Rewriting Conjunctive Queries over Description Logic Knowledge Bases

Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks

Computing Laboratory
University of Oxford
Oxford, UK
{hector.perez-urbina,boris.motik,ian.horrocks}@comlab.ox.ac.uk

Abstract. We consider the problems of conjunctive query answering and rewriting for information integration systems in which a Description Logic ontology is used to provide a global view of the data. We present a resolution-based query rewriting algorithm for DL-Lite$^+$ ontologies, and use it to show that query answering in this setting is NLogSpace-complete with respect to data complexity. We also show that our algorithm produces an optimal rewriting when the input ontology is expressed in the language DL-Lite. Finally, we sketch an extended version of the algorithm that would, we are confident, be optimal for several DL languages with data complexity of query answering ranging from LogSpace to PTime-complete.

1 Introduction

The use of ontologies as conceptual views over data repositories has proven to be useful in a variety of different scenarios. In Information Integration (II) [15], Enterprise Application Integration (EAI) [14], and the Semantic Web [11], ontologies are used to represent the domain of a given application. This provides users with a coherent global view of the information, thus hiding the details of data organization. In this paper, we focus on II systems in which an ontology is used to provide transparent access to several independent information sources. Typically, such a system consists of a *global* ontology, representing the structure of the application domain, a set of (relational) schemas representing the structure of the *local* information sources, and a set of mappings that relates the global ontology to the local schemas.

The global ontology is often expressed in a Description Logic (DL). DLs are a family of knowledge representation formalisms that model a given domain in terms of concepts (unary predicates), roles (binary predicates), and individuals (constants) [2]. A DL Knowledge Base (KB) consists of a *terminological* component $\mathcal{T}$ called the TBox, and an *assertional* component $\mathcal{A}$ called the ABox. In analogy to databases, the TBox can be seen as a conceptual schema and the ABox as a (partial) instantiation of the schema. The syntax of DLs can be restricted in a variety of ways to trade off the expressive power against computational complexity, and thus to obtain a representation language that is suitable for the application at hand.

The main task of an II system is to provide a service for answering a query Q over the global ontology using information in the local sources. This service can be realized via query rewriting: the query over the global ontology can be rewritten into a query that is then evaluated over the local sources [10]. Calvanese et al. [5] showed that query rewriting in global-as-view II systems—that is, systems in which concepts and roles from the global ontology are mapped to the local sources by a query over one or more sources [15]—can be solved in two stages. Given a global ontology $\mathcal{T}$ expressed in the description logic DL-Lite [7] and a query Q over $\mathcal{T}$, we can first eliminate $\mathcal{T}$—that is, we can compute a query Q' (which depends on Q and $\mathcal{T}$) such that, for every ABox $\mathcal{A}$, the answers of Q over $\mathcal{T}$ and $\mathcal{A}$, and the answers of Q' over $\mathcal{A}$ coincide; this problem is known as *query rewriting w.r.t. DL TBoxes.* We can then deal with the mappings by unfolding—that is, by replacing each atom in Q' with its definition in the mappings. Assuming mappings are as in [5], this second step is rather straightforward; in contrast, the rewriting of Q and $\mathcal{T}$ into Q' is the main technical problem in the overall algorithm. Therefore, in the rest of this paper, we consider the problem of query rewriting w.r.t. DL TBoxes; the application of our results in an II setting is then straightforward and can be done as in [5].

The rewriting algorithm by Calvanese et al. for the DL-Lite family of languages has been used to show that query answering in DL-Lite is in LogSpace w.r.t. data complexity [7]. Similarly, Rosati used a rewriting algorithm for DL TBoxes expressed in the $\mathcal{EL}$ family of languages [1] to show that query answering in $\mathcal{EL}$ is PTime-complete [18].

In this paper we explore the gap between these two results, and investigate the case where the TBox is expressed in DL-Lite$^+$—a language for which query answering is known to be NLogSpace-hard [7]. We present a query rewriting algorithm for DL-Lite$^+$ and use it to show that query answering in DL-Lite$^+$ can be implemented in NLogSpace, thus closing an open problem from [7]. Moreover, we show that our algorithm exhibits "pay-as-you-go" behavior: it produces an optimal rewriting for TBoxes expressed in a subset of DL-Lite$^+$ for which query answering is in LogSpace. Finally, we provide a sketch showing how this algorithm could be straightforwardly extended to deal with members of the $\mathcal{EL}$ family and beyond. In fact, we are confident that such an algorithm would not only deal with the full spectrum of languages from DL-Lite to $\mathcal{EL}$ extended with inverse roles, universal quantifier, and functionality assertions, but would be optimal with respect to data complexity for all such languages.

2 Preliminaries

2.1 Description Logic DL-Lite$^+$

For A an *atomic concept* and P an *atomic role*, a DL-Lite$^+$ *basic concept* has the form A, $\exists P$, or $\exists P.A$. A *TBox* is a set of *inclusion assertions* of the form $B_1 \sqsubseteq B_2$ or $P_1 \sqsubseteq P_2$, where B_1 and B_2 are basic concepts, and P_1 and P_2 are atomic roles. Without loss of generality, we assume that no TBox contains assertions of the form $\exists P.A \sqsubseteq \exists S.B$: without affecting satisfiability of the TBox,

Table 1. Semantics of DL-Lite$^+$

Semantics of concepts:	Semantics of assertions:	
	$\mathcal{I} \models A(a)$	iff $a^\mathcal{I} \in A^\mathcal{I}$
$(\exists P)^\mathcal{I} = \{x \mid \exists y.\langle x,y\rangle \in P^\mathcal{I}\}$	$\mathcal{I} \models P(a,b)$	iff $(a^\mathcal{I}, b^\mathcal{I}) \in P^\mathcal{I}$
$(\exists P.A)^\mathcal{I} = \{x \mid \exists y.\langle x,y\rangle \in P^\mathcal{I} \wedge y \in A^\mathcal{I}\}$	$\mathcal{I} \models B_1 \sqsubseteq B_2$	iff $B_1^\mathcal{I} \subseteq B_2^\mathcal{I}$
	$\mathcal{I} \models P_1 \sqsubseteq P_2$	iff $P_1^\mathcal{I} \subseteq P_2^\mathcal{I}$

each assertion of this form can be replaced with $\exists P.A \sqsubseteq C$ and $C \sqsubseteq \exists S.B$, for C a fresh atomic concept. An *ABox* is a set of *membership assertions* of the form $A(a)$ or $P(a,b)$, where A is an atomic concept, P is an atomic role, and a and b are constants. A DL-Lite$^+$ *knowledge base* (KB) $\mathcal{K}$ is a tuple $\langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox.

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 concept C to a subset $C^\mathcal{I}$ of $\triangle^\mathcal{I}$, each role P to a subset $P^\mathcal{I}$ of $\triangle^\mathcal{I} \times \triangle^\mathcal{I}$, and each constant a to an element $a^\mathcal{I}$ of $\triangle^\mathcal{I}$. The function $\cdot^\mathcal{I}$ is extended to concepts as shown in the left part of Table 1. An interpretation $\mathcal{I}$ is a model of an inclusion or membership assertion α, written $\mathcal{I} \models \alpha$, if $\mathcal{I}$ and α satisfy the conditions shown in the right part of Table 1. An interpretation $\mathcal{I}$ is a *model* of a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, written $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ satisfies each of the inclusion assertions in $\mathcal{T}$ and each of the membership assertions in $\mathcal{A}$. A KB $\mathcal{K}$ is *satisfiable* if it has at least one model; furthermore, $\mathcal{K}$ *logically implies* an assertion α, written $\mathcal{K} \models \alpha$, if all models of $\mathcal{K}$ are also models of α.

DL-Lite is obtained from DL-Lite$^+$ by disallowing concepts of the form $\exists P.A$ on the left-hand side of inclusion assertions $B_1 \sqsubseteq B_2$. The definition of DL-Lite in [7] additionally allows for inverse roles. Extending DL-Lite$^+$ with inverse roles results in a logic with a PTime-hard query answering problem [7]. Since our goal in this paper is to investigate NLogSpace-complete DLs, we do not consider inverse roles in this paper.

2.2 Conjunctive and Datalog Queries

We use the well-known notions of a first-order signature, terms, variables, and atoms. A *Horn clause* is an expression of the form $H \leftarrow B_1 \wedge \cdots \wedge B_m$, where H is a possibly empty atom and $\{B_i\}$ is a set of atoms. H is called the *head* and the set $\{B_i\}$ is called the *body*. With $\square$ we denote the *empty clause* that has no body atoms and whose head atom is $\bot$. A Horn clause C is *safe* if all variables occurring in the head also occur in the body. A Horn clause is a *fact* if it is safe and does not contain body atoms; instead of $H \leftarrow$, we usually denote such clauses as H. With $\mathsf{var}(C)$ we denote the number of variables in a clause C. The *depth* of a term is defined as $\mathsf{depth}(t) = 0$ for t a constant or a variable and $\mathsf{depth}(f(s_1,\ldots,s_m)) = 1 + \max(\mathit{depth}(s_1),\ldots,\mathit{depth}(s_m))$; the depth of an atom is defined as $\mathsf{depth}(R(t_1,\ldots,t_n)) = \max(\mathsf{depth}(t_1),\ldots,\mathsf{depth}(t_n))$; and the depth of a Horn clause C is $\mathsf{depth}(C) = \max(\mathsf{depth}(H),\mathsf{depth}(B_1),\ldots,\mathsf{depth}(B_m))$.

A *datalog* program P is a set of function-free, safe Horn clauses. The *extensional database (EDB) predicates* of P are those that do not occur in the head atom of any Horn clause in P; all other predicates are called *intensional database (IDB) predicates.* Furthermore, P is *linear* if each Horn clause in P contains at most one IDB predicate in the body.

A *datalog query* Q is a tuple $\langle Q_P, P\rangle$, where Q_P is a *query predicate* and P is a datalog program. A datalog query $Q = \langle Q_P, P\rangle$ is called a *union of conjunctive queries* if Q_P is the only IDB predicate in P and the body of each clause in P does not contain Q_P; furthermore, Q is a *conjunctive query* if it is a union of conjunctive queries and P contains exactly one Horn clause; finally, Q is a *linear datalog query* if P is a linear datalog program. A tuple of constants $\vec{a}$ is an *answer* of a datalog query $Q = \langle Q_P, P\rangle$ on a DL-Lite$^+$ knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ if and only if $\mathcal{K} \cup P \models Q_P(\vec{a})$, where P is considered to be a set of universally quantified implications with the usual first-order semantics; the set of all answers of Q on $\mathcal{K}$ is denoted by $\mathsf{ans}(Q, \mathcal{K})$.

2.3 Resolution with Free Selection

Resolution with free selection is a well-known calculus that can be used to decide satisfiability of a set of Horn clauses N [4]. The calculus is parameterized by a *selection function* S that assigns to each Horn clause C a nonempty set of atoms such that either $S(C) = \{H\}$ or $S(C) \subseteq \{B_i\}$. The atoms in $S(C)$ are said to be *selected* in C. The resolution calculus with free selection $\mathcal{R}$ consists of the following *resolution inference rule.*

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

As usual, we make a technical assumption that the premises do not have variables in common. The atoms B_i and C must be selected in the corresponding premises by a selection function and $\sigma = \mathsf{MGU}(B_i, C)$; that is, σ is the *most general unifier* of B_i and C as defined in [3]. The two clauses above the inference line are called the *premises* and the clause below the line is called the *resolvent.*

An *inference* is an application of an inference rule to concrete premises. A set of Horn clauses N is *saturated* by $\mathcal{R}$ if, for any two premises $P_1, P_2 \in N$, the set N contains a clause that is equivalent to the resolvent of P_1 and P_2 up to variable renaming. A *derivation* by $\mathcal{R}$ from a set of Horn clauses N is a sequence of sets of Horn clauses $N = N_0, N_1, \ldots$ such that, for each $i \geq 0$, we have that $N_{i+1} = N_i \cup \{C\}$, where C is the conclusion of an inference by $\mathcal{R}$ from premises in N_i. A derivation is said to be *fair* if, for any i and any two Horn clauses $P_1, P_2 \in N_i$ to which resolution is applicable, some $j \geq i$ exists such that $R \in N_j$, where R is the resolvent between P_1, P_2. The limit N_∞ of a (possibly infinite) fair derivation from a set of Horn clauses N is defined as $N_\infty = \bigcup_i N_i$. It is well known that N_∞ is saturated by $\mathcal{R}$ and does not depend on the derivation [4]. A set of Horn clauses N is satisfiable if and only if $\square \notin N_\infty$. A clause C is said to be *derivable* from N iff $C \in N_\infty$.

3 Answering Conjunctive Queries in DL-Lite$^+$

Given a DL-Lite$^+$ TBox $\mathcal{T}$ and a conjunctive query $Q = \langle Q_P, \{Q_C\}\rangle$, our goal is to compute a *rewriting* $\mathsf{rew}(Q, \mathcal{T})$—that is, a query such that, for each ABox $\mathcal{A}$, evaluating $\mathsf{rew}(Q, \mathcal{T})$ over $\mathcal{A}$ and evaluating the query Q directly over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ produces exactly the same answers. We derive this algorithm in two steps. In this section, we first show how to compute the set of answers $\mathsf{ans}(Q, \mathcal{K})$ directly; then, in Section 4 we use this result to derive the rewriting algorithm.

It is well known that $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable, where $\Xi(\mathcal{K})$ is the set of clauses obtained by transforming $\mathcal{K}$ as shown in Table 2. 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 outlined by Joyner [12].

Given $\mathcal{K}$ and Q, with $\mathcal{N}$ we denote the set of clauses of the forms shown in Table 2 that can be constructed using the signature of $\mathcal{K}$ and Q. It is not difficult to see that $\mathcal{N}$ is finite assuming that $\mathcal{K}$ and Q are finite. Furthermore, clearly, if we translate $\mathcal{K}$ into a set of clauses $\Xi(\mathcal{K})$, then $\Xi(\mathcal{K}) \cup \{Q_C\} \subseteq \mathcal{N}$. Finally, we saturate $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ using $\mathcal{R}^{DL}$—a suitably parameterized resolution with free selection calculus. Since $\mathcal{R}^{DL}$ is sound and complete, in order to obtain a decision procedure we only need to show that each saturation terminates. This is done in the key Lemma 2, which shows that the resolvent of any two premises in $\mathcal{N}$ by $\mathcal{R}^{DL}$ is also a clause in $\mathcal{N}$. This immediately implies termination: in the worst case, the saturation derives the entire set $\mathcal{N}$, which is finite.

We now formalize our calculus. We first define the set of clauses $\mathcal{N}$ and parameterize suitably the calculus of resolution with free selection.

Definition 1. *Let $\mathcal{K}$ be a DL-Lite$^+$ knowledge base and $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query. The set of clauses $\Xi(\mathcal{K})$ is obtained by transforming $\mathcal{K}$ as shown in Table 2. Furthermore, the set of DL-Lite$^+$ clauses $\mathcal{N}$ is the set of all clauses of types shown in Table 2 constructed using the symbols in Q_C and $\Xi(\mathcal{K})$.*

With $\mathcal{R}^{DL}$ we denote the resolution calculus with free selection parameterized with the following selection function S.

- *In a clause C of type A1–K2, the selection function S selects the atoms that are underlined in Table 2.*
- *In a clause C of type Q1, the selection function S selects the head atom if C contains functional terms in the head or if the body of C is empty; otherwise, S selects all deepest body atoms of C.*

For N a set of clauses, N_∞ is the limit of a fair derivation from N by $\mathcal{R}^{DL}$.

The principles outlined before Definition 1 allow us only to check whether some tuple $\vec{a}$ is an answer to Q over $\mathcal{K}$. Often, however, we need to compute the entire set $\mathsf{ans}(Q, \mathcal{K})$. This can be done using the *answer literal* technique [9]: instead of saturating $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$, we can saturate just $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$. The following lemma shows that by doing so we shall compute all answers to Q over $\mathcal{K}$.

Table 2. Clause Set $\mathcal{N}$ for $Q = \langle Q_P, \{Q_C\}\rangle$ and $\mathcal{K}$

Type	DL-Lite$^+$ clause	DL-Lite$^+$ axiom
A1	$\underline{A(a)}$	$A(a)$
A2	$\underline{P(a,b)}$	$P(a,b)$
T1	$B(x) \leftarrow \underline{A(x)}$	$A \sqsubseteq B$
T2	$\underline{P(x, f^i_A(x))} \leftarrow A(x)$	$A \sqsubseteq \exists P.B$
T3	$\underline{B(f^i_A(x))} \leftarrow A(x)$	
T4	$B(x) \leftarrow \underline{P(x,y)}$	$\exists P \sqsubseteq B$
T5	$B(x) \leftarrow \underline{P(x,y)} \wedge A(y)$	$\exists P.A \sqsubseteq B$
T6	$S(x,y) \leftarrow \underline{P(x,y)}$	$P \sqsubseteq S$
K1	$B(a) \leftarrow \underline{A(b)}$	
K2	$B_2(x) \leftarrow \underline{B_1(f^i_A(x))} \wedge A(x)$	
Q1	$Q_P(\vec{t}) \leftarrow \bigwedge L_i(\vec{t_i})$	

Note 1. We use A and B to denote atomic concepts, P and S for atomic roles, L_i for atomic concepts or roles, and $\vec{t}$ (possibly subscripted) for a tuple of terms. Each axiom of the form $A \sqsubseteq \exists P.B$ is uniquely associated with a function symbol f^i_A. For each clause C of type Q1, (i) $\mathsf{var}(C) \leq \mathsf{var}(Q_C)$, (ii) $\mathsf{depth}(C) \leq \max(1, \mathsf{var}(Q_C) - \mathsf{var}(C))$, and (iii) if a variable x occurs in a functional term in C, then x occurs in all functional terms in C.

Lemma 1. *Let $\mathcal{K}$ be a DL-Lite$^+$ knowledge base, $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query, and $\vec{a}$ a tuple of constants. Then, we have $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ if and only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup \{Q_C\})_\infty$.*

Proof. Clearly, $\mathcal{K} \cup \{Q_C\} \models Q_P(\vec{a})$ iff $\mathcal{K} \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ is unsatisfiable; we now prove that the latter is the case iff $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup \{Q_C\})_\infty$. The $(\Leftarrow)$ direction is trivial. For the $(\Rightarrow)$ direction, note that $\Xi(\mathcal{K}) \cup \{Q_C\}$ does not contain a clause with the empty head, so a saturation of $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$ cannot derive the empty clause. Furthermore, the predicate Q_P does not occur in the body of any clause in $\Xi(\mathcal{K}) \cup \{Q_C\}$; hence, $\mathcal{R}^{DL}$ can derive the empty clause from $\Xi(\mathcal{K}) \cup \{Q_C,\ \bot \leftarrow Q_P(\vec{a})\}$ only if $Q_P(\vec{a}) \in (\Xi(\mathcal{K}) \cup \{Q_C\})_\infty$. □

Thus, to compute $\mathsf{ans}(Q, \mathcal{K})$, we simply need to saturate $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$. The following key lemma shows that such a saturation terminates.

Lemma 2. *For each two clauses $C_1, C_2 \in \mathcal{N}$ and C_r the resolvent of C_1 and C_2 by $\mathcal{R}^{DL}$, we have that $C_r \in \mathcal{N}$.*

Proof. Let C_1 and C_2 be some clauses of $\mathcal{N}$, and let C_r be a resolvent of C_1 and C_2 by $\mathcal{R}^{DL}$. The possible inferences by $\mathcal{R}^{DL}$ on C_1 and C_2 are summarized in Table 3. As can be seen from the table, if C_1 and C_2 are of types A1–K2, then C_r is also of type A1–K2.

Assume that C_1 is of type Q1, satisfying properties (i)–(iii) of Table 2. If the head atom $Q_P(\vec{t})$ of C_1 is selected, then resolution is not possible, since no clause in $\mathcal{N}$ contains Q_P in the body. If a unary body atom $A(t)$ of C_1 is selected, then

Table 3. Inferences of $\mathcal{R}^{DL}$ on $\mathcal{N}$

A1 + T1 = A1:

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

A2 + T4 = A1:

$$\frac{\underline{P(a,b)} \quad B(x) \leftarrow \underline{P(x,y)}}{B(a)}$$

A2 + T5 = K1:

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

A2 + T6 = A2:

$$\frac{\underline{P(a,b)} \quad S(x,y) \leftarrow \underline{P(x,y)}}{S(a,b)}$$

T1 + T3 = T3:

$$\frac{C(x) \leftarrow \underline{B(x)} \quad \underline{B(f_A^i(x))} \leftarrow A(x)}{C(f_A^i(x)) \leftarrow A(x)}$$

Q1 + A1 = Q1:

$$\frac{Q_P(\vec{u}) \leftarrow \underline{A(t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{A(a)}}{Q_P(\vec{u})\sigma \leftarrow \bigwedge L_i(\vec{t_i})\sigma}$$

T2 + T4 = T1:

$$\frac{\underline{P(x, f_A^i(x))} \leftarrow A(x) \quad B(x) \leftarrow \underline{P(x,y)}}{B(x) \leftarrow A(x)}$$

T2 + T5 = K2:

$$\frac{\underline{P(x, f_A^i(x))} \leftarrow A(x) \quad B(x) \leftarrow \underline{P(x,y)} \wedge C(y)}{B(x) \leftarrow C(f_A^i(x)) \wedge A(x)}$$

T2 + T6 = T2:

$$\frac{\underline{P(x, f_A^i(x))} \leftarrow A(x) \quad S(x,y) \leftarrow \underline{P(x,y)}}{S(x, f_A^i(x)) \leftarrow A(x)}$$

K1 + A1 = A1:

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

K2 + T3 = T1:

$$\frac{C(x) \leftarrow \underline{B(f_A^i(x))} \wedge A(x) \quad \underline{B(f_A^i(x))} \leftarrow A(x)}{C(x) \leftarrow A(x)}$$

Q1 + A2 = Q1:

$$\frac{Q_P(\vec{u}) \leftarrow \underline{P(s,t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{P(a,b)}}{Q_P(\vec{u})\sigma \leftarrow \bigwedge L_i(\vec{t_i})\sigma}$$

Q1 + T3 = Q1:

$$\frac{Q_P(\vec{u}) \leftarrow \underline{A(t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{A(f_B^i(x))} \leftarrow B(x)}{Q_P(\vec{u})\sigma \leftarrow B(x)\sigma \wedge \bigwedge L_i(\vec{t_i})\sigma}$$

Q1 + T2 = Q1:

$$\frac{Q_P(\vec{u}) \leftarrow \underline{P(s,t)} \wedge \bigwedge L_i(\vec{t_i}) \quad \underline{P(x, f_A^i(x))} \leftarrow A(x)}{Q_P(\vec{u})\sigma \leftarrow A(x)\sigma \wedge \bigwedge L_i(\vec{t_i})\sigma}$$

Note 2. The notation $A + B = C$ denotes that "resolving a clause of type A with a clause of type B produces a clause of type C."

C_2 can be of type A1 or T3; we now show that C_r satisfies properties (i)–(iii) of Table 2.

– If C_2 is of type A1, unification is possible only if the term t is either a constant a or a variable y. In the former case, the unifier σ is empty; in the latter case, $\sigma = \{y \mapsto a\}$. Clearly, $\mathsf{var}(C_r) \leq \mathsf{var}(C_1)$ and $\mathsf{depth}(C_r) = \mathsf{depth}(C_1)$, so C_r satisfies (i) and (ii). Furthermore, since $A(t)$ is the deepest atom in C_1, the clause C_1 does not contain functional terms, so C_r does not contain them either; hence, C_r satisfies (iii) vacuously.
– If C_2 is of type T3, unification is possible only if the term t is a variable or a functional term.
  - If t is a variable y, then $\sigma = \{y \mapsto f_B^i(x)\}$. Clearly, $\mathsf{var}(C_r) = \mathsf{var}(C_1)$, so C_r satisfies (i). Furthermore, $\mathsf{depth}(C_1) = 0$ and $\mathsf{depth}(C_r) \leq 1$, so C_r satisfies (ii). Finally, every occurrence of y is replaced with $f_B^i(x)$, and C_1 does not contain functional terms, so (iii) holds as well.
  - If t is a functional term $f_B^i(s)$, the unifier is of the form $\sigma = \{x \mapsto s\}$. Clearly, $\mathsf{var}(C_r) = \mathsf{var}(C_1)$, so C_r satisfies (i). Furthermore, since no term in C_1 is deeper than $f_B^i(s)$, we have $\mathsf{depth}(C_r) \leq \mathsf{depth}(C_1)$, so C_r satisfies (ii). Finally, the inference does not introduce new functional terms, so C_r satisfies (iii).

If a binary atom $P(s,t)$ is selected in C_1, then C_2 can be of type A2 or T2. We now show that C_r satisfies properties (i)–(iii) of Table 2.

– If C_2 is of type A2, the unification is possible only if the terms s and t are not functional terms. If they are both constants, the substitution σ is empty; otherwise, σ maps s, t, or both to the corresponding constants in C_2. Clearly, $\mathsf{var}(C_r) \leq \mathsf{var}(C_1)$, so C_r satisfies (i). Furthermore, $\mathsf{depth}(C_r) = \mathsf{depth}(C_1)$, so C_r satisfies (ii). Finally, since $P(s,t)$ is the deepest atom in C_1, the clause C_1 does not contain functional terms, so C_r satisfies (iii) vacuously.
– If C_2 is of type T2, unification is possible only if the term t is a variable or a functional term.
  - If t is a variable x_t, then $\sigma = \{x_t \mapsto f_A^i(s), x \mapsto s\}$. Due to the occurs-check in unification, x_t cannot occur in s. The inference thus decreases the number of variables of C_1 in C_r by one: $\mathsf{var}(C_r) = \mathsf{var}(C_1) - 1$, so C_r satisfies (i). Furthermore, C_1 satisfies (iii), so x_t does not occur in a functional term in C_1 (because it does not occur in s). Hence, even though x_t is mapped to a functional term, $\mathsf{depth}(C_r) = \mathsf{depth}(C_1) + 1$, so C_r satisfies (ii). Finally, since every occurrence of x_t is replaced with $f_A^i(s)$, C_r satisfies (iii) as well.
  - Assume that t is a functional term $f_A^i(t')$. If s does not occur in t', then s is a variable x_s and $\sigma = \{x \mapsto t', x_s \mapsto t'\}$. If s occurs in t', the only way for the inference to be possible is if $t' = s$, so $\sigma = \{x \mapsto t'\}$. In both cases, $\mathsf{var}(C_r) \leq \mathsf{var}(C_1)$ and $\mathsf{depth}(C_r) \leq \mathsf{depth}(C_1)$, so C_r satisfies properties (i) and (ii). Furthermore, the inference does not introduce new functional terms, so C_r satisfies (iii) as well.

This covers all possible forms of C_1 and C_2, so the lemma holds. □

This lemma now straightforwardly implies that a saturation of $\Xi(\mathcal{K}) \cup \{Q_C\}$ terminates, so we can use it to compute $\mathsf{ans}(Q, \mathcal{K})$.

Lemma 3. *For $\mathcal{K}$ a DL-Lite$^+$ knowledge base and $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query, the saturation of $\Xi(\mathcal{K}) \cup \{Q_C\}$ by $\mathcal{R}^{DL}$ terminates.*

Proof. The clause set $\mathcal{N}$ for Q and $\mathcal{K}$ is finite. Moreover, no clause is ever deleted during the saturation process and, by Lemma 2, $\mathcal{N}$ is closed under $\mathcal{R}^{DL}$. Hence, in the worst case, $\mathcal{R}^{DL}$ derives all clauses in $\mathcal{N}$ and then terminates. □

We believe that the result in this section can relatively easily be extended to more expressive DLs, such as $\mathcal{EL}$ extended with inverse roles, universal quantifiers in the implication consequent, and functionality assertions. The key to achieving this is to extend the clause set in Table 2 to cover the new constructors, and to prove that Lemma 2 still holds. We believe we can do this along the lines of [17, 16]; the main technical difficulty is to precisely describe the interaction between the new types of clause and clauses of type Q1. Once this is done, however, the rest of this paper (i.e., the material in the following two sections), should hold with only minor changes. The resulting algorithm would deal with the full spectrum of languages from DL-Lite to extended $\mathcal{EL}$, and we are confident that it would still be optimal with respect to data complexity for all such languages.

4 Rewriting Conjunctive Queries in DL-Lite$^+$

The algorithm from the previous section allows us to compute the answers to a conjunctive query $Q = \langle Q_P, \{Q_C\}\rangle$ over a DL-Lite$^+$ knowledge base $\mathcal{K}$. If we ask the same query over different ABoxes, the algorithm will repeat a lot of unnecessary work, since the query answering algorithm depends on both $\mathcal{T}$ and $\mathcal{A}$. In this section, we present an algorithm for *query rewriting*: given Q and a TBox $\mathcal{T}$, we compute a datalog query $\mathsf{rew}(Q, \mathcal{T})$ such that, for any ABox $\mathcal{A}$, the sets of answers of Q over $\langle \mathcal{T}, \mathcal{A}\rangle$ and of $\mathsf{rew}(Q, \mathcal{T})$ over $\mathcal{A}$ are the same. Thus, our algorithm eliminates the TBox and reduces the problem of answering conjunctive queries in DL-Lite$^+$ to the problem of answering datalog queries.

A distinguishing feature of our algorithm is that it exhibits "pay-as-you-go" behavior: the resulting datalog program $\mathsf{rew}(Q, \mathcal{T})$ is optimal for the input TBox $\mathcal{T}$. If $\mathcal{T}$ is in DL-Lite$^+$, then $\mathsf{rew}(Q, \mathcal{T})$ consists of a union of conjunctive queries and a linear datalog program. We use this fact in Section 5 to establish novel data complexity bounds for conjunctive query answering. Furthermore, if $\mathcal{T}$ is in DL-Lite, then $\mathsf{rew}(Q, \mathcal{T})$ is a union of conjunctive queries. Hence, our algorithm generalizes the approach from [7].

We derive the rewriting algorithm in two phases: in Section 4.1, we show how to convert $\Xi(\mathcal{T})$ into a nonoptimal datalog program by eliminating function symbols; then, in Section 4.2 we present an additional step that ensures that the rewritten query is of optimal form.

4.1 Elimination of Function Symbols

The following definition summarizes the first step of our rewriting algorithm.

Definition 2. *For $Q = \langle Q_P, \{Q_C\}\rangle$ a conjunctive query and $\mathcal{T}$ a DL-Lite$^+$ TBox, $\mathsf{ff}(Q,\mathcal{T})$ is the set that contains exactly all function-free clauses contained in $(\Xi(\mathcal{T}) \cup \{Q_C\})_\infty$.*

We next show that, for each ABox $\mathcal{A}$, we have $\{Q_C\} \cup \mathcal{T} \cup \mathcal{A} \models Q_P(\vec{a})$ if and only if $\mathsf{ff}(Q,\mathcal{T}) \cup \mathcal{A} \models Q_P(\vec{a})$. Thus, $\mathsf{ff}(Q,\mathcal{T})$ is a rewriting of Q and $\mathcal{T}$, albeit not necessarily an optimal one. We prove the claim proof-theoretically: we show that $Q_P(\vec{a})$ is derivable from $\{Q_C\} \cup \mathcal{T} \cup \mathcal{A}$ if and only if it is derivable from $\mathsf{ff}(Q,\mathcal{T}) \cup \mathcal{A}$. To this end, we first prove that we can always "postpone" the inferences with the ABox clauses in the saturation of $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$—that is, we can first perform all inferences with nonground clauses only, and then perform the inferences involving a ground clause.

Lemma 4. *Let $Q = \langle Q_P, \{Q_C\}\rangle$ be a conjunctive query, $\mathcal{T}$ a DL-Lite$^+$ TBox, and $\mathcal{A}$ an ABox. For each clause C of type Q1 derivable from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$, a clause C' of type Q1 is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$ such that, for G the subset of all clauses of type A1 and A2 in $(\mathsf{ff}(Q,\mathcal{T}) \cup \mathcal{A})_\infty$, we have $\{C'\} \cup G \models C$.*

Proof. We prove the claim by induction on the height of a derivation tree by which C is derived from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$. If the derivation tree has height zero, then C must be the clause Q_C, so the claim follows trivially for $C' = Q_C$. Assume that the claim holds for each clause derived from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$ by a derivation tree of height n, and consider a clause C derived by a derivation tree of height $n+1$. The clause C is obtained by resolving some clauses C_1 and C_2. According to Table 3, one of the premises has to be of type Q1, so we denote it by C_1; the other premise C_2 can be of type A1, A2, T2, or T3. By the induction hypothesis, some clause C_1' of type Q1 is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$ such that $\{C_1'\} \cup G \models C_1$. We now consider the different forms that C_2 can have.

Assume that C_2 is of type A1 or A2. From Table 3 we can see that each derivation of a clause of type A1 or A2 involves only function-free clauses, so $C_2 \in G$. The inductive claim now trivially holds for $C' = C_1'$.

Assume that C_2 is of type T2 or T3. We assume that it is of the form $L(\vec{t}) \leftarrow A(x)$; then, C_1 must contain in the body a counterpart atom $L(\vec{q})$. By examining the inferences between DL-Lite$^+$ clauses shown in Table 3, we can see that C_2 is derivable from $\Xi(\mathcal{T})$. Note that G contains only ground clauses of types A1 and A2; thus, since $\{C_1'\} \cup G \models C_1$, a subset $\{G_i(\vec{a_i})\} \subseteq G$ exists such that resolving C_1' on body literals $\{G_i(\vec{g_i})\}$ with the elements of $\{G_i(\vec{a_i})\}$ produces C_1. Furthermore, all such resolution inferences just remove body atoms. Therefore, if C_1 is to contain the atom $L(\vec{q})$ in the body, the clause C_1' must contain an atom $L(\vec{s})$ in its body. Hence, C_1' is of the form (1), and C_1 is of the form (2), where δ maps some variables to constants in $\{G_i(\vec{a_i})\}$ such that $L(\vec{s})\delta = L(\vec{q})$. Finally, resolving C_1 and C_2 produces the clause C, which is of the form (3) for $\sigma = \mathsf{MGU}(L(\vec{s})\delta, L(\vec{t}))$.

$$C_1' = Q_P(\vec{u}) \leftarrow L(\vec{s}) \wedge \bigwedge G_i(\vec{g_i}) \wedge \bigwedge M_j(\vec{m_j}) \tag{1}$$

$$C_1 = Q_P(\vec{u})\delta \leftarrow L(\vec{s})\delta \wedge \bigwedge M_j(\vec{m_j})\delta \tag{2}$$

$$C = Q_P(\vec{u})\delta\sigma \leftarrow A(x)\sigma \wedge \bigwedge M_j(\vec{m_j})\delta\sigma \tag{3}$$

Note that no inference used to derive C_1 changes the number of function symbols of C_1'; therefore, $L(\vec{s})$ is the deepest literal of C_1'. Furthermore, each variable of C_1' that is replaced by δ with a constant clearly does not occur in $L(\vec{s})\delta$; hence, the substitutions δ and σ have disjoint domains, and $\sigma = \delta\sigma$.

We now transform this derivation into a derivation in which all inferences with ABox clauses are performed after all inferences with only TBox clauses. Let C' be the clause obtained by resolving C_1' and C_2; we can assume that C' has the form (4), where $\sigma' = \mathsf{MGU}(L(\vec{s}), L(\vec{t}))$.

$$C' = Q_P(\vec{u})\sigma' \leftarrow A(x)\sigma' \wedge \bigwedge G_i(\vec{g_i})\sigma' \wedge \bigwedge M_j(\vec{m_j})\sigma' \tag{4}$$

Since $L(\vec{s})$ is the deepest literal of C_1', the inference between C_1' and C_2 satisfies the selection function of the calculus $\mathcal{R}^{DL}$. 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 x be a variable that occurs in $L(\vec{s})$ and that is replaced by δ with a constant. Clearly, σ does not contain such x; hence, without loss of generality, we can assume (*) that σ' does not contain such variables either—that is, instead of mapping x to a term in $L(\vec{t})$, we can assume that the corresponding term is mapped to x.

Let D now be the clause obtained by resolving the literals $G_i(\vec{g_i})\sigma'$ in C' with the ground clauses $\{G_i(\vec{a_i})\}$. This inference is possible due to assumption (*), so D has the following form, where δ' maps some variables of C' to constants.

$$D = Q_P(\vec{u})\sigma'\delta' \leftarrow A(x)\sigma'\delta' \wedge \bigwedge M_j(\vec{m_j})\sigma'\delta' \tag{5}$$

Due to (*), σ and σ' have the same domain which is disjoint with the domain of δ, so $\sigma = \sigma'\delta$. None of the variables occurring in $\{G_i(\vec{g_i})\}$ is in the domain of σ', so $\delta = \delta'$. Since $\sigma = \sigma'\delta$ and $\delta = \delta'$, we have $\sigma = \sigma'\delta'$. Moreover, since $\sigma = \delta\sigma$, we have $\sigma = \delta\sigma = \sigma'\delta'$, so $C = D$, which proves our claim. $\square$

This lemma now allows us to prove the desired relationship between the answers of Q on $\mathcal{T}$ and $\mathcal{A}$ and the answers of $\mathsf{ff}(Q, \mathcal{T})$ on $\mathcal{A}$.

Lemma 5. *Let $Q = \langle Q_P, \{Q_C\}\rangle$ be a conjunctive query, $\mathcal{T}$ a DL-Lite$^+$ TBox, and $\mathcal{A}$ an ABox. Then, $\vec{a} \in \mathsf{ans}(Q, \langle \mathcal{T}, \mathcal{A}\rangle)$ if and only if $\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A} \models Q_P(\vec{a})$.*

Proof. ($\Leftarrow$) Note that $\mathsf{ff}(Q, \mathcal{T}) \cup \mathcal{A} \subseteq (\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A})_\infty$, which trivially implies this direction of the claim.

($\Rightarrow$) Assume that $Q_P(\vec{a})$ is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\} \cup \mathcal{A}$. Since $Q_P(\vec{a})$ is of type Q1, by Lemma 4, a clause C' of type Q1 is derivable from $\Xi(\mathcal{T}) \cup \{Q_C\}$

such that, for G the subset of all clauses of type A1 and A2 in $(\mathsf{ff}(Q,\mathcal{T}) \cup \mathcal{A})_\infty$, we have $\{C'\} \cup G \models C$. Since $Q_P(\vec{a})$ does not contain function symbols, C' cannot contain function symbols either, so $C' \in \mathsf{ff}(Q,\mathcal{T})$. Thus, $Q_P(\vec{a})$ is implied by $\mathsf{ff}(Q,\mathcal{T}) \cup G$ so, by the definition of G, we have $\mathsf{ff}(Q,\mathcal{T}) \cup \mathcal{A} \models Q_P(\vec{a})$. □

4.2 Optimizing the Rewriting through Unfolding

By Lemma 5, the datalog program $\mathsf{ff}(Q,\mathcal{T})$ is a rewriting of Q w.r.t. $\mathcal{T}$; however, it is not necessarily optimal for the TBox $\mathcal{T}$ at hand. In particular, the program $\mathsf{ff}(Q,\mathcal{T})$ can contain clauses of type T1, T4, and T6; hence, we must assume that each predicate in $\mathsf{ff}(Q,\mathcal{T})$ can be an IDB predicate, so a clause of type T4 is not a linear datalog rule. Our goal, however, is to ensure that the rewriting consists of a linear datalog program and a union of conjunctive queries; furthermore, if $\mathcal{T}$ is a DL-Lite TBox, the rewriting should be a union of conjunctive queries only. Thus, in this section we introduce a further *unfolding* step that transforms $\mathsf{ff}(Q,\mathcal{T})$ into a datalog program of an optimal form.

Definition 3. *The* unfolding *of* $L(\vec{x}) \leftarrow \bigwedge M_i(\vec{m_i})$ *in* $N(\vec{n}) \leftarrow L(\vec{x'}) \wedge \bigwedge P_j(\vec{p_j})$ *is the clause* $N(\vec{n})\sigma \leftarrow \bigwedge M_i(\vec{m_i})\sigma \wedge \bigwedge P_j(\vec{p_j})\sigma$, *where* $\sigma = \mathsf{MGU}(L(\vec{x}), L(\vec{x'}))$.

Given two sets of safe Horn clauses R *and* U, *let* R_U *be the smallest set such that* $R \subseteq R_U$ *and, for each unfolding* C_r *of a clause* $C_1 \in R \cap U$ *in a clause* $C_2 \in R$, *we have that* $C_r \in R_U$. *The* unfolding *of* R *w.r.t.* U *is defined as* $\mathsf{unfold}(R,U) = R_U \setminus U$.

We shall apply the unfolding step for $R = \mathsf{ff}(Q,\mathcal{T})$ and U the set of all clauses of types T1, T4, and T6. Since unfolding eliminates all clauses of type T6, all atomic roles thus become EDB predicates; thus, the resulting set of clauses, apart from the clauses of type Q1, is a linear datalog program. Moreover, since all clauses of types T1 and T4 are also eliminated, the resulting set of clauses becomes a union of conjunctive queries whenever $\mathcal{T}$ is a DL-Lite TBox. Before proceeding, however, we show that unfolding does not change the set of "relevant" consequences of a datalog program.

Lemma 6. *Let* R *and* U *be sets of safe Horn clauses. For any set of facts* A *and for any predicate* F *that does not occur in* U, *we have* $R \cup A \models F(\vec{a})$ *if and only if* $\mathsf{unfold}(R,U) \cup A \models F(\vec{a})$.

Proof. ($\Leftarrow$) 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 $R \cup A \models C$.

($\Rightarrow$) Let $\mathcal{H}$ be the hyperresolution calculus—that is, the resolution calculus in which all the body literals are selected whenever possible, and in which all selected literals are resolved in one step. It is well known that, if $R \models F(\vec{a})$, then a derivation tree T for $F(\vec{a})$ from R by $\mathcal{H}$ exists [4]. We represent such a tree T as a tuple $\langle T_N, \delta, \lambda \rangle$ for T_N the set of nodes where we denote with $t.i$ the i-th child of $t \in T_N$ and with ϵ the root node; δ is a function that maps each node $t \in T_N$ to a fact $\delta(t)$; and λ is a function that maps each node $t \in T_N$ to a clause

$\lambda(t)$ such that $\delta(t)$ is derived by hyperresolving each $\delta(t.i)$ with the i-th body literal of $\lambda(t)$. If t is a leaf node, then $\delta(t) \in A$ and $\lambda(t)$ is undefined.

We now inductively define a function $\sigma(t)$ as follows: starting from the leaves upwards, for each $t \in T_N$, 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 $\sigma(t.i) \notin U$ or $\delta(t.i) \in A$; furthermore, we call t a *surviving* node iff $\sigma(t) \notin U$ 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 is easy to see that, for each node t, the fact $\delta(t)$ can be derived by hyperresolving $\sigma(t)$ with the set of facts $\{\delta(t_1), \ldots, \delta(t_n)\}$, where $t_1, \ldots, t_n$ are exactly all the closest surviving nodes to t. Note that, for every node $t \in T_N$, 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 F does not occur in U, we have $F(\vec{a}) \notin U$. Furthermore, $\delta(\epsilon) = F(\vec{a})$, so the clause $\sigma(\epsilon)$ contains F in the head, and $\sigma(\epsilon) \notin U$. Thus, ϵ is a surviving node, so $\delta(\epsilon)$ can be derived from $\mathsf{unfold}(R, U) \cup A$. □

We are now ready to define the rewriting of a conjunctive query Q with respect to a TBox $\mathcal{T}$ expressed in DL-Lite$^+$.

Definition 4. *The* rewriting $\mathsf{rew}(Q, \mathcal{T})$ *of a conjunctive query* $Q = \langle Q_P, \{Q_C\}\rangle$ *w.r.t. a DL-Lite*$^+$ *TBox* $\mathcal{T}$ *is the query* $\langle Q_P, \mathsf{unfold}(R, U)\rangle$*, where* $R = \mathsf{ff}(Q, \mathcal{T})$ *and* U *is the subset of* $\mathcal{N}$ *of all clauses of type T1, T4, and T6.*

We now state the main property of the reduction algorithm.

Theorem 1. *For a conjunctive query* Q*, a DL-Lite*$^+$ *TBox* $\mathcal{T}$*, and an ABox* $\mathcal{A}$*, we have* $\mathsf{ans}(Q, \langle \mathcal{T}, \mathcal{A}\rangle) = \mathsf{ans}(\mathsf{rew}(Q, \mathcal{T}), \mathcal{A})$.

Proof. Without loss of generality, we can assume that Q_P does not occur in $\Xi(\mathcal{T})$. Then, the claim of this theorem follows straightforwardly from Lemmata 5 and 6. □

We now prove two important properties about the structure of the rewriting. We use these properties in Section 5 to prove complexity results about answering conjunctive queries over DL-Lite$^+$.

Lemma 7. *Let* $Q = \langle Q_P, \{Q_C\}\rangle$ *be a conjunctive query,* $\mathcal{T}$ *a DL-Lite*$^+$ *TBox, and* $\mathsf{rew}(Q, \mathcal{T}) = \langle Q_P, P\rangle$*. Then,* P *can be split into disjoint subsets* U_Q *and* U_C *such that* $\langle Q_P, U_Q\rangle$ *is a union of conjunctive queries and* $\langle Q_P, U_C\rangle$ *is a linear datalog query.*

Proof. Let $U_Q \subseteq P$ be the set of all clauses of type Q1 in P. By the definition of clauses of type Q1, Q_P is the head predicate of every clause in U_Q, and Q_P does not appear in the body of a clause in U_Q, so $\langle Q_P, U_Q\rangle$ is a union of conjunctive queries. Let $U_C = P \setminus U_Q$. The program $\mathsf{ff}(Q, \mathcal{T})$ contains clauses of types T1, T4, T5, and T6. Hence, U_C is obtained by unfolding clauses of types T1, T4, and

T6 in clauses of types T1, T4, T5, and T6, and then by removing all clauses of types T1, T4, and T6. Thus, U_C contains clauses of type T5 and clauses of the form $B(x) \leftarrow P(x,y) \wedge S(y,z)$ that are obtained by unfolding a clause of type T4 in a clause of type T5. Clearly, no clause in U_C contains a role predicate in the head, so all role predicates are EDB predicates. Furthermore, clauses of type T5 can contain a unary predicate in the head, so unary predicates can be IDB predicates; however, IDB predicates can occur only in a clause of type T5 in the body, so all such clauses are linear. Thus, U_C is a linear datalog program. $\square$

Lemma 8. *For $Q = \langle Q_P, \{Q_C\} \rangle$ a conjunctive query and $\mathcal{T}$ a DL-Lite TBox, $\mathsf{rew}(Q, \mathcal{T})$ is a union of conjunctive queries.*

Proof. Let $\mathsf{rew}(Q, \mathcal{T}) = \langle Q_P, P \rangle$. Since $\mathcal{T}$ is a DL-Lite TBox, the set $\Xi(\mathcal{T})$ does not contain clauses of type T5. Thus, $\mathsf{ff}(Q, \mathcal{T})$ contains only clauses of types Q1, T1, T4, and T6. Clauses of types T1, T4, and T6 are unfolded in clauses of type Q1, so $\mathsf{rew}(Q, \mathcal{T})$ is a union of conjunctive queries. $\square$

5 Complexity Analysis

It is well known that the problem of deciding $P \models A(\vec{a})$ for P a linear datalog program is NLogSpace-complete with respect to data complexity [8]. We were not able to find in the literature a generalization of this result for the case where P consists of a linear datalog program and a union of conjunctive queries; therefore, before proceeding, we show that this is indeed the case.

Lemma 9. *For $Q = \langle Q_P, Q_C \rangle$ a union of conjunctive queries, P a linear datalog program, and A a set of facts, deciding $P \cup Q_C \cup A \models Q_P(\vec{a})$ can be performed in* NLogSpace *in the size of A.*

Proof. If $P \cup Q_C \cup A \models Q_P(\vec{a})$, then $Q_P(\vec{a})$ can be derived from the set of clauses $P \cup Q_C \cup A$ using SLD resolution [4]. 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 $R \in P$ 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 P and A multiplied by the number of the query rules 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 $P \cup Q_C \cup A$ 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.

Since we are interested in data complexity, 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 rules in $P \cup Q_C$, 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. $\square$

We now apply Lemma 9 to show that answering conjunctive queries over DL-Lite$^+$ knowledge bases is NLogSpace-complete.

Theorem 2. *For a conjunctive query $Q = \langle Q_P, \{Q_C\} \rangle$ and a DL-Lite$^+$ knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, deciding whether $\vec{a} \in \mathsf{ans}(Q, \mathcal{K})$ is* NLogSpace*-complete w.r.t. data complexity.*

Proof. In [6], it was shown that checking entailment of a ground concept assertion is NLogSpace-hard if we allow for assertions of the form $\exists P.A \sqsubseteq B$. Membership follows immediately from Theorem 1, Lemmata 7 and 9, and the observation that the size of $\mathsf{rew}(Q, \mathcal{T})$ does not depend on the size of $\mathcal{A}$. $\square$

By Lemma 8, if $\mathcal{T}$ is a DL-Lite TBox, then $\mathsf{rew}(Q, \mathcal{T})$ is a union of conjunctive queries, so we can compute answers to $\mathsf{rew}(Q, \mathcal{T})$ over $\mathcal{A}$ in LogSpace with respect to data complexity [13], just as is the case in [7].

6 Conclusion

Motivated by the use of DL ontologies in Information Integration systems, we have presented a resolution-based algorithm for rewriting conjunctive queries over DL-Lite$^+$ TBoxes. We have used this algorithm to show that query answering in DL-Lite$^+$ can be implemented in NLogSpace w.r.t. data complexity. Together with the hardness result from [6], it follows that query answering in DL-Lite$^+$ is NLogSpace-complete with respect to data complexity,which closes what was, to the best of our knowledge, an open problem. Moreover, we have shown that our algorithm exhibits good "pay-as-you-go" behavior: on the subset of DL-Lite$^+$ for which query answering is in LogSpace, our algorithm is also worst-case optimal.

As part of our future work, we plan to extend the technique to deal with more expressive DLs, and in particular with an extended version of $\mathcal{EL}$; a sketch describing how this could be done was given at the end of Section 3. Such an algorithm would be optimal for the full spectrum of languages from DL-Lite to extended $\mathcal{EL}$—that is, languages for which the data complexity of query answering ranges from LogSpace to PTime-complete. Finally, we plan to implement our query answering technique in a prototype Information Integration system—we have established a promising relationship with researchers at the University of Newcastle who are using Information Integration in their ComparaGRID project,[1] and we plan to use ComparaGRID as an evaluation framework for our prototype system.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope. In *Proc. IJCAI-05*, pages 364–369, Edinburgh, Scotland, 2005.
2. F. Baader and W. Nutt. *Basic Description Logics*, chapter 2, pages 47–100. Cambridge University Press, 2003.
3. F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.
4. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–100. North Holland, 2001.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. MASTRO-I: Efficient integration of relational data through DL ontologies. In *DL-07*, 2007.
6. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in Description Logics. In *Proc. DL 2005*, 2005.
7. 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. *J. of Automated Reasoning*, 9:385–429, 2007.
8. E. Grädel. Capturing complexity classes by fragments of second-order logic. *Theor. Comput. Sci.*, 101:35–57, 1992.
9. 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*, pages 183–208. Edinburgh University Press, 1969.
10. A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
11. J. Heflin and J. Hendler. A portrait of the semantic web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.
12. W. H. Joyner. Resolution strategies as decision procedures. *J. ACM*, 23(3):398–417, 1976.
13. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *Proc. PODS '90*, pages 299–313, 1990.
14. J. Lee, K. Siau, and S. Hong. Enterprise integration with ERP and EAI. *Commun. ACM*, 46(2):54–60, 2003.
15. M. Lenzerini. Data Integration: a theoretical perspective. In *Proc. PODS '02*, pages 233–246, New York, NY, USA, 2002. ACM Press.
16. B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
17. H. De Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-Based Methods for Modal Logics. *Logic Journal of the IGPL*, 8(3):265–292, 2000.
18. R. Rosati. On conjunctive query answering in EL. In *DL-07*. CEUR Electronic Workshop Proceedings, 2007.

[1] `http://www.comparagrid.org`