the other cases is analogous. Let C_2 be a clause of the form (4.26) shown below. Note that every clause of type 2 is safe and contains at most one variable; therefore, since S^{lite} selects the covering atoms of this type of clauses, we have that if $D(\vec{s})$ is not ground, then $C_2 \in (\Xi(\mathcal{T}))_{\mathcal{R}^{\text{lite}}}$.

We know there is a derivation tree for C_1 from $\{C_1'\} \cup \mathcal{G}$. Therefore, given the fact that $\mathcal{G}$ contains only ground unit clauses, we have that a set $\{D_k(\vec{a_k})\} \subseteq \mathcal{G}$ exists such that resolving C_1' on body atoms $\{D_k(\vec{r_k})\}$ with the atoms of $\{D_k(\vec{a_k})\}$ produces C_1. Furthermore, all such resolution inferences just remove body atoms; hence, since C_1 is to be resolved with C_2, the clause C_1' is of the form (4.27), and C_1 of the form (4.28), where δ maps the variables occurring in $\{D_k(\vec{r_k})\}$ to some constants occurring in $\{D_k(\vec{a_k})\}$. Note that no inference used to derive C_1 changes the number of function symbols of C_1'; therefore, since $R(s,t)\delta$ is deepest in C_1, we have that $R(s,t)$ is deepest in C_1'. Moreover, each variable of C_1' that is replaced by δ with a constant clearly

does not occur in $R(s,t)\delta$; hence, the substitutions δ and σ have disjoint domains, and $\sigma = \delta\sigma$. Resolving C_1 and C_2 produces C, a clause of the form (4.29), where $\sigma = \mathsf{MGU}(R(s,t)\delta, R(x, f(x)))$.

$$C_2 = \underline{R(x, f(x))} \leftarrow [A(x)] \tag{4.26}$$

$$C_1' = Q_P(\vec{u}) \leftarrow \underline{R(s,t)} \wedge \bigwedge D_j(\vec{w_j}) \wedge \bigwedge D_k(\vec{r_k}) \tag{4.27}$$

$$C_1 = Q_P(\vec{u})\delta \leftarrow \underline{R(s,t)\delta} \wedge \bigwedge D_j(\vec{w_j})\delta \tag{4.28}$$

$$C = Q_P(\vec{u})\delta\sigma \leftarrow [A(x)\sigma] \wedge \bigwedge D_j(\vec{w_j})\delta\sigma \tag{4.29}$$

We now transform the derivation for C into a derivation in which all inferences with clauses in $\mathcal{G}$ are performed in the end. Let C' be the clause obtained by resolving C_1' and C_2. Given the form of C_1' and C_2, the clause C' is of the form (4.30), where $\sigma' = \mathsf{MGU}(R(s,t), R(x, f(x)))$.

$$C' = Q_P(\vec{u})\sigma' \leftarrow [A(x)\sigma'] \wedge \bigwedge D_j(\vec{w_j})\sigma' \wedge \bigwedge D_k(\vec{r_k})\sigma' \tag{4.30}$$

Since $R(s,t)$ is deepest in C_1', the inference between C_1' and C_2 satisfies the selection function of $\mathcal{R}^{\text{lite}}$. Moreover, since both C_1' and C_2 are derivable from $\Xi(\mathcal{T}) \cup Q_C$, the clause C' is derivable from $\Xi(\mathcal{T}) \cup Q_C$ as well. Let D now be the clause obtained by resolving $\{D_k(\vec{r_k})\sigma'\}$ in C' with atoms of $\mathcal{G}$. The clause D has form (4.31), where δ' maps some variables of C' to constants.

$$D = Q_P(\vec{u})\sigma'\delta' \leftarrow [A(x)\sigma'\delta'] \wedge \bigwedge D_j(\vec{w_j})\sigma'\delta' \tag{4.31}$$

We now prove that $C = D$. Given the forms of C and D, and since $\sigma = \delta\sigma$, it suffices to show that $\delta\sigma = \sigma'\delta'$. Let y be a variable that occurs in $R(s,t)$ that is replaced by δ with a constant. Clearly, σ does not contain such y; therefore, without loss of generality we can assume that (*) if σ maps a variable z to a constant $y\delta$, then σ' maps z to y. Note that σ' may still map a variable y_1 occurring in δ to another

variable y_2 occurring in δ, in which case we have that $y_1\delta = y_2\delta$. Due to (*), σ and σ' have the same domain modulo variables such as y_1. Even though $\{D_k(\vec{r_k})\}$ and $\{D_k(\vec{r_k})\sigma'\}$ are not necessarily the same (both y_1 and y_2 can occur in $\{D_k(\vec{r_k})\}$, while only y_2 can occur in $\{D_k(\vec{r_k})\sigma'\}$), note that since σ' maps y_1 to another *variable* y_2, the atoms $\{D_k(\vec{r_k})\sigma'\}$ can be resolved with the ground atoms $\{D_k(\vec{a_k})\}$ via δ'. So, we know that σ and σ' have the same domain modulo variables such as y_1; moreover, the only difference between δ and δ' is that δ maps both y_1 and y_2 to the same constant, while δ' does not contain a mapping for y_1; therefore, $\delta\sigma = \sigma'\delta'$. □

We now demonstrate the desired relationship between the sets $\mathsf{ans}(Q, \mathcal{K})$ and $\mathsf{ans}(\langle Q_P, \mathsf{ff}^{\text{lite}}(Q, \mathcal{T})\rangle, \mathcal{A})$. By Theorem 4.3.1, it follows that $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$. According to Lemma 4.4.1, we have that there is a clause $C' \in (\Xi(\mathcal{T}) \cup Q_C)_{\mathcal{R}^{\text{lite}}}$ of type 1, such that there is a derivation tree for $Q_P(\vec{a})$ from $\{C'\} \cup \mathcal{G}$. Since $Q_P(\vec{a})$ does not contain function symbols, C' does not contain function symbols either, so $C' \in \mathsf{ff}^{\text{lite}}(Q, \mathcal{T})$. Hence, by the definition of $\mathcal{G}$, it follows that there is a derivation tree for $Q_P(\vec{a})$ from $\mathsf{ff}^{\text{lite}}(Q, \mathcal{T}) \cup \mathcal{A}$. Lemma 4.4.2 follows immediately.

Lemma 4.4.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL-Lite$_R$ KB and $Q = \langle Q_P, Q_C\rangle$ a CQ. We have that $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}(\langle Q_P, \mathsf{ff}^{lite}(Q, \mathcal{T})\rangle, \mathcal{A})$.*

According to Lemma 4.4.2, the DQ $\langle Q_P, \mathsf{ff}^{\text{lite}}(Q, \mathcal{T})\rangle$ is a rewriting of Q with respect to $\mathcal{T}$. We note, however, that such a rewriting is not necessarily optimal for DL-Lite$_R$. We illustrate the point with an example.

Example 4.4.3. Consider a DL-Lite$_R$ TBox $\mathcal{T}$ consisting of the following axioms:

$$\exists \text{believesInDeity} \sqsubseteq \text{Theist}$$

$$\text{praysToDeity} \sqsubseteq \text{believesInDeity}$$

The TBox $\mathcal{T}$ states that those who believe in a deity are theists, and that anyone who prays to a deity believes in it.

According to Table 4.1, the set $\Xi(\mathcal{T})$ consists of the following clauses:

$$\text{Theist}(x) \leftarrow \underline{\text{believesInDeity}(x, y)}$$
$$\text{believesInDeity}(x, y) \leftarrow \underline{\text{praysToDeity}(x, y)}$$

Consider the query Q: $Q_P(x) \leftarrow \underline{\text{Theist}(x)}$. It is not difficult to verify that $\mathsf{ff}^{\text{lite}}(Q, \mathcal{T}) = \Xi(\mathcal{T}) \cup \{Q\}$. This is so because $\Xi(\mathcal{T}) \cup \{Q\}$ does not contain function symbols and it is already saturated by $\mathcal{R}^{\text{lite}}$. In the case of DL-Lite$_R$, rewritings can be expressed as UCQs [CDL+07]. In this case, however, $\langle Q_P, \mathsf{ff}^{\text{lite}}(Q, \mathcal{T})\rangle$ is a DQ; therefore, this rewriting is not optimal for DL-Lite$_R$. △

4.4.2 Optimizing the Rewriting

We now introduce a step that can be used to transform the DQ $\langle Q_P, \mathsf{ff}^{\text{lite}}(Q, \mathcal{T})\rangle$ into a UCQ. This is the second step of $\mathsf{RQR}^{\text{lite}}$ and it is based on the notion of *unfolding*. We formally define unfolding as follows.

Definition 4.4.2. Let C_1 be a clause of the form $D_1(\vec{r}) \leftarrow \bigwedge D_i(\vec{s_i})$ and let C_2 be a clause of the form $D_2(\vec{t}) \leftarrow D_1(\vec{u}) \wedge \bigwedge D_j(\vec{v_j})$. The unfolding of C_1 in C_2 is the clause $D_2(\vec{t})\sigma \leftarrow \bigwedge D_i(\vec{s_i})\sigma \wedge \bigwedge D_j(\vec{v_j})\sigma$, where $\sigma = \mathsf{MGU}(D_1(\vec{r}), D_1(\vec{u}))$. The clause C_1 is said to have been unfolded into C_2.

Let R and U be sets of safe Horn clauses. With R_U we denote the smallest set such that $R \subseteq R_U$, and for every unfolding C_r of a clause $C_1 \in R_U \cap U$ in a clause $C_2 \in R_U$, we have that there is a clause $C'_r \in R_U$ that is equivalent to C_r up to variable renaming. The unfolding of R with respect to U is defined as $\mathsf{unfold}(R, U) = R_U \setminus U$. △

We now show that unfolding clauses does not change the set of "relevant" consequences (ground facts) of a datalog program.

Lemma 4.4.3. *Let R and U be sets of Horn clauses. For any set of facts A and for any predicate D that does not occur in U, we have $R \cup A \models D(\vec{a})$ if and only if* $\mathsf{unfold}(R, U) \cup A \models D(\vec{a})$.

Proof. For the ($\Leftarrow$) direction, note that $R \models R_U$ and $\mathsf{unfold}(R, U) \subseteq R_U$; therefore, for each clause C, if $\mathsf{unfold}(R, U) \cup A \models C$ then $R \cup A \models C$.

For the ($\Rightarrow$) direction, let $\Sigma = \langle T, \delta, \lambda \rangle$ be the derivation tree for $D(\vec{a})$. We now inductively define a function $\sigma(t)$ as follows: starting from the leaves upwards, for each $t \in T$, we set $\sigma(t)$ to be the clause obtained from $\lambda(t)$ by unfolding each $\sigma(t.i)$ in the i-th body atom of $\lambda(t)$ provided that $\sigma(t.i) \in U$ or $\delta(t.i) \in A$; furthermore, we call t a *surviving* node if and only if $\sigma(t) \notin U$ or $\delta(t) \in A$. We say that a node t_2 is the *closest surviving node* to t_1 if t_2 is a surviving node, if it is a descendent of t_1, and no node on the path between t_1 and t_2 is a surviving node. By the inductive definition of σ, it can be seen that, for each node t, the fact $\delta(t)$ can be derived by resolving $\sigma(t)$ with the set of facts $\{\delta(t_1), \ldots, \delta(t_n)\}$ in one step, where $t_1, \ldots, t_n$ are exactly all the closest surviving nodes to t. Note that, for every node $t \in T$, we have $\sigma(t) \in R_U$. Moreover, if t is a surviving node, then $\sigma(t) \in \mathsf{unfold}(R, U)$. Therefore, if t is a surviving node, the fact $\delta(t)$ can be derived from $\mathsf{unfold}(R, U) \cup A$. Since the predicate D does not occur in U, we have $D(\vec{a}) \notin U$. Furthermore, $\delta(\epsilon) = D(\vec{a})$, so the clause $\sigma(\epsilon)$ contains D in the head, and $\sigma(\epsilon) \notin U$. Thus, ϵ is a surviving node, so $\delta(\epsilon)$ can be derived from $\mathsf{unfold}(R, U) \cup A$. □

In order to obtain a rewriting of Q with respect to $\mathcal{T}$, the idea is to unfold certain types of clauses in $\mathsf{ff}^{\text{lite}}(Q, \mathcal{T})$. We are ready to define the rewriting in terms of $\mathsf{ff}^{\text{lite}}(Q, \mathcal{T})$.

Definition 4.4.3. Let $Q = \langle Q_P, Q_C \rangle$ be a CQ and $\mathcal{T}$ a DL-Lite$_R$ TBox. We define $\mathsf{RQR}^{\text{lite}}(Q, \mathcal{T}) = \langle Q_P, \mathsf{unfold}(\mathsf{ff}^{\text{lite}}(Q, \mathcal{T}), U) \rangle$, where U is the set that contains every

clause $C \in \mathcal{N}^{\text{lite}}$ of one of the following forms:

$$A(x) \leftarrow B(x) \tag{4.32}$$

$$A(x) \leftarrow R(x, y) \tag{4.33}$$

$$A(x) \leftarrow R(y, x) \tag{4.34}$$

$$R(x, y) \leftarrow S(x, y) \tag{4.35}$$

$$R(x, y) \leftarrow S(y, x) \tag{4.36}$$

△

Example 4.4.4. Consider the DL-Lite$_R$ TBox $\mathcal{T}$ and the query Q introduced in Example 4.4.3. It can be readily verified that $\mathsf{RQR}^{\text{lite}}(Q, \mathcal{T})$ is a UCQ consisting of the following queries:

$$Q_P(x) \leftarrow \text{Theist}(x)$$

$$Q_P(x) \leftarrow \text{believesInDeity}(x, y)$$

$$Q_P(x) \leftarrow \text{praysToDeity}(x, y)$$

△

Theorem 4.4.1 follows from Lemma 4.4.2 and Lemma 4.4.3, since without loss of generality we can assume that Q_P does not occur in any of the clauses that are to be unfolded to obtain $\mathsf{RQR}^{\text{lite}}(Q, \mathcal{T})$.

Theorem 4.4.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_R$ KB and Q a CQ. We have that $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}(\mathsf{RQR}^{lite}(Q, \mathcal{T}), \mathcal{A})$.*

4.4.3 Form of the Rewriting

We now consider the form of the rewriting; we show that $\mathsf{RQR}^{\text{lite}}(Q, \mathcal{T})$ is a UCQ.

Lemma 4.4.4. *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox and $Q = \langle Q_P, Q_C \rangle$ a CQ. We have that $\mathsf{RQR}^{lite}(Q, \mathcal{T}) = \langle Q_P, Q'_C \rangle$ is a UCQ.*

Input: Conjunctive query Q, DL-Lite$_R$ TBox $\mathcal{T}$
$\mathsf{CGLLR}(Q, \mathcal{T}) = \{Q\}$;
repeat
  foreach *query* $Q' \in \mathsf{CGLLR}(Q, \mathcal{T})$ **do**
    (reformulation) **foreach** *atom* D *in* Q' **do**
      foreach *axiom* $\alpha \in \mathcal{T}$ **do**
        if α *is applicable to* D **then**
          $\mathsf{CGLLR}(Q, \mathcal{T}) = \mathsf{CGLLR}(Q, \mathcal{T}) \cup \{Q'[D/\mathsf{ref}(D, \alpha)]\}$;
        end
      end
    end
    (reduction) **forall** *atoms* D_1, D_2 *in* Q' **do**
      if D_1 *and* D_2 *unify* **then**
        $\sigma = \mathsf{MGU}(D_1, D_2)$;
        $\mathsf{CGLLR}(Q, \mathcal{T}) = \mathsf{CGLLR}(Q, \mathcal{T}) \cup \{\lambda(Q'\sigma))\}$;
      end
    end
  end
until *no query unique up to variable renaming can be added to* $\mathsf{CGLLR}(Q, \mathcal{T})$;
return $\mathsf{CGLLR}(Q, \mathcal{T})$;

Algorithm 1: The CGLLR algorithm

Proof. By analyzing the inferences shown in Table 4.3, one can see that saturating a set of clauses of types 1, 2, or 3.1–3.4 by $\mathcal{R}^{\text{lite}}$ produces only clauses of the same types. Therefore, $\mathsf{ff}^{\text{lite}}(Q, \mathcal{T})$ contains only clauses of type 1 or of the form (4.32)–(4.36), which implies that the datalog program Q'_C only contains clauses of type 1 given the definition of unfolding. □

4.5 The CGLLR Algorithm

In this section we briefly describe CGLLR—the CQ rewriting algorithm for DL-Lite$_R$ developed by Calvanese et al [CDL+07]. As mentioned at the beginning of this chapter, we present an empirical evaluation of RQR$^{\text{lite}}$ and CGLLR in Chapter 8 in order to analyze the type and size of the rewritings they produce. In order to better understand the results of this evaluation, in this section we also discuss the main differences between the algorithms.

The algorithm computes a rewriting $\mathsf{CGLLR}(Q, \mathcal{T})$ of Q with respect to $\mathcal{T}$ by using the axioms of $\mathcal{T}$ as rewriting rules and applying them to the body atoms of Q. The algorithm is shown in Algorithm 1. The partial function ref takes as input an axiom α and an atom D, and returns an atom $\mathsf{ref}(D, \alpha)$ as follows:

- If $D = A(x)$, then we have that (i) if $\alpha = B \sqsubseteq A$, then $\mathsf{ref}(D, \alpha) = B(x)$; (ii) if $\alpha = \exists P \sqsubseteq A$, then $\mathsf{ref}(D, \alpha) = P(x, \_)$; and (iii) if $\alpha = \exists P^- \sqsubseteq A$, then we have that $\mathsf{ref}(D, \alpha) = P(\_, x)$.

- If $D = P(x, \_)$, then we have that (i) if $\alpha = A \sqsubseteq \exists P$, then $\mathsf{ref}(D, \alpha) = A(x)$; (ii) if $\alpha = \exists S \sqsubseteq \exists P$, then $\mathsf{ref}(D, \alpha) = S(x, \_)$; and (iii) if $\alpha = \exists S^- \sqsubseteq \exists P$, then $\mathsf{ref}(D, \alpha) = S(\_, x)$.

- If $D = P(\_, x)$, then we have that (i) if $\alpha = A \sqsubseteq \exists P^-$, then $\mathsf{ref}(D, \alpha) = A(x)$; (ii) if $\alpha = \exists S \sqsubseteq \exists P^-$, then $\mathsf{ref}(D, \alpha) = S(x, \_)$; and (iii) if $\alpha = \exists S^- \sqsubseteq \exists P^-$, then $\mathsf{ref}(D, \alpha) = S(\_, x)$.

- If $D = P(x, y)$, then we have that (i) if either $\alpha = S \sqsubseteq P$ or $\alpha = S^- \sqsubseteq P^-$, then $\mathsf{ref}(D, \alpha) = S(x, y)$; and (ii) if either $\alpha = S \sqsubseteq P^-$ or $\alpha = S^- \sqsubseteq P$, then $\mathsf{ref}(D, \alpha) = S(y, x)$.

If $\mathsf{ref}(D, \alpha)$ is defined for α and D we say that α is *applicable* to D. The expression $Q[D/D']$ denotes the CQ obtained from Q by replacing the body atom D with a new atom D'. The function λ takes as input a CQ Q and returns a new CQ obtained by replacing each variable that occurs only once in Q with the symbol '_'.

Starting with the original query Q, CGLLR continues to produce queries through its two steps until no new query can be produced. In the *reformulation* step the algorithm rewrites the body atoms of a given query Q' by using applicable TBox axioms as rewriting rules, generating a new query for every atom reformulation. Then, in the *reduction* step the algorithm produces a new query $\lambda(Q'\sigma)$ for each pair of body atoms of Q' that unify.

CGLLR and RQR$^{\text{lite}}$ mainly differ in the way they handle existential quantification. This difference is twofold: first, while RQR$^{\text{lite}}$ deals with axioms containing existential quantifiers on the right-hand side by introducing function symbols, CGLLR does so by restricting the applicability of such axioms and relying on the reduction step; second, unlike RQR$^{\text{lite}}$, CGLLR does not handle qualified existential axioms natively—that is, there is no rewriting rule for axioms of the form $A \sqsubseteq \exists R.B$; instead, the algorithm employs a preprocessing step in which each such axiom occurring in $\mathcal{T}$ is replaced with a set of axioms $\{A \sqsubseteq \exists P_1, \exists P_1^- \sqsubseteq B, P_1 \sqsubseteq R\}$, where P_1 is a new atomic role not occurring in $\mathcal{T}$. We explore these differences and their impact on the size of the rewritings by means of an example.

Example 4.5.1. Consider a DL-Lite$_R$ TBox $\mathcal{T}$ that consists of the axiom

$$\text{Catholic} \sqsubseteq \exists \text{hasReligion.ChristianReligion}, \tag{4.37}$$

which states that all catholics have the religion ChristianReligion, and the query Q

$$Q_P(x) \leftarrow \underline{\text{hasReligion}(x, y)} \wedge \underline{\text{ChristianReligion}(y)}. \tag{4.38}$$

We first analyze the execution of CGLLR. First, as discussed previously, CGLLR cannot handle axiom (4.37) natively, so the axiom must be preprocessed into the following axioms:

$$\text{Catholic} \sqsubseteq \exists R_{aux} \tag{4.39}$$

$$\exists R_{aux}^- \sqsubseteq \text{ChristianReligion} \tag{4.40}$$

$$R_{aux} \sqsubseteq \text{hasReligion} \tag{4.41}$$

In the first iteration, axiom (4.41) is applicable to the atom hasReligion(x, y) in (4.38). Similarly, axiom (4.40) is applicable to ChristianReligion(y) in (4.38). Therefore, we obtain the following queries in the reformulation step:

$$Q_P(x) \leftarrow R_{aux}(x, y) \wedge \text{ChristianReligion}(y) \tag{4.42}$$

$$Q_P(x) \leftarrow \text{hasReligion}(x, y) \wedge R_{aux}(\_\,, y) \tag{4.43}$$

In this iteration no query can be obtained in the reduction step. In the next iteration, axiom (4.39) is not applicable to the atom $R_{aux}(x, y)$ in (4.42) because y occurs in (4.42) in more than one place. Axiom (4.39) is not applicable to $R_{aux}(x, y)$ because CGLLR does not keep track of information about role successors; in fact, if we naively allowed existential quantification axioms to be applied, the resulting algorithm would become unsound. To illustrate this point, suppose that (4.39) were applicable to $R_{aux}(x, y)$ in (4.42) and $\mathsf{ref}(R_{aux}(x, y), (4.39)) = \text{Catholic}(x)$; we would then obtain the query

$$Q_P(x) \leftarrow \text{Catholic}(x) \wedge \text{ChristianReligion}(y). \tag{4.44}$$

Note that the relation between x and y is lost—that is, the fact that the individual represented by y must be a hasReligion-successor of the individual represented by x is not captured by query (4.44).

Although the applicability of (4.39) is restricted, axiom (4.40) is applicable to ChristianReligion(y) in (4.42). Similarly, axiom (4.41) is applicable to hasReligion(x, y) in (4.43). Both reformulations produce the query

$$Q_P(x) \leftarrow R_{aux}(x, y) \wedge R_{aux}(\_\,, y). \tag{4.45}$$

In the next iteration, no axiom is applicable to any body atom of (4.45) so no query is added in the reformulation step. In the reduction step, however, the algorithm produces

$$Q_P(x) \leftarrow R_{aux}(x, \_\,) \tag{4.46}$$

by unifying the body atoms of (4.45). In the following iteration, axiom (4.39) is applicable to the only body atom of (4.46), producing

$$Q_P(x) \leftarrow \text{Catholic}(x). \tag{4.47}$$

Note that without the reduction step, the algorithm would not have produced query (4.47). It can be easily verified that no more new queries can be produced; thus, CGLLR returns $\{(4.38), (4.42), (4.43), (4.45), (4.46), (4.47)\}$.

We now analyze the execution of $\mathsf{RQR}^{\mathrm{lite}}$. According to Table 4.1, axiom (4.37) is translated into the following clauses:

$$\underline{\mathrm{hasReligion}(x, f(x))} \leftarrow \mathrm{Catholic}(x) \tag{4.48}$$

$$\underline{\mathrm{ChristianReligion}(f(x))} \leftarrow \mathrm{Catholic}(x) \tag{4.49}$$

The saturation step $\mathsf{RQR}^{\mathrm{lite}}$ produces

$$[(4.38) + (4.48)] \quad Q_P(x) \leftarrow \mathrm{Catholic}(x) \wedge \underline{\mathrm{ChristianReligion}(f(x))} \tag{4.50}$$

$$[(4.38) + (4.49)] \quad Q_P(x) \leftarrow \underline{\mathrm{hasReligion}(x, f(x))} \wedge \mathrm{Catholic}(x) \tag{4.51}$$

$$[(4.38) + (4.51)] \quad Q_P(x) \leftarrow \underline{\mathrm{Catholic}(x)} \tag{4.52}$$

Note the difference between queries (4.44) and (4.50). Since the function symbol f is uniquely associated with axiom (4.37), unlike query (4.44), query (4.50) captures the fact that the individual represented by $f(x)$ must be a hasReligion-successor of the individual represented by x. It can easily be verified that no other clause is produced in the first step. Clearly, $\mathsf{ff}^{\mathrm{lite}}(Q, \mathcal{T}) = \{(4.38), (4.52)\}$. In this case, there is no unfolding to be done, so $\mathsf{RQR}^{\mathrm{lite}}$ returns $\{(4.38), (4.52)\}$. △

As shown in the above example, the lack of a native handling of qualified existential quantification by CGLLR can lead to an increase in the size of the rewritings (because of the auxiliary symbols introduced). The reduction step alone, however, can also lead to larger rewritings. This situation arises especially in the common case where part of the data of the ABox describes a graph. Consider the following example.

Example 4.5.2. Consider a DL-Lite$_R$ TBox $\mathcal{T}$ consisting of the axiom

$$\mathrm{Religion} \sqsubseteq \exists \mathrm{hasDeity}, \tag{4.53}$$

which states that a religion has at least one deity, and the query

$$Q_P(x) \leftarrow \text{hasDeity}(x, y) \wedge \text{hasDeity}(z, y) \wedge \text{hasDeity}(z, w) \wedge \text{hasDeity}(x, w). \quad (4.54)$$

When using CGLLR, axiom (4.53) is not applicable to any body atom of query (4.54) so no query is produced in the reformulation step. However, every pair of body atoms in query (4.54) unify, and it is easy to see that for each query of this form with m body atoms, CGLLR produces $\binom{m}{2}$ new queries in the reduction step. Eventually, the reduction step produces the query

$$Q_P(x) \leftarrow \text{hasDeity}(x, \_). \quad (4.55)$$

Axiom (4.53) is now applicable to the only body atom of query (4.55) and the following query is produced in the reformulation step:

$$Q_P(x) \leftarrow \text{Religion}(x) \quad (4.56)$$

Note, however, that several queries needed to be produced in the reduction step in order to produce query (4.56) in the reformulation step. △

Note that for every query Q' produced from a query Q in the reduction step, there is a substitution σ such that $Q\sigma \subseteq Q'$, in which case we say that Q *subsumes* Q'. It is well known that every query that is subsumed by another can be discarded after the rewriting has been computed without affecting completeness [CL97]; therefore, all such clauses can be safely removed a posteriori. In our example, query (4.54) subsumes query (4.55) by the substitution $\sigma = \{z \mapsto x, w \mapsto y\}$; therefore, query (4.55) can be safely discarded from the final rewriting. In this case, however, note that query (4.55) subsumes query (4.54) as well; therefore, it is sensible to eliminate query (4.54) instead since the latter query is larger. Since both queries subsume each other, we say that they are *equivalent*. Moreover, since query (4.55) is the minimal equivalent subquery of query (4.54), we say that query (4.55) is a *condensation* of query

(4.54) [BG01]. The potential generation of condensations by the reduction step is a distinctive characteristic of CGLLR.

Summing up, the use of function symbols makes $\mathsf{RQR}^{\text{lite}}$ more goal-oriented, in the sense that it does not need to derive the "irrelevant" (subsumed) queries produced by the reduction step of CGLLR in order to be complete. Moreover, $\mathsf{RQR}^{\text{lite}}$ handles qualified existential axioms natively, whereas CGLLR needs to encode them away by introducing new roles and axioms.

4.6 Chapter Summary

In this chapter we presented $\mathsf{RQR}^{\text{lite}}$—a simpler version of RQR scaled down to the basic DL DL-Lite$_R$. In Section 4.3 we presented a resolution-based CQ answering algorithm for DL-Lite$_R$. The algorithm takes as input a CQ Q and a DL-Lite$_R$ KB $\mathcal{K}$ and computes the set of certain answers $\mathsf{ans}(Q, \mathcal{K})$. The algorithm produces such answers by saturating the set of clauses corresponding to Q and $\mathcal{K}$ using the suitably parameterized RFS calculus $\mathcal{R}^{\text{lite}}$.

In Section 4.4 we modified the answering algorithm in order to derive the CQ rewriting algorithm $\mathsf{RQR}^{\text{lite}}$. The algorithm takes as input a CQ Q and a DL-Lite$_R$ TBox $\mathcal{T}$ and computes a UCQ $\mathsf{RQR}^{\text{lite}}(Q, \mathcal{T})$ that is a rewriting of Q with respect to $\mathcal{T}$. The algorithm produces such a rewriting by saturating the set of clauses corresponding to Q and $\mathcal{T}$ using $\mathcal{R}^{\text{lite}}$ and then transforming the resulting set into a UCQ by pruning and unfolding clauses of certain types.

To conclude the chapter, in Section 4.5 we presented CGLLR—the CQ rewriting algorithm for DL-Lite$_R$ developed by Calvanese et al. $\mathsf{RQR}^{\text{lite}}$ and CGLLR mainly differ in the way they handle existential quantification. This difference is twofold: first, while $\mathsf{RQR}^{\text{lite}}$ deals with axioms containing existential quantifiers on the right-hand side by introducing function symbols, CGLLR does so by restricting the applicability of such axioms and relying on the reduction step; second, unlike $\mathsf{RQR}^{\text{lite}}$, CGLLR does

not handle qualified existential axioms natively; instead, the algorithm requires a preliminary step in which each such axiom occurring in $\mathcal{T}$ is replaced with a set of auxiliary axioms.

Chapter 5

Answering Queries for $\mathcal{ELHIO}^{\neg}$

In this chapter we consider the problems of CQ answering and CQ rewriting for $\mathcal{ELHIO}^{\neg}$—an expressive extension of the basic DL $\mathcal{EL}$ [BBL05]. As we shall show in Section 5.3, $\mathcal{ELHIO}^{\neg}$ is one of the most expressive Horn DLs for which CQ answering is polynomial with respect to data complexity.

In Section 5.1, we apply Joyner's principles and the answering literal technique described in Chapter 4 to derive a CQ answering algorithm. As we shall see, a major difficulty of solving CQ answering for $\mathcal{ELHIO}^{\neg}$ is that the existence of nominals necessitates reasoning with equality. It is well known that reasoning with equality using a resolution-based calculus can lead to nontermination. We address this problem with a novel way of reasoning with equality, which makes our answering algorithm interesting in its own right.

Then, in Section 5.2, we show how to modify the answering algorithm to derive RQR—our CQ rewriting algorithm for $\mathcal{ELHIO}^{\neg}$. RQR exhibits "pay-as-you-go" behavior: if the input $\mathcal{T}$ is expressible in $\mathcal{ELHIO}^{\neg}$, then the computed rewriting $\mathsf{RQR}(Q, \mathcal{T})$ is a datalog query; if $\mathcal{T}$ is expressible in DL-Lite^{+}, then $\mathsf{RQR}(Q, \mathcal{T})$ consists of a UCQ and a linear datalog program; finally, if $\mathcal{T}$ is expressible in DL-Lite$_R$, then $\mathsf{RQR}(Q, \mathcal{T})$ is a UCQ. Therefore, as we shall show in Section 5.3, RQR not only handles a variety of DLs ranging from DL-Lite$_{core}$ up to $\mathcal{ELHIO}^{\neg}$, but it is worst-case optimal with respect to data complexity for all these DLs.

Table 5.1: Translating an $\mathcal{ELHIO}$ KB $\mathcal{K}$ into a set of clauses $\Xi(\mathcal{K})$

$\mathcal{ELHIO}$ clause	$\mathcal{ELHIO}$ axiom
$A(a)$	$A(a)$ $\{a\} \sqsubseteq A$
$P(a, b)$	$P(a, b)$
$x \approx a \leftarrow A(x)$	$A \sqsubseteq \{a\}$
$A_2(x) \leftarrow A_1(x)$	$A_1 \sqsubseteq A_2$
$A_3(x) \leftarrow A_1(x) \wedge A_2(x)$	$A_1 \sqcap A_2 \sqsubseteq A_3$
$P(x, f(x)) \leftarrow A(x)$	$A \sqsubseteq \exists P$
$P(x, f(x)) \leftarrow A_1(x)$ $A_2(f(x)) \leftarrow A_1(x)$	$A_1 \sqsubseteq \exists P.A_2$
$P(f(x), x) \leftarrow A(x)$	$A \sqsubseteq \exists P^-$
$P(f(x), x) \leftarrow A_1(x)$ $A_2(f(x)) \leftarrow A_1(x)$	$A_1 \sqsubseteq \exists P^-.A_2$
$A(x) \leftarrow P(x, y)$	$\exists P \sqsubseteq A$
$A_2(x) \leftarrow P(x, y) \wedge A_1(y)$	$\exists P.A_1 \sqsubseteq A_2$
$A(x) \leftarrow P(y, x)$	$\exists P^- \sqsubseteq A$
$A_2(x) \leftarrow P(y, x) \wedge A_1(y)$	$\exists P^-.A_1 \sqsubseteq A_2$
$S(x, y) \leftarrow P(x, y)$	$P \sqsubseteq S, P^- \sqsubseteq S^-$
$S(x, y) \leftarrow P(y, x)$	$P^- \sqsubseteq S, P \sqsubseteq S^-$

Note 5.1.1. Each axiom of the form $A \sqsubseteq \exists R$ or $A_1 \sqsubseteq \exists R.A_2$ is uniquely associated with a distinct function symbol f.

5.1 CQ Answering

Our objective is to compute the set of certain answers $\mathsf{ans}(Q, \mathcal{K})$ for a given CQ Q and an $\mathcal{ELHIO}^{\neg}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. We develop our answering algorithm under the assumption that $\mathcal{K}$ is satisfiable. In the same vein as described in Chapter 4, it can be shown that in case $\mathcal{K}$ is satisfiable, negative inclusions are not needed for CQ answering; therefore, in the rest of the chapter we assume that $\mathcal{T}$ does not contain negative inclusions—that is, $\mathcal{T}$ is an $\mathcal{ELHIO}$ TBox.

By the definition of certain answers, it follows that $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if

the logic program $\Xi(\mathcal{K}) \cup Q_C \cup \{\bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable, where $\Xi(\mathcal{K})$ is the set of clauses that corresponds to the clausal normal form [Fit96] of the FOL formula $\pi(\mathcal{K})$ (see Definition 3.3.3), and $\bot$ is the logical constant False. It is easy to see that $\Xi(\cdot)$ corresponds to the transformation shown in Table 5.1.

As can be seen, the resulting set of clauses might contain the equality predicate $\approx$. For example, the $\mathcal{ELHIO}$ axiom $\mathsf{Pope} \sqsubseteq \{\mathsf{BenedictXVI}\}$ is transformed into a clause $x \approx \mathsf{BenedictXVI} \leftarrow \mathrm{Pope}(x)$ that intuitively says that if an individual a is an instance of Pope, then a is equal to BenedictXVI. Therefore, to check whether $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ by resolution we need a calculus capable of dealing with equality.

Resolution-based calculi such as paramodulation and superposition [BG01] have been specially designed to improve efficiency of reasoning with equality. We base our work, however, on Resolution with Free Selection, mainly because being able to design an ad-hoc selection function is key to attaining worst-case optimality for RQR. Equality can be handled in an RFS calculus by treating the equality predicate $\approx$ as an ordinary predicate and axiomatizing its properties with a set of clauses $E_{\mathcal{T}}$ such that $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E_{\mathcal{T}})$. We formally define $E_{\mathcal{T}}$ as follows.

Definition 5.1.1. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB. Let $E_{\mathcal{T}}$ be the set containing exactly the following clauses, where (i) one functional monotonicity clause is instantiated for every functional symbol f occurring in $\Xi(\mathcal{T})$, (ii) one unary substitutivity clause is instantiated for every unary predicate A occurring in $\Xi(\mathcal{T})$, and (iii) one binary substitutivity 1 clause and one binary substitutivity 2 clause are instantiated for every binary predicate P occurring in $\Xi(\mathcal{T})$:[1]

$$x \approx x \qquad \text{(reflexivity)}$$

$$x \approx y \leftarrow y \approx x \qquad \text{(symmetry)}$$

$$x \approx z \leftarrow x \approx y \wedge y \approx z \qquad \text{(transitivity)}$$

[1]Without loss of generality we assume that all the symbols in $\mathcal{A}$ occur in $\mathcal{T}$.

$$f(x) \approx f(y) \leftarrow x \approx y \qquad \text{(functional monotonicity)}$$

$$A(y) \leftarrow A(x) \wedge x \approx y \qquad \text{(unary substitutivity)}$$

$$P(x,z) \leftarrow P(x,y) \wedge y \approx z \qquad \text{(binary substitutivity 1)}$$

$$P(z,x) \leftarrow P(y,x) \wedge y \approx z \qquad \text{(binary substitutivity 2)}$$

△

It is well known, however, that saturating a set of clauses containing $E_{\mathcal{T}}$ often causes the generation of a large (or even infinite) number of clauses [BG01]. In order to avoid this problem, we develop an *approximation of equality* $E'_{\mathcal{T}}$. Our approximation of equality $E'_{\mathcal{T}}$ "weakens" the axiomatization of equality $E_{\mathcal{T}}$ in three ways: first, the relation $\approx$ is only required to be (implicitly) symmetric; second, the substitutivity clauses are limited to be applicable to certain constants only ($\mathcal{O}$-constants); and third, functional monotonicity is dropped. We formally define $E'_{\mathcal{T}}$ as follows.

Definition 5.1.2. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB. A constant o occurring in $\Xi(\mathcal{K})$ is called an $\mathcal{O}$-constant if it appears in a clause $C \in \Xi(\mathcal{K})$ of the form $x \approx o \leftarrow A(x)$.

We define the set $E'_{\mathcal{T}}$ as follows. Let $\mathcal{O}$ be a fresh predicate not occurring in $\Xi(\mathcal{T})$. If there is no $\mathcal{O}$-constant o occurring in $\Xi(\mathcal{K})$, then $E'_{\mathcal{T}} = \emptyset$; otherwise let $E'_{\mathcal{T}}$ be the set that contains the following clauses, where (i) one $\mathcal{O}$-constant identification clause is instantiated for every $\mathcal{O}$-constant o occurring in $\Xi(\mathcal{T})$, (ii) one unary substitutivity$_{\mathcal{O}}$ clause is instantiated for every unary predicate A occurring in $\Xi(\mathcal{T})$, and (iii) one binary substitutivity$_{\mathcal{O}}$ 1 clause and one binary substitutivity$_{\mathcal{O}}$ 2 clause are instantiated for every binary predicate P occurring in $\Xi(\mathcal{T})$:

$$\mathcal{O}(o) \qquad (\mathcal{O}\text{-constant identification})$$

$$A(y) \leftarrow A(x) \wedge x \approx y \wedge \mathcal{O}(y) \qquad (\text{unary substitutivity}_{\mathcal{O}})$$

$$P(x,z) \leftarrow P(x,y) \wedge y \approx z \wedge \mathcal{O}(z) \qquad (\text{binary substitutivity}_{\mathcal{O}}\ 1)$$

$$P(z,x) \leftarrow P(y,x) \wedge y \approx z \wedge \mathcal{O}(z) \qquad (\text{binary substitutivity}_{\mathcal{O}}\ 2)$$

△

In the next section, we present an algorithm for computing the set of approximate answers $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ using an RFS calculus. As we shall see, $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ is a subset of $\mathsf{ans}(Q, \mathcal{K})$ and the latter set can be obtained from the former.

5.1.1 Computing Approximate Answers Using Resolution

Our algorithm for computing $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ amounts to saturating the logic program $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C$ by $\mathcal{R}^{DL}$—the RFS calculus defined in Definition 5.1.4 (given later)—and returning each tuple $\vec{a}$ such that $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$. Before showing the correctness of our answering algorithm, we present an example in which we give informal intuitive explanations.

Example 5.1.1. Consider an $\mathcal{ELHIO}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ about monotheistic religions. Let $\mathcal{T}$ contain the following axioms:

$$\begin{aligned} \mathsf{Religion} &\sqsubseteq \exists \mathsf{hasDevotee} \\ \exists \mathsf{hasDevotee}^{-}.\mathsf{Religion} &\sqsubseteq \mathsf{Theist} \\ \mathsf{Theist} &\sqsubseteq \exists \mathsf{believesIn}.\{\mathsf{God}\} \\ \{\mathsf{FSM}\} &\sqsubseteq \{\mathsf{God}\} \end{aligned}$$

The TBox $\mathcal{T}$ states that a religion has at least one devotee, that someone who is a devotee of a religion is a theist, that a theist is someone who believes in God, and that FSM and God are the same individual.

Let $\mathcal{A}$ contain the following assertions:[2]

$$\underline{\mathsf{Religion}(\mathsf{Pastafarism})} \tag{5.1}$$

$$\underline{\mathsf{hasDeity}(\mathsf{Pastafarism}, \mathsf{FSM})} \tag{5.2}$$

$$\underline{\mathsf{Mighty}(\mathsf{FSM})} \tag{5.3}$$

[2]We underline the atoms selected by the selection function of $\mathcal{R}^{DL}$.

Finally, consider the following query Q:

$$Q_P(z) \leftarrow \underline{\text{hasDevotee}(x, y)} \wedge \underline{\text{believesIn}(y, z)} \wedge \underline{\text{hasDeity}(x, z)} \wedge \underline{\text{Mighty}(z)} \tag{5.4}$$

Note that Pastafarism is a religion, so it has at least one devotee who believes in God. FSM is mighty, and it is the deity of Pastafarism. Therefore, since FSM and God denote the same individual, we expect $\{\mathsf{God}, \mathsf{FSM}\} \subseteq \mathsf{ans}(Q, \mathcal{K})$.

We now show how to compute the set $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. We start by translating $\mathcal{T}$ into clauses and computing $E'_{\mathcal{T}}$. According to Table 5.1, the set $\Xi(\mathcal{T})$ contains the following clauses:[3]

$$\underline{\text{hasDevotee}(x, f_1(x))} \leftarrow \text{Religion}(x) \tag{5.5}$$

$$\text{Theist}(x) \leftarrow \underline{\text{hasDevotee}(y, x)} \wedge \text{Religion}(y) \tag{5.6}$$

$$\underline{\text{believesIn}(x, f_2(x))} \leftarrow \text{Theist}(x) \tag{5.7}$$

$$\underline{A_1(f_2(x))} \leftarrow \text{Theist}(x) \tag{5.8}$$

$$\underline{x \approx \mathsf{God}} \leftarrow A_1(x) \tag{5.9}$$

$$\underline{x \approx \mathsf{God}} \leftarrow A_2(x) \tag{5.10}$$

$$\underline{A_2(\mathsf{FSM})} \tag{5.11}$$

According to Definition 5.1.2, the approximation of equality $E'_{\mathcal{T}}$ for $\mathcal{T}$ contains the following clauses:

$$\underline{\mathcal{O}(\mathsf{God})} \tag{5.12}$$

$$\text{Mighty}(y) \leftarrow \text{Mighty}(x) \wedge x \approx y \wedge \underline{\mathcal{O}(y)} \tag{5.13}$$

$$\text{hasDeity}(x, z) \leftarrow \text{hasDeity}(x, y) \wedge y \approx z \wedge \underline{\mathcal{O}(z)} \tag{5.14}$$

$$\text{believesIn}(x, z) \leftarrow \text{believesIn}(x, y) \wedge y \approx z \wedge \underline{\mathcal{O}(z)} \tag{5.15}$$

Intuitively, clause (5.12) identifies God as an $\mathcal{O}$-*constant:* a constant that occurs in a clause of the form $x \approx o \leftarrow A(x)$ (cf. clauses (5.9) and (5.10)). Clauses (5.13)–(5.15)

[3] We show only the clauses that are relevant for the derivation of the answers to Q. We introduce A_1 and A_2 in order to normalize $\mathcal{T}$ (see Section 3.3).

intuitively say that, if an individual is equal to an $\mathcal{O}$-constant, then it can be replaced with that $\mathcal{O}$-constant. Restricting substitutivity to $\mathcal{O}$-constants reduces the number of clauses generated during the saturation.

After the logic program $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup \{Q\}$ is computed, we saturate it using our RFS calculus $\mathcal{R}^{DL}$. Note that this calculus is parameterized in a particular way, which determines the selected atoms in each clause and, consequently, the inferred clauses. The saturation $(\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup \{Q\})_{\mathcal{R}^{DL}}$ contains the following clauses, where we denote God with G and Pastafarism with P:[4]

$$[(5.12)+(5.14)] \quad \mathrm{hasDeity}(x,\mathsf{G}) \leftarrow \mathrm{hasDeity}(x,y) \wedge \underline{y \approx \mathsf{G}} \tag{5.16}$$

$$[(5.16)+(5.10)] \quad \mathrm{hasDeity}(x,\mathsf{G}) \leftarrow \underline{\mathrm{hasDeity}(x,y)} \wedge A_2(y) \tag{5.17}$$

$$[(5.17)+(5.2)] \quad \mathrm{hasDeity}(\mathsf{P},\mathsf{G}) \leftarrow \underline{A_2(\mathsf{FSM})} \tag{5.18}$$

$$[(5.18)+(5.11)] \quad \underline{\mathrm{hasDeity}(\mathsf{P},\mathsf{G})} \tag{5.19}$$

$$[(5.12)+(5.13)] \quad \mathrm{Mighty}(\mathsf{G}) \leftarrow \mathrm{Mighty}(x) \wedge \underline{x \approx \mathsf{G}} \tag{5.20}$$

$$[(5.20)+(5.10)] \quad \mathrm{Mighty}(\mathsf{G}) \leftarrow \underline{\mathrm{Mighty}(x)} \wedge \underline{A_2(x)} \tag{5.21}$$

$$[(5.21)+(5.3)] \quad \mathrm{Mighty}(\mathsf{G}) \leftarrow \underline{A_2(\mathsf{FSM})} \tag{5.22}$$

$$[(5.22)+(5.11)] \quad \underline{\mathrm{Mighty}(\mathsf{G})} \tag{5.23}$$

$$[(5.12)+(5.15)] \quad \mathrm{believesIn}(x,\mathsf{G}) \leftarrow \mathrm{believesIn}(x,y) \wedge \underline{y \approx \mathsf{G}} \tag{5.24}$$

$$[(5.24)+(5.9)] \quad \mathrm{believesIn}(x,\mathsf{G}) \leftarrow \underline{\mathrm{believesIn}(x,y)} \wedge A_1(y) \tag{5.25}$$

$$[(5.25)+(5.7)] \quad \mathrm{believesIn}(x,\mathsf{G}) \leftarrow \mathrm{Theist}(x) \wedge \underline{A_1(f_2(x))} \tag{5.26}$$

$$[(5.26)+(5.8)] \quad \mathrm{believesIn}(x,\mathsf{G}) \leftarrow \underline{\mathrm{Theist}(x)} \tag{5.27}$$

$$[(5.5)+(5.6)] \quad \underline{\mathrm{Theist}(f_1(x))} \leftarrow \mathrm{Religion}(x) \tag{5.28}$$

$$[(5.28)+(5.27)] \quad \underline{\mathrm{believesIn}(f_1(x),\mathsf{G})} \leftarrow \mathrm{Religion}(x) \tag{5.29}$$

$$[(5.4)+(5.19)] \quad Q_P(\mathsf{G}) \leftarrow \underline{\mathrm{hasDevotee}(\mathsf{P},y)} \wedge \underline{\mathrm{believesIn}(y,\mathsf{G})} \wedge \underline{\mathrm{Mighty}(\mathsf{G})} \tag{5.30}$$

$$[(5.30)+(5.23)] \quad Q_P(\mathsf{G}) \leftarrow \underline{\mathrm{hasDevotee}(\mathsf{P},y)} \wedge \underline{\mathrm{believesIn}(y,\mathsf{G})} \tag{5.31}$$

[4]The expression $[(x)+(y)]$ next to each clause states that the corresponding clause was obtained by resolving clauses (x) and (y).

$$[(5.31)+(5.29)] \quad Q_P(\mathsf{G}) \leftarrow \underline{\text{hasDevotee}(\mathsf{P}, f_1(x))} \wedge \text{Religion}(x) \tag{5.32}$$

$$[(5.32)+(5.5)] \quad Q_P(\mathsf{G}) \leftarrow \underline{\text{Religion}(\mathsf{P})} \tag{5.33}$$

$$[(5.33)+(5.1)] \quad \underline{Q_P(\mathsf{G})} \tag{5.34}$$

It can be shown that (5.34) is the only clause of the form $Q_P(\vec{a})$ contained in $(\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup \{Q\})_{\mathcal{R}^{DL}}$; so, our algorithm returns $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \{\mathsf{God}\}$.

We can obtain $\mathsf{ans}(Q, \mathcal{K})$ from $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ by querying $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$ for all individuals that are equal to God. In our example, it is possible to show that the only such individual is FSM, so $\mathsf{ans}(Q, \mathcal{K}) = \{\mathsf{God}, \mathsf{FSM}\}$. △

According to the principles outlined in Chapter 4, in order to show that our answering algorithm is correct we need to identify a suitable clause set $\mathcal{N}^{DL}$ that is closed under $\mathcal{R}^{DL}$. We formally define $\mathcal{N}^{DL}$ as follows.

Definition 5.1.3. Let $\mathcal{K}$ be an $\mathcal{ELHIO}$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. The clause set $\mathcal{N}^{DL}$ for $Q = \langle Q_P, Q_C \rangle$ and $\Xi(\mathcal{K})$ is the set containing every safe clause C constructed with the symbols in Q and $\Xi(\mathcal{K})$ such that:

- (i) C is of one of the types shown in Table 5.2.
- If C is of type 1, then
  - (ii) $\mathsf{varNo}(C) \leq \mathsf{varNo}(C')$ where C' is the only clause in Q_C,
  - (iii) $\mathsf{depth}(C) \leq \max(1, \mathsf{varNo}(C') - \mathsf{varNo}(C))$ where C' is the only clause in Q_C, and
  - (iv) if a variable x occurs in a functional term in C, then x occurs in every functional term in C.
- If C is of type 2, then
  - (v) $\mathsf{varNo}(C) \leq 1$, and

Table 5.2: $\mathcal{ELHIO}$ types of clause

Type	$\mathcal{ELHIO}$ clause
1	$Q_P(\vec{s}) \leftarrow \bigwedge D_i(\vec{t_i})$
2	$D(\vec{s}) \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x) \wedge \bigwedge C_k(f(x))$
3.1	$A(x) \leftarrow \underline{P(x,y)} \wedge [B(y)]$
3.2	$A(x) \leftarrow \underline{P(y,x)} \wedge [B(y)]$
3.3	$P(x,y) \leftarrow \underline{S(x,y)}$
3.4	$P(x,y) \leftarrow \underline{S(y,x)}$
3.5	$P(x,a) \leftarrow \underline{P(x,y)} \wedge A(y)$
3.6	$P(a,x) \leftarrow \underline{P(y,x)} \wedge A(y)$
3.7	$P(x,a_1) \leftarrow \underline{P(x,a_2)} \wedge A(a_1)$
3.8	$P(a_1,x) \leftarrow \underline{P(a_2,x)} \wedge A(a_1)$
4.1	$\underline{x \approx a} \leftarrow A(x)$
4.2	$A(y) \leftarrow A(x) \wedge x \approx y \wedge \underline{\mathcal{O}(y)}$
4.3	$P(x,z) \leftarrow P(x,y) \wedge y \approx z \wedge \underline{\mathcal{O}(z)}$
4.4	$P(z,x) \leftarrow P(y,x) \wedge y \approx z \wedge \underline{\mathcal{O}(z)}$
4.5	$A(a) \leftarrow A(x) \wedge \underline{x \approx a}$
4.6	$P(x,a) \leftarrow P(x,y) \wedge \underline{y \approx a}$
4.7	$P(a,x) \leftarrow P(y,x) \wedge \underline{y \approx a}$
4.8	$\underline{\mathcal{O}(a)}$

Note 5.1.2. D (possibly with subscripts) denotes a unary or a binary predicate; x, y, and z denote variables; a (possibly with subscripts) denotes a constant; s and t (possibly with subscripts) denote terms; and $[B(y)]$ denotes a possible occurrence of the atom $B(y)$ in the clause.

– (vi) the head of C is of the form $A(a)$, $A(x)$, $A(f(x))$, $P(a,b)$, $P(a,x)$, $P(x,a)$, $P(a,f(x))$, $P(f(x),a)$, $P(x,f(x))$, or $P(f(x),x)$.

△

We now define the calculus $\mathcal{R}^{DL}$.

Definition 5.1.4. With $\mathcal{R}^{DL}$ we denote the RFS calculus parameterized with the selection function S^{DL} defined as follows.

- If $C \in \mathcal{N}^{DL}$ is of type 1, then S^{DL} selects the head atom H if the body is empty or if H contains function symbols; otherwise, S^{DL} selects all deepest body atoms.

- If $C \in \mathcal{N}^{DL}$ is of type 2, then S^{DL} selects the head atom H if the body is empty or if the depth of H is greater than the maximal depth of the body; otherwise, S^{DL} selects all deepest covering body atoms.
- If $C \in \mathcal{N}^{DL}$ is of type 3.1–3.8, or 4.1–4.8, then S^{DL} selects the atom underlined in Table 5.2.

△

Clearly, $\mathcal{N}^{DL}$ is finite assuming that Q and $\Xi(\mathcal{K})$ are finite; moreover, it can be verified that $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C \subseteq \mathcal{N}^{DL}$. Since $\mathcal{R}^{DL}$ is sound and complete [BG01], in order to obtain a decision procedure we only need to show that each saturation terminates. We do this in Lemma 5.1.1 by showing that $\mathcal{N}^{DL}$ is closed under $\mathcal{R}^{DL}$.

We make the standard assumption of equational theorem proving that $\approx$ is implicitly symmetric; thus, each atom $s \approx t$ should be also read as $t \approx s$—that is, the two forms should be considered interchangeable and both forms should be considered when applying $\mathcal{R}^{DL}$ inferences to clauses containing equality atoms.

Lemma 5.1.1. *Let $\mathcal{K}$ be an $\mathcal{ELHIO}$ KB, Q a CQ, and $\mathcal{N}^{DL}$ the clause set for $\Xi(\mathcal{K})$ and Q. For every two clauses C_1 and C_2 in $\mathcal{N}^{DL}$, and every resolvent C_r of C_1 and C_2 by $\mathcal{R}^{DL}$, we have that $C_r \in \mathcal{N}^{DL}$.*

Proof. The possible inferences are summarized in Table 5.3. If neither C_1 nor C_2 is of type 1, it is straightforward to check that $C_r \in \mathcal{N}^{DL}$; hence, we assume that C_1 is of type 1. For resolution to be possible, C_2 must be of type 2 and the selection function S^{DL} must select the head $D(\vec{v})$ of C_2. By the definition of S^{DL}, if $D(\vec{v})$ is of the form $A(a)$ or $P(a, b)$, then the body of C_2 is empty. Therefore, the inference only reduces the number of body atoms of C_1, so C_r is of type 1. We now consider the case where $D(\vec{v})$ is of the form $P(\alpha, f(x))$ for α a constant a or the variable x; the proof for the other forms of $D(\vec{v})$ is analogous. Let $D(\vec{t})$ be of the form $P(s, u)$. For unification to be possible, the term u must be a variable x_u or a functional term of the form $f(u')$.

Table 5.3: Inferences of $\mathcal{R}^{DL}$ on $\mathcal{N}^{DL}$

1 + 2 = 1:

$$\frac{Q_P(\vec{s}) \leftarrow \underline{D(\vec{t})} \wedge \bigwedge D_i(\vec{u_i}) \quad \underline{D(\vec{v})} \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x)}{Q_P(\vec{s})\sigma \leftarrow A_i(a_i)\sigma \wedge \bigwedge B_j(x)\sigma \wedge \bigwedge D_i(\vec{u_i})\sigma}$$

(a) $\sigma = \mathsf{MGU}(D(\vec{t}), D(\vec{v}))$

2 + 2 = 2:

$$\frac{D(\vec{s}) \leftarrow \bigwedge A_i(a_i) \wedge \underline{A(b)} \quad \underline{A(b)}}{D(\vec{s}) \leftarrow \bigwedge A_i(a_i)}$$

(a) $D(\vec{s})$ is of the form $A(a)$ or $P(a, b)$.

2 + 2 = 2:

$$\frac{D(\vec{s}) \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x) \wedge \underline{B(x)} \quad \underline{B(b)}}{D(\vec{s})\sigma \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(b)}$$

(a) $D(\vec{s})$ is of the form $A(a)$, $A(x)$, $P(a, b)$, $P(a, x)$, or $P(x, a)$.
(b) $\sigma = \{x \mapsto b\}$

2 + 2 = 2:

$$\frac{D(\vec{s}) \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x) \wedge \underline{B(x)} \quad \underline{B(f(x))} \leftarrow \bigwedge A'_i(a_i) \wedge \bigwedge B'_j(x)}{D(\vec{s})\sigma \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(f(x)) \wedge \bigwedge A'_i(a_i) \wedge \bigwedge B'_j(x)}$$

(a) $D(\vec{s})$ is of the form $A(a)$, $A(x)$, $P(a, b)$, $P(a, x)$, or $P(x, a)$.
(b) $\sigma = \{x \mapsto f(x)\}$

2 + 2 = 2:

$$\frac{D(\vec{s}) \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x) \wedge \bigwedge C_k(f(x)) \wedge \underline{C(f(x))} \quad \underline{C(f(x))} \leftarrow \bigwedge A'_i(a_i) \wedge \bigwedge B'_j(x)}{D(\vec{s}) \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x) \wedge \bigwedge C_k(f(x)) \wedge \bigwedge A'_i(a_i) \wedge \bigwedge B'_j(x)}$$

(a) $D(\vec{s})$ is of the form $A(a)$, $A(x)$, $A(f(x))$, $P(a, b)$, $P(a, x)$, $P(x, a)$, $P(a, f(x))$, $P(f(x), a)$, $P(x, f(x))$, or $P(f(x), x)$.

3.1 + 2 = 2:

$$\frac{A(x) \leftarrow \underline{P(x, y)} \wedge [B(y)] \quad \underline{P(s, t)} \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x)}{A(s) \leftarrow \bigwedge A_i(a_i) \wedge \bigwedge B_j(x) \wedge [B(t)]}$$

3.2–3.8 + 2 = 2 are analogous.

4.2 + 4.8 = 4.5:

$$\frac{A(y) \leftarrow A(x) \wedge x \approx y \wedge \underline{\mathcal{O}(y)} \quad \underline{\mathcal{O}(a)}}{A(a) \leftarrow A(x) \wedge x \approx a}$$

4.3 + 4.8 = 4.6 and 4.4 + 4.8 = 4.7 are analogous.

4.5 + 4.1 = 2:

$$\frac{A(a) \leftarrow A(x) \wedge \underline{x \approx a} \quad \underline{x \approx a} \leftarrow B(x)}{A(a) \leftarrow A(x) \wedge B(x)}$$

4.6 + 4.1 = 3.5 and 4.7 + 4.1 = 3.6 are analogous.

4.5 + 4.1 = 2:

$$\frac{A(a) \leftarrow A(x) \wedge \underline{x \approx a} \quad \underline{x \approx b} \leftarrow B(x)}{A(a) \leftarrow A(b) \wedge B(a)}$$

4.6 + 4.1 = 3.7 and 4.7 + 4.1 = 3.8 are analogous.

Note 5.1.3. The notation $X + Y = Z$ denotes that resolving a clause of type X with a clause of type Y produces a clause of type Z.

- Let u be a variable x_u. If α is also a variable x, then $\sigma = \{x \mapsto s, x_u \mapsto f(s)\}$. If α is a constant a, then $\sigma = \{s \mapsto a, x_u \mapsto f(x)\}$. In the former case, due to the occurs-check in unification, x_u cannot occur in s. In the latter case, x_u cannot occur in s because s has to be a variable different from x_u or the constant a. The inference thus decreases the number of variables by one in both cases, so C_r satisfies condition (ii). Moreover, in the former case, since C_1 satisfies condition (iv), x_u does not occur in any functional term in C_1 (because it does not occur in s). Furthermore, in the latter case, given the selection function of $\mathcal{R}^{DL}$, since s is either a variable or a constant, C_1 has no functional terms. Hence, even though x_u is mapped to a functional term, we have that $\mathsf{depth}(C_r) \leq \mathsf{depth}(C_1) + 1$, but since the number of variables has been decreased by one in C_r, condition (iii) is satisfied. Finally, since every occurrence of x_u is replaced with the same term, the resolvent C_r satisfies condition (iv) as well.

- Let u be a functional term $f(u')$. If α is a variable x, then we have that either $s = u'$ and $\sigma = \{x \mapsto u'\}$; s is a variable x_s (not occurring in u') and $\sigma = \{x \mapsto u', x_s \mapsto u'\}$; or u' is a variable x'_u and $\sigma = \{x \mapsto s, x'_u \mapsto s\}$. If α is a constant a, then $\sigma = \{s \mapsto a, x \mapsto u'\}$. In all cases, the inference does not increase the number of variables or function symbols; therefore, C_r satisfies conditions (ii) and (iii). Furthermore, the inference does not introduce new function symbols, so C_r satisfies condition (iv) as well.

□

It can be shown along the lines of Lemma 4.3.1 that $\vec{a} \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ if and only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$. Therefore, Theorem 5.1.1 immediately follows given that Lemma 5.1.1 holds.

Theorem 5.1.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. The saturation of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C$ by $\mathcal{R}^{DL}$ terminates and*

$$\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \{\vec{a} \mid Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}\}.$$

In the next section, we show how to compute $\mathsf{ans}(Q, \mathcal{K})$ from $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$.

5.1.2 Computing the Certain Answers

Our goal is to show how to compute $\mathsf{ans}(Q, \mathcal{K})$ from $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. We first define a set $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ that depends on $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$, and then show that $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \mathsf{ans}(Q, \mathcal{K})$.

Definition 5.1.5. Let $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \{\langle a_{1,1}, ..., a_{1,m} \rangle, ..., \langle a_{n,1}, ..., a_{n,m} \rangle\}$ for a CQ Q and an $\mathcal{ELHIO}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. Let $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \bigcup_{i=1}^{n} \times_{j=1}^{m} r_{a_{i,j}}$, where $r_{a_{i,j}}$ is the set containing exactly $a_{i,j}$ and every constant b such that $\langle a_{i,j}, b \rangle$ is contained in the transitive closure of $\mathsf{ans}(Q_{eq}(x, y) \leftarrow x \approx y, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. △

The idea is to replace each constant $a_{i,j}$ with the set of constants that are equal to it with respect to $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. We explain the intuition behind the definition of $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ with an example.

Example 5.1.2. Consider an $\mathcal{ELHIO}$ TBox $\mathcal{T}$ containing the following axioms:

$$\{\mathsf{FSM}\} \sqsubseteq \{\mathsf{God}\}$$

$$\{\mathsf{God}\} \sqsubseteq \{\mathsf{Zeus}\}$$

Consider an ABox $\mathcal{A}$ containing the following assertions:

$$\mathrm{Mighty}(\mathsf{FSM})$$

$$\mathrm{Omniscient}(\mathsf{God})$$

$$\mathrm{Omnipotent}(\mathsf{Zeus})$$

Consider the following query Q:

$$Q_P(x) \leftarrow \text{Mighty}(x) \wedge \text{Omniscient}(x) \wedge \text{Omnipotent}(x)$$

Clearly, FSM, God and Zeus are equal according to $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. Therefore, we have that $\mathsf{ans}(Q, \mathcal{K}) = \{\mathsf{FSM}, \mathsf{God}, \mathsf{Zeus}\}$.

We now explain how to use our answering approach in order to compute $\mathsf{ans}(Q, \mathcal{K})$. The first step is to compute $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ as explained in Section 5.1.1. By saturating $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$ with $\mathcal{R}^{DL}$, it can be shown that $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \{\mathsf{God}, \mathsf{Zeus}\}$.

The second step consists of computing $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. It can be shown that

$$\{\langle \mathsf{FSM}, \mathsf{God} \rangle, \langle \mathsf{God}, \mathsf{FSM} \rangle, \langle \mathsf{Zeus}, \mathsf{God} \rangle, \langle \mathsf{God}, \mathsf{Zeus} \rangle\}$$

is a subset of $\mathsf{ans}(Q_{\text{eq}}(x, y) \leftarrow x \approx y, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. Therefore, it can be readily verified that $r_{\mathsf{God}} = r_{\mathsf{Zeus}} = \{\mathsf{FSM}, \mathsf{God}, \mathsf{Zeus}\}$.

Clearly, substituting each tuple $\langle a \rangle$ in $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ with r_a yields the set $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \{\mathsf{FSM}, \mathsf{God}, \mathsf{Zeus}\}$. △

In the rest of the section we show several properties required to prove that $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \mathsf{ans}(Q, \mathcal{K})$.

In our proofs we use the notion of *representative*. Intuitively, the idea is to define a unique representative term for every set of terms that are equal according to $\mathcal{K}$. The role of the representative is to "gather" all the information of all terms that are mutually equal: we ensure that everything that follows from $\mathcal{K}$ for the terms that are mutually equal also follows from $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$ for the corresponding representative term. Therefore, we argue that it is possible to answer queries over $\mathcal{K}$ by answering them over $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$, and then "expanding" the answers to the set of represented terms. We formally define the representative $[s]_I$ of a term s as follows.

Definition 5.1.6. Let $\prec_{\mathcal{K}}$ be a total well-founded order on the set of terms occurring in the Herbrand universe of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$ such that, for all terms s and t, we have that

$\mathsf{depth}(s) < \mathsf{depth}(t)$ implies $s \prec_{\mathcal{K}} t$, and for every $\mathcal{O}$-constant o and every term s, we have that $o \prec_{\mathcal{K}} s$.

Let I be the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. For all terms s and t occurring in I, we say that s and t are $\approx$-connected with respect to I, written $s \approx^*_I t$, if $s = t$ or there are terms $u_1, ..., u_n$ such that $\{s \approx u_1, u_1 \approx u_2, ..., u_n \approx t\} \subseteq I$. For every term s occurring in I, with $\mathsf{min}(s)$ we denote the term such that $s \approx^*_I \mathsf{min}(s)$, and there is no other term t such that $s \approx^*_I t$ and $t \prec_I \mathsf{min}(s)$.

For every term s occurring in the Herbrand universe of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$, we define the representative $[s]_I$ of s with respect to I inductively as follows: if s is a constant or if there is an $\mathcal{O}$-constant o such that $s \approx o \in I$, then $[s]_I = \mathsf{min}(s)$; otherwise, s is of the form $f(s')$ and $[s]_I = \mathsf{min}(f([s']_I))$.

For $\vec{s} = \langle s_1, ..., s_n \rangle$ a tuple of n terms, with $[\vec{s}]_I$ we denote the tuple $\langle [s_1]_I, ..., [s_n]_I \rangle$. △

We now show that $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$ ensures the "propagation of information" from every term s to its representative $[s]_I$ (cf. Lemma 5.1.5)—that is, we show that

$$\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \models A(s) \text{ implies } \Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \models A([s]_I) \text{ and}$$
$$\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \models P(s,t) \text{ implies } \Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \models P([s]_I, [t]_I).$$

In order to do so, we first prove a property of the binary atoms occurring in the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$ (cf. Lemma 5.1.2), and two properties of the representative function $[\cdot]_I$ (cf. Lemma 5.1.3 and Lemma 5.1.4).

Lemma 5.1.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. Every binary atom D occurring in I is of the form (i) $P(s, f(s))$, (ii) $P(f(s), s)$, (iii) $P(s, o)$, or (iv) $P(o, s)$, where s is a term and o is a constant.*

Proof. If $D \in \mathcal{A}$, then D is of form (iii) or (iv). Otherwise, by analyzing the type of clauses with binary head in Table 5.1, it can be readily checked that if D was derived

through one of these types of clause, then it is of the form (i)–(iv). By analyzing the binary substitutivity$_{\mathcal{O}}$ clauses of $E'_{\mathcal{T}}$, it can be checked that if D was derived through one of these types of clause, then it is of the form (iii) or (iv). There is no other type of clause with binary head in $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. The claim follows from these facts since I is minimal. □

Lemma 5.1.3. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. For every term s such that $s \neq \mathsf{min}(s)$, there exist $\mathcal{O}$-constants $o_1, ..., o_n$ such that $\{s \approx o_1, o_1 \approx o_2, ..., o_n \approx \mathsf{min}(s)\} \subseteq I$.*

Proof. We first show that, for all $\mathcal{O}$-constants o and o', if $t \approx o \in I$ and $t \approx o' \in I$ for some term t, then $o \approx o' \in I$. Since I is minimal, the assertion $t \approx o'$ was derived by a clause $x \approx o' \leftarrow A(x)$ and a fact $A(t) \in I$ for some predicate A. We have $t \approx o \in I$; moreover, since o is an $\mathcal{O}$-constant, we have $\mathcal{O}(o) \in I$. Therefore, the unary substitutivity$_{\mathcal{O}}$ clauses in $E'_{\mathcal{T}}$ then ensures $A(o) \in I$, which implies $o \approx o' \in I$.

Now, since $s \neq \mathsf{min}(s)$, by the definition of $\mathsf{min}(s)$, there exist terms $u_1, ..., u_n$ such that $\Gamma \subseteq I$ for $\Gamma = \{s \approx u_1, u_1 \approx u_2, ..., u_n \approx \mathsf{min}(s)\}$. Given the type of clauses contained in $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$, every equality assertion in I is of the form $t \approx o$, where t is a term and o is an $\mathcal{O}$-constant. Assume now that some term u_i is not an $\mathcal{O}$-constant. Then, Γ contains equalities of the form $u_i \approx o$ and $u_i \approx o'$. By the previous paragraph, I then contains $o \approx o'$; hence, u_i can be eliminated from Γ. By successively eliminating all u_j that are not $\mathcal{O}$-constants, we obtain a subset of the equalities of I that satisfy our claim. □

Lemma 5.1.4. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. For every term s, every constant o, all predicates A and P, and every $n \geq 0$, we have that*

(i) $A(f_1...f_n(s)) \in I$ implies $A(f_1...f_n([s]_I)) \in I$,

(ii) $P(f_1...f_n(s), f_0f_1...f_n(s)) \in I$ implies $P(f_1...f_n([s]_I), f_0f_1...f_n([s]_I)) \in I$,

(iii) $P(f_1...f_n(s), f_2...f_n(s)) \in I$ *implies* $P(f_1...f_n([s]_I), f_2...f_n([s]_I)) \in I$,

(iv) $P(f_1...f_n(s), o) \in I$ *implies* $P(f_1...f_n([s]_I), [o]_I) \in I$,

(v) $P(o, f_1...f_n(s)) \in I$ *implies* $P([o]_I, f_1...f_n([s]_I)) \in I$, *and*

(vi) $f_1...f_n(s) \approx o$ *implies* $f_1...f_n([s]_I) \approx o$.

Proof. Let D be an atom of one of the following forms:

$$A(f_1...f_n(s)) \tag{5.35}$$

$$P(f_1...f_n(s), f_0 f_1...f_n(s)) \tag{5.36}$$

$$P(f_1...f_n(s), f_2...f_n(s)) \tag{5.37}$$

$$P(f_1...f_n(s), o) \tag{5.38}$$

$$P(o, f_1...f_n(s)) \tag{5.39}$$

$$f_1...f_n(s) \approx o \tag{5.40}$$

We prove the claim by induction on the height h of the derivation tree of D.

The base case is $h = 0$. Since I is minimal, D is of the form $A(a)$ or $P(a, b)$. We consider the case where D is of the form $P(a, b)$; the other case can be proved analogously. By the definition of $[\cdot]_I$, we have that $[a]_I = \mathsf{min}(a)$ and $[b]_I = \mathsf{min}(b)$. If $a = \mathsf{min}(a)$ and $b = \mathsf{min}(b)$, then the claim trivially follows. We consider the case when $a \neq \mathsf{min}(a)$ and $b \neq \mathsf{min}(b)$; the case where $a \neq \mathsf{min}(a)$ and $b = \mathsf{min}(b)$, and vice versa can be shown analogously. It follows by Lemma 5.1.3 that $\mathcal{O}$-constants $o_1, ..., o_n$ exist such that $\{a \approx o_1, o_1 \approx o_2, ..., o_n \approx \mathsf{min}(a)\} \subseteq I$; similarly, we have that $\mathcal{O}$-constants $o'_1, ..., o'_m$ exist such that $\{b \approx o'_1, o'_1 \approx o'_2, ..., o'_m \approx \mathsf{min}(b)\} \subseteq I$. Due to the substitutivity$_{\mathcal{O}}$ clauses in $E'_{\mathcal{T}}$, we have $P(o_i, b) \in I$ for $1 \leq i \leq n$; and since $o_n \approx \mathsf{min}(a)$, we have $P(\mathsf{min}(a), b) \in I$. Analogously, we have $P(\mathsf{min}(a), o'_j) \in I$ for $1 \leq j \leq m$; and since $o'_m \approx \mathsf{min}(b)$, we have that $P(\mathsf{min}(a), \mathsf{min}(b)) \in I$.

We now assume the claim holds for h and we prove the claim for $h + 1$. We first consider the case where D is of the form (5.35). Given the clauses contained

in $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$, we have that D could have been derived using a clause C of one the following forms:

$$A(x) \leftarrow B(x) \tag{5.41}$$

$$A(x) \leftarrow B(x) \wedge C(x) \tag{5.42}$$

$$A(x) \leftarrow P(x, y) \wedge B(y) \tag{5.43}$$

$$A(x) \leftarrow P(y, x) \wedge B(y) \tag{5.44}$$

$$A(y) \leftarrow A(x) \wedge x \approx y \wedge \mathcal{O}(y) \tag{5.45}$$

$$A(f(x)) \leftarrow B(x) \tag{5.46}$$

- If C is of the form (5.41) or (5.42), then by the induction hypothesis we have that the claim holds for every antecedent atom required to apply C, which implies that $A(f_1...f_n([s]_I)) \in I$.

- If C is of the form (5.43), then $\{P(f_1...f_n(s), t), B(t)\} \subseteq I$ for some term t. Lemma 5.1.2 implies that the term t can be of the form o, $f_0 f_1...f_n(s)$, or $f_2...f_n(s)$. In all cases, by the induction hypothesis we have that the claim holds for every antecedent atom required to apply C; therefore, $A(f_1...f_n([s]_I)) \in I$. The proof is analogous if C is of the form (5.44).

- If C is of the form (5.45), then we have that D is of the form $A(o)$ for some $\mathcal{O}$-constant o. By the definition of $[\cdot]_I$, we have that $[o]_I = \mathsf{min}(o)$. If $o = \mathsf{min}(o)$, then the claim trivially follows, so we assume that $o \neq \mathsf{min}(o)$. Given Lemma 5.1.3 and the substitutivity$_{\mathcal{O}}$ clauses of $E'_{\mathcal{T}}$, we have that $A(\mathsf{min}(o)) \in I$.

- If C is of the form (5.46), then we consider two cases: $n > 0$ and $n = 0$. In the former case, we have that $B(f_2...f_n(s)) \in I$. By the induction hypothesis we have $B(f_2...f_n([s]_I)) \in I$, so $A(f_1...f_n([s]_I)) \in I$. For $n = 0$, note that, given the form of C, s is of the form $s = f(s')$ and $B(s') \in I$. We consider the possible forms of $[f(s')]_I$.

– If an $\mathcal{O}$-constant o exists such that $f(s') \approx o \in I$, then $[f(s')]_I = \mathsf{min}(o)$. Given Lemma 5.1.3 and the substitutivity$_{\mathcal{O}}$ clauses of $E'_{\mathcal{T}}$, we have that $A(\mathsf{min}(o)) \in I$ since $A(f(s')) \in I$.

– Otherwise, we have $[f(s')]_I = \mathsf{min}(f([s']_I))$. If there is an $\mathcal{O}$-constant o such that $f([s']_I) \approx o \in I$, then $[f(s')]_I = \mathsf{min}(f([s']_I)) = \mathsf{min}(o)$. By the induction hypothesis we have that $B([s']_I) \in I$, so $A(f([s']_I)) \in I$. Therefore, it follows from Lemma 5.1.3 and the substitutivity$_{\mathcal{O}}$ clauses of $E'_{\mathcal{T}}$ that $A(\mathsf{min}(o)) \in I$. Finally, if there is no $\mathcal{O}$-constant o such that $f([s']_I) \approx o \in I$, we have that $[f(s')]_I = \mathsf{min}(f([s']_I)) = f([s']_I)$ and the claim follows since $A(f([s']_I)) \in I$.

We now consider the case where D is of the form (5.36)–(5.39). The atom D could have been derived using a clause C of one the following forms:

$$P(x, y) \leftarrow S(x, y) \tag{5.47}$$

$$P(x, y) \leftarrow S(y, x) \tag{5.48}$$

$$P(x, z) \leftarrow P(x, y) \wedge y \approx z \wedge \mathcal{O}(z) \tag{5.49}$$

$$P(z, x) \leftarrow P(y, x) \wedge y \approx z \wedge \mathcal{O}(z) \tag{5.50}$$

$$P(x, f(x)) \leftarrow A(x) \tag{5.51}$$

$$P(f(x), x) \leftarrow A(x) \tag{5.52}$$

- If C is of the form (5.47) or (5.48), then by the induction hypothesis we have that the claim holds for every antecedent atom required to apply C, which implies that the claim holds for all possible forms of D.

- If C is of the form (5.49), then D is of the form (5.38). Clearly, in order for C to be applied, we have that $\{P(f_1...f_n(s), t), t \approx o, \mathcal{O}(o)\} \subseteq I$ for some term t. Lemma 5.1.2 implies that the term t can be of the form o', $f_0 f_1...f_n(s)$, or $f_2...f_n(s)$. In all cases, by the induction hypothesis the claim holds for

$P(f_1...f_n(s), t)$ and $t \approx o$, which is enough to derive $P(f_1...f_n([s]_I), o)$ using C. By the definition of $[\cdot]_I$ we have that $[o]_I = \mathsf{min}(o)$. If $o = \mathsf{min}(o)$, then the claim trivially follows, so we assume that $o \neq \mathsf{min}(o)$. By Lemma 5.1.3 and the substitutivity$_{\mathcal{O}}$ clauses of $E'_{\mathcal{T}}$, we have that $P(f_1...f_n([s]_I), \mathsf{min}(o)) \in I$; the claim can be shown analogously if C is of the form (5.50).

- If C is of the form (5.51), then D is of the form (5.36) and $A(f_1...f_n(s)) \in I$. By the induction hypothesis we have that $A(f_1...f_n([s]_I)) \in I$; therefore, we have that $P(f_1...f_n([s]_I), f_0 f_1...f_n([s]_I)) \in I$; the claim can be shown analogously if C is of the form (5.52).

Finally, we consider the case where D is of the form (5.40). The atom D could have been derived using a clause C of the following form:

$$x \approx o \leftarrow A(x) \tag{5.53}$$

We have that $A(f_1...f_n(s)) \in I$. Moreover, $A(f_1...f_n([s]_I)) \in I$ by the induction hypothesis; therefore, $f_1...f_n([s]_I) \approx o \in I$. □

We can now show the desired relationship between a term and its representative.

Lemma 5.1.5. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. For all terms s and t, and all predicates A and P occurring in I, we have that*

(i) $A(s) \in I$ implies $A([s]_I) \in I$, and

(ii) $P(s,t) \in I$ implies $P([s]_I, [t]_I) \in I$.

Proof. Claim (i) follows from Lemma 5.1.4. We now consider claim (ii). Let D be an atom of the form $P(s,t)$. In case D is of the form $P(a,b)$, $P(s,a)$, or $P(a,s)$, the claim follows from Lemma 5.1.4. We now consider the case where D is of the form $P(s, f(s))$; the case where D is of the form $P(f(s), s)$ can be shown analogously. We analyze the possible forms of $[f(s)]_I$.

- If there is an $\mathcal{O}$-constant o such that $f(s) \approx o \in I$, then $[f(s)]_I = \mathsf{min}(o)$. By Lemma 5.1.4 we have that $f([s]_I) \approx o \in I$ and $P([s]_I, f([s]_I)) \in I$. By Lemma 5.1.3 and the substitutivity$_{\mathcal{O}}$ clauses of $E'_{\mathcal{T}}$ we have that $P([s]_I, \mathsf{min}(o)) \in I$.

- Otherwise, we have that $[f(s)]_I = \mathsf{min}(f([s]_I))$. If there is an $\mathcal{O}$-constant o such that $f([s]_I) \approx o \in I$, then $[f(s)]_I = \mathsf{min}(f([s]_I)) = \mathsf{min}(o)$, and the claim follows again by Lemma 5.1.3 and the substitutivity$_{\mathcal{O}}$ clauses of $E'_{\mathcal{T}}$. Finally, if there is no $\mathcal{O}$-constant o such that $f([s]_I) \approx o \in I$, we have that $[f(s)]_I = \mathsf{min}(f([s]_I)) = f([s]_I)$, and the claim follows since we have that $P([s]_I, f([s]_I)) \in I$.

□

We now show that $[\cdot]_I$ "compensates" for the loss of functional monotonicity: we show that $[s]_I = [t]_I$ implies $[f(s)]_I = [f(t)]_I$ (cf. Lemma 5.1.6). We point out that in order to do so, it suffices to show that $[f(s)]_I = [f([s]_I)]_I$. We illustrate the point with an example.

Example 5.1.3. Assume that (*) $[f(s)]_I = [f([s]_I)]_I$ for every term s and every function symbol f. Now, suppose that

$$[\mathsf{FSM}]_I = [\mathsf{Zeus}]_I = \mathsf{God}.$$

Let religionOf be a function symbol. It follows from assumption (*) that

$$[\text{religionOf}(\mathsf{FSM})]_I = [\text{religionOf}(\mathsf{God})]_I,$$
$$[\text{religionOf}(\mathsf{Zeus})]_I = [\text{religionOf}(\mathsf{God})]_I.$$

Therefore, as can be seen, if assumption (*) holds, then $[\mathsf{FSM}]_I = [\mathsf{Zeus}]_I$ implies $[\text{religionOf}(\mathsf{FSM})]_I = [\text{religionOf}(\mathsf{Zeus})]_I$. △

Lemma 5.1.6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. For every term s and every function symbol f, we have that $[f(s)]_I = [f([s]_I)]_I$.*

Proof. We show the claim by analyzing the possible forms of $[f(s)]_I$.

- If there is an $\mathcal{O}$-constant o such that $f(s) \approx o \in I$, then $[f(s)]_I = \mathsf{min}(o)$. By Lemma 5.1.4 we have that $f([s]_I) \approx o \in I$; therefore, $[f([s]_I)]_I = \mathsf{min}(o)$.
- Otherwise, we have that $[f(s)]_I = \mathsf{min}(f([s]_I))$. If there is an $\mathcal{O}$-constant o such that $f([s]_I) \approx o \in I$, then $\mathsf{min}(f([s]_I)) = \mathsf{min}(o)$ and $[f([s]_I)]_I = \mathsf{min}(o)$. Finally, if there is no $\mathcal{O}$-constant o such that $f([s]_I) \approx o \in I$, then we have that $\mathsf{min}(f([s]_I)) = f([s]_I)$ and $[f([s]_I)]_I = f([s]_I)$.

□

We now concentrate on showing the crucial fact that $[\vec{a}]_I \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $[\vec{a}]_I \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ (cf. Lemma 5.1.10). Note that, in order to show this claim, it suffices to show that $\Xi(\mathcal{K}) \cup E_{\mathcal{T}} \models D([\vec{s}]_I)$ if and only if $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \models D([\vec{s}]_I)$ for every atom $D([\vec{s}]_I)$. Moreover, since both sets—$\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$ and $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$—are sets of Horn clauses, we can simply show that $D([\vec{s}]_I) \in J$ if and only if $D([\vec{s}]_I) \in I$, where J is the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$ and I is the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. We show this by constructing an interpretation I_∞ from I such that $D([\vec{s}]_I) \in I_\infty$ if and only if $D([\vec{s}]_I) \in I$, and proving that I_∞ is the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$.

Intuitively, we obtain I_∞ from I by adding to I the set of facts needed to make $\approx$ a *congruence relation*: we propagate every logical consequence for $[s]_I$ to all the terms represented by $[s]_I$, and we ensure that $\approx$ is a transitive, symmetric and reflexive relation in I_∞, and that it conforms to functional monotonicity. We formally define I_∞ as follows.

Definition 5.1.7. Let I be the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$. Let $\mathcal{O}_I$ be the set of exactly all the atoms of the form $\mathcal{O}(o)$ contained in I.

We construct I_∞ as follows. Let I_0 be the smallest set of facts that satisfy the following conditions for all predicates A and P, all terms s and t, and every constant o:

(i) $I \setminus \mathcal{O}_I \subseteq I_0$;

(ii) if $A(s) \in I$, then $A(s') \in I_0$ for every term s' such that $[s']_I = s$;

(iii) if $P(s,t) \in I$, then $P(s',t') \in I_0$ for all terms s' and t' such that $[s']_I = s$ and $[t']_I = t$; and

(iv) if $s \approx o \in I$, then $s' \approx o \in I_0$ for every term s' such that $[s']_I = s$.

Let I_i, for $1 \leq i \leq \infty$, be the smallest set of facts that satisfy the following conditions for all terms s, t, and u, and every function symbol f:

(v) $I_{i-1} \subseteq I_i$;

(vi) $s \approx s \in I_i$;

(vii) if $s \approx t \in I_i$, then $t \approx s \in I_i$;

(viii) if $s \approx u \in I_i$ and $u \approx t \in I_i$, then $s \approx t \in I_i$; and

(ix) if $s \approx t \in I_i$, then $f(s) \approx f(t) \in I_i$.

Let $I_\infty = \bigcup I_i$. △

We now show that I_∞ is the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E_\mathcal{T}$. We do so in two steps: we first show that I_∞ is a model of $\Xi(\mathcal{K}) \cup E_\mathcal{T}$ (cf. Lemma 5.1.8) and then we show that I_∞ is minimal (cf. Lemma 5.1.9). In order to do so, we first show an important property of $[\cdot]_I$ in Lemma 5.1.7.

Lemma 5.1.7. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_\mathcal{T}$. For all terms s and t, we have that $s \approx t \in I_\infty$ if and only if $[s]_I = [t]_I$.*

Proof. We first prove that $s \approx t \in I_\infty$ implies $[s]_I = [t]_I$. We prove the claim by induction on the level i in the construction of I_i for $0 \leq i \leq \infty$. In case $i = 0$, it follows from the definition of I_0 that either $s \approx t \in I$ or $[s]_I \approx t \in I$. In both cases, the claim follows from the definition of $[\cdot]_I$. We now assume that the claim holds for i and prove the claim for $i + 1$. The atom $s \approx t$ was derived due to conditions (vi)–(ix). The claim trivially follows for condition (vi). For condition (vii) we have that $t \approx s \in I_i$. By the induction hypothesis we have that $[t]_I = [s]_I$, so the claim holds. For condition (viii) we have that there is a term u such that $s \approx u \in I_i$ and $u \approx t \in I_i$. By the induction hypothesis we have that $[s]_I = [t]_I = [u]_I$, so the claim holds. We now consider condition (ix). Let s be of the form $f(s')$ and t be of the form $f(t')$ for some function symbol f. We have that $s' \approx t' \in I_i$. By the induction hypothesis we have that $[s']_I = [t']_I$, and by Lemma 5.1.6, we have that $[f(s')]_I = [f(t')]_I$. We have shown that $s \approx t \in I_i$ implies $[s]_I = [t]_I$ for any $0 \leq i \leq \infty$. The claim for I_∞ follows from the fact that $s \approx t \in I_\infty$ implies that there is an i such that $s \approx t \in I_i$.

We now prove that $[s]_I = [t]_I$ implies $s \approx t \in I_\infty$. It follows from the definition of $[\cdot]_I$ and Lemma 5.1.6 that there are three cases for which $[s]_I = [t]_I$: (I) $s = t$; (II) there are terms $u_1, ..., u_m$ such that $\{s \approx u_1, u_1 \approx u_2, ..., u_m \approx t\} \subseteq I$; or (III) s is of the form $f(s')$, the term t is of the form $f(t')$, and $[s']_I = [t']_I$. Let $s = f_1...f_n(s'')$ and $t = f_1...f_n(t'')$ for the longest possible common prefix $f_1...f_n$ of s and t. We prove the claim by induction on n. If $n = 0$, then only cases (I) and (II) apply. The claim follows from conditions (vi) and (viii), respectively, of the definition of I_∞. We now assume that the claim holds for n and prove the claim for $n + 1$. In cases (I) and (II), the claim follows again from conditions (vi) and (viii), respectively. In case (III) we have that s is of the form $f_1...f_{n+1}(s'')$, the term t is of the form $f_1...f_{n+1}(t'')$, and $[f_2...f_{n+1}(s'')]_I = [f_2...f_{n+1}(t'')]_I$. By the induction hypothesis we have that $f_2...f_{n+1}(s'') \approx f_2...f_{n+1}(t'') \in I_\infty$. Therefore, there is an integer i such that

$f_2 ... f_{n+1}(s'') \approx f_2 ... f_{n+1}(t'') \in I_i$ and the claim follows due to condition (ix) of the construction of I_∞. □

We are ready to show that I_∞ is a model of $\Xi(\mathcal{K}) \cup E_\mathcal{T}$.

Lemma 5.1.8. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_\mathcal{T}$. Then, I_∞ is a Herbrand model of $\Xi(\mathcal{K}) \cup E_\mathcal{T}$.*

Proof. We first show that I_0 is a model of $\Xi(\mathcal{K})$. The set $\Xi(\mathcal{K})$ contains clauses of the form (5.41)–(5.44), (5.46)–(5.48), and (5.51)–(5.53).

We now show that I_0 is a model of clauses of form (5.43); the proof is analogous for clauses of form (5.41), (5.42), (5.44), (5.47), and (5.48). We show that if $P(s,t) \in I_0$ and $B(t) \in I_0$, then $A(t) \in I_0$. If we have that $P(s,t) \in I$ and $B(t) \in I$, then $A(t) \in I$ because I is a model of $\Xi(\mathcal{K})$. Therefore, $A(t) \in I_0$ since $I \setminus \mathcal{O}_I \subseteq I_0$. Otherwise, we first assume that $P(s,t) \notin I$ and $B(t) \in I$. It follows from the definition of I_0 that $P([s]_I, [t]_I) \in I$; moreover, Lemma 5.1.5 implies that $B([t]_I) \in I$. Therefore, we have that $A([t]_I) \in I$ and according to condition (ii) of the definition of I_0, we have that $A(t) \in I_0$; the claim can be shown analogously for the case where $P(s,t) \in I$ and $B(t) \notin I$. Finally, we assume that $P(s,t) \notin I$ and $B(t) \notin I$. It follows from the definition of I_0 that $P([s]_I, [t]_I) \in I$ and $B([t]_I) \in I$; therefore, we have that $A([t]_I) \in I$ and according to condition (ii) of the definition of I_0, we have that $A(t) \in I_0$.

We now show that I_0 is a model of clauses of form (5.51). We show that if $A(s) \in I_0$, then $P(s, f(s)) \in I_0$. If we have that $A(s) \in I$, then $P(s, f(s)) \in I$ because I is a model of $\Xi(\mathcal{K})$. Therefore, $P(s, f(s)) \in I_0$, since $I \setminus \mathcal{O}_I \subseteq I_0$. Otherwise, it follows from the definition of I_0 that $A([s]_I) \in I$; therefore, we have that $P([s]_I, f([s]_I)) \in I$. By Lemma 5.1.5, we have that $P([[s]_I]_I, [f([s]_I)]_I) \in I$. The function $[\cdot]_I$ is idempotent by definition, so we have $P([s]_I, [f([s]_I)]_I) \in I$; moreover, by Lemma 5.1.6, we have that $[f([s]_I)]_I = [f(s)]_I$, so $P([s]_I, [f(s)]_I) \in I$. Now, accord-

ing to condition (iii) of the definition of I_0, we have that $P(s, f(s)) \in I_0$; the proof is analogous for clauses of form (5.46) and (5.52).

We finally show that I_0 is a model of clauses of form (5.53). We show that if $A(s) \in I_0$, then $s \approx o \in I_0$. If we have that $A(s) \in I$, then $s \approx o \in I$ because I is a model of $\Xi(\mathcal{K})$. Therefore, $s \approx o \in I_0$, since $I \setminus \mathcal{O}_I \subseteq I_0$. Otherwise, it follows from the definition of I_0 that $A([s]_I) \in I$; therefore, we have that $[s]_I \approx o \in I$. Now, according to condition (iv) of the definition of I_0, we have that $s \approx o \in I_0$.

Since I_0 is a model of $\Xi(\mathcal{K})$, we have that I_∞ is also a model of $\Xi(\mathcal{K})$ since $I_0 \subseteq I_\infty$, only atoms of the form $s \approx t$ are added to I_i for every $i > 0$, and no clause in $\Xi(\mathcal{K})$ contains the predicate $\approx$ in the body.

We now show that I_∞ is a model of $E_\mathcal{T}$. It follows from the definition of I_i for $i > 0$ that I_∞ is a model of the reflexivity, symmetry, transitivity, and functional monotonicity clauses of $E_\mathcal{T}$. Therefore, we need only consider the substitutivity clauses of $E_\mathcal{T}$. We consider binary substitutivity 1. We show that if $P(s, t) \in I_\infty$ and $t \approx u \in I_\infty$, then $P(s, u) \in I_\infty$. Since no atom of the form $P(s, t)$ is added to I_∞ in the construction of I_i for $i > 0$, we have that $P(s, t) \in I_0$. In case $P(s, t) \in I$, by Lemma 5.1.5, we have $P([s]_I, [t]_I) \in I$; and in case $P(s, t) \notin I$, it follows from the definition of I_0 that $P([s]_I, [t]_I) \in I$. Since $t \approx u \in I_\infty$, Lemma 5.1.7 implies that $[t]_I = [u]_I$. Therefore, according to condition (iii) of the definition of I_0 we have that $P(s, u) \in I_0$, which implies that $P(s, u) \in I_\infty$ since $I_0 \subseteq I_\infty$; the claim can be shown analogously for other substitutivity clauses. □

We are ready to show that I_∞ is minimal.

Lemma 5.1.9. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_\mathcal{T}$. Every fact $D \in I_\infty$ is derivable from $\Xi(\mathcal{K}) \cup E_\mathcal{T}$.*

Proof. Note that D is of the form $s \approx t$, $A(s)$ or $P(s, t)$. We first consider the case where $D \in I$. Let $\Sigma = \langle T, \delta, \lambda \rangle$ be the derivation tree of D. We transform Σ into a

new derivation tree Σ' as follows: for every node $n \in T$, if $\delta(n)$ is of the form $\mathcal{O}(o)$, then remove n from T; and if $\lambda(n)$ contains an atom of the form $\mathcal{O}(y)$, then remove it from $\lambda(n)$. Given the substitutivity clauses contained in $E_{\mathcal{T}}$ it follows that Σ' is a derivation tree for D such that for every node $n \in T'$ we have that $\lambda(n) \in \Xi(\mathcal{K}) \cup E_{\mathcal{T}}$. Therefore, D is derivable from $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$.

We now consider the case where $D \notin I$. In case D is of the form $s \approx t$, we have that it was added to I_∞ due to condition (iv) of the definition of I_0, or conditions (vi)–(ix) of the definition of I_i for $i > 0$. For condition (iv), it follows from the definition of I_0 that $[s]_I \approx t \in I$; and since $I \setminus \mathcal{O}_I \subseteq I_\infty$, we have $[s]_I \approx t \in I_\infty$. Since $[s]_I = [[s]_I]_I$, Lemma 5.1.7 implies that $s \approx [s]_I \in I_\infty$. Therefore, the atom $s \approx t$ is derivable from $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$ given the transitivity clause of $E_{\mathcal{T}}$. For conditions (vi)–(ix), it is easy to see that $s \approx t$ is derivable from $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$ given the reflexivity, symmetry, transitivity, and functional monotonicity clauses of $E_{\mathcal{T}}$, respectively. We now assume that D is of the form $P(s,t)$; the unary case can be shown analogously. Since no atom of the form $P(s,t)$ is added to I_∞ in the construction of I_i for $i > 0$, we have that $P(s,t) \in I_0$. In case $P(s,t) \in I$, by Lemma 5.1.5, we have $P([s]_I, [t]_I) \in I$; and in case $P(s,t) \notin I$, it follows from the definition of I_0 that $P([s]_I, [t]_I) \in I$. As already shown, there is a derivation tree Σ' for $P([s]_I, [t]_I)$. Since $[s]_I = [[s]_I]_I$ and $[t]_I = [[t]_I]_I$, Lemma 5.1.7 implies that $s \approx [s]_I \in I_\infty$ and $t \approx [t]_I \in I_\infty$. We obtain a derivation tree Σ'' for $P(s,t)$ from Σ' by applying the binary substitutivity clauses of $E_{\mathcal{T}}$ twice. Therefore, $P(s,t)$ is derivable from $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$. □

By Lemma 5.1.8 we have that I_∞ is a Herbrand model of $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$, and by Lemma 5.1.9 we have that every fact $D \in I_\infty$ is derivable from $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$. Therefore, I_∞ is the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$. Lemma 5.1.10 follows from this fact, since by definition of I_∞, we have that $D([\vec{s}]_I) \in I_\infty$ if and only if $D([\vec{s}]_I) \in I$.

Lemma 5.1.10. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB, I the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}}$, and Q a CQ. We have that $[\vec{a}]_I \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $[\vec{a}]_I \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$.*

We are finally ready to show that $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \mathsf{ans}(Q, \mathcal{K})$. We recall Definition 5.1.5: given $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \{\langle a_{1,1}, ..., a_{1,m} \rangle, ..., \langle a_{n,1}, ..., a_{n,m} \rangle\}$, we have that $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \bigcup_{i=1}^{n} \times_{j=1}^{m} r_{a_{i,j}}$, where $r_{a_{i,j}}$ is the set containing $a_{i,j}$ and exactly every constant b such that $\langle a_{i,j}, b \rangle$ is contained in the transitive closure of $\mathsf{ans}(Q_{eq}(x, y) \leftarrow x \approx y, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$.

Lemma 5.1.11. *Let $\mathcal{K}$ be an $\mathcal{ELHIO}$ KB and Q a CQ. We have that $\mathsf{ans}(Q, \mathcal{K}) = \mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$.*

Proof. We first show that $\mathsf{ans}(Q, \mathcal{K}) \subseteq \mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. Suppose that $\langle a_1, ..., a_n \rangle$ is contained in $\mathsf{ans}(Q, \mathcal{K})$. It follows from the definition of $[\cdot]_I$ and Lemma 5.1.7 that $a_i \approx [a_i]_I \in I_\infty$. Since Lemma 5.1.8 and Lemma 5.1.9 imply that I_∞ is the minimal Herbrand model of $\Xi(\mathcal{K}) \cup E_{\mathcal{T}}$, we have that $\mathcal{K} \models a_i \approx [a_i]_I$. Therefore, $\langle a_1, ..., a_n \rangle \in \mathsf{ans}(Q, \mathcal{K})$ implies $\langle [a_1]_I, ..., [a_n]_I \rangle \in \mathsf{ans}(Q, \mathcal{K})$. By Lemma 5.1.10, we have that $\langle [a_1]_I, ..., [a_n]_I \rangle \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. It follows from the definition of $[\cdot]_I$ and Lemma 5.1.3 that for every b, if $[b]_I \neq b$, then there exist $\mathcal{O}$-constants $o_1, ..., o_m$ such that $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \models \{b \approx o_1, o_1 \approx o_2, ..., o_m \approx [b]_I\}$. Therefore, for each $r_{[a_i]_I}$ as defined in the definition of $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$, we have that $a_i \in r_{[a_i]_I}$. Hence, $\langle a_1, ..., a_n \rangle \in \mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$.

We now show that $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) \subseteq \mathsf{ans}(Q, \mathcal{K})$. Suppose that $\langle a_1, ..., a_n \rangle$ is contained in $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. As already noted, for every b, if $[b]_I \neq b$, then there exist $\mathcal{O}$-constants $o_1, ..., o_m$ such that $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \models \{b \approx o_1, o_1 \approx o_2, ..., o_m \approx [b]_I\}$. Therefore, it follows from the definition of $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ that

$$\langle a_1, ..., a_n \rangle \in \mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) \text{ implies } \langle [a_1]_I, ..., [a_n]_I \rangle \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}).$$

By Lemma 5.1.10, we have that $\langle [a_1]_I, ..., [a_n]_I \rangle \in \mathsf{ans}(Q, \mathcal{K})$. As already noted, we have that $\mathcal{K} \models a_i \approx [a_i]_I$; therefore, $\langle a_1, ..., a_n \rangle \in \mathsf{ans}(Q, \mathcal{K})$. □

5.2 CQ Rewriting

In this section we present RQR—a CQ rewriting algorithm for $\mathcal{ELHIO}^{\neg}$. We derive this algorithm from the answering algorithm presented in Section 5.1. Our goal is to obtain worst-case optimal rewritings for various sublanguages of $\mathcal{ELHIO}^{\neg}$—that is, if $\mathcal{T}$ is expressible in DL-Lite^{+}, then the rewriting should consist of a UCQ and a linear datalog program; and if $\mathcal{T}$ is expressible in DL-Lite$_R$, then the rewriting should be a UCQ.

RQR takes as input Q and $\mathcal{T}$, and proceeds in two steps: in the *saturation step*, the set $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup Q_C$ is saturated using $\mathcal{R}^{DL}$ and the resulting logic program is transformed into a datalog program $\mathsf{ff}(Q, \mathcal{T})$; and in the *unfolding step*, this datalog program is transformed into a DQ $\mathsf{RQR}(Q, \mathcal{T})$ of optimal form. In the rest of this section, we first show how to convert $(\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$ into a datalog program and then we present an optimization step to obtain rewritings of optimal form.

5.2.1 Elimination of Function Symbols

The following definition summarizes the first step of RQR.

Definition 5.2.1. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. With $\mathsf{ff}(Q, \mathcal{T})$ we denote the set that contains exactly every function-free clause contained in $(\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$. △

We now show that the DQ $\langle Q_P, \mathsf{ff}(Q, \mathcal{T}) \rangle$ is a rewriting of Q with respect to $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}}$, albeit not necessarily an optimal one. To this end, we first prove that in order to compute the answers to Q over an $\mathcal{ELHIO}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we can always "postpone" the inferences involving ground facts of the form $A(a)$ or $P(a, b)$ in a saturation of $\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C$.

Lemma 5.2.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and $Q = \langle Q_P, Q_C \rangle$ a CQ. For every clause C of type 1, if $C \in (\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$, then there is a clause C' of type 1 such that $C' \in (\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$ and there is a derivation tree for C from $\{C'\} \cup \mathcal{G}$, where $\mathcal{G}$ is the set containing exactly every clause of the form $A(a)$ or $P(a, b)$ contained in $(\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A})_{\mathcal{R}^{DL}}$.*

Proof. We prove the claim by induction on the height of a derivation tree of C. If the derivation tree has height zero, then C' is the only clause contained in Q_C, and the claim trivially follows. We assume that the claim holds for each clause derived by a derivation tree of height n, and consider a clause C derived by a derivation tree of height $n + 1$. Let C_1 and C_2 be the clauses that are resolved to obtain C. Without loss of generality we assume that C_1 is of type 1 and C_2 is of type 2. Hence, by the induction hypothesis, there is a clause C'_1 of type 1 such that $C'_1 \in (\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$ and C_1 can be derived from $\{C'_1\} \cup \mathcal{G}$.

The selection function S^{DL} selects a deepest covering body atom in C_1 and the head $D(\vec{s})$ of C_2, which is of the form $A(a)$, $A(f(x))$, $R(a, b)$, $R(a, f(x))$, $R(f(x), a)$, $R(x, f(x))$, or $R(f(x), x)$.

If $D(\vec{s})$ is ground, it follows from the definition of $\mathcal{R}^{DL}$ that the body of C_2 is empty. Given the fact that S^{DL} selects the deepest atoms in clauses of type 2, and that function symbols do not unify with constants, we have that every ground clause of type 2 has function-free premises. Therefore, we have that $C_2 \in \mathcal{G}$, and the claim follows for $C' = C'_1$.

We now consider the case where $D(\vec{s})$ is not ground. We show the claim for the case where $D(\vec{s})$ is of the form $R(\alpha, f(x))$, for α a constant a or the variable x; the proof for the other cases is analogous. Let C_2 be a clause of the form (5.54). Note that every clause of type 2 is safe and contains at most one variable; therefore, since S^{DL} selects the covering atoms of this type of clauses, we have that if $D(\vec{s})$ is not ground, then $C_2 \in (\Xi(\mathcal{T}) \cup E'_{\mathcal{T}})_{\mathcal{R}^{DL}}$.

There is a derivation tree for C_1 from $\{C_1'\} \cup \mathcal{G}$; therefore, given the fact that $\mathcal{G}$ contains only ground unit clauses, a set $\{D_k(\vec{a_k})\} \subseteq \mathcal{G}$ exists such that resolving C_1' on body atoms $\{D_k(\vec{r_k})\}$ with the atoms of $\{D_k(\vec{a_k})\}$ produces C_1. Furthermore, all such resolution inferences just remove body atoms; hence, since C_1 is to be resolved with C_2, the clause C_1' is of the form (5.55), and C_1 of the form (5.56), where δ maps the variables occurring in $\{D_k(\vec{r_k})\}$ to some constants occurring in $\{D_k(\vec{a_k})\}$. Note that no inference used to derive C_1 changes the number of function symbols of C_1'; therefore, since $R(s,t)\delta$ is deepest in C_1, we have that $R(s,t)$ is deepest in C_1'. Moreover, each variable of C_1' that is replaced by δ with a constant clearly does not occur in $R(s,t)\delta$; hence, the substitutions δ and σ have disjoint domains, and $\sigma = \delta\sigma$. Resolving C_1 and C_2 produces C, a clause of the form (5.57), where $\sigma = \mathsf{MGU}(R(s,t)\delta, R(\alpha, f(x)))$.

$$C_2 = \underline{R(\alpha, f(x))} \leftarrow \bigwedge A_i(t_i) \tag{5.54}$$

$$C_1' = Q_P(\vec{u}) \leftarrow \underline{R(s,t)} \wedge \bigwedge D_j(\vec{w_j}) \wedge \bigwedge D_k(\vec{r_k}) \tag{5.55}$$

$$C_1 = Q_P(\vec{u})\delta \leftarrow \underline{R(s,t)\delta} \wedge \bigwedge D_j(\vec{w_j})\delta \tag{5.56}$$

$$C = Q_P(\vec{u})\delta\sigma \leftarrow \bigwedge A_i(t_i)\sigma \wedge \bigwedge D_j(\vec{w_j})\delta\sigma \tag{5.57}$$

We now transform this derivation into a derivation in which all inferences with clauses in $\mathcal{G}$ are performed in the end. Let C' be the clause obtained by resolving C_1' and C_2. Given the form of C_1' and C_2, the clause C' is of the form (5.58), where $\sigma' = \mathsf{MGU}(R(s,t), R(\alpha, f(x)))$.

$$C' = Q_P(\vec{u})\sigma' \leftarrow \bigwedge A_i(t_i)\sigma' \wedge \bigwedge D_j(\vec{w_j})\sigma' \wedge \bigwedge D_k(\vec{r_k})\sigma' \tag{5.58}$$

Since $R(s,t)$ is deepest in C_1', the inference between C_1' and C_2 satisfies the selection function of $\mathcal{R}^{DL}$. Moreover, since both C_1' and C_2 are derivable from $\Xi(\mathcal{T}) \cup E_{\mathcal{T}}' \cup Q_C$, the clause C' is derivable from $\Xi(\mathcal{T}) \cup E_{\mathcal{T}}' \cup Q_C$ as well. Let D now be the clause obtained by resolving $\{D_k(\vec{r_k})\sigma'\}$ in C' with atoms of $\mathcal{G}$. The

clause D has form (5.59), where δ' maps some variables of C' to constants.

$$D = Q_P(\vec{u})\sigma'\delta' \leftarrow \bigwedge A_i(t_i)\sigma'\delta' \wedge \bigwedge D_j(\vec{w_j})\sigma'\delta' \tag{5.59}$$

We now prove that $C = D$. Given the forms of C and D, and since $\sigma = \delta\sigma$, it suffices to show that $\delta\sigma = \sigma'\delta'$.

We first assume that $\alpha = x$, so C_2 has head $R(x, f(x))$. Let y be a variable that occurs in $R(s,t)$ that is replaced by δ with a constant. Clearly, σ does not contain such y; therefore, without loss of generality we can assume that (*) if σ maps a variable z to a constant $y\delta$, then σ' maps z to y. Note that σ' may still map a variable y_1 occurring in δ to another variable y_2 occurring in δ, in which case we have that $y_1\delta = y_2\delta$. Due to (*), σ and σ' have the same domain modulo variables such as y_1. Even though $\{D_k(\vec{r_k})\}$ and $\{D_k(\vec{r_k})\sigma'\}$ are not necessarily the same (both y_1 and y_2 can occur in $\{D_k(\vec{r_k})\}$, while only y_2 can occur in $\{D_k(\vec{r_k})\sigma'\}$), note that since σ' maps y_1 to another *variable* y_2, the atoms $\{D_k(\vec{r_k})\sigma'\}$ can be resolved with the ground atoms $\{D_k(\vec{a_k})\}$ via δ'. So, we know that σ and σ' have the same domain modulo variables such as y_1; moreover, the only difference between δ and δ' is that δ maps both y_1 and y_2 to the same constant, while δ' does not contain a mapping for y_1; therefore, $\delta\sigma = \sigma'\delta'$.

We now assume that $\alpha = a$, so C_2 has head $R(a, f(x))$. We consider the forms of $R(s,t)\delta$. For the inference between C_1 and C_2 to be possible, the term $s\delta$ cannot be a functional term, and $t\delta$ cannot be a constant. Therefore, $R(s,t)\delta$ can be of the form (i) $R(x_s, x_t)$, (ii) $R(a, x_t)$, (iii) $R(x_s, f(t'\delta))$, or (iv) $R(a, f(t'\delta))$. Without loss of generality we assume that if $R(s,t)\delta$ is of form (i), then $\sigma = \{x_s \mapsto a, x_t \mapsto f(x)\}$; if it is of form (ii), then $\sigma = \{x_t \mapsto f(x)\}$; if it is of form (iii), then $\sigma = \{x_s \mapsto a, x \mapsto t'\delta\}$; and if it is of form (iv), then $\sigma = \{x \mapsto t'\delta\}$. Since $t\delta$ cannot be a constant, x_t is not in the domain of δ. In case x_s is not in the domain of δ either, without loss of generality we can assume that σ' and σ have the same domain. As can be seen, this domain is

disjoint from the domain of δ; therefore, we have that $\sigma = \sigma'\delta$ and $\delta = \delta'$. Therefore, $\delta\sigma = \sigma'\delta'$ since $\sigma = \delta\sigma$. In case x_s is in the domain of δ, we have that δ has to map x_s to a in order for the inference between C_1 and C_2 to be possible. Moreover, $R(s,t)\delta$ is of form (ii) or (iv). Without loss of generality we assume that σ and σ' have the same domain modulo x_s. Moreover, since x_s is the only variable of δ that is mapped to a term (namely, a) by σ', we have that δ and δ' are the same modulo the mapping concerning x_s. Furthermore, both δ and σ' map x_s to the same constant (namely, a); therefore, $\delta\sigma = \sigma'\delta'$. □

We now demonstrate the desired relationship between $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ and $\mathsf{ans}(\langle Q_P, \mathsf{ff}(Q,\mathcal{T})\rangle, \mathcal{A})$. By Theorem 5.1.1, it follows that $\vec{a} \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ if and only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$. By Lemma 5.2.1, we have that there is a clause $C' \in (\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup Q_C)_{\mathcal{R}^{DL}}$ of type 1, such that there is a derivation tree for $Q_P(\vec{a})$ from $\{C'\} \cup \mathcal{G}$. Since $Q_P(\vec{a})$ does not contain function symbols, C' does not contain function symbols either, so $C' \in \mathsf{ff}(Q,\mathcal{T})$. Hence, by the definition of $\mathcal{G}$, it follows that there is a derivation tree for $Q_P(\vec{a})$ from $\mathsf{ff}(Q,\mathcal{T}) \cup \mathcal{A}$. Lemma 5.2.2 follows immediately.

Lemma 5.2.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ be an $\mathcal{ELHIO}$ KB and $Q = \langle Q_P, Q_C\rangle$ a CQ. We have that $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \mathsf{ans}(\langle Q_P, \mathsf{ff}(Q,\mathcal{T})\rangle, \mathcal{A})$.*

According to Lemma 5.2.2, the DQ $\langle Q_P, \mathsf{ff}(Q,\mathcal{T})\rangle$ is a rewriting of Q with respect to $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}}$. We note, however, that $\mathsf{ff}(Q,\mathcal{T})$ is not necessarily optimal for DL-Lite$^+$. We illustrate the point with an example.

Example 5.2.1. Consider a DL-Lite$^+$ TBox $\mathcal{T}$ consisting of the following axioms:

$$\exists \text{hasFemaleAncestor}.\text{Jewish} \sqsubseteq \text{Jewish}$$

$$\text{hasMother} \sqsubseteq \text{hasFemaleAncestor}$$

The TBox $\mathcal{T}$ states that those who have a Jewish female ancestor are Jewish, and that anyone's mother is his/her female ancestor.

Since $\mathcal{T}$ does not contain $\mathcal{O}$-constants, then $E'_{\mathcal{T}} = \emptyset$; therefore $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}}$ consists of the following clauses:

$$\text{Jewish}(x) \leftarrow \underline{\text{hasFemaleAncestor}(x, y)} \wedge \text{Jewish}(y)$$
$$\text{hasFemaleAncestor}(x, y) \leftarrow \underline{\text{hasMother}(x, y)}$$

Consider the query Q: $Q_P(x) \leftarrow \underline{\text{Jewish}(x)}$. It is not difficult to verify that $\mathsf{ff}(Q, \mathcal{T}) = \Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup \{Q\}$. This is so because $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup \{Q\}$ does not contain function symbols and it is already saturated by $\mathcal{R}^{DL}$. In the case of DL-Lite^{+}, a worst-case optimal rewriting consists of a UCQ and a *linear* DQ [PUMH08a]. In this case, however, predicates Jewish and hasFemaleAncestor are both IDB predicates; therefore, the datalog program $\mathsf{ff}(Q, \mathcal{T}) \setminus \{Q\}$ is not linear. △

5.2.2 Optimizing the Rewriting

We now introduce a second step that can be used to transform the DQ $\langle Q_P, \mathsf{ff}(Q, \mathcal{T})\rangle$ into a DQ of optimal form. This is the second step of RQR and it is based on the notion of *unfolding* defined in Chapter 4. We define the rewriting in terms of $\mathsf{ff}(Q, \mathcal{T})$ as follows.

Definition 5.2.2. Let $Q = \langle Q_P, Q_C\rangle$ be a CQ, and $\mathcal{T}$ an $\mathcal{ELHIO}$ TBox. We define $\mathsf{RQR}(Q, \mathcal{T}) = \langle Q_P, \mathsf{unfold}(\mathsf{ff}(Q, \mathcal{T}), U)\rangle$, where U is the set that contains every clause $C \in \mathcal{N}^{DL}$ of one of the following forms:

$$A(x) \leftarrow B(x) \tag{5.60}$$
$$A(x) \leftarrow R(x, y) \tag{5.61}$$
$$A(x) \leftarrow R(y, x) \tag{5.62}$$
$$R(x, y) \leftarrow S(x, y) \tag{5.63}$$

$$R(x, y) \leftarrow S(y, x) \tag{5.64}$$

△

Example 5.2.2. Consider the DL-Lite^{+} TBox $\mathcal{T}$ and the query Q introduced in Example 5.2.1. It can be readily verified that given $\mathsf{RQR}(Q, \mathcal{T}) = \langle Q_P, Q_C \rangle$, Q_C consists of the following clauses:

$$Q_P(x) \leftarrow \text{Jewish}(x)$$
$$\text{Jewish}(x) \leftarrow \text{hasFemaleAncestor}(x, y) \wedge \text{Jewish}(y)$$
$$\text{Jewish}(x) \leftarrow \text{hasMother}(x, y) \wedge \text{Jewish}(y)$$

As can be seen, Q_C consists of a set of CQs (with only one element) and a linear datalog program. △

Theorem 5.2.1 follows from Lemma 5.2.2 and Lemma 4.4.3, since without loss of generality we can assume that Q_P does not occur in any of the clauses that are to be unfolded to obtain $\mathsf{RQR}(Q, \mathcal{T})$.

Theorem 5.2.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and Q a CQ. We have that $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}}) = \mathsf{ans}(\mathsf{RQR}(Q, \mathcal{T}), \mathcal{A})$.*

5.2.3 Form and Size of the Rewriting

We first consider the form of the rewriting. We show important properties of the structure of $\mathsf{RQR}(Q, \mathcal{T})$ that are used in Section 5.3 to prove complexity results about CQ answering in $\mathcal{ELHIO}^{\neg}$.

Lemma 5.2.3. *Let $\mathsf{RQR}(Q, \mathcal{T}) = \langle Q_P, Q'_C \rangle$ for a CQ $Q = \langle Q_P, Q_C \rangle$ and an $\mathcal{ELHIO}$ TBox $\mathcal{T}$. We have that (i) $\mathsf{RQR}(Q, \mathcal{T})$ is a DQ; (ii) if $\mathcal{T}$ is a DL-Lite^{+} TBox, then Q'_C consists of a set of CQs and a linear datalog program; and (iii) if $\mathcal{T}$ is a DL-Lite$_R$ TBox, then $\mathsf{RQR}(Q, \mathcal{T})$ is a UCQ.*

Proof. We first consider claim (i). It is immediate to see that, since $\mathsf{ff}(Q, \mathcal{T})$ does not contain function symbols, we have that $\mathsf{RQR}(Q, \mathcal{T})$ does not contain function symbols either. Therefore, the claim holds.

We now consider claim (ii). It follows from Table 5.1 and the definition of DL-Lite$^+$ that $E'_{\mathcal{T}} = \emptyset$, and $\Xi(\mathcal{T})$ only contains clauses of type 3.3, 3.4, or 2 of the form $B(x) \leftarrow A(x)$, $P(x, f(x)) \leftarrow A(x)$, or $B(f(x)) \leftarrow A(x)$. By analyzing the inferences shown in Table 5.3, one can see that saturating a set of clauses of these forms by $\mathcal{R}^{DL}$ produces only clauses of the same forms plus clauses of the form $A(x) \leftarrow B(f(x)) \wedge C(x)$. Therefore, $\mathsf{ff}(Q, \mathcal{T})$ contains only clauses of type 1 or of the form (5.60), (5.61), (5.63), or $A(x) \leftarrow R(x, y) \wedge B(y)$. Hence, the datalog program Q'_C only contains clauses of type 1, and of the form $A(x) \leftarrow R(x, y) \wedge B(y)$ or $A(x) \leftarrow R(x, y) \wedge S(y, z)$ (which are obtained by unfolding a clause of the form (5.61) into a clause of the form $A(x) \leftarrow R(x, y) \wedge B(y)$). Clearly, every binary predicate in Q'_C is an EDB predicate. Moreover, even though there could be unary IDB predicates in Q'_C, we have that there can be at most one unary atom in the clauses of Q'_C which are not of type 1. Therefore Q'_C consists of a set of CQs and a linear datalog program.

We finally consider claim (iii). It follows from Table 5.1 and the definition of DL-Lite$_R$ that $E'_{\mathcal{T}} = \emptyset$. Moreover, $\Xi(\mathcal{T})$ only contains clauses of type 3.1 with one body atom, 3.2 with one body atom, 3.3, 3.4, or 2 of the forms $B(x) \leftarrow A(x)$, $P(x, f(x)) \leftarrow A(x)$, $P(f(x), x) \leftarrow A(x)$, or $B(f(x)) \leftarrow A(x)$. By analyzing the inferences shown in Table 5.3, one can see that saturating a set of clauses of these forms by $\mathcal{R}^{DL}$ produces only clauses of the same forms. Therefore, $\mathsf{ff}(Q, \mathcal{T})$ contains only clauses of type 1 or of the form (5.60)–(5.64). Clearly, the datalog program Q'_C only contains clauses of type 1; hence, the claim follows. □

We now provide estimates on the maximal number of clauses generated by RQR and the maximal size of $\mathsf{RQR}(Q, \mathcal{T})$.

Lemma 5.2.4. *Let $\mathcal{T}$ be an $\mathcal{ELHIO}$ TBox and Q a CQ. We have that (i) the maximal number M of clauses generated by* RQR *is $O(2^{t^q})$ and (ii) the size $|\mathsf{RQR}(Q,\mathcal{T})|$ of $\mathsf{RQR}(Q,\mathcal{T})$ is $O(2^{tq})$, where $t = |\mathcal{T}|$ and $q = |Q|$.*

Proof. We first consider claim (i). Clearly, we have that $M = |\mathcal{N}^{DL}| + F$, where $|\mathcal{N}^{DL}|$ is the size of the clause set and F is the maximal number of clauses produced in the unfolding step. We provide upper bounds for $|\mathcal{N}^{DL}|$ and F.

We first consider $|\mathcal{N}^{DL}|$. We analyze clauses of type 1 since they are the longest and deepest clauses in $\mathcal{N}^{DL}$. Let p be the number of predicate names occurring in $\Xi(\mathcal{T})$, f the number of constants and function symbols occurring in $\Xi(\mathcal{T})$, v a bound on the number of variables for the clauses of $\Xi(\mathcal{T})$, d a bound on the maximal depth for the clauses of $\Xi(\mathcal{T})$, and b be the number of body atoms of Q_C. It can be readily verified that it is possible to build pf^dv unary atoms and $p(f^dv)^2$ binary atoms with the symbols of $\Xi(\mathcal{T})$. For clauses of type 1, however, both d and v are bounded by $\mathsf{varNo}(Q_C)$ and consequently by b. Therefore, we can build no more than $U = pf^bb$ unary and $B = p(f^bb)^2$ binary atoms for clauses of type 1. Moreover, it is easy to see that we can build no more than $H = (f^bb)^a$ head atoms for clauses of type 1, where a is the arity of Q_C.

We now determine the *length* of the longest clause of type 1—that is, we determine the maximal number of *occurrences* of unary and binary body atoms in any clause of type 1. The number of unary body atoms of a clause of type 1 may be augmented by resolving it with a clause of type 2; however, no inference augments the number of binary atoms, which are bounded by b. Therefore, the longest clause C in $\mathcal{N}^{DL}$ contains no more than one head atom, U unary body atoms and b binary body atoms.

Given the possible atoms that can be built for clauses of type 1 and the length of the longest clause of type 1, it can be readily verified that there can be no more than $H2^UB^b$ clauses of type 1 in $\mathcal{N}^{DL}$. Note that p and f depend on $\mathcal{T}$, and a and b depend on Q. Let $t = |\mathcal{T}|$ and $q = |Q|$. We have that $H = O((t^qq)^q)$, $U = O(tt^qq)$,

and $B = O((t(t^q q)^2)^q)$; therefore, the number of clauses of type 1 contained in $\mathcal{N}^{DL}$ is

$$O((t^q q)^q 2^{tt^q q}(t(t^q q)^2)^q) = O(t^{3q^2+q} q^{3q} 2^{qt^{q+1}}) = O(2^{t^q}).$$

Since clauses of type 1 are the longest and deepest clauses in $\mathcal{N}^{DL}$, we have that $|\mathcal{N}^{DL}| = O(2^{t^q})$.

We now consider F. Note that only the function-free clauses of $\mathcal{N}^{DL}$ are unfolded; therefore, we first need to determine the size $|\mathsf{ff}(Q, \mathcal{T})|$ of $\mathsf{ff}(Q, \mathcal{T})$. We do so by analyzing the number of possible clauses of type 1 in $\mathcal{N}^{DL}$ with a depth equal to 0. For such clauses, we can build no more than $H_0 = (fb)^a$ head atoms, $U_0 = pfb$ unary body atoms, and $B_0 = p(fb)^2$ binary body atoms. The length of the longest clause of type 1 with depth 0 is the same as before; therefore, there can be no more than $H_0 2^{U_0} {B_0}^b$ clauses of type 1 in $\mathsf{ff}(Q, \mathcal{T})$. Since p and f depend on t, and a and b depend on q, we have that $H_0 = O((tq)^q)$, $U_0 = O(ttq)$, and $B_0 = O((t(tq)^2)^q)$; therefore, the number of clauses of type 1 contained in $\mathsf{ff}(Q, \mathcal{T})$ is

$$O((tq)^q 2^{t^2 q}(t(tq)^2)^q) = O(t^{4q} q^{3q} 2^{t^2 q}) = O(2^{tq}).$$

The unfolding step may augment the number of binary body atoms in clauses of type 1. Each unary atom of a clause $C \in \mathsf{ff}(Q, \mathcal{T})$ can be replaced with a binary atom. Therefore, for every such C, the unfolding step produces no more than p^{U_0} clauses. Hence, the number of clauses of type 1 produced in the unfolding step is the number of clauses of this type contained in $\mathsf{ff}(Q, \mathcal{T})$ multiplied by p^{U_0}. Consequently, the number of clauses of type 1 produced in the unfolding step is

$$O(2^{tq} t^{ttq}) = O(2^{tq}).$$

Since clauses of type 1 are the longest and deepest clauses produced in the unfolding step, we have that $F = O(2^{tq})$.

We conclude that the number of clauses M generated by our rewriting algorithm is

$$O(|\mathcal{N}^{DL}| + F) = O(2^{t^q} + 2^{tq}) = O(2^{t^q}).$$

We now consider claim (ii). The number of clauses of type 1 contained in the rewriting $\mathsf{RQR}(Q, \mathcal{T})$ is the number of clauses of this type contained in $\mathsf{ff}(Q, \mathcal{T})$ plus the new clauses of this type produced in the unfolding step, that is

$$O(2^{tq} + 2^{tq}) = O(2^{tq}).$$

Since clauses of type 1 are the longest and deepest clauses in $\mathsf{ff}(Q, \mathcal{T})$ and in the set of clauses produced in the unfolding step, we have that $|\mathsf{RQR}(Q, \mathcal{T})| = O(2^{tq})$. □

Note that according to Lemma 5.2.4, even though the number of clauses generated by our algorithm is doubly exponential with respect to $|\mathcal{T}|$ and $|Q|$, the number of clauses in the computed rewriting $\mathsf{RQR}(Q, \mathcal{T})$ is only exponential with respect to $|\mathcal{T}|$ and $|Q|$.

5.3 Data Complexity

CQ answering is typically considered an important reasoning task in scenarios where the size of the ABox is considerably larger than the size of the TBox. In this kind of scenario, it is usual to measure the complexity of CQ answering with respect to the size of the ABox only. This notion of complexity—called *data complexity*—was first identified by Vardi [Var82] in the context of relational databases.

We are interested in determining the data complexity of answering CQs with respect to $\mathcal{ELHIO}^{\neg}$ KBs. In order to make this notion precise, we recall the CQ entailment decision problem for $\mathcal{ELHIO}^{\neg}$—the problem of deciding whether a tuple of constants $\vec{a}$ is in $\mathsf{ans}(Q, \mathcal{K})$ for a CQ Q and an $\mathcal{ELHIO}^{\neg}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. When we say that CQ answering for $\mathcal{ELHIO}^{\neg}$ is in a complexity class $\mathcal{C}$ with respect to data

complexity, we mean that the corresponding CQ entailment problem is in $\mathcal{C}$ assuming that both Q and $\mathcal{T}$ are fixed. Similarly, when we say that CQ answering for $\mathcal{ELHIO}^{\neg}$ is $\mathcal{C}$-hard (respectively, $\mathcal{C}$-complete) with respect to data complexity, we mean that the corresponding CQ entailment problem is $\mathcal{C}$-hard (respectively, $\mathcal{C}$-complete) assuming that both Q and $\mathcal{T}$ are fixed.

In the rest of the section we analyze the data complexity of CQ answering for $\mathcal{ELHIO}^{\neg}$ and we show that RQR produces optimal rewritings with respect to data complexity for various sublanguages of $\mathcal{ELHIO}^{\neg}$. Additionally, we present upper bounds concerning complexity measured with respect to the TBox only, the query only, and all inputs combined.

Theorem 5.3.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and Q a CQ. Deciding whether $\vec{a} \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ is* PTIME*-complete with respect to $|\mathcal{A}|$.*

Proof. It has been shown in [CGL+06] that the problem of instance checking is already PTIME-hard if the language in which $\mathcal{K}$ is modeled allows for TBox axioms of the form $\exists P.A \sqsubseteq B$ and $A \sqcap B \sqsubseteq C$. Moreover, it is well known that, given a datalog program DP, deciding whether $DP \models P(\vec{a})$ can be done in PTIME with respect to the number of facts in DP [DEGV97]. According to Lemma 5.2.3 we have that $\mathsf{RQR}(Q, \mathcal{T})$ is a DQ; therefore, since its size does not depend on $\mathcal{A}$, deciding whether $\vec{a} \in \mathsf{ans}(\mathsf{RQR}(Q, \mathcal{T}), \mathcal{A})$ is PTIME-complete with respect to $|\mathcal{A}|$. The claim follows from this fact since Theorem 5.2.1 holds. □

Verifying whether an $\mathcal{ELHIO}^{\neg}$ $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable can be reduced to answering a set of queries (that depend on $\mathcal{T}$) over $\langle \mathcal{T}_{\text{PI}}, \mathcal{A} \rangle$ in the same vein as described in Chapter 4 for DL-Lite$_R$. Therefore, Corollary 5.3.1 follows from Theorem 5.3.1.

Corollary 5.3.1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}^{\neg}$ KB. Verifying whether $\mathcal{K}$ is unsatisfiable can be done in* PTIME *with respect to $|\mathcal{A}|$.*

Moreover, since negative inclusions are not relevant for CQ answering in case an $\mathcal{ELHIO}^{\neg}$ KB is satisfiable, Corollary 5.3.2 also follows from Theorem 5.3.1.

Corollary 5.3.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable $\mathcal{ELHIO}^{\neg}$ KB and Q a CQ. Deciding whether $\vec{a} \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ is* PTime*-complete with respect to $|\mathcal{A}|$.*

It is well-known that adding other common DL features to $\mathcal{ELHIO}^{\neg}$, such as disjunction or qualified universal quantification, would cause CQ answering to become coNP-hard with respect to data complexity [CGL+06]. Therefore, $\mathcal{ELHIO}^{\neg}$ is one of the most expressive languages for which CQ answering remains tractable with respect to data complexity.

We now consider the optimality of the rewriting. Before proceeding, we show that a datalog program DP consisting of a set of CQs and a linear datalog program can be evaluated in NLogSpace with respect to to the number of facts in DP.

Lemma 5.3.1. *Let $Q = \langle Q_P, Q_C \rangle$ be a UCQ and DP a linear datalog program. Deciding whether $DP \cup Q_C \models Q_P(\vec{a})$ can be performed in* NLogSpace *with respect to $|A|$, where A is the set of facts contained in DP.*

Proof. If $DP \cup Q_C \models Q_P(\vec{a})$, then $Q_P(\vec{a})$ can be derived from the set of clauses $DP \cup Q_C$ using SLD resolution [BG01]. First, we nondeterministically choose a query $Q_i \in Q_C$ and ground it by nondeterministically choosing a set of constants from A. We then initialize the *goal* G to be the resolvent of Q_i and $\leftarrow Q_P(\vec{a})$; if resolution is not possible, the algorithm halts. Then, we start the following loop. We first eliminate all atoms with EDB predicates in G by resolving them with facts in A; if some atom cannot be resolved, the algorithm halts. If G has an empty body, the algorithm accepts. Otherwise, we nondeterministically choose a rule $R \in DP$ and generate its grounding R' by nondeterministically choosing a set of constants from A. Finally, we set our goal G to be the resolvent between R' and G; if this is not possible, the algorithm halts. We now repeat the loop. To ensure termination, we

maintain a counter that is initialized in the beginning to the number of ground clauses of DP and A multiplied by the number of the queries in Q_C. We decrease the counter after each pass through the loop, and we terminate the loop if the counter reaches zero. Clearly, if the algorithm accepts, then SLD resolution for $Q_P(\vec{a})$ from $DP \cup Q_C$ exists. Conversely, if an SLD resolution exists, then we can assume that each ground instance of a rule is used only once, so an accepting run of our algorithm exists. It is possible to make such an assumption since without loss of generality we can assume that every ground fact occurs on each branch of a derivation tree only once: multiple occurrences of a ground fact would clearly lead to unnecessary cyclic computations.

Since we are interested in determining the data complexity of this procedure, the number of predicates p and their arity r is bounded. Hence, if A contains c constants, we can describe each ground atom in $p \cdot r \cdot \lceil \log(c) \rceil$ bits. The number of atoms in G depends on the number of clauses in $DP \setminus A \cup Q_C$, so storing G requires $k_1 \lceil \log(c) \rceil$ bits for k_1 a constant that does not depend on c. Finally, the number of ground clauses depends polynomially on c, so we can store the counter using $k_2 \lceil \log(c) \rceil$ bits for k_2 a constant that does not depend on c. Clearly, the algorithm requires $k \lceil \log(c) \rceil$ bits of space in total for k a constant that does not depend on c. The algorithm is nondeterministic, so it can be implemented in NLogSpace. □

Since negative inclusions are not taken into account by our rewriting algorithm, it follows from Lemma 5.2.3 that if $\mathcal{T}$ is an $\mathcal{ELHIO}^{\neg}$ TBox, then $\mathsf{RQR}(Q, \mathcal{T})$ is a DQ; if $\mathcal{T}$ is a DL-Lite^{+} TBox, then $\mathsf{RQR}(Q, \mathcal{T})$ consists of a UCQ and a linear datalog program; and if $\mathcal{T}$ is a DL-Lite$_R$ TBox, then $\mathsf{RQR}(Q, \mathcal{T})$ is a UCQ. A DQ can be evaluated in PTime with respect to data complexity [DEGV97]; a UCQ with a linear datalog program, in NLogSpace with respect to data complexity (Lemma 5.3.1); and a UCQ, in AC^0 with respect to data complexity [AHV95]. Therefore, our rewriting algorithm not only deals with various DLs from DL-Lite$_{core}$ up to $\mathcal{ELHIO}^{\neg}$, but it is worst-case optimal with respect to data complexity for all these logics.

Even though data complexity is the most significant complexity measure in scenarios where the size of the ABox dominates, complexity of CQ answering can also be measured with respect to the TBox only, the query only, and all inputs combined. Upper bounds concerning these types of complexity may be derived from our rewriting algorithm. We present these complexity results in Lemma 5.3.2.

Lemma 5.3.2. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{ELHIO}$ KB and Q a CQ. Deciding whether $\vec{a} \in \mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ is in 2*EXPTIME *with respect to $|\mathcal{T}|$, with respect to $|Q|$, and with respect to $|\mathcal{K}| + |Q|$.*

Proof. According to Lemma 5.2.4, the maximal number of clauses produced by our algorithm is $O(2^{t^q})$. Therefore, computing $\mathsf{RQR}(Q, \mathcal{T})$ takes an exponential number of steps with respect to $|\mathcal{T}|$, and a doubly exponential number of steps with respect to $|Q|$, and with respect to $|\mathcal{K}| + |Q|$. It follows from [DEGV97] that $\mathsf{RQR}(Q, \mathcal{T})$ can be evaluated over $\mathcal{A}$ in an exponential number of steps with respect to $|\mathsf{RQR}(Q, \mathcal{T})|$. By Lemma 5.2.4, we have that $|\mathsf{RQR}(Q, \mathcal{T})| = O(2^{qt})$. Therefore, evaluating $\mathsf{RQR}(Q, \mathcal{T})$ over $\mathcal{A}$ can be done in a doubly exponential number of steps with respect to $|\mathcal{T}|$, with respect to $|Q|$, and with respect to $|\mathcal{K}| + |Q|$. The claim follows from these facts since Theorem 5.2.1 holds. □

We believe that we can tighten the upper complexity bounds stated by Lemma 5.3.2 by adjusting the definition of $\mathcal{N}^{DL}$ to make it as small as possible. Nevertheless, our complexity analysis suggests that our rewriting-based technique is more likely to be useful in practice when dealing with relatively small TBoxes and queries. It is reasonable to assume that queries are small in practice; moreover, it is quite common for applications dealing with very large datasets to use relatively small schemas. Therefore, we expect our rewriting-based CQ answering technique to be useful in practice for many typical scenarios.

5.4 Chapter Summary

In this chapter we presented answering and rewriting algorithms for $\mathcal{ELHIO}^{\neg}$. The main difficulty of CQ answering for $\mathcal{ELHIO}^{\neg}$ is the need for reasoning with equality. In Section 5.1 we presented a novel way of reasoning with equality; our approach is based on the notion of the so-called approximation of equality $E'_{\mathcal{T}}$—a set of clauses that approximates the characteristics of the equality relation.

In Section 5.1 we presented a resolution-based CQ answering algorithm. The algorithm takes as input a CQ Q and an $\mathcal{ELHIO}^{\neg}$ KB $\mathcal{K}$ and computes the set of certain answers $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. The algorithm produces these answers by saturating the set of clauses corresponding to Q, $\mathcal{K}$, and $E'_{\mathcal{T}}$ using the suitably parameterized RFS calculus $\mathcal{R}^{DL}$. We showed that it is possible to compute $\mathsf{ans}(Q, \mathcal{K})$ from $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ and presented a procedure to do so.

In Section 5.2 we modified the answering algorithm in order to derive the CQ rewriting algorithm RQR. The algorithm takes as input a CQ Q and an $\mathcal{ELHIO}^{\neg}$ TBox $\mathcal{T}$ and computes a DQ $\mathsf{RQR}(Q, \mathcal{T})$ that is a rewriting of Q with respect to $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}}$. The algorithm produces the rewriting in two steps: the set of clauses corresponding to Q, $\mathcal{T}$, and $E'_{\mathcal{T}}$ is saturated by $\mathcal{R}^{DL}$ and then the resulting set is transformed into a DQ by pruning and unfolding clauses of certain types. As shown in Section 5.2.3, RQR exhibits "pay-as-you-go" behavior: if $\mathcal{T}$ is expressible in $\mathcal{ELHIO}^{\neg}$, then $\mathsf{RQR}(Q, \mathcal{T})$ is a DQ; if $\mathcal{T}$ is expressible in DL-Lite^{+}, then $\mathsf{RQR}(Q, \mathcal{T})$ consists of a UCQ and a linear datalog program; and if $\mathcal{T}$ is expressible in DL-Lite$_R$, then $\mathsf{RQR}(Q, \mathcal{T})$ is a UCQ.

Finally, in Section 5.3 we presented a data complexity analysis. The results presented in this section lead to the novel result that CQ answering over satisfiable $\mathcal{ELHIO}^{\neg}$ KBs is PTime-complete with respect to data complexity. Moreover, based of the form of the rewritings produced by RQR, we showed that this algorithm is optimal with respect to data complexity for various DLs ranging from DL-Lite$_{core}$

up to $\mathcal{ELHIO}^{\neg}$, which makes it an extension and a generalization of the existing approaches.

Chapter 6

Related Work

As briefly discussed in Chapter 1, the development of RQR was motivated by existing resolution-based approaches to reasoning with DLs. In this chapter, we recapitulate these approaches and, additionally, we describe existing work related to the problems of CQ answering and CQ rewriting for DLs.

6.1 Resolution-Based Decision Procedures for DLs

The direct correspondence between the DL $\mathcal{ALC}$ [BCM+03] and a multi-modal version of the modal logic K [BBW06], first observed by Schild [Sch91], opened the possibility of reasoning in DLs using first-order logic techniques. Based on the work of Joyner [Joy76], for instance, Fermüller et al. [FTZL93] proposed various resolution-based decision procedures for fragments of first-order logic, including the fragment that corresponds to the translation of $\mathcal{ALC}$. These and other related approaches have been implemented in the system MSPASS [HS00]—an extension of the general-purpose theorem prover SPASS [WSH+07] that provides various translations for modal and DL formulae. MSPASS was primarily designed to handle modal logics; nevertheless, it can also be used for checking the satisfiability of $\mathcal{ALC}$ concepts.

More recently, Kazakov, Motik, and Krötzsch et al. have studied the application of resolution-based approaches for solving various DL reasoning problems, such as concept satisfiability and instance checking. Kazakov [Kaz06] applied the resolution

framework of Bachmair and Ganzinger [BG90, BG94, BG01] to derive a range of decision procedures for various inference problems for the $\mathcal{EL}$ family of languages [BBL05]. His procedures use standard resolution-based calculi enhanced with simplification rules based on the general notion of redundancy [BG01]. Motik [Mot06] studied the relationship between very expressive DLs and deductive databases [CGT90]. His main contribution is a resolution-based algorithm that reduces a $\mathcal{SHIQ}$(D) KB $\mathcal{K}$ to a disjunctive datalog program [CGT90] $\mathsf{DD}(\mathcal{K})$ such that $\mathcal{K}$ and $\mathsf{DD}(\mathcal{K})$ entail the same set of ground facts. Thus, the standard inference problems over $\mathcal{K}$ can be solved over $\mathsf{DD}(\mathcal{K})$ using deductive database techniques. Motik's reduction algorithm has been implemented in the system KAON2.[1] Similar to the work of Motik, Krötzsch et al. proposed a reduction of ELP KBs to datalog programs [KRH08]. ELP is a rule-based tractable knowledge representation language that generalizes the well–known tractable DLs $\mathcal{EL}^{++}$ [BBL05] and DLP [GHVD03].

The main difference between our work and the aforementioned approaches is that none of them considers the problem of CQ answering.

6.2 CQ Answering for DLs

The problem of CQ answering for very expressive DLs of the $\mathcal{SH}$ family [HST99] has been considered by Hustadt et al., Glimm, Ortiz et al., and Dolby et al. Hustadt et al. [HMS05] proposed a resolution-based algorithm for answering CQs with simple roles only for the DL $\mathcal{SHIQ}$. Their algorithm is based on a novel inference rule so-called decomposition. Glimm [Gli07] presented CQ answering algorithms for the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$. The algorithm for $\mathcal{SHIQ}$ has been used to show that the data complexity of CQ answering for $\mathcal{SHIQ}$ is coNP-complete. Ortiz et al. [OCE08] proposed a CQ answering algorithm for the DLs $\mathcal{SHIQ}$, $\mathcal{SHOQ}$, and $\mathcal{SHOI}$ for CQs without transitive roles. The algorithm has been used to show that for each DL of the

[1] http://kaon2.semanticweb.org/

$\mathcal{SH}$ family except $\mathcal{SHOIQ}$, answering CQs without transitive roles is coNP-complete with respect to data complexity. A decidability result for CQ answering without transitive roles for $\mathcal{SHOIQ}$ has been recently presented by Glimm and Rudolph [GR09]. Dolby et al. [DFK+08] considered the problem of answering CQs without undistinguished variables for the DL $\mathcal{SHIN}$. Their algorithm has been implemented in the reasoner SHER.[2]

The problem of CQ answering for less expressive DLs has been considered by Krisnadhi and Lutz, Krötzsch and Rudolph, and Eiter et al. Krisnadhi and Lutz [KL07] studied the data complexity for various extensions of the DL $\mathcal{EL}$. In particular, they proposed an algorithm for CQ answering for $\mathcal{ELH}^f$. Their algorithm is used to show that the data complexity of CQ answering in this setting is PTime-complete. Krötzsch and Rudolph [KR07] studied the problem of CQ answering with role composition over various extensions of the DL $\mathcal{EL}$. In particular, they proposed an algorithm for a decidable fragment of $\mathcal{EL}^{++}$, for which CQ answering is PTime-complete with respect to data complexity. Eiter et al. [EGOv08] presented a CQ answering algorithm for the DL Horn-$\mathcal{SHIQ}$—a fragment of $\mathcal{SHIQ}$ which, analogously to Horn logic, does not allow to represent disjunctive knowledge. Their algorithm is used to show that the data complexity of CQ answering in this setting is PTime-complete.

The main difference between our work and the aforementioned approaches is that they do not consider the problem of CQ rewriting for DLs.

6.3 CQ Rewriting for DLs

The problem of CQ rewriting for DLs has been considered by Calvanese et al. and Rosati. Notably, the evaluation of the rewritings produced by these approaches can be delegated to existing (deductive) database systems, thus taking advantage of the query optimization strategies provided by such systems.

[2] http://www.alphaworks.ibm.com/tech/sher

Calvanese et al. [CDL+07] proposed CGLLR—a CQ rewriting algorithm for the DL-Lite family of languages—and used it to show that the data complexity of CQ answering in this setting is in AC^0. As described in Section 4.5, CGLLR takes as input a CQ Q and a DL-Lite$_R$ TBox $\mathcal{T}$, and produces a UCQ that is a rewriting of Q with respect to $\mathcal{T}$. This algorithm has been implemented in systems such as QuOnto, Owlgres, and ONTOSEARCH2. A minor extension of CGLLR was recently presented by Corona et al. [CRS09]. The main difference with respect to our approach is that both of these algorithms consider DLs of the DL-Lite family only.

Rosati [Ros07] considered the problem of CQ rewriting for $\mathcal{ELH}$ [BBL05]. He proposed an algorithm that transforms a CQ Q and a $\mathcal{ELH}$ TBox $\mathcal{T}$ into a DQ that is a rewriting of Q with respect to $\mathcal{T}$. Rosati's algorithm was used to show that the data complexity of CQ answering for $\mathcal{ELH}$ is PTIME-complete. The main difference with respect to our approach is that Rosati does not consider DLs including inverse roles, nominals and negation such as $\mathcal{ELHIO}^\neg$. Moreover, unlike our technique, his rewriting technique is not worst-case optimal for DL-Lite$^+$ or DL-Lite$_R$.

Finally, Lutz et al. [LTW09] proposed a nonstandard CQ rewriting technique for various DLs of the $\mathcal{EL}$ family of languages [BBL05]. The central idea of their approach is to incorporate the consequences of the TBox into the ABox to have as much explicit information as possible. Notably, their algorithm enables the use of standard relational database systems as the basis for query execution. The main difference with respect to our approach is that Lutz et al. do not consider the standard problem of CQ rewriting for DLs. In fact, while our approach tries to rewrite the query with respect to the TBox only, the approach of Lutz et al. tries to rewrite both the query *and* the ABox with respect to the TBox. Moreover, unlike us, Lutz et al. do not consider DLs including nominals such as $\mathcal{ELHIO}^\neg$.

6.4 CQ Answering over Incomplete Databases

There is a close relation between answering queries for DLs and incomplete databases. An incomplete database is defined by a set of constraints and a partial database instance (usually defined as a set of sound views) [AHV95]. In analogy to DLs, the set of constraints can be seen as a TBox and the partial database instance as some ABox. The literature on the subject is vast (to mention just a few, see [CK06,KLW95, SE07, GL88, CLPS08, Cal07, FM07, CGK08, CGL09]); however, unlike our approach, none of the several existing approaches considers DL constraints.

6.5 Chapter Summary

In this chapter we presented existing work related to the problems of CQ answering and rewriting for DLs, and the use of resolution as the basis for decision procedures over KBs. As discussed in this chapter, our contributions—CQ answering and rewriting algorithms for $\mathcal{ELHIO}^{\neg}$—differ from existing approaches in various aspects.

We presented various resolution-based approaches to reasoning with DLs in Section 6.1; unlike these approaches, we consider the problem of CQ answering for DLs. Existing approaches to solving the problem of CQ answering for DLs have been presented in Section 6.2; none of such approaches, however, considers the problem of CQ rewriting for DLs. As shown in Section 5.3, our CQ rewriting algorithm exhibits "pay-as-you-go" behavior; therefore, it can be seen as an extension and a generalization of the standard rewriting approaches presented in Section 6.3. Finally, unlike the approaches to CQ answering for incomplete databases presented in Section 6.4, we consider the problem of CQ answering for DL constraints.

Part III

Practical Considerations

Chapter 7

Query Rewriting in Practice

As discussed in Section 5.2, RQR takes as input a CQ Q and an $\mathcal{ELHIO}^{\neg}$ TBox $\mathcal{T}$ and produces a DQ $\mathsf{RQR}(Q, \mathcal{T})$ that is a rewriting of Q with respect to $\mathcal{T}$. Clearly, given the form of the rewritings produced by RQR, it is possible to evaluate them using existing (deductive) database technology. Unfortunately, as shown in Example 7.0.1, the size of $\mathsf{RQR}(Q, \mathcal{T})$ is worst-case exponential with respect to the size of Q and $\mathcal{T}$ even for the rather inexpressive DL DL-Lite$_R$.

Example 7.0.1. Consider the following DL-Lite$_R$ TBox

$$\mathcal{T} = \{R_1 \sqsubseteq R_2, R_2 \sqsubseteq R_3, ..., R_{n-1} \sqsubseteq R_n\}$$

and the query $Q = Q_P(x_0) \leftarrow R_n(x_0, x_1) \wedge R_n(x_1, x_2) \wedge ... \wedge R_n(x_{m-1}, x_m)$ posed over $\mathcal{T}$. It can be readily verified that that $\mathsf{RQR}(Q, \mathcal{T})$ contains n^m queries. △

Example 7.0.1 shows that queries containing concepts or roles with many subsumees can lead to large rewritings. Moreover, the example implies that CQ answering even for DL-Lite$_R$ is NP-hard with respect to the size of Q and $\mathcal{T}$ (in fact, it is NP-complete for DL-Lite$_R$ [CDL+07]).

Given the potentially large size of the rewritings, on the one hand $\mathsf{RQR}(Q, \mathcal{T})$ may be costly to compute and on the other hand evaluation by existing (deductive) database systems may be costly or even unfeasible. Trying to produce small rewrit-

ings is therefore of critical importance to the practical application of the rewriting approach to CQ answering.

In this chapter, we first explain how to use RQR to answer CQs using existing (deductive) database systems in Section 7.1. Then, in Section 7.2, we introduce an alternative unfolding for RQR that can be used to obtain UCQs even for many TBoxes that go beyond DL-Lite$_R$. In Section 7.3, we present various optimizations aimed at reducing the size of the rewritings produced by RQR. Finally, in Section 7.4, we describe REQUIEM—a prototypical implementation of RQR that implements various of the optimizations described in this chapter. We shall use REQUIEM in order to test the practicality of our rewriting approach; we present the results of an empirical evaluation of REQUIEM in Chapter 8.

7.1 Answering Queries using RQR

In this section we describe how to use RQR in order to compute the certain answers $\mathsf{ans}(Q, \mathcal{K})$ of a CQ Q with respect to a satisfiable $\mathcal{ELHIO}^{\neg}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$.[1] We assume that the data represented by $\mathcal{A}$ has been stored in a relational database.[2] We illustrate the process by means of an example.

Example 7.1.1. Suppose we have a relational database DB containing a table `Professor` with attributes `name`, `department`, and `telephone`; and a table `Student` with attributes `name`, `major`, `address`, and `tutor`. Additionally, suppose we have the following TBox $\mathcal{T}$ as a conceptual schema over DB:

$$\text{Teacher} \sqsubseteq \exists \text{teaches}$$

$$\text{Professor} \sqsubseteq \text{Teacher}$$

[1] The process described in this section actually refers to computing $\mathsf{ans}(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$. In case $\mathcal{K}$ contains $\mathcal{O}$-constants, it is necessary to compute $\mathsf{ans}'(Q, \Xi(\mathcal{K}) \cup E'_{\mathcal{T}})$ as defined in Definition 5.1.5 to guarantee that all answers are retrieved.

[2] Note that this approach can be generalized to having multiple databases by using a data federation tool.

$$\exists \text{hasTutor}^- \sqsubseteq \text{Professor}^*$$

The TBox $\mathcal{T}$ states that teachers teach at least someone, that professors are teachers, and that the range of the role hasTutor is Professor.

In order to answer queries posed to $\mathcal{T}$ using DB, it is necessary to define a set of *mappings* $\mathcal{M}$ from the concepts and roles in $\mathcal{T}$ to data in DB. Mappings from the TBox to the database are typically defined using expressions of the form $D \mapsto Q_D$, where D is a concept or role occurring in the TBox and Q_D is an SQL query over the database. This type of mappings is known as Global-as-View or GaV. A detailed discussion on the different types of mappings is out of the scope of this thesis; the interested reader is referred to [Len02]. In our example, the set of mappings $\mathcal{M}$ between $\mathcal{T}$ and DB is defined as follows:

$$\text{Professor} \mapsto \texttt{SELECT Name FROM Professor}$$

$$\text{hasTutor} \mapsto \texttt{SELECT Name, Tutor FROM Student}$$

The first step to answer a query Q posed over $\mathcal{T}$ is to use RQR to obtain $\mathsf{RQR}(Q, \mathcal{T})$. For example, consider the query

$$Q_P(x) \leftarrow \text{teaches}(x, y) \tag{7.1}$$

posed over $\mathcal{T}$. In this case, it can be shown that $\mathsf{RQR}(Q, \mathcal{T})$ contains (7.1) and the following queries:

$$Q_P(x) \leftarrow \text{Teacher}(x) \tag{7.2}$$

$$Q_P(x) \leftarrow \text{Professor}(x) \tag{7.3}$$

$$Q_P(x) \leftarrow \text{hasTutor}(y, x) \tag{7.4}$$

In this case, since $\mathsf{RQR}(Q, \mathcal{T})$ is a UCQ we can use the mappings to transform it into an SQL query and send it to the RDBMS where DB resides for evaluation. In this case, transforming $\mathsf{RQR}(Q, \mathcal{T})$ into an SQL query $\mathsf{sql}(\mathsf{RQR}(Q, \mathcal{T}))$ amounts to using

$\mathcal{M}$ to replace each concept or role D occurring in a query contained in $\mathsf{RQR}(Q, \mathcal{T})$ with the corresponding SQL query Q_D and forming the union of the resulting queries. Note that $\mathcal{M}$ does not contain a mapping for every concept and role of $\mathcal{T}$. The answer to any query containing an atom for which there is no mapping will necessarily be empty so we can discard such queries. Consequently, queries (7.1) and (7.2) can be discarded. As a result, we obtain the following rewritten SQL query:

$$\mathsf{sql}(\mathsf{RQR}(Q, \mathcal{T})) = \texttt{SELECT Name FROM Professor UNION}$$
$$\texttt{SELECT Tutor FROM Student}$$

This query can now be directly evaluated in the RDBMS where DB resides in order to compute the answers of the original query Q. △

As discussed in Section 5.2, the rewriting $\mathsf{RQR}(Q, \mathcal{T})$ might be a recursive DQ and thus necessitate the use of a deductive DBMS for evaluation. In order to understand why this is so, consider the following example.

Example 7.1.2. Suppose we want to rewrite the query $Q = Q_P(x) \leftarrow \text{Student}(x)$ with respect to a TBox $\mathcal{T}$ containing the following axiom:

$$\exists \text{hasClassmate.Student} \sqsubseteq \text{Student} \tag{7.5}$$

The TBox $\mathcal{T}$ states that anybody who has a classmate who is a student is also a student.

It can be shown that RQR will produce the following rewriting $\mathsf{RQR}(Q, \mathcal{T})$:

$$Q_P(x) \leftarrow \text{Student}(x) \tag{7.6}$$
$$\text{Student}(x) \leftarrow \text{hasClassmate}(x, y) \wedge \text{Student}(y) \tag{7.7}$$

Intuitively, the reason RQR produced a DQ as opposed to a UCQ is that clause (7.7) is recursive; therefore, $\mathsf{RQR}(Q, \mathcal{T})$ cannot be transformed into a UCQ through unfolding. △

Existing deductive DBMSs, such as IRIS or XSB, allow for the specification of external data sources. Therefore, in case $\mathsf{RQR}(Q, \mathcal{T})$ is a DQ, the mappings $\mathcal{M}$ can be used to specify such an external data source in the deductive DBMS in order for $\mathsf{RQR}(Q, \mathcal{T})$ to be evaluated.

7.2 Producing UCQs beyond DL-Lite$_R$

As discussed in Section 5.2, the rewriting $\mathsf{RQR}(Q, \mathcal{T})$ might be a DQ in case $\mathcal{T}$ is not expressible in DL-Lite$_R$. In many cases, though, the TBox does not contain (or imply) recursive axioms of the form of (7.5), which might allow us to transform $\mathsf{RQR}(Q, \mathcal{T})$ into a UCQ. Obtaining UCQs is interesting from a practical point of view as that would allow the use of highly optimized RDBMSs (as opposed to deductive database systems) for evaluating $\mathsf{RQR}(Q, \mathcal{T})$. Consider the following example:

Example 7.2.1. Suppose that RQR computed the following rewriting $\mathsf{RQR}(Q, \mathcal{T})$:

$$Q_P(x) \leftarrow \text{Teacher}(x) \tag{7.8}$$

$$\text{Teacher}(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y) \tag{7.9}$$

By *unfolding* (7.9) into (7.8), the rewritingFly can be transformed into the following UCQ:

$$Q_P(x) \leftarrow \text{Teacher}(x)$$

$$Q_P(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y)$$

△

We present in Algorithm 2 a procedure that can be used to transform a nonrecursive DQ Q into a UCQ. As can be seen, the idea is to unfold nonrecursive clauses in Q. It follows from the definition of unfolding (see Section 5.2) that after unfolding every clause with head predicate P, this predicate is no longer an IDB predicate (because

Input: DQ $Q = \langle Q_P, Q_C \rangle$
$Q'_C = Q_C$;
foreach *IDB predicate* $P \neq Q_P$ *occurring in* Q_C **do**
  $C_P = \{C \mid C \in Q'_C$ with head predicate $P\}$;
  foreach *clause* $C \in C_P$ **do**
    if P *occurs in the body of* C **then**
      return $\langle Q_P, Q'_C \rangle$;
    end
  end
  $Q'_C = \mathsf{unfold}(Q'_C, C_P)$;
end
return $\langle Q_P, Q'_C \rangle$;

Algorithm 2: Greedy Unfolding

all unfolded clauses are discarded). Therefore, provided we are able to unfold every clause, in the end Q'_C will contain only one IDB predicate, namely Q_P, which means that $\langle Q_P, Q'_C \rangle$ is a UCQ since Q_P does not occur in the body of any clause. The soundness of this procedure follows from Lemma 4.4.3; termination follows from the fact that the clauses to be unfolded are nonrecursive and there is a finite number of them.

It is well known that transforming a DQ into a UCQ may, in the worst case, increase the size of the query exponentially. In Chapter 8 we analyze the impact of the greedy unfolding procedure on the size of $\mathsf{RQR}(Q, \mathcal{T})$ for realistic queries and TBoxes.

7.3 Reducing the Size of the Rewriting

In this section we present various optimizations aimed at reducing the size of the rewritings produced by RQR. Many of the discussed optimizations are also applicable to other rewriting algorithms, such as CGLLR (see Section 4.5).

7.3.1 Empty EDB predicates

The first set of optimizations we consider consist of eliminating from $\mathsf{RQR}(Q, \mathcal{T})$ every clause that contains *empty EDB predicates*—that is, EDB predicates with an empty extension with respect to a given ABox. Intuitively, the reason we can do this is that such clauses cannot be used to derive new facts, which effectively means that they are irrelevant for answering $\mathsf{RQR}(Q, \mathcal{T})$. We illustrate the idea with an example.

Example 7.3.1. Consider the following datalog program D:

$$Q_P(x) \leftarrow \text{hasTutor}(y, x) \tag{7.10}$$

$$Q_P(x) \leftarrow \text{Professor}(x) \tag{7.11}$$

Suppose that we have is an ABox $\mathcal{A}$ containing a list of professors *only*. If we wanted to evaluate D over $\mathcal{A}$, it is reasonable to discard clause (7.10) from D given the fact that we know that $\mathcal{A}$ only contains professors. In other words, we can discard (7.10) from D because it contains an empty EDB predicate of D with respect to $\mathcal{A}$, namely hasTutor. △

As we shall see, $\mathsf{RQR}(Q, \mathcal{T})$ may contain empty EDB predicates with respect to some given ABox; and more interestingly, it also may contain empty EDB predicates with respect to *any* ABox.

7.3.1.1 Mappings

As explained in Section 7.1, in order to answer a query Q using RQR, it is necessary to define mappings between the TBox $\mathcal{T}$ and the data source DB where the instance data resides. The mappings between $\mathcal{T}$ and DB do not have to be complete for $\mathcal{T}$—that is, they do not have to be defined for every concept or role of $\mathcal{T}$. Therefore, we can use the mappings to discover the set of EDB predicates in $\mathsf{RQR}(Q, \mathcal{T})$ that are empty with respect to DB, and then discard every clause in $\mathsf{RQR}(Q, \mathcal{T})$ that contains such predicates.

7.3.1.2 TBox Normalization

The rewriting $\mathsf{RQR}(Q, \mathcal{T})$ may contain empty EDB predicates with respect to any ABox due to the normalization of $\mathcal{T}$. As discussed in Section 3.3, every $\mathcal{ELHIO}^{\neg}$ TBox admits a normal form. In fact, RQR requires the input TBox to be in this normal form. In order to normalize $\mathcal{T}$ it may be necessary to introduce "auxiliary" artificial concepts. Consider the following example.

Example 7.3.2. Consider a TBox $\mathcal{T}$ containing the following axiom:

$$\text{Professor} \sqsubseteq \exists \text{teaches}.\exists \text{hasTutor} \tag{7.12}$$

Axiom (7.12) states that all professors teach someone who has a tutor.

It can be shown that the normalized version of $\mathcal{T}$ contains the following axioms:

$$\text{Professor} \sqsubseteq \exists \text{teaches}.A_1$$

$$A_1 \sqsubseteq \exists \text{hasTutor}$$

As can be seen, the auxiliary concept A_1 was introduced in the normalization process. △

Predicates corresponding to the auxiliary concepts (or roles) introduced in the normalization of $\mathcal{T}$ may occur in $\mathsf{RQR}(Q, \mathcal{T})$. Without loss of generality we can assume that for any ABox the extension of such auxiliary predicates is empty. Therefore, we can eliminate from $\mathsf{RQR}(Q, \mathcal{T})$ every clause containing an auxiliary predicate P provided that P is an EDB predicate in $\mathsf{RQR}(Q, \mathcal{T})$.

7.3.1.3 Approximation of Equality

Another reason $\mathsf{RQR}(Q, \mathcal{T})$ may contain empty EDB predicates with respect to any ABox is that auxiliary predicates may be introduced by the approximation of equality $E'_{\mathcal{T}}$. As discussed in Section 5.1, the approximation of equality is a set of clauses that we use in order to reason with equality for answering CQs over $\mathcal{ELHIO}^{\neg}$ KBs.

According to its definition, the set $E'_{\mathcal{T}}$ for a given $\mathcal{ELHIO}^{\neg}$ $\mathcal{T}$ includes two artificial predicates: $\mathcal{O}$ and $\approx$. We can thus assume that for any ABox the extension of these predicates is empty.

It follows from the definition of $\mathcal{R}^{DL}$ that clauses of the form $\mathcal{O}(o)$ (type 4.8) or $x \approx o \leftarrow A(x)$ (type 4.1) have already been unfolded in $\mathsf{RQR}(Q, \mathcal{T})$. Therefore, Lemma 4.4.3 implies that we can discard all such clauses from $\mathsf{RQR}(Q, \mathcal{T})$. Given the types of clause in the clause set $\mathcal{N}^{DL}$, it can be readily verified that after all clauses of types 4.8 and 4.1 have been eliminated from $\mathsf{RQR}(Q, \mathcal{T})$, predicates $\mathcal{O}$ and $\approx$ are EDB predicates in $\mathsf{RQR}(Q, \mathcal{T})$. Therefore, as we have assumed that their extension is empty for any ABox, we can eliminate from $\mathsf{RQR}(Q, \mathcal{T})$ every clause containing either the predicate $\mathcal{O}$ or the predicate $\approx$.

7.3.2 Clause Subsumption

The optimizations we describe in this section are based on the notion of *clause subsumption*. Intuitively, a clause C_1 subsumes another clause C_2 if C_1 is more general (or less restrictive) than C_2. Consider the following example.

Example 7.3.3. Consider the following two clauses:

$$\text{Teacher}(x) \leftarrow \text{Professor}(x) \tag{7.13}$$

$$\text{Teacher}(x) \leftarrow \text{Professor}(x) \wedge \text{Supervisor}(x) \tag{7.14}$$

Intuitively, according to clause (7.13), one only needs to know that a certain individual is a professor in order to derive that he/she is also a teacher. In contrast, clause (7.14) requires that he/she also be a supervisor. In this sense, (7.13) is less restrictive than (7.14) and we say that (7.13) subsumes (7.14). △

We formally define clause subsumption as follows.

Definition 7.3.1. Let C_1 and C_2 be Horn clauses of the form $H \leftarrow B_1 \wedge ... \wedge B_n$ and $H' \leftarrow B'_1 \wedge ... \wedge B'_m$, respectively. The clause C_1 *subsumes* C_2 if there exists a substitution σ such that $\{H\sigma, B_1\sigma, ..., B_n\sigma\} \subseteq \{H', B'_1, ..., B'_n\}$.

A Horn clause C_1 is said to be a *condensation* of a Horn clause C_2 if C_1 is the minimal subclause of C_2 such that C_1 subsumes C_2 and vice versa. △

7.3.2.1 A Posteriori Clause Subsumption

The notion of clause subsumption can be used in order to reduce the size of $\mathsf{RQR}(Q, \mathcal{T})$, both in the size of each clause and in the number of clauses. A straightforward optimization that can be used in order to reduce the size of the clauses contained in $\mathsf{RQR}(Q, \mathcal{T})$ consists of replacing each clause in $\mathsf{RQR}(Q, \mathcal{T})$ with its condensation. Clearly, since any clause is subsumed by its condensation and vice versa, the two clauses are equivalent; therefore, replacing clauses with their condensations does no affect the set of certain answers. Additionally, one can reduce the number of clauses in $\mathsf{RQR}(Q, \mathcal{T})$ by eliminating every subsumed clause. It follows from [CL97] that eliminating subsumed clauses from a DQ does not affect the set of certain answers.

It is important to note that applying the subsumption optimization *alone* (without the condensation replacements) might produce *different* rewritings with RQR and CGLLR. This is due to the fact that CGLLR's reduction step may produce condensations (see Section 4.5). We illustrate this point with an example.

Example 7.3.4. Consider a DL-Lite$_R$ TBox $\mathcal{T}$ consisting of the axiom

$$\exists \text{teaches}^- \sqsubseteq \text{Student}, \tag{7.15}$$

which states that someone that is taught is a student, and the query

$$Q_P(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y). \tag{7.16}$$

It can be shown that CGLLR produces a set containing (7.16) and the following queries:

$$Q_P(x) \leftarrow \text{teaches}(x, y) \wedge \text{teaches}(\_\,, y) \tag{7.17}$$

$$Q_P(x) \leftarrow \text{teaches}(x, \_\,) \tag{7.18}$$

Note that query (7.18) was produced in the reduction step from (7.17) and it is a condensation of (7.17). By applying the a posteriori query subsumption check we have that query (7.18) subsumes query (7.16), so the latter is discarded. Note, however, that query (7.18) subsumes query (7.17) and *vice versa*. Therefore, since it is sensible to discard the larger query, the condensation (7.18) is kept and query (7.17) is discarded instead. It is easy to see that in the end we obtain {(7.18)}.

When using RQR axiom (7.15) is translated into the following clause:

$$\text{Student}(x) \leftarrow \text{teaches}(y, x) \tag{7.19}$$

It can be shown that RQR produces a set containing (7.16) and the following clause:

$$Q_P(x) \leftarrow \text{teaches}(x, y) \wedge \text{teaches}(z, y) \tag{7.20}$$

Since (7.20) subsumes (7.16), it is easy to see that after the query subsumption check we obtain {(7.20)}. As can be seen, (7.18) is smaller than (7.20). △

7.3.2.2 Clause Subsumption on the Fly

We now describe two other well-known optimizations based on clause subsumption: forward and backward subsumption [BG01]. These two techniques, however, are based on a slightly different notion of clause subsumption. The essential difference is that clauses are to be considered *multisets* as opposed to sets. We illustrate this subtle but important difference with an example.

Example 7.3.5. Consider the following clauses:

$$Q_P(x) \leftarrow \text{teaches}(x, y) \wedge \text{teaches}(z, y) \tag{7.21}$$

$$Q_P(x') \leftarrow \text{teaches}(x', y') \tag{7.22}$$

Consider the substitution $\sigma = \{x \mapsto x', y \mapsto y', z \mapsto x'\}$. It can be readily verified that (7.21)$\sigma \subseteq$ (7.22); therefore, clause (7.21) subsumes clause (7.22) according to Definition 7.3.1. In contrast, if we adopt the multiset semantics, since (7.21) has more atoms than (7.22), there is no substitution σ such that (7.21)$\sigma \subseteq$ (7.22); therefore, (7.21) does not subsume (7.22) under the multiset semantics. △

Both forward and backward subsumption compare each new clause C_1 produced in the saturation step with the set of previously generated clauses. In forward subsumption, C_1 is discarded if the set of clauses already contains a clause C_2 that subsumes C_1 under the multiset semantics; in backward subsumption, C_2 is removed from the set of clauses if it is subsumed by C_1 under the multiset semantics.

Another well-known optimization that can be applied to eliminate clauses on the fly is a syntactic tautology check. This optimization consists of discarding every clause C produced in the saturation or unfolding step if C contains a body atom identical to its head atom. Intuitively, the reason is that such a clause cannot be used to derive new ground facts, which renders it irrelevant for query answering.

Since RQR is based on a resolution calculus, the three optimizations described in this section can be straightforwardly applied without affecting completeness [BG01]. In the case of CGLLR, however, forward subsumption cannot be (straightforwardly) applied: as discussed in Section 4.5, every query produced in the reduction step is subsumed by another previously produced query; forward subsumption would thus effectively eliminate the reduction step and so compromise completeness. It is not clear whether backward subsumption can be applied to CGLLR without affecting completeness.

7.3.3 Dependency Graph

The optimization we describe in this section is based on the notion of the so-called *dependency graph*. We formally define the dependency graph of a set of clauses as follows.

Definition 7.3.2. Given a set of Horn clauses LP, the *dependency graph* $\mathcal{G}(LP)$ of LP is constructed as follows: every predicate P occurring in LP is a node in $\mathcal{G}(LP)$ and there is an edge from a predicate P_1 to a predicate P_2 if there is a clause $C \in LP$ such that P_1 occurs in the head of C and P_2 occurs in the body of C. △

Intuitively, we can use the dependency graph $\mathcal{G}(LP)$ of a given set of clauses LP to identify the set of clauses $LP' \subseteq LP$ that are relevant in order to compute the extension of a given predicate P. Consider the following example.

Example 7.3.6. Consider the following set of clauses LP:

$$\text{Teacher}(x) \leftarrow \text{Professor}(x) \tag{7.23}$$

$$\text{Student}(x) \leftarrow \text{hasTutor}(y, x) \tag{7.24}$$

Note that neither Student nor hasTutor are reachable from the predicate Teacher in $\mathcal{G}(LP)$; therefore, we can deduce that clause (7.24) is not relevant for the computation of the extension of Teacher. △

By computing the set of predicates that are reachable from the head predicate Q_P of the query Q in the dependency graph $\mathcal{G}(\mathsf{RQR}(Q, \mathcal{T}))$ of the rewriting, one can identify the set of clauses that are "relevant" for the query and discard the set of clauses that correspond to the unreachable parts of the graph.

Note that in case $\mathsf{RQR}(Q, \mathcal{T})$ is a UCQ, its dependency graph consists of a star-like graph with the query predicate Q_P in the center. Therefore, all predicates are reachable from it, which means that the dependency graph optimization will not get

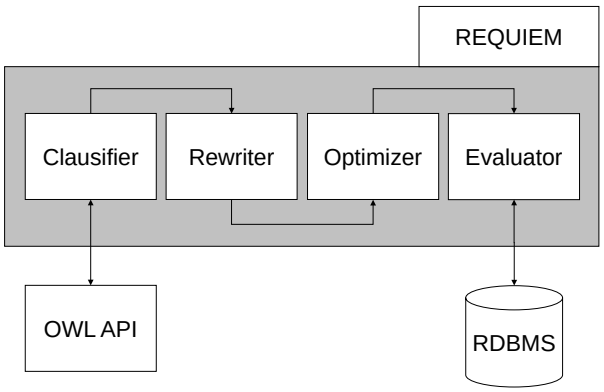


Figure 7.1: Architecture of REQUIEM

rid of any clause in $\mathsf{RQR}(Q, \mathcal{T})$. In other words, the dependency graph optimization is only useful in case $\mathsf{RQR}(Q, \mathcal{T})$ is not a UCQ. Therefore, this optimization is not useful for reducing the size of the rewritings produced by $\mathsf{RQR}^{\text{lite}}$ or CGLLR.

7.4 REQUIEM

In order to test the practicality of our rewriting approach we implemented RQR in a system that we called REQUIEM (REsolution-based QUery rewrIter for Expressive Models).[3] The system is written in Java 1.6 and fully implements the CQ rewriting algorithm described in Chapter 5.

Figure 7.1 describes the technical architecture of REQUIEM. As can be seen, the system is composed of four main components:

- The *Clausifier* is responsible for translating an $\mathcal{ELHIO}^{\neg}$ TBox into a set of clauses $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}}$, where $\mathcal{T}$ is the normalized version of the input TBox (see Section 3.3). This component interacts with the OWL API [BVL03] in order to perform TBox manipulation tasks, such as loading and parsing.

[3] http://www.comlab.ox.ac.uk/projects/requiem/

- The *Rewriter* is the central component of REQUIEM; it is composed of two subcomponents: the *Saturator* and the *Unfolder* (not shown in the figure). The Saturator is responsible for computing $\mathsf{ff}(Q, \mathcal{T})$. Computing this datalog program involves computing the saturation of $\Xi(\mathcal{T}) \cup E'_{\mathcal{T}} \cup Q_C$ by $\mathcal{R}^{DL}$. The Saturator implements the syntactic tautology check and the forward subsumption optimization described in this chapter; the latter optimization is optional. The Unfolder is responsible for computing $\mathsf{RQR}(Q, \mathcal{T})$. This component implements the greedy unfolding procedure described in this chapter; this type of unfolding is optional.

- The *Optimizer* is responsible for reducing the size of $\mathsf{RQR}(Q, \mathcal{T})$ a posteriori. This component implements the optimizations regarding empty EDB predicates and the dependency graph described in this chapter as well as the a posteriori clause subsumption check; all of these optimizations are optional.

- The *Evaluator* is responsible for computing the answer of the optimized version of $\mathsf{RQR}(Q, \mathcal{T})$ over a relational database stored in an external RDBMS. Computing the answers involves transforming the optimized version of $\mathsf{RQR}(Q, \mathcal{T})$ into an SQL query according to a set of mappings. This component can only be used when the given rewriting is a UCQ.

We have implemented a simple GUI application in order to facilitate the use of REQUIEM. The main window of this application is shown in Figure 7.2. As can be seen, one can select a TBox or ontology as an OWL file, and pose a CQ in datalog notation; moreover, one can specify the connection settings for the RDBMS where the instance data resides (currently both MySQL and PostgreSQL are supported), and select a file containing the mappings between the TBox and the database (as defined in Section 7.1). Once the relevant information has ben specified, one can use the application to rewrite the query with respect to the TBox, to obtain an SQL

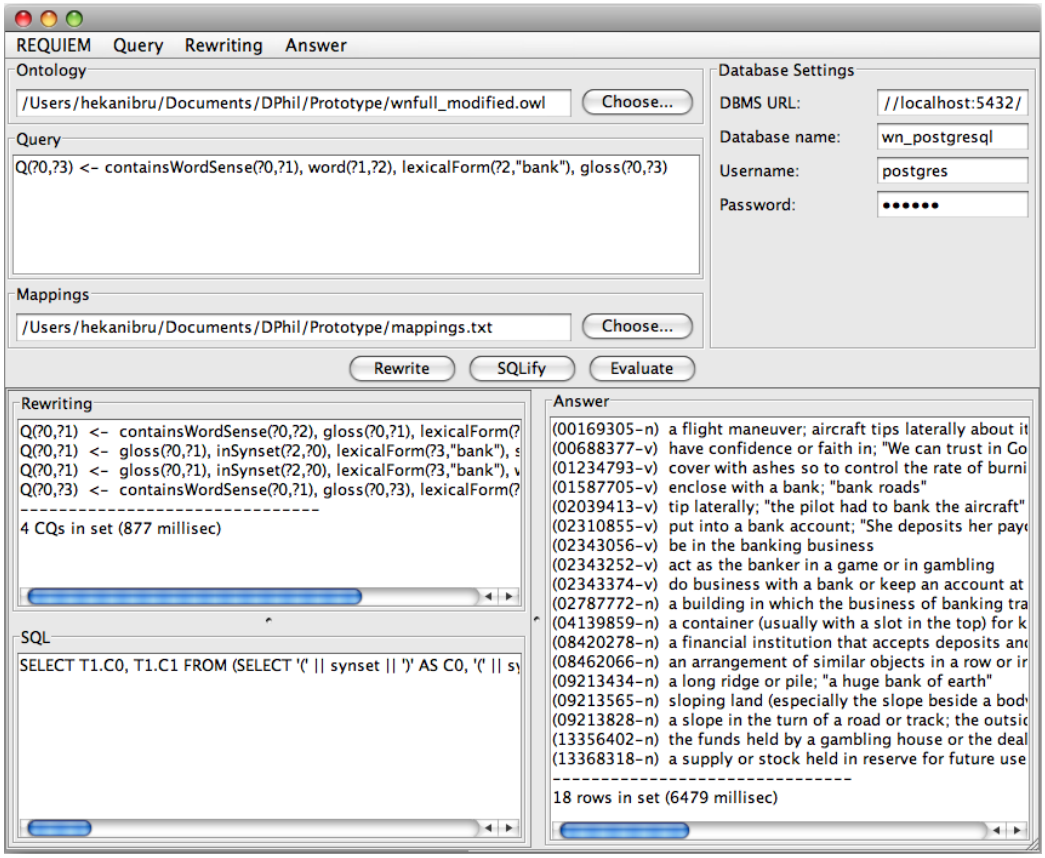


Figure 7.2: GUI application for REQUIEM

query for the resulting rewriting, and to evaluate this SQL query over the RDBMS where the instance data resides.

7.5 Chapter Summary

A clear advantage of the rewriting approach to CQ answering is that the evaluation of the rewritings can be delegated to existing (deductive) DBMSs. However, even though existing database systems are highly optimized, the practicality of query rewriting is not straightforward given the potential large size of the rewritings. Clearly, trying to produce small rewritings is of critical importance to the practical application of the CQ rewriting approach to CQ answering.

In this chapter we illustrated how to use RQR to answer CQ queries using existing (deductive) database technology. Moreover, we presented various optimizations that

can be applied in order to reduce the size of the rewritings. Finally, we described REQUIEM—a prototypical implementation of RQR that implements various of the optimizations described in this chapter. We used REQUIEM in order to conduct an empirical evaluation of our rewriting approach. We present the results of this evaluation in Chapter 8.

Chapter 8

Empirical Evaluation

Although the development of RQR has provided new theoretical insights regarding the relationship between DLs and datalog programs, our primary motivation for developing it was to obtain a CQ rewriting algorithm that can be useful in practice. As described in Chapter 7, we implemented RQR in a system called REQUIEM to validate the practicality of our approach. In this chapter, we present the results of an empirical evaluation of REQUIEM in which we analyzed its performance with respect to the size of the rewritings it produces and the time it takes to compute them. In Section 8.1 we consider DL-Lite$_R$ TBoxes and in Section 8.2 we consider TBoxes expressible in logics of the $\mathcal{EL}$ family. Additionally, in Section 8.3, we present an empirical evaluation of REQUIEM's performance in the context of a real use case identified by researchers of the Japanese National Institute of Information and Communications Technology.

REQUIEM can be configured to apply various combinations of the optimizations described in Chapter 7. For our empirical evaluation, we shall consider several of such combinations which we will refer to as follows:

- R: syntactic tautology check.
- RS: R plus an a posteriori clause subsumption check.
- RF: RS plus forward subsumption, dependency graph pruning, and empty EDB predicates pruning.

- RG: RF plus greedy unfolding.
- RC: RG plus an a posteriori condensation replacement.

In the discussion we consider each combination as a different implementation.

8.1 Rewriting Queries over DL-Lite$_R$ TBoxes

In this section we present an empirical evaluation of REQUIEM when used with DL-Lite$_R$ TBoxes.

As discussed in Sections 4.5 and 6.3, Calvanese et al. developed CGLLR—a CQ rewriting algorithm for DL-Lite$_R$ [CDL+07]. This algorithm has been implemented in various systems—such as QuOnto, Owlgres, and ONTOSEARCH2. Moreover, recent empirical evaluations [PT07, CGL+08] suggest that the algorithm can be useful in practice in realistic scenarios.

Like RQR, CGLLR can be used to rewrite a CQ Q with respect to a DL-Lite$_R$ TBox to produce a UCQ. RQR differs from CGLLR mainly in the way it handles existential quantification in the TBox. As described in Section 4.5, CGLLR handles existential quantification by relying on a so-called reduction step. Additionally, unlike RQR, CGLLR requires the elimination of qualified existential quantification axioms using an encoding that introduces new "auxiliary" roles. Both the reduction step and the introduction of auxiliary roles can increase the size of the rewritings produced by CGLLR. Therefore, we conjecture that RQR will often produce smaller rewritings than CGLLR. The main goal of our evaluation is to gather empirical evidence to support this conjecture.

Our evaluation mainly consisted of comparing R to C—a prototypical implementation of CGLLR. The comparison uses a benchmark suite containing realistic DL-Lite$_R$ TBoxes and test queries as well as some artificial TBoxes and queries designed to

highlight the differences between the two algorithms. The benchmark suite also included versions of the TBoxes in which qualified existential quantification axioms have been explicitly encoded using auxiliary roles, as this allowed us to compare the two implementations in cases where RQR's native handling of qualified existential quantification is not advantageous.

Both algorithms are amenable to optimizations that can reduce the size of the rewritings (see Section 7.3). In order to compare the optimized versions of the two algorithms we additionally ran our tests with RS and CS, both of which first compute the rewriting as in the original version, and then apply an a posteriori clause subsumption check to the result. Note that CS eliminates queries that contain auxiliary properties (introduced by the encoding of qualified existential axioms) *before* performing the clause subsumption check.

8.1.1 Test Setting

We considered four implementations for our tests: R, RS, C, and CS. The main goal of the evaluation is to compare the different implementations with respect to the *size of the rewritings* they produce. Simply counting the number of CQs in each rewriting might not provide a fair comparison as the queries themselves could differ in size; we therefore additionally measured the total number of symbols needed to represent the complete rewriting in the standard datalog notation. We also measured the time taken for each rewriting procedure. In view of our relatively naive implementations, however, this may not provide a very meaningful measure of the likely cost of the rewriting process. We therefore also measured the number of *inferences* performed by each algorithm, where by an inference we mean the derivation of a query. Note that the number of inferences is not necessarily the same as the number of queries in the final rewriting since an algorithm may derive the same query more than once.

Tests were performed on a PC running Windows XP with a 2.59 GHz Intel processor and 1.87 GB of RAM. We used Java 1.6.0 Update 7 with a maximum heap size of 1 GB. Tests were halted if execution time exceeded 600 seconds.

8.1.2 TBoxes and Queries

The test set mainly consisted of DL-Lite$_R$ TBoxes that were developed in the context of real applications. `V` is a TBox capturing information about European history that was developed in the EU-funded VICODI project.[1] `S` is a TBox capturing information about European Union financial institutions that was developed for ontology-based data access [RMLC08]. `U` is a DL-Lite$_R$ version of LUBM[2]—a benchmark TBox developed at Lehigh University for testing the performance of TBox management and reasoning systems—that describes the organizational structure of universities. `A` is a TBox capturing information about abilities, disabilities, and devices; it was developed to allow ontology-based data access for the South African National Accessibility Portal [KAGC08].

We additionally included two synthetic TBoxes in our tests in order to provide a controlled scenario to help us understand the impact of CGLLR's reduction step. `P1` and `P5` model information about graphs: nodes are represented by individuals and vertices are assertions of the form edge(a, b). The TBox `P5` contains concepts representing paths of length 1–5, while `P1` contains a concept representing paths of length 1 only.

Finally, for every TBox containing qualified existential axioms, we created a TBox where the relevant axioms have been replaced by applying the encoding required in CGLLR. We included these TBoxes in order to measure the impact of the encoding and the saturation step separately. These TBoxes are identified with the name of the original TBox and the suffix `X`. In our discussion, we refer to these TBoxes as

[1] http://www.vicodi.org/
[2] http://swat.cse.lehigh.edu/projects/lubm/

Table 8.1: Number of concepts, roles, and axioms of considered DL-Lite$_R$ TBoxes

	V	S	U	A	P1	P5	UX	AX	P5X
Concepts	194	18	34	74	2	6	35	74	6
Roles	10	12	26	5	1	1	31	31	5
Axioms	222	51	127	137	2	10	137	189	18

the *AUX TBoxes* and we refer to the others as the *original TBoxes*. The number of concepts, roles, and axioms of the considered TBoxes are shown in Table 8.1.

Except for the queries regarding the synthetic TBoxes P1, P5, and P5X, our test queries are based on canonical examples of queries used in the application where the corresponding TBox was developed. All queries are presented in Appendix A.1.

8.1.3 Results

Figure 8.1 shows the results of our evaluation. For each TBox and query, the column "Clauses" shows the number of CQs in the rewriting, the column "Symbols" shows the number of symbols needed to represent the rewriting in datalog notation, the column "Inferences" shows the number of inferences that were performed by each implementation to compute the rewritings (note that the number of inferences for R and RS, and for C and CS, is always the same), and the column "Time" shows the number of milliseconds taken to compute the rewritings.

Comparing R to C with respect to the original TBoxes, it can be seen that R produced smaller rewritings than C in 25 out of 30 cases and equally sized rewritings in the remaining 5 cases. Moreover, R was often faster and performed fewer inference steps, particularly in nontrivial cases (i.e., where both implementations took more than 1,000 ms). The differences in the size of the rewritings are often significant: in the fifth queries over U, S, and P5, for example, the rewritings produced by C contain between two and four times as many queries (and contain correspondingly larger numbers of symbols). In the fifth query over A, C had already produced more than

		Clauses				Symbols				Inferences		Time			
O	Q	R	C	RS	CS	R	C	RS	CS	R/RS	C/CS	R	C	RS	CS
V	1	15	15	15	15	454	454	454	454	14	14	16	1	31	1
	2	10	11	10	10	762	812	762	762	9	10	16	1	47	15
	3	72	72	72	72	6525	6525	6525	6525	117	117	31	31	62	47
	4	185	185	185	185	11911	11911	11911	11911	328	328	62	47	141	110
	5	30	150	30	30	3761	16255	3761	3761	59	475	31	78	63	110
S	1	6	6	6	6	158	158	158	158	48	9	15	1	16	1
	2	160	204	2	2	11422	13680	154	66	689	876	78	78	109	141
	3	480	1194	4	4	56536	121674	488	232	3130	7523	438	922	1062	2875
	4	960	1632	4	4	111092	173088	468	224	5841	10270	828	1109	2171	4156
	5	2880	11487	8	8	466896	1602203	1320	648	24332	87324	9829	80031	34681	233958
U	1	2	5	2	2	118	286	118	118	26	4	16	1	31	1
	2	148	287	1	1	10378	18496	68	30	307	589	47	62	93	125
	3	224	1260	4	4	29376	151848	516	372	570	4228	94	531	203	640
	4	1628	5364	2	2	113270	348782	124	56	5028	18541	797	4610	4093	8390
	5	2960	9245	10	10	279266	822279	932	506	13085	43306	3234	16219	15262	29204
A	1	402	783	27	27	21933	39593	901	901	934	1958	94	157	265	350
	2	103	1812	50	50	7122	116137	3783	3783	308	4986	47	687	78	719
	3	104	4763	104	104	10108	413760	10108	10108	461	14454	78	4187	93	4266
	4	492	7251	224	224	33454	461549	16069	16069	1405	21377	156	8000	422	8250
	5	624	-	624	-	70320	-	70320	-	2618	-	328	-	1031	-
P1	1	2	2	2	2	42	42	42	42	1	1	1	1	1	1
	2	2	3	2	2	70	92	70	70	4	2	1	16	16	30
	3	2	7	2	2	98	262	98	98	9	9	1	16	16	30
	4	2	16	2	2	126	734	126	126	16	38	1	16	16	30
	5	2	33	2	2	154	1824	154	154	25	129	15	31	16	47
P5	1	6	14	6	6	122	298	122	122	9	13	1	1	1	1
	2	10	86	10	10	286	2814	286	286	75	141	15	15	15	31
	3	13	538	13	13	486	23740	486	486	428	1348	94	141	95	142
	4	15	3620	15	15	708	200696	708	708	2238	12471	922	3546	1045	3607
	5	16	25256	16	16	938	1690902	938	938	11350	113277	24172	265875	30000	270520
UX	1	5	5	5	5	286	286	286	286	39	4	16	1	31	1
	2	240	286	1	1	16208	18496	68	30	510	589	78	78	187	156
	3	1008	1248	12	12	125304	151848	1452	1060	3240	4228	641	532	2093	2484
	4	5000	5358	5	5	332000	348782	283	131	17188	18541	5969	4719	36247	38189
	5	8000	9220	25	25	737000	822279	2240	1220	37635	43306	19141	15844	106055	108034
AX	1	782	783	41	41	39527	39549	1399	1385	1785	1958	218	156	766	765
	2	1781	1812	1431	1431	114537	116137	91576	91145	5073	4986	1016	688	5547	5282
	3	4752	4763	4466	4466	413030	413760	388457	388303	14629	14454	7375	4125	49452	45876
	4	7100	7251	3159	3159	454807	461549	199604	197955	21137	21377	13032	8047	72781	79377
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P5X	1	14	14	14	14	298	298	298	298	25	13	15	16	16	1
	2	77	86	25	25	2616	2814	784	728	182	141	31	15	63	47
	3	390	530	58	58	18574	23452	2480	2326	1241	1348	156	141	406	609
	4	1953	3476	179	179	120232	193608	10010	9520	7832	12471	2375	3469	9610	26438
	5	9766	23744	718	-	737890	1597158	50120	-	47100	-	69454	249177	280381	-

Figure 8.1: Results of the DL-Lite$_R$ Evaluation

42,000 queries when it exceeded the 600 second time limit; in contrast, R completed the rewriting in less than 350 ms and produced only 624 queries.

When we compare R to RS, we can see that they produced the same rewritings in 19 out of 30 cases. In some cases, however, the rewriting produced by R was much larger: in the fifth query over S, for example, R's rewriting contained 2,880 queries compared to only 8 for RS. As we might expect given the larger rewritings produced by C, it produced the same rewritings as CS in only 6 out of 30 cases. The differences in size were also generally larger: in the fifth query over S, for example,

C's rewriting contained 11,487 queries compared to only 8 for CS. The size of the largest rewritings produced by C suggests that performing query subsumption tests over these rewritings can be costly. In the fifth query over S, for example, CS takes nearly three times as long as C.

Comparing RS to CS, we can see that RS produced the same rewritings as CS in 21 out of 30 cases, larger rewritings in 8 cases, and a smaller rewriting in the remaining case (due to the fact that CS exceeded the maximum time of 600 seconds). The differences in size are minimal and the number of queries in the rewritings is always the same (with the exception of the case where CS exceeded the time limit). The reason why CS produced slightly smaller rewritings is due to the generation of condensations (see Section 7.3.2). Modifying RS to replace queries with their condensations before the clause subsumption check is straightforward.

Finally, if we turn our attention to the AUX TBoxes, we can see that R still produced smaller rewritings than C in 12 out of 15 cases, although the differences were less marked. Moreover, R still performed less inferences than C in 9 out of 15 cases. In contrast to the results with the original TBoxes, R was slower than C in 10 out of 15 cases; the differences, however, were generally small.

Our results suggest that R will produce significantly smaller rewritings than C and will be significantly faster, particularly in cases where the queries are relatively complex and/or the TBoxes contain a relatively large number of qualified existential axioms. The size of the rewritings produced in some cases also means that a clause subsumption check may be prohibitively costly in practice with CGLLR, even when clauses containing auxiliary properties are removed before performing the check. Moreover, the results for the AUX TBoxes suggest that the reduction step alone has a negative impact on the size of the rewritings—that is, the introduction of auxiliary properties does contribute to producing large rewritings, but it is not the only cause.

As mentioned in Section 6.3, a minor extension to CGLLR has been recently presented [CRS09]. Unlike CGLLR, the new version of the algorithm handles existential quantification natively. As previously mentioned, our experiments with the AUX TBoxes suggest that the reduction step alone has a negative impact on the size of the rewritings; therefore, we expect R to produce smaller rewritings than an implementation of this new algorithm as well.

8.2 Rewriting Queries over $\mathcal{EL}$ TBoxes

In this section we present an empirical evaluation of REQUIEM when used with TBoxes expressed in logics of the $\mathcal{EL}$ family.

As shown in Section 5.2, given a CQ Q and an $\mathcal{ELHIO}^{\neg}$ TBox $\mathcal{T}$, the rewriting $\mathsf{RQR}(Q, \mathcal{T})$ might be a DQ in case $\mathcal{T}$ is not expressible in DL-Lite$_R$. However, as explained in Section 7.2, it is possible to use a greedy unfolding in order to transform nonrecursive DQs into UCQs, which means that it may be possible to produce UCQs as rewritings even in case $\mathcal{T}$ is not expressible in DL-Lite$_R$. We conjecture that using greedy unfolding with REQUIEM will result in the system often producing UCQs in practice for TBoxes that go beyond DL-Lite$_R$.

The main goal of the evaluation is twofold: on the one hand, we want to gather empirical evidence to support our conjecture that REQUIEM enhanced with greedy unfolding can often produce UCQs ever for TBoxes that go beyond DL-Lite$_R$; and on the other hand, we want to test the practicality of REQUIEM for the $\mathcal{EL}$ family by analyzing the size of the produced rewritings. Our evaluation mainly consisted of measuring the impact of the greedy unfolding by comparing RF to RG. The comparison uses a benchmark suite containing realistic $\mathcal{EL}$ TBoxes and test queries.

Table 8.2: Number of concepts, roles, and axioms of considered $\mathcal{EL}$ TBoxes

	N	U	NM	GM
Concepts	27,652	43	192	99
Roles	70	25	18	46
Axioms	395,124	171	3074	166

8.2.1 Test Setting

We considered two implementations in our tests: RF and RG. The main goal of the evaluation is to compare the implementations with respect to the form and size of the rewritings produced. We measured the number of clauses and the total number of symbols needed to represent the complete rewriting in the standard datalog notation; additionally, for each rewriting we specified whether it is a UCQ or a DQ. Finally, note that the number of inferences for RF and RG is the same; therefore we only measured the time taken for each rewriting procedure.

Tests were performed on a PC running Windows XP with a 2.59 GHz Intel processor and 1.87 GB of RAM. We used Java 1.6.0 Update 7 with a maximum heap size of 1 GB. Tests were halted if execution time exceeded 600 seconds.

8.2.2 TBoxes and Queries

The test set consisted of $\mathcal{EL}$ TBoxes that were developed in the context of real applications. N is an $\mathcal{ELHI}$ version of the well-known NCI[3] ontology—a public thesaurus produced by the National Cancer Institute. U is an $\mathcal{ELHI}$ version of the university bechmark ontology[4] developed at Lehigh University. We decided to include two additional TBoxes, NM and GM, that were obtained in the context of the Health-e-Child project[5] using the modularization algorithm of Cuenca-Grau et al. [GHKS08]. NM is a module of NCI that contains information about genes and proteins, while GM is a

[3] http://www.mindswap.org/2003/CancerOntology/
[4] http://swat.cse.lehigh.edu/projects/lubm/
[5] http://www.health-e-child.org/

		Clauses		Symbols		Time		Type	
O	Q	RF	RG	RF	RG	RF	RG	RF	RG
N	1	35	35	1235	1235	142689	133141	UCQ	UCQ
	2	171	171	4975	4975	2297	3110	UCQ	UCQ
	3	220	220	14508	14508	186050	213706	UCQ	UCQ
	4	9109	9109	936485	936485	234269	253659	UCQ	UCQ
	5	176	176	12171	12171	310145	334522	UCQ	UCQ
U	1	24	24	3532	3532	94	78	UCQ	UCQ
	2	272	11	13512	1005	219	328	DQ	UCQ
	3	267	37	13133	4577	141	188	DQ	UCQ
	4	279	36	14582	4800	140	1922	DQ	UCQ
	5	267	1	13269	158	438	188	DQ	UCQ
NM	1	6	6	133	133	15	16	UCQ	UCQ
	2	16	16	529	529	16	16	UCQ	UCQ
	3	2	2	95	95	16	1	UCQ	UCQ
	4	128	128	12431	12431	93	94	UCQ	UCQ
	5	548	548	53961	53961	796	798	UCQ	UCQ
GM	1	136	46	7638	5788	235	2485	DQ	DQ
	2	136	22	7638	2578	203	1953	DQ	DQ
	3	142	46	8130	5788	204	2453	DQ	DQ
	4	137	23	7669	2609	203	1985	DQ	DQ
	5	142	28	8010	2950	219	2000	DQ	DQ

Figure 8.2: Results of the $\mathcal{EL}$ Evaluation

module of the well-known GALEN[6] ontology; the module contains information about procedures and laboratory tests. The number of concepts, roles, and axioms of the considered TBoxes are shown in Table 8.2.

Our test queries, presented in Appendix A.2, are based on canonical examples of queries used in the application where the corresponding TBox was developed.

8.2.3 Results

Figure 8.2 shows the results of our evaluation. For each TBox and query, the column 'Clauses' shows the number of clauses in the rewriting, the column 'Symbols' shows the number of symbols needed to represent the rewriting in datalog notation, the column 'Time' shows the number of milliseconds taken to compute the rewritings, and the column 'Type' shows the type of the rewritings (either UCQ or DQ).

Analyzing the type of rewritings produced by both implementations, we can see that, on the one hand, RF produced UCQs in 11 out of 20 cases and DQs in the remaining 9 cases; on the other hand, RG produced UCQs in 15 out of 20 cases and DQs in the remaining 5 cases.

[6] http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/galen.owl

Comparing the implementations with respect to their performance, RF often performed better than RG. Interestingly, the differences are not very significant in the cases where RG produced UCQs; while in the cases where RG produced DQs, the differences can be significant (e.g., queries over GM).

Analyzing the size of the rewritings produced by both implementations, we can see that both implementations can produce large rewritings especially if the TBox is relatively large (e.g., fourth query over N). In contrast, the results regarding NM and GM suggest that both implementations perform well with relatively small TBoxes.

Comparing the implementations with respect to the size of the rewritings they produced, RG produced smaller rewritings than RF in 9 out of 20 cases, and equally large rewritings in the remaining 11 cases. The differences in the size of the rewritings are often significant: in the second and fifth queries over U, for example, the rewritings produced by RF contain more than 250 more clauses than those produced by RG. These results were somewhat surprising as we expected the greedy unfolding to have a negative effect on the size of the rewritings. After a careful analysis of the rewritings, we discovered that the greedy unfolding typically produced clauses that had already been produced. We illustrate this situation with an example.

Example 8.2.1. Suppose RG produced the following rewriting before applying the greedy unfolding:

$$\text{Student}(x) \leftarrow \text{hasSupervisor}(x, y) \wedge \text{Professor}(y) \tag{8.1}$$

$$\text{PhDStudent}(x) \leftarrow \text{Student}(x) \wedge \text{Teacher}(x) \tag{8.2}$$

$$Q_P(x) \leftarrow \text{Teacher}(x) \wedge \text{hasSupervisor}(x, y) \wedge \text{Professor}(y) \tag{8.3}$$

$$Q_P(x) \leftarrow \text{Student}(x) \wedge \text{Teacher}(x) \tag{8.4}$$

$$Q_P(x) \leftarrow \text{PhDStudent}(x) \tag{8.5}$$

Let us now analyze what RG would produce by applying the greedy unfolding. The IDB predicates are Student and PhDStudent. Let us start by unfolding the

clauses with head predicate Student. After unfolding clause (8.1), we obtain:

$$\text{PhDStudent}(x) \leftarrow \text{hasSupervisor}(x, y) \wedge \text{Professor}(y) \wedge \text{Teacher}(x) \tag{8.6}$$

$$Q_P(x) \leftarrow \text{Teacher}(x) \wedge \text{hasSupervisor}(x, y) \wedge \text{Professor}(y) \tag{8.7}$$

$$Q_P(x) \leftarrow \text{Student}(x) \wedge \text{Teacher}(x) \tag{8.8}$$

$$Q_P(x) \leftarrow \text{PhDStudent}(x) \tag{8.9}$$

We can now unfold the remaining IDB predicate PhDStudent. After unfolding clause (8.6), we obtain $\{(8.7), (8.8), (8.9)\}$. As can be seen, the greedy unfolding actually reduced the size of the rewriting. Moreover, note that we would get the same result if we unfolded the IDB predicate PhDStudent first. △

Our results suggest that RF and RG will often perform well and produce small rewritings in practice, especially when dealing with relatively small TBoxes. Additionally, our results suggest that RG will often produce relatively small UCQs. This is an encouraging result since it means that, in realistic scenarios, we will often be able to use an off-the-shelf RDBMS for CQ answering even over TBoxes that go beyond DL-Lite$_R$.

8.3 A Real Use Case: the Japanese WordNet

In this section we present an empirical evaluation of REQUIEM when used to *answer* queries posed over a KB with a relatively large ABox. The main goal of the evaluation is to verify whether REQUIEM can be successfully used in this setting.

Our evaluation mainly consisted of using RC to rewrite queries and then evaluate the rewritings over a relatively large relational database. Our test KB is a Japanese version of WordNet—a well-known large lexical database created by researchers at Princeton University [Fel98]—developed by the Japanese National Institute of Information and Communications Technology [HIK08].

8.3.1 Test Setting

We considered the RC implementation in our tests. The main goal of the evaluation is to analyze the performance of RC both when computing the rewritings of typical queries and when evaluating the computed rewritings over a relatively large relational database. We measured the number of clauses and the total number of symbols needed to represent the complete rewriting in the standard datalog notation. We also measured the time taken by the implementation to compute the rewriting. Finally, we measured the number of tuples in the answer and the time taken by the implementation to evaluate the rewriting.

Tests were performed on a PC running Windows XP with a 2.59 GHz Intel processor and 1.87 GB of RAM. We used Java 1.6.0 Update 7 with a maximum heap size of 1 GB. Tests were halted if execution time exceeded 600 seconds.

8.3.2 TBox and Queries

The test set consisted of a DL-Lite$_R$ version of the RDF/OWL representation of WordNet developed by the W3C,[7] that we call W. We decided to use a DL-Lite$_R$ ontology in order to ensure that every rewriting could be evaluated in an RDBMS. W contains 16 concepts, 48 roles, and 141 axioms.

We considered 5 canonical test queries which are based on typical examples of queries used in the WordNet project. These queries, presented in Appendix A.3, were chosen in collaboration with researchers of the Japanese National Institute of Informatics.

8.3.3 Database and Mappings

Our test database, called DB$_W$, is a PostgreSQL version of a relational SQLlite3 database developed by the Japanese National Institute of Information and Commu-

[7] http://www.w3.org/TR/wordnet-rdf/

Table 8.3: Tables and number of tuples contained in DB_W

Table	Tuples
ancestor	873002
link_def	25
pos_def	8
sense	560693
synlink	567200
synset	117659
word	242420
xlink	102669

nications Technology.[8] The tables contained in the database, as well as the number of tuples for each table are shown in Table 8.3.

The mappings between W and DB_W were defined as explained in Section 7.1 and were developed in collaboration with researchers of the Japanese National Institute of Informatics. All mappings are presented in Appendix B.

8.3.4 Results

Figure 8.3 shows the results of our evaluation. For each query, the column 'Clauses' shows the number of queries in the rewriting, the column 'Symbols' shows the number of symbols needed to represent the rewriting in datalog notation, the column 'Rew. Time' shows the number of milliseconds taken to compute the rewritings, the column 'Tuples' shows the number of tuples in the answer, and the column 'Eva. Time' shows the number of milliseconds taken to evaluate the rewritings.

As can be seen, the size of the rewritings produced by RC are quite small. Moreover, the time the implementation took to compute the rewritings was always less than 100 ms. In contrast, the evaluation time can be significantly larger especially when retrieving a large set of tuples (e.g. first query).

Our results suggest that RC will often perform well and produce small rewritings in practice, especially when dealing with relatively small TBoxes. Moreover, even

[8] http://nlpwww.nict.go.jp/wn-ja/data/0.92/wnjpn.db

Q	Clauses	Symbols	Rew. Time	Tuples	Eva. Time
1	64	1934	96	117659	23254
2	31	975	49	82735	9622
3	4	328	66	18	6182
4	8	692	59	50	6737
5	1	56	30	6	506

Figure 8.3: Results of the Japanese WordNet Evaluation

though the evaluation time can be significantly larger than the rewriting time, our results suggest that RC can be used for answering queries in a relatively short time even over KBs containing hundreds of thousands of instances.

8.4 Chapter Summary

In this chapter we presented an empirical evaluation of REQUIEM—an implementation of RQR. Our evaluation consisted of using REQUIEM to rewrite typical queries over realistic TBoxes expressible in logics of the DL-Lite and $\mathcal{EL}$ families, as well as using REQUIEM in a real use case to answer CQs over a KB with a relatively large ABox.

The results presented in Section 8.1 suggest that, in spite of being a more general procedure, RQR is more efficient than CGLLR—the standard CQ rewriting algorithm for DL-Lite$_R$. However, it can be seen that R may still produce very large rewritings, especially with large TBoxes. In such cases, our evaluation suggests that several of the optimizations described in Section 7.3, such as clause subsumption, can help to significantly reduce the size of the rewritings.

The results presented in Section 8.2 suggest that, in realistic scenarios, it will often be the case that we can use RG to obtain UCQs even over TBoxes that go beyond DL-Lite$_R$. Moreover, as can be seen, transforming nonrecursive DQs into UCQs typically does not have a negative impact on the size of the rewritings. Finally, our results suggest that RG will often perform well in practice and produce relatively small rewritings, even in case the rewritings are DQs.

The results presented in Section 8.3 suggest that RC can be successfully used in

conjunction with current database technology to answer typical CQs over realistic DL-Lite$_R$ KBs with relatively large ABoxes.

As a final remark, note that since the types of query/scenario we considered are limited, we cannot conclusively state that our approach will be useful in practice in general.

Part IV

Finale

Chapter 9

Discussion

We conclude the thesis with a review of the work presented and an assessment of the extent to which the objectives set out in Chapter 1 have been met. We present a summary of the main contributions in Section 9.1 and discuss their significance in Section 9.2. Finally, in Section 9.3, we suggest directions for future work.

9.1 Thesis Achievements

Motivated by the the successful application of resolution-based techniques to reasoning with DLs, we developed RQR—a CQ rewriting algorithm for the DL $\mathcal{ELHIO}^{\neg}$. We used the algorithm to show that CQ answering for $\mathcal{ELHIO}^{\neg}$ is PTIME-complete with respect to data complexity, which makes it one of the most expressive DLs for which CQ answering is tractable with respect to data complexity.

RQR is based on a CQ answering algorithm that is interesting in its own right as it employs a novel way of reasoning with equality in the context of resolution: instead of axiomatizing equality in the standard way, which can lead to nontermination, we showed that CQ answering can be solved by reasoning with an approximation of equality that limits the number of derived clauses.

Notably, RQR exhibits "pay-as-you-go" behavior: if $\mathcal{T}$ is expressible in $\mathcal{ELHIO}^{\neg}$, then the computed rewriting $\mathsf{RQR}(Q, \mathcal{T})$ is a DQ; if $\mathcal{T}$ is expressible in DL-Lite^{+}, then $\mathsf{RQR}(Q, \mathcal{T})$ consists of a UCQ and a linear datalog program; finally, if $\mathcal{T}$ is expressible

in DL-Lite$_R$, then RQR$(Q, \mathcal{T})$ is a UCQ. Therefore, RQR not only handles a variety of DLs ranging from DL-Lite$_{core}$ up to $\mathcal{ELHIO}^{\neg}$, but it is worst-case optimal with respect to data complexity for all these DLs. RQR can thus be seen as a generalization and an extension of the existing rewriting approaches.

In order to test the practicality of RQR, we implemented it in a system called REQUIEM and conducted an empirical evaluation. Our test data mainly consisted of typical queries and realistic TBoxes expressible in various languages of the DL-Lite and $\mathcal{EL}$ families. The results of our evaluation suggest that RQR is more efficient than the standard rewriting algorithm for the DL-Lite family; and more importantly, that various straightforward optimizations can be applied to RQR in order to obtain smaller rewritings in practice. We used an optimized version of REQUIEM in conjunction with an RDBMS in order to answer CQs with respect to a real KB developed in the context of the Japanese WordNet project. The results of our evaluation suggest that our approach to query answering via query rewriting can be successfully applied in practice to answer CQs over realistic KBs.

Besides the development of RQR, our contributions include the novel result that CQ answering for DL-Lite$^+$ is NLogSpace-complete with respect to data complexity, thus closing the open question of whether CQ answering for this DL could be solved in NLogSpace. Moreover, we present new proofs for the known results that CQ answering for $\mathcal{EL}$, $\mathcal{ELH}$, and $\mathcal{ELHI}$ is PTime-complete with respect to data complexity [Ros07, KL07, KR07] and in AC^0 for DL-Lite$_R$ and DL-Lite$_{core}$ [CDL$^+$07].

The main aim of this research was to investigate whether one can use resolution-based techniques in order to solve the problem of CQ answering for various DLs via query rewriting. In particular, our objective was to develop an efficient resolution-based CQ rewriting algorithm for expressive yet tractable DLs that produces small rewritings in practice. We believe that our objectives haven been successfully met

with the development of RQR given the favorable results obtained from the empirical evaluation of REQUIEM.

9.2 Significance of the Results

As described in Chapter 7, TBoxes can be used as conceptual schemas to provide an intuitive and unified view over one or more data repositories, allowing queries to be independent of the structure and location of the data. The use of data repositories to store instance data is becoming increasingly important, for instance in the semantic Web, due to the widespread use of DLs and the scalability requirements of many data-intensive applications.

In an attempt to provide a DL that allows for practical CQ answering over KBs with large ABoxes, the new version of OWL includes a so-called profile that has been specifically designed to allow CQ answering via query rewriting. This profile, called OWL 2 QL, roughly corresponds to the DL DL-Lite$_R$; therefore, a straightforward extension of RQR can be used in order to rewrite queries over OWL 2 QL TBoxes.

Based on the results of our empirical evaluation, we expect RQR to be useful in practice in order to answer CQs over OWL 2 QL KBs. Moreover, unlike CGLLR, RQR offers the important advantage that it can handle most of the OWL 2 EL profile in addition to OWL 2 QL. As discussed in Chapter 5, in case the TBox is not expressible in DL-Lite$_R$, the rewriting is no longer guaranteed to be a UCQ, thus requiring the use of a deductive database system for its evaluation. However, our empirical evaluation shows that, in realistic scenarios, RQR enhanced with so-called greedy unfolding will often produce relatively small UCQs for TBoxes that go beyond DL-Lite$_R$; therefore, it will often be possible to use RQR in conjunction with an off-the-shelf RDBMs to answer CQs even over OWL 2 EL KBs.

9.3 Future Work

For our future work, an important theoretical challenge is to extend RQR to support other useful features of DLs, such as number restrictions or data types. While we expect the extension regarding the latter to be fairly straightforward, the extension regarding the former would be much more challenging given the interactions between number restrictions and nominals. Other theoretical future work includes a detailed analysis of the type of recursive rewritings produced by RQR in order to precisely determine when recursion could be reduced or eliminated, and whether this is beneficial as it could have a negative impact on the size of the rewritings.

With respect to the practical challenges, an extension of REQUIEM so that it can evaluate the produced rewritings using a deductive database system remains to be realized. As discussed in Section 7.1, doing so would involve defining external sources for the database system using the mappings. Given the form of the mappings we consider, we expect the extension of the evaluator module to handle DQs to be straightforward. Additionally, it would be interesting to analyze the performance of REQUIEM when answering CQs with respect to DL-Lite$_R$ KBs using different ways of storing the ABox in the database. Depending on the type of rewritings and the implementation of the used RDBMS, having a certain structure in the data might increase the speed of the evaluation process. Finally, another interesting future endeavor is the extension of REQUIEM to transform it into a full DL reasoner. This would require, on the one hand, the definition of various reductions from the standard reasoning problems to the problem of CQ answering in the same vein as the reduction of KB satisfiability described in Section 4.2; and on the other hand, the straightforward implementation of data types.

Bibliography

[ACL+05] Andrea Acciarri, Diego Calvanese, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. QUONTO: QUerying ONTOlogies. In *In Proc. of AAAI 2005*, pages 1670–1671, 2005.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ Envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 364–369. Professional Book Center, 2005.

[BBLS94] Domenico Beneventano, Sonia Bergamaschi, Stefano Lodi, and Claudio Sartori. Terminological Logics for Schema Design and Query Processing in OODBs. In Franz Baader, Martin Buchheit, Manfred A. Jeusfeld, and Werner Nutt, editors, *Reasoning about Structured Objects: Knowledge Representation Meets Databases, Proceedings of 1st Workshop KRDB'94, Saarbrücken, Germany, September 20-22, 1994*, volume 1 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1994.

[BBW06] Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2006.

[BCDG01] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML Class Diagrams using Description Logic Based Systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*, volume 44 of *CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-44/*, 2001.

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

[BDNS98] M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. A Refined Architecture for Terminological Systems: Terminology = Schema + Views. *Artif. Intell.*, 99(2):209–260, 1998.

[BFL83] R.J. Brachman, R.E. Fikes, and H.J. Levesque. Krypton: A Functional Approach to Knowledge Representation. *Computer*, 16(10):67–73, Oct. 1983.

[BFT95] Paolo Bresciani, Enrico Franconi, and Sergio Tessaris. Implementing and Testing Expressive Description Logics: a Preliminary Report. In *Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the First International KRUSE Symposium*, pages 131–139, 1995.

[BG90] Leo Bachmair and Harald Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In *CADE-10: Proceedings of the tenth international conference on Automated deduction*, pages 427–441, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[BG94] L. Bachmair and H. Ganzinger. Rewrite-based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BG01] L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In Robinson and Voronkov [RV01], chapter 2, pages 19–99.

[BH91] Franz Baader and Bernhard Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *In Proceedings of the First International Workshop on Processing Declarative Knowledge*, pages 67–86. Springer-Verlag, 1991.

[BHGS01] Sean Bechhofer, Ian Horrocks, Carole A. Goble, and Robert Stevens. OilEd: a Reason-able Ontology Editor for the Semantic Web. In *KI/GAI*, pages 396–408, 2001.

[BIG94] José Miguel Blanco, Arantza Illarramendi, and Alfredo Goñi. Building a Federated Relational Database System: An Approach Using a Knowledge-Based System. *Int. J. Cooperative Inf. Syst.*, 3(4):415–456, 1994.

[BIW94] J.I. Berman, H.H. Moore IV, and J.R. Wright. The CLASSIC and PROSE Stories: Enabling Technologies for Knowledge-Based Systems. *AT&T Technical Journal*, 73:69–80., 1994.

[BJNS94] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between Queries to Object-Oriented Databases. *Inf. Syst.*, 19(1):33–54, 1994.

[BL85] Ronald J. Brachman and Hector J. Levesque. *Readings in Knowledge Representation*. Morgan Kaufmann Pub, 1985.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. In *Scientific American*. 2001.

[BLR03] Alex Borgida, Maurizio Lenzerini, and Riccardo Rosati. Description Logics for Databases. pages 462–484, 2003.

[BLS06] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—A Polynomial-Time Reasoner for Life Science Ontologies. In U. Furbach and N. Shankar, editors, *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNCS*, pages 287–291, Seattle, WA, USA, August 17–20 2006. Springer.

[BMPSB99] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, and Alexander Borgida. "Reducing" CLASSIC to Practice: Knowledge Representation Theory Meets Reality. *Artif. Intell.*, 114(1-2):203–237, 1999.

[Bor96] A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[Bra77] Ronald J. Brachman. What's in a Concept: Structural Foundations for Semantic Networks. *Int. Journal of Man-Machine Studies*, 9(2):127–152, 1977.

[Bra78] R. J. Brachman. Structured Inheritance Networks. In W. A. Woods and R. J. Brachman, editors, *Research in Natural Language Understanding*, Quarterly Progress Report No. 1, BBN Report No. 3742, pages 36–78. Bolt, Beranek and Newman Inc., Cambridge, Mass., 1978.

[Bre95] Paolo Bresciani. Querying Databases from Description Logics. In *In Proc. of 2nd Workshop KRDB'95*, pages 1–4, 1995.

[BS85] Ronald J. Brachman and James G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[BS01] F. Baader and W. Snyder. Unification Theory. In Robinson and Voronkov [RV01], chapter 8, pages 445–532.

[BvHH+04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Technical report, World Wide Web Consortium, 2004.

[BVL03] Sean Bechhofer, Raphael Volz, and Phillip W. Lord. Cooking the Semantic Web with the OWL API. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 659–675. Springer, 2003.

[Cal07] Andrea Calì. Querying Incomplete Data with Logic Programs: ER Strikes Back. In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *ER*, volume 4801 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 2007.

[CDGL+98] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description Logic Framework for Information Integration. In Cohn et al. [CSS98], pages 2–13.

[CDGR99] Diego Calvanese, Giuseppe De Giacomo, and Riccardo Rosati. Data Integration and Reconciliation in Data Warehousing: Conceptual Modeling and Reasoning Support. *Network and Information Systems*, 2(4):413–432, 1999.

[CDL98] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the Decidability of Query Containment under Constraints. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '98)*, pages 149–158, Seattle, WA, USA, June 1–3 1998. ACM Press.

[CDL+05] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable Description Logics for Ontologies. In M. M. Veloso and S. Kambhampati, editors, *Proc. of the 20th National Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, Pittsburgh, PA, USA, July 9–13 2005. AAAI Press.

[CDL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning*, 9:385–429, 2007.

[CFH+07] Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris, editors. *Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8-10 June, 2007*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[CGK08] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In Gerhard Brewka and Jérôme Lang, editors, *KR*, pages 70–80. AAAI Press, 2008.

[CGL99] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Modeling and Querying Semi-Structured Data. *Networking and Information Systems*, 2(2):253–273, 1999.

[CGL+06] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data Complexity of Query Answering in Description Logics. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *KR*, pages 260–270. AAAI Press, 2006.

[CGL+08] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati, and Marco Ruzzi. Data Integration through DL-Lite$_A$ Ontologies. In Schewe and Thalheim [ST08], pages 26–47.

[CGL09] Andrea Calì, Georg Gottlob, and Thomas Lukasiewickz. A General Datalog-based Framework for Tractable Query Answering over Ontologies. In *PODS '09: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 69–78, New York, NY, USA, 2009. ACM.

[CGT90] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, 1990.

[CK06] Andrea Calì and Michael Kifer. Containment of Conjunctive Object Meta-Queries. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *VLDB*, pages 942–952. ACM, 2006.

[CL97] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., Orlando, FL, USA, 1997.

[CLN98] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description Logics for Conceptual Data Modeling. pages 229–263, 1998.

[CLPS08] Andrea Calì, Thomas Lukasiewicz, Livia Predoiu, and Heiner Stuckenschmidt. Tightly Integrated Probabilistic Description Logic Programs for Representing Ontology Mappings. In Sven Hartmann and Gabriele Kern-Isberner, editors, *FoIKS*, volume 4932 of *Lecture Notes in Computer Science*, pages 178–198. Springer, 2008.

[CRS09] Claudio Corona, Marco Ruzzi, and Domenico Fabio Savo. Fillign the Gap between OWL 2 QL and QuOnto: ROWLKit. In *In procceding of the International Workshop on Description Logics DL 2009*, 2009.

[CSS98] Anthony G. Cohn, Lenhard K. Schubert, and Stuart C. Shapiro, editors. *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998*. Morgan Kaufmann, 1998.

[DBSB90] P. T. Devanbu, R. J. Brachman, P. G. Selfridge, and B. W. Ballard. LaSSIE—a Knowledge-Based Software Information System. In *ICSE '90: Proceedings of the 12th international conference on Software engineering*, pages 249–261, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[DEGV97] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. In *IEEE Conference on Computational Complexity*, pages 82–101, 1997.

[DFK+08] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Li Ma, Edith Schonberg, Kavitha Srinivas, and Xingzhi Sun. Scalable Grounded Conjunctive Query Evaluation over Large and Expressive Knowledge Bases. In Sheth et al. [SSD+08], pages 403–418.

[DGINR96] Giuseppe De Giacomo, Luca Iocchi, Daniele Nardi, and Riccardo Rosati. Moving a Robot: the KR&R Approach at Work. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 198–209. Morgan Kaufmann Publishers, 1996.

[DLN+92] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Bernhard Hollunder, Werner Nutt, and Alberto Marchetti Spaccamela. The Complexity of Existential Quantification in Concept Languages. *Artif. Intell.*, 53(2-3):309–327, 1992.

[DLNN91a] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The Complexity of Concept Languages. In *Information and Computation*, pages 151–162. Morgan Kaufmann, 1991.

[DLNN91b] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. Tractable Concept Languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 458–463, 1991. Best Paper Award.

[DRPM06] S. Derriere, A. Richard, and A. Preite-Martinez. An Ontology of Astronomical Object Types for the Virtual Observatory. In *Proc. of the 26th meeting of the IAU: Virtual Observatory in Action: New Science, New Technology, and Next Generation Facilities*, pages 17–18, Prague, Czech Republic, August 21–22 2006.

[EGOv08] Thomas Eiter, Georg Gottlob, Magdalena Ortiz, and Mantas Šimkus. Query Answering in the Description Logic Horn-$\mathcal{SHIQ}$. In *JELIA '08: Proceedings of the 11th European conference on Logics in Artificial Intelligence*, pages 166–179, Berlin, Heidelberg, 2008. Springer-Verlag.

[Fel98] Christiane Fellbaum, editor. *WordNet: an Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[Fit96] Melvin Fitting. *First-Order Logic and Automated Theorem Proving (2nd ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[FM07] Ariel Fuxman and Renée J. Miller. First-Order Query Rewriting for Inconsistent Databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007.

[FTZL93] C. Fermuller, T. Tammet, N. Zamov, and Alexander Leitsch. *Resolution Methods for the Decision Problem*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1993.

[GBBI96] Alfredo Goñi, Jesús Bermúdez, José Miguel Blanco, and Arantza Illarramendi. Using Reasoning of Description Logics for Query Processing in Multidatabase Systems. In *KRDB*, 1996.

[GH07] Christine Golbreich and Ian Horrocks. The OBO to OWL Mapping, GO to OWL 1.1! In *OWLED*, 2007.

[GHH+07] Christine Golbreich, Matthew Horridge, Ian Horrocks, Boris Motik, and Rob Shearer. OBO and OWL: Leveraging Semantic Web Technologies for the Life Sciences. In *Proc. of the 6th Int. Semantic Web Conference (ISWC 2007)*, LNCS, Busan, South Korea, November 11-15 2007. Springer.

[GHKS08] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular Reuse of Ontologies: Theory and Practice. *JAIR*, 31:273–318, 2008.

[GHM+08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. OWL 2: the Next Step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *WWW*, pages 48–57, 2003.

[Gia95] Giuseppe De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Universit'a di Roma "La Sapienza", 1995.

[GKV97] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the Decision Problem for Two-Variable First-Order Logic. *The Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

[GL88] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. of the 5th Int. Conf. on Logic Programming (ICLP '88)*, pages 1070–1080, Seattle, WA, USA, August 15–19 1988. MIT Press.

[GL94a] G. De Giacomo and M. Lenzerini. Boosting the Correspondence between Description Logics and Propositional Dynamic Logics. In *AAAI '94: Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 205–212, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[GL94b] G. De Giacomo and M. Lenzerini. Concept Language with Number Restrictions and Fixpoints, and its Relationship with μ-Calculus. In *In ECAI'94*, 1994.

[GL94c] R. V. Guha and Douglas B. Lenat. Enabling Agents to work together. *Commun. ACM*, 37(7):126–142, 1994.

[GL96] G. De Giacomo and M. Lenzerini. TBox and ABox Reasoning in Expressive Description Logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327. Morgan Kaufmann Publishers, 1996.

[Gli07] Birte Glimm. *Querying Description Logic Knowledge Bases*. PhD thesis, The University of Manchester, Manchester, United Kingdom, 2007.

[Goo05] John Goodwin. Experiences of using OWL at the Ordnance Survey. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter F. Patel-Schneider, editors, *OWLED*, volume 188 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[GR09] Birte Glimm and Sebastian Rudolph. Conjunctive Query Entailment: Decidable in Spite of O, I, and Q. In *Proceedings of the of the 2000 Description Logic Workshop (DL-09)*. CEUR Workshop Proceedings, 2009.

[Gra98] Erich Gradel. Guarded Fragments of First-Order Logic: a Perspective for new Description Logics. In *In Proc. of 1998 Int. Workshop on Description Logics DL '98, Trento, CEUR Electronic Workshop Proceedings*, pages 1–1. Publications, 1998.

[Gra99] Erich Gradel. On the Restraining Power of Guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.

[Gre69] C. Green. Theorem Proving by Resolution as a Basis for Question-Answering Systems. In B. Meltzer and D. Michie, editors, *4th Annual Machine Intelligence Workshop*, page 183208. Edinburgh University Press, 1969.

[GZB06] Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The Foundational Model of Anatomy in OWL: Experience and Perspectives. *Web Semant.*, 4(3):181–195, 2006.

[Hay77] Patrick J. Hayes. In Defense of Logic. In *Procceding of the 5th Int. Joint Conf. on Artificial Intelligence (IJCAI'77)*, page 559565, 1977.

[Hay79] Patrick J. Hayes. The Logic of Frames. In D. Metzing, editor, *Frame Conceptions and Text Understanding*, pages 46–61. deextasciitilde-Gruyter, Berlin, 1979.

[Hay04] Patrick Hayes. RDF model theory. W3C Recommendation, February 2004. Available at http://www.w3.org/TR/rdf-mt/.

[HdCD+05] Frank W. Hartel, Sherri de Coronado, Robert Dionne, Gilberto Fragoso, and Jennifer Golbeck. Modeling a Description Logic Vocabulary for Cancer Research. *J. of Biomedical Informatics*, 38(2):114–129, 2005.

[HH01] Jeff Heflin and James Hendler. A Portrait of the Semantic Web in Action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.

[HIK08] Kiyotaka Uchimoto Masao Utiyama Hitoshi Isahara, Fransis Bond and Kyoko Kanzaki. Development of the Japanese WordNet.

In Bente Maegaard Joseph Mariani Jan Odjik Stelios Piperidis Daniel Tapias Nicoletta Calzolari (Conference Chair), Khalid Choukri, editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may 2008. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

[HM99] V. Haarslev and R. Möller. RACE System Description. In *Proc. of DL99, International Workshop on Description Logics, Linköping*, pages 130–132, 1999.

[HM01] V. Haarslev and R. Möller. RACER System Description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *LNAI*, pages 701–706, Siena, Italy, June 18–23 2001. Springer.

[HMS05] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. A Decomposition Rule for Decision Procedures by Resolution-based Calculi. In Franz Baader and Andrei Voronkov, editors, *Proc. of the 11th Int. Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2004)*, volume 3452 of *LNAI*, pages 21–35, Montevideo, Uruguay, March 14–18 2005. Springer.

[HMvdSW04] V. Haarslev, R. Möller, R. van der Straeten, and M. Wessel. Extended Query Facilities for Racer and an Application to Software-Engineering Problems. In *Proceedings of the 2004 International Workshop on Description Logics (DL-2004), Whistler, BC, Canada, June 6-8*, pages 148–157, 2004.

[HNSs90] Bernhard Hollunder, Werner Nutt, and Manfred Schmidt-schau. Subsumption Algorithms for Concept Description Languages. In *In ECAI-90*, pages 348–353. Pitman Publishing, 1990.

[Hor98a] Ian Horrocks. The FaCT System. In *TABLEAUX '98: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312, London, UK, 1998. Springer-Verlag.

[Hor98b] Ian Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR '98)*, pages 636–647, Trento, Italy, June 2–5 1998. Morgan Kaufmann Publishers.

[HPSv03] I. Horrocks, P. F. Patel-Schneider, and F. vanHarmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[HS99] Ian Horrocks and Ulrike Sattler. A Description Logic with Transitive and Inverse Roles and Role Hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

[HS00] U. Hustadt and R. A. Schmidt. MSPASS: Modal Reasoning by Translation and First-Order Resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000.

[HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical Reasoning for Expressive Description Logics. In *LPAR '99: Proceedings of the*

6th International Conference on Logic Programming and Automated Reasoning, pages 161–180, London, UK, 1999. Springer-Verlag.

[HSTT00] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide Query Containment under Constraints using a Description Logic. In M. Parigot and A. Voronkov, editors, *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *LNAI*, pages 326–343, Reunion Island, France, November 11–12 2000. Springer.

[Joy76] William H. Joyner. Resolution Strategies as Decision Procedures. *J. ACM*, 23(3):398–417, 1976.

[KAGC08] C. Maria Keet, Ronell Alberts, Aurona Gerber, and Gibson Chimamiwa. Enhancing Web Portals with Ontology-Based Data Access: The Case Study of South Africa's Accessibility Portal for People with Disabilities. In *OWLED*, 2008.

[Kaz06] Yevgeny Kazakov. *Saturation-Based Decision Procedures for Extensions of the Guarded Fragment*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, March 2006.

[KFNM04] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *International Semantic Web Conference*, pages 229–243, 2004.

[KL07] Adila Krisnadhi and Carsten Lutz. Data Complexity in the $\mathcal{EL}$ family of DLs. In Calvanese et al. [CFH+07].

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *J. ACM*, 42(4):741–843, 1995.

[KPH05] Aditya Kalyanpur, Bijan Parsia, and James A. Hendler. A Tool for Working with Web Ontologies. *Int. J. Semantic Web Inf. Syst.*, 1(1):36–49, 2005.

[KR07] Markus Krötzsch and Sebastian Rudolph. Conjunctive Queries for $\mathcal{EL}$ with Role composition. In Calvanese et al. [CFH+07].

[KRH08] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. ELP: Tractable Rules for OWL 2. In Sheth et al. [SSD+08], pages 649–664.

[LAF+05] Lee Lacy, Gabriel Aviles, Karen Fraser, William Gerber, Alice M. Mulvehill, and Robert Gaskill. Experiences using OWL in Military Applications. In *Proc. of the OWL: Expreiences and Directions Workshop (OWLED 2005)*, volume 188 of *CEUR WS Proceedings*, Galway, Ireland, November 11–12 2005.

[Len02] Maurizio Lenzerini. Data Integration: a Theoretical Perspective. In Lucian Popa, editor, *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[Lif96] Vladimir Lifschitz. Foundations of Logic Programming. pages 69–127, 1996.

[Llo84] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[LR96] Alon Y. Levy and Marie-Christine Rousset. Using Description Logics to Model and Reason About Views. In *KRDB*, 1996.

[LTW09] C. Lutz, D. Toman, and F. Wolter. Conjunctive Query Answering in the Description Logic $\mathcal{EL}$ using a Relational Database System. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence IJCAI09*. AAAI Press, 2009.

[Mac91] R. MacGregor. The Evolving Technology of Classification-Based Knowledge Representation Systems. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 385–400. Kaufmann, San Mateo, 1991.

[MDW91] Eric Mays, Robert Dionne, and Robert Weida. K-Rep System Overview. *SIGART Bull.*, 2(3):93–97, 1991.

[Min81] M. Minsky. A Framework for Representing Knowledge. In J. Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press, Cambridge, MA, 1981.

[Mot06] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, January 2006.

[MS06] Boris Motik and Ulrike Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In M. Hermann and A. Voronkov, editors, *Proc. of the 13th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*, volume 4246 of *LNCS*, pages 227–241, Phnom Penh, Cambodia, November 13–17 2006. Springer.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In Frank Pfenning, editor, *Proc. of the 21st Conference on Automated Deduction (CADE-21)*, volume 4603 of *LNAI*, pages 67–83, Bremen, Germany, July 17–20 2007. Springer.

[MW98] Deborah L. McGuinness and Jon R. Wright. Conceptual Modelling for Configuration: a Description Logic-Based Approach. *Artif. Intell. Eng. Des. Anal. Manuf.*, 12(4):333–344, 1998.

[Neb90a] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[Neb90b] Bernhard Nebel. Terminological Reasoning is Inherently Intractable. *Artificial Intelligence*, 43:235–249, 1990.

[OCE08] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data Complexity of Query Answering in Expressive Description Logics via Tableaux. *J. Autom. Reason.*, 41(1):61–98, 2008.

[Pel91] Christof Peltason. The BACK System—an Overview. *SIGART Bull.*, 2(3):114–119, 1991.

[Ps98] Peter F. Patel-schneider. DLP System Description. In *Collected Papers from the International Description Logics Workshop (DL'98*, pages 87–89, 1998.

[PSMBR91] Peter F. Patel-Schneider, Deborah L. McGuinness, Ronald J. Brachman, and Lori Alperin Resnick. The CLASSIC Knowledge Representation System: guiding Principles and Implementation Rationale. *SIGART Bull.*, 2(3):108–113, 1991.

[PST97] Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of Two-Variable Logic with Counting. In *LICS '97: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, page 318, Washington, DC, USA, 1997. IEEE Computer Society.

[PT07] Jeff Z. Pan and Edward Thomas. Approximating OWL-DL Ontologies. In *AAAI*, pages 1434–1439. AAAI Press, 2007.

[PUHM09a] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient Query Answering for OWL 2. In *In Proceedings of the 8th International Semantic Web Conference (ISWC 2009), Chantilly, Virginia, USA*, 2009.

[PUHM09b] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Practical Aspects of Query Rewriting for OWL 2. In *In Proceedings of the 5th OWL: Experiences and Directions Workshop (OWLED 2009), Chantilly, Virginia, USA*, 2009.

[PUMH08a] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Rewriting Conjunctive Queries over Description Logic Knowledge Bases. In Schewe and Thalheim [ST08], pages 199–214.

[PUMH08b] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Rewriting Conjunctive Queries under Description Logic Constraints. In *Proceedings of the International Workshop on Logics in Databases*, May 2008.

[PUMH09a] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. A Comparison of Query Rewriting Techniques for DL-lite. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[PUMH09b] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable Query Answering and Rewriting under Description Logic Constraints. *Journal of Applied Logic*, 2009. To appear.

[Qui67] M. Ross Quillian. Word Concepts: a Theory and Simulation of some basic Capabilities. *Behavioral Science*, 12:410430, 1967.

[Ras06] Rob Raskin. Semantic Web for Earth and Environmental Terminology (SWEET). Jet Propulsion Laboratory, California Institute of Technology, 2006. Available at http://sweet.jpl.nasa.gov/.

[RMLC08] Mariano Rodriguez-Muro, Lina Lubyte, and Diego Calvanese. Realizing Ontology Based Data Access: A Plug-in for Protégé. In *Proc. of the Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008)*, pages 286–289. IEEE Computer Society Press, 2008.

[Ros07] Riccardo Rosati. On Conjunctive Query Answering in $\mathcal{EL}$. In Calvanese et al. [CFH+07].

[RR06] A. L. Rector and J. E. Rogers. Reasoning Web: Ontological and Practical Issues in using a Description Logic to represent Medical Concept Systems: Experience from GALEN. In *Heidelberg*, pages 197–231. Springer Verlag, January 2006.

[RRL05] Alan Ruttenberg, Jonathan Rees, and Joanne Luciano. Experience using OWL DL for the Exchange of Biological Pathway Information. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter F. Patel-Schneider, editors, *OWLED*, volume 188 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[RV01] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[Ryc96] Nestor Rychtyckyj. DLMS: an Evaluation of KL-ONE in the Automobile Industry. In Lin Padgham, Enrico Franconi, Manfred Gehrke, Deborah L. McGuinness, and Peter F. Patel-Schneider, editors, *Description Logics*, volume WS-96-05 of *AAAI Technical Report*, pages 60–69. AAAI Press, 1996.

[SC98] K.A. Spackman and l K.E. Campbel. Compositional Concept Representation using SNOMED: towards further convergence of Clinical Terminologies. In *AMIA Symposium*, 1998.

[Sch91] Klaus Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. In J. Mylopoulos and R. Reiter, editors, *Proc. of 12th Int. Joint Conf. on Artificial Intelligence (IJCAI '91)*, pages 466–471, Sydney, Australia, August 24–30 1991. Morgan Kaufmann Publishers.

[Sch95] Klaus Schild. *Querying Knowledge and Data Bases by a Universal Description Logic with Recursion*. PhD thesis, Universitat des Saarlandes, 1995.

[SDCS05] Amandeep S. Sidhu, Tharam S. Dillon, Elizabeth Chang, and Baldev S. Sidhu. Protein Ontology Development using OWL. In *Proc. of the OWL: Expreiences and Directions Workshop (OWLED 2005)*, volume 188 of *CEUR WS Proceedings*, Galway, Ireland, November 11–12 2005.

[SE07] Mantas Simkus and Thomas Eiter. DNC: Decidable Non-monotonic Disjunctive Logic Programs with Function Symbols. In Nachum Der-

showitz and Andrei Voronkov, editors, *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 514–530. Springer, 2007.

[SLL+04] Dagobert Soergel, Boris Lauser, Anita C. Liang, Frehiwot Fisseha, Johannes Keizer, and Stephen Katz. Reengineering Thesauri for New Applications: The AGROVOC Example. *J. Digit. Inf.*, 4(4), 2004.

[SP06] Evren Sirin and Bijan Parsia. Pellet System Description. In *Description Logics*, 2006.

[SS89] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR '89)*, pages 421–431, Toronto, Canada, May 15–18 1989. Morgan Kaufmann Publishers.

[SSD+08] Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors. *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*. Springer, 2008.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[ST08] Klaus-Dieter Schewe and Bernhard Thalheim, editors. *Semantics in Data and Knowledge Bases, Third International Workshop, SDKB 2008, Nantes, France, March 29, 2008, Revised Selected Papers*, volume 4925 of *Lecture Notes in Computer Science*. Springer, 2008.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNAI*, pages 292–297, Seattle, WA, USA, August 17–20 2006. Springer.

[TM93] Catarci T. and Lenzerini M. Representing and using Interschema Knowledge in Cooperative Information Systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[Var82] Moshe Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. Landweber, editors, *Proc. of the 14th annual ACM Symposium on Theory of Computing (STOC '82)*, pages 137–146, San Francisco, CA, USA, May 5–7 1982. ACM Press.

[Woo75] William A. Woods. What's in a Link: Foundations for Semantic Networks. pages 35–82, 1975. From: Bobrow & Collins, Representation and Understanding.

[WSH+07] C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System Description: SPASS Version 3.0. In F. Pfenning, editor, *Automated Deduction—CADE-21*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 514–520. Springer, 2007.

Appendix A

Test Queries

A.1 Queries for DL-Lite$_R$ TBoxes

- V

$$Q_1(x_0) \leftarrow \text{Location}(x_0)$$

$$Q_2(x_0, x_1) \leftarrow \text{MilitaryPerson}(x_0) \wedge \text{hasRole}(x_1, x_0) \wedge \text{related}(x_0, x_2)$$

$$Q_3(x_0, x_1) \leftarrow \text{TimeDependantRelation}(x_0) \wedge \text{hasRelationMember}(x_0, x_1) \wedge \text{Event}(x_1)$$

$$Q_4(x_0, x_1) \leftarrow \text{Object}(x_0) \wedge \text{hasRole}(x_0, x_1) \wedge \text{Symbol}(x_1)$$

$$Q_5(x_0) \leftarrow \text{Individual}(x_0) \wedge \text{hasRole}(x_0, x_1) \wedge \text{Scientist}(x_1) \wedge \text{hasRole}(x_0, x_2) \wedge \text{Discoverer}(x_2) \wedge \text{hasRole}(x_0, x_3) \wedge \text{Inventor}(x_3)$$

- S

$$Q_1(x_0) \leftarrow \text{StockExchangeMember}(x_0)$$

$$Q_2(x_0, x_1) \leftarrow \text{Person}(x_0) \wedge \text{hasStock}(x_0, x_1) \wedge \text{Stock}(x_1)$$

$$Q_3(x_0, x_1, x_2) \leftarrow \text{FinantialInstrument}(x_0) \wedge \text{belongsToCompany}(x_0, x_1) \wedge \text{Company}(x_1) \wedge \text{hasStock}(x_1, x_2) \wedge \text{Stock}(x_2)$$

$$Q_4(x_0, x_1, x_2) \leftarrow \text{Person}(x_0) \wedge \text{hasStock}(x_0, x_1) \wedge \text{Stock}(x_1) \wedge \text{isListedIn}(x_1, x_2) \wedge \text{StockExchangeList}(x_2)$$

$$Q_5(x_0, x_1, x_2, x_3) \leftarrow \text{FinantialInstrument}(x_0) \wedge \text{belongsToCompany}(x_0, x_1) \wedge \text{Company}(x_1) \wedge \text{hasStock}(x_1, x_2) \wedge \text{Stock}(x_2) \wedge \text{isListedIn}(x_1, x_3) \wedge \text{StockExchangeList}(x_3)$$

- `U/UX`

$$Q_1(x_0) \leftarrow \text{worksFor}(x_0, x_1) \wedge \text{affiliatedOrganizationOf}(x_1, x_2)$$

$$Q_2(x_0, x_1) \leftarrow \text{Person}(x_0) \wedge \text{teacherOf}(x_0, x_1) \wedge \text{Course}(x_1)$$

$$Q_3(x_0, x_1, x_2) \leftarrow \text{Student}(x_0) \wedge \text{advisor}(x_0, x_1) \wedge \text{FacultyStaff}(x_1) \wedge \text{takesCourse}(x_0, x_2) \wedge \text{teacherOf}(x_1, x_2) \wedge \text{Course}(x_2)$$

$$Q_4(x_0, x_1) \leftarrow \text{Person}(x_0) \wedge \text{worksFor}(x_0, x_1) \wedge \text{Organization}(x_1)$$

$$Q_5(x_0) \leftarrow \text{Person}(x_0) \wedge \text{worksFor}(x_0, x_1) \wedge \text{University}(x_1) \wedge \text{hasAlumnus}(x_1, x_0)$$

- `A/AX`

$$Q_1(x_0) \leftarrow \text{Device}(x_0) \wedge \text{assistsWith}(x_0, x_1)$$

$$Q_2(x_0) \leftarrow \text{Device}(x_0) \wedge \text{assistsWith}(x_0, x_1) \wedge \text{UpperLimbMobility}(x_1)$$

$$Q_3(x_0) \leftarrow \text{Device}(x_0) \wedge \text{assistsWith}(x_0, x_1) \wedge \text{Hear}(x_1) \wedge \text{affects}(x_2, x_1) \wedge \text{Autism}(x_2)$$

$$Q_4(x_0) \leftarrow \text{Device}(x_0) \wedge \text{assistsWith}(x_0, x_1) \wedge \text{PhysicalAbility}(x_1)$$

$$Q_5(x_0) \leftarrow \text{Device}(x_0) \wedge \text{assistsWith}(x_0, x_1) \wedge \text{PhysicalAbility}(x_1) \wedge \text{affects}(x_2, x_1) \wedge \text{Quadriplegia}(x_2)$$

- `P1/P5/P5X`

$$Q_1(x_0) \leftarrow \text{edge}(x_0, x_1)$$

$$Q_2(x_0) \leftarrow \text{edge}(x_0, x_1) \wedge \text{edge}(x_1, x_2)$$

$Q_3(x_0) \leftarrow \text{edge}(x_0, x_1) \wedge \text{edge}(x_1, x_2) \wedge \text{edge}(x_2, x_3)$

$Q_4(x_0) \leftarrow \text{edge}(x_0, x_1) \wedge \text{edge}(x_1, x_2) \wedge \text{edge}(x_2, x_3) \wedge \text{edge}(x_3, x_4)$

$Q_5(x_0) \leftarrow \text{edge}(x_0, x_1) \wedge \text{edge}(x_1, x_2) \wedge \text{edge}(x_2, x_3) \wedge \text{edge}(x_3, x_4) \wedge \text{edge}(x_4, x_5)$

A.2 Queries for $\mathcal{EL}$ TBoxes

- `N`

  $Q_1(x_0) \leftarrow \text{Organ\_System}(x_0)$

  $Q_2(x_0) \leftarrow \text{Chemical\_Modifier}(x_0)$

  $Q_3(x_0) \leftarrow \text{Blood}(x_1) \wedge \text{Cell}(x_0) \wedge \text{rAnatomic\_Structure\_Has\_Location}(x_0, x_1)$

  $Q_4(x_0) \leftarrow \text{Cytokine}(x_0) \wedge \text{Growth\_Factor}(x_1) \wedge$
  $\text{rProtein\_Has\_Biochemical\_Function}(x_0, x_1)$

  $Q_5(x_0) \leftarrow \text{Hematopoietic\_Body\_System}(x_1) \wedge \text{Organ}(x_0) \wedge$
  $\text{rAnatomic\_Structure\_is\_Physical\_Part\_of}(x_0, x_1)$

- `U`

  $Q_1(x_0) \leftarrow \text{Department}(x_2) \wedge \text{GraduateStudent}(x_0) \wedge \text{University}(x_1) \wedge$
  $\text{memberOf}(x_0, x_2) \wedge \text{subOrganizationOf}(x_2, x_1) \wedge$
  $\text{udergraduateDegreeFrom}(x_0, x_1)$

  $Q_2(x_0) \leftarrow \text{Professor}(x_0) \wedge \text{emailAddress}(x_0, x_3) \wedge \text{name}(x_0, x_2) \wedge \text{telephone}(x_0, x_4) \wedge$
  $\text{worksFor}(x_0, x_1)$

  $Q_3(x_0) \leftarrow \text{Course}(x_1) \wedge \text{Student}(x_0) \wedge \text{takesCourse}(x_0, x_1) \wedge \text{teacherOf}(x_2, x_1)$

  $Q_4(x_0) \leftarrow \text{Department}(x_1) \wedge \text{Student}(x_0) \wedge \text{emailAddress}(x_0, x_3) \wedge$
  $\text{memberOf}(x_0, x_1) \wedge \text{subOrganizationOf}(x_1, x_2)$

  $Q_5(x_0) \leftarrow \text{Course}(x_2) \wedge \text{Faculty}(x_1) \wedge \text{Student}(x_0) \wedge \text{advisor}(x_0, x_1) \wedge$
  $\text{takesCourse}(x_0, x_2) \wedge \text{teacherOf}(x_1, x_2)$

- `NM`

$$Q_1(x_0) \leftarrow \text{Organisms}(x_0)$$

$$Q_2(x_0) \leftarrow \text{Gene}(x_0)$$

$$Q_3(x_0) \leftarrow \text{Islets\_of\_Langerhans}(x_1) \wedge \text{rProtein\_Expressed\_In\_Tissue}(x_0, x_1)$$

$$Q_4(x_0) \leftarrow \text{Cytokine}(x_0) \wedge \text{Growth\_Factor}(x_1) \wedge \text{rProtein\_Has\_Biochemical\_Function}(x_0, x_1)$$

$$Q_5(x_0) \leftarrow \text{Protein}(x_0) \wedge \text{Growth\_Factor}(x_1) \wedge \text{rProtein\_Has\_Biochemical\_Function}(x_0, x_1)$$

- `GM`

$$Q_1(x_0) \leftarrow \text{PlateletCountProcedure}(x_0)$$

$$Q_2(x_0) \leftarrow \text{WestergrenESRProcedure}(x_0)$$

$$Q_3(x_0) \leftarrow \text{InvestigationAct}(x_0) \wedge \text{PlateletCount}(x_1) \wedge \text{isToDetermine}(x_0, x_1)$$

$$Q_4(x_0) \leftarrow \text{hasSubprocess}(x_0, x_1)$$

$$Q_5(x_0) \leftarrow \text{InvestigationAct}(x_0) \wedge \text{hasSubprocess}(x_0, x_1)$$

A.3 Queries for the Japanese WordNet

$$Q_1(x_0) \leftarrow \text{Synset}(x_0)$$

$$Q_2(x_0) \leftarrow \text{NounSynset}(x_0)$$

$$Q_3(x_3) \leftarrow \text{containsWordSense}(x_0, x_1) \wedge \text{word}(x_1, x_2) \wedge \text{lexicalForm}(x_2, \text{“bank”}) \wedge \text{gloss}(x_0, x_3)$$

$$Q_4(x_3) \leftarrow \text{hyponymOf}(x_0, x_3) \wedge \text{containsWordSense}(x_0, x_1) \wedge \text{word}(x_1, x_2) \wedge \text{lexicalForm}(x_2, \text{“bank”})$$

$$Q_5(x_0, x_1) \leftarrow \text{similarTo}(\text{“[00337404-a]”}, x_0) \wedge \text{gloss}(x_0, x_1)$$

Appendix B

Mappings for the Japanese WordNet

AdjectiveSynset ↦
```
SELECT '['||synset||']' AS C0 FROM synset
WHERE pos = 'a'
```

AdverbSynset ↦
```
SELECT '['||synset||']' AS C0 FROM synset
WHERE pos = 'r'
```

NounSynset ↦
```
SELECT '['||synset||']' AS C0 FROM synset
WHERE pos = 'n'
```

VerbSynset ↦
```
SELECT '['||synset||']' AS C0 FROM synset
WHERE pos = 'v'
```

AdjectiveWordSense ↦
```
SELECT '['||synset||'+'||wordid||']' AS C0
FROM sense WHERE substring(synset from
char_length(synset)) = 'a'
```

AdverbWordSense ↦
```
SELECT '['||synset||'+'||wordid||']' AS C0
FROM sense WHERE substring(synset from
char_length(synset)) = 'r'
```

NounWordSense ↦
```
SELECT '['||synset||'+'||wordid||']' AS C0
FROM sense WHERE substring(synset from
```

```
                    char_length(synset)) = 'n'
VerbWordSense ↦ SELECT '['||synset||'+'||wordid||']' AS C0
                    FROM sense WHERE substring(synset from
                    char_length(synset)) = 'v'
          Word ↦ SELECT '['||wordid||']' AS C0 FROM word
containsWordSense ↦ SELECT '['||synset||']' AS C0,
                    '['||synset||'+'||wordid||']' AS C1 FROM sense
          word ↦ SELECT '['||synset||'+'||wordid||']' AS C0,
                    '['||wordid||']' AS C1 FROM sense
  hypernymOf ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                    AS C1 FROM synlink WHERE link = 'hype'
   holonymOf ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                    AS C1 FROM synlink WHERE link = 'holo'
memberHolonymOf ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                    AS C1 FROM synlink WHERE link = 'hmem'
 partHolonymOf ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                    AS C1 FROM synlink WHERE link = 'hprt'
substanceHolonymOf ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                    AS C1 FROM synlink WHERE link = 'hsub'
    attribute ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                    AS C1 FROM synlink WHERE link = 'attr'
      entails ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                    AS C1 FROM synlink WHERE link = 'enta'
      causes ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
```

```
                AS C1 FROM synlink WHERE link = 'caus'
seeAlso ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                AS C1 FROM synlink WHERE link = 'also'
similarTo ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                AS C1 FROM synlink WHERE link = 'sim'
antonymOf ↦ SELECT '['||synset1||']' AS C0, '['||synset2||']'
                AS C1 FROM synlink WHERE link = 'ants'
lexicalForm ↦ SELECT '['||wordid||']' AS C0, lemma AS C1
                FROM word
gloss ↦ SELECT '['||synset||']' AS C0, def AS C1
                FROM synset_def
```