Data Complexity of Query Answering in Description Logics

Diego Calvanese[1], **Giuseppe De Giacomo**[2], **Domenico Lembo**[2], **Maurizio Lenzerini**[2], **Riccardo Rosati**[2]

[1] Faculty of Computer Science
Free University of Bozen-Bolzano
Piazza Domenicani 3
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

[2] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113
I-00198 Roma, Italy
lastname@dis.uniroma1.it

Abstract

In this paper we study data complexity of answering conjunctive queries over Description Logic knowledge bases constituted by an ABox and a TBox. In particular, we are interested in characterizing the FOL-reducibility and the polynomial tractability boundaries of conjunctive query answering, depending on the expressive power of the Description Logic used to specify the knowledge base. FOL-reducibility means that query answering can be reduced to evaluating queries over the database corresponding to the ABox. Since first-order queries can be expressed in SQL, the importance of FOL-reducibility is that, when query answering enjoys this property, we can take advantage of Data Base Management System (DBMS) techniques for both representing data, i.e., ABox assertions, and answering queries via reformulation into SQL. What emerges from our complexity analysis is that the Description Logics of the *DL-Lite* family are the maximal logics allowing conjunctive query answering through standard database technology. In this sense, they are the first Description Logics specifically tailored for effective query answering over very large ABoxes.

Introduction

The idea of using ontologies as a conceptual view over data repositories is becoming more and more popular. For example, in Enterprise Application Integration(Lee, Siau, & Hong 2003), Data Integration (Lenzerini 2002), and the Semantic Web (Heflin & Hendler 2001), the intensional level of the application domain can be profitably represented by an ontology, so that clients can rely on a shared conceptualization when accessing the services provided by the system. In these contexts, the set of instances of the concepts in the ontology is to be managed in the data layer of the system architecture (e.g., in the lowest of the three tiers of the Enterprise Software Architecture), and, since instances correspond to the data items of the underlying information system, such a layer constitutes a very large (much larger than the intensional level of the ontology) repository, to be stored in secondary storage (see (Borgida *et al.* 1989)).

When clients access the application ontology, it is very likely that one of the main services they need is the one of answering complex queries over the extensional level of the ontology (obviously making use of the intensional level as well in producing the answer). Here, by complex we mean that it does not suffice to ask for the instances of concepts, but we need at least expressing conjunctive conditions on the extensional level. Given the size of the instance repository, when measuring the computational complexity of query answering (and reasoning in general) the most important parameter is the size of the data. In other words, we are interested in the so-called *data complexity* of query answering.

In this paper we consider conjunctive queries (CQs) specified over ontologies expressed in Description Logics (DL), and study the data complexity of the query answering problem. Since an ontology in DL is essentially a knowledge base (KB) constituted by a TBox and an ABox, the problem we address is the one of computing the answers to a CQ that are logical consequences of the TBox and the ABox, where complexity is measured with respect to the size of the ABox only. Note that we borrow the notion of data complexity from the database literature (Vardi 1982), on the premise that an ABox can be naturally viewed as a relational database.

We are interested in characterizing the FOL-reducibility and the polynomial tractability boundaries of conjunctive query answering, depending on the expressive power of the DL used to specify the KB. We say that query answering is FOL-reducible in a DL $\mathcal{L}$, if for every conjunctive query q over an $\mathcal{L}$ TBox $\mathcal{T}$, there is a first-order query q' such that for all ABoxes $\mathcal{A}$ the answers to q with respect to the KB $(\mathcal{T}, \mathcal{A})$ are the same as the answers to q' over the database corresponding to the ABox $\mathcal{A}$. Since first-order queries can be expressed in SQL, the importance of FOL-reducibility is that, when query answering enjoys this property, we can take advantage of Data Base Management System (DBMS) techniques for both representing data, i.e., ABox assertions, and answering queries via reformulation into SQL[1]. Notably, in this case, the data complexity of conjunctive query answering over ontologies is the one of FOL queries over databases, i.e., LOGSPACE.

We are also interested to know for which DLs we go beyond FOL. For this purpose, we consider the LOGSPACE boundary of the problem. Indeed, we single out those DLs for which query answering becomes NLOGSPACE-

[1] We consider here the kernel of the SQL-92 standard, i.e., we see SQL as an implementation of relational algebra.

hard and PTIME,-hard respectively. From the complexity characterization of query languages, it follows that those DLs require at least the power of linear recursive Datalog (NLOGSPACE), and general recursive Datalog (PTIME). Note that, although very interesting and promising Datalog engines exist, query optimization strategies for this query language are not sufficiently mature yet to deal with complex applications with millions of instances in the extensional level. Finally, we address the problem of going even beyond PTIME, by exhibiting DLs for which query answering is polynomially intractable.

More precisely, the contributions of the paper are the following.

- We discuss DLs for which conjunctive query answering is FOL-reducible. In this class, we essentially find the family of *DL-Lite* (Calvanese *et al.* 2005) languages. Notably, the two simplest DLs of this family (namely, *DL-Lite*$_{\mathcal{F},\sqcap}$ and *DL-Lite*$_{\mathcal{R},\sqcap}$) are rich enough to express basic ontology languages, e.g., extensions of (the DL subset of) RDFS[2] or fragments of OWL-DL[3]; conceptual data models, e.g., Entity-Relationship (Abiteboul, Hull, & Vianu 1995); and object-oriented formalisms, e.g., basic UML class diagrams[4]. We also show that we can extend these two languages by adding n-ary relations, and still retain FOL-reducibility. We show that the DLs of the *DL-Lite* family are maximally expressive DLs for which query answering is FOL reducible.
- We show that minimal additions to the languages considered above bring data complexity of conjunctive query answering to NLOGSPACE-hardness and PTIME-hardness, thus losing the possibility of reformulating queries in first-order logic. In spite of the fact that we conjecture that for such languages query answering is polynomially tractable (in NLOGSPACE and PTIME, respectively), these hardness results tell us that in query answering we cannot take advantage of state-of-the-art database query optimization strategies, and this might hamper practical feasibility for very large ABoxes.
- Finally, we establish coNP-hardness of conjunctive query answering with respect to data complexity for surprisingly simple DLs. In particular, we show that we get intractability as soon as the DL is able to express simple forms of union.

What emerges from our complexity analysis is that the DLs of the *DL-Lite* family are "maximal" DLs which allow for answering conjunctive queries through standard database technology. In this sense, they are the first DLs specifically tailored for effective query answering over large amounts of data.

The paper is organized as follows. In the next section we introduce some preliminaries which will be useful for the subsequent discussion. In the sections on FOL-reducibility of *DL-Lite* and *DLR-Lite* we present DLs for which query answering is FOL-reducible. Then, we deal with DLs for which query answering goes beyond LOGSPACE: we first identify DLs for which query answering is NLOGSPACE-hard; then we characterize DLs for which query answering is PTIME-hard; and finally we identify DLs for which query answering is coNP-hard. In the last two sections we overview related work, and we draw some conclusions.

[2] http://www.w3.org/TR/rdf-schema/

[3] http://www.w3.org/TR/owl-features/

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

Preliminaries

Description Logics (DLs) (Baader *et al.* 2003) are logics that represent the domain of interest in terms of *concepts*, denoting sets of objects, and *roles*, denoting binary relations between (instances of) concepts. Complex concept and role expressions are constructed starting from a set of atomic concepts and roles by applying suitable constructs. Different DLs allow for different constructs. In this paper, we distinguish between the constructs that are allowed in the concepts in the left-hand side (Cl) and those in the right-hand side (Cr) of inclusion assertions (see later).

As a concrete example of a DL, we focus on *DL-Lite*$_{core}$, which serves as core language for the family of *DL-Lite* languages discussed in the rest of the paper. The language for *DL-Lite*$_{core}$ concepts and roles is defined as follows:

$$\begin{array}{rcl} Cl & \longrightarrow & A \mid \exists R \\ Cr & \longrightarrow & A \mid \exists R \mid \neg A \mid \neg \exists R \\ R & \longrightarrow & P \mid P^- \end{array}$$

where Cl (resp., Cr) denotes a concept used in the left-hand side (resp., right-hand side) of an inclusion assertion, A denotes an atomic concept, P an atomic role, and P^- its inverse.

We observe that we might include $Cl_1 \sqcup Cl_2$ in the constructs for the left-hand side of the inclusion assertions and $Cr_1 \sqcap Cr_2$ in the constructs for the right-hand side. In this way, however, we would not extend the expressive capabilities of the language, since these constructs can be simulated by considering that $Cl_1 \sqcup Cl_2 \sqsubseteq Cr$ is equivalent to the pair of assertions $Cl_1 \sqsubseteq Cr$ and $Cl_2 \sqsubseteq Cr$, and that $Cl \sqsubseteq Cr_1 \sqcap Cr_2$ is equivalent to $Cl \sqsubseteq Cr_1$ and $Cl \sqsubseteq Cr_2$. Similarly, we might add $\bot$ to the constructs for the left-hand side and $\top$ to those for the right-hand side.

The semantics of a DL is given in terms of interpretations, where an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of an interpretation domain $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation over $\Delta^{\mathcal{I}}$. In particular for the constructs of *DL-Lite*$_{core}$ we have:

$$\begin{array}{rcl} A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\ P^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\ (P^-)^{\mathcal{I}} & = & \{(o_2, o_1) \mid (o_1, o_2) \in P^{\mathcal{I}}\} \\ (\exists R)^{\mathcal{I}} & = & \{o \mid \exists o' . (o, o') \in R^{\mathcal{I}}\} \\ (\neg A)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\ (\neg \exists R)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus \exists R^{\mathcal{I}} \end{array}$$

A DL *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ represents the domain of interest and consists of two parts, a *TBox* $\mathcal{T}$, representing intensional knowledge, and an *ABox* $\mathcal{A}$, representing extensional knowledge. A TBox is formed by a set of *inclusion assertions* of the form

$$Cl \sqsubseteq Cr$$

where Cl and Cr are formed using the constructs allowed by the particular DL used, e.g., for *DL-Lite*$_{core}$ we can have the constructs described above. Such an inclusion assertion expresses that all instances of concept Cl are also instances of concept Cr. Apart from the above inclusion assertions, some DLs that we consider in this paper, and that go beyond *DL-Lite*$_{core}$ allow for other forms of assertions in the TBox (see later).

An ABox is formed by a set of *membership assertions* on atomic concepts and on atomic roles:

$$A(a), \qquad P(a_1, a_2)$$

stating respectively that the object (denoted by the constant) a is an instance of A and that the pair (a_1, a_2) of objects is an instance of the role P. Other forms of ABoxes have also been proposed (Baader *et al.* 2003), but we will not consider them in this paper.

We now specify the semantics of both inclusion and membership assertions. An interpretation $\mathcal{I}$ is a *model* of an inclusion assertion $Cl \sqsubseteq Cr$ if $Cl^{\mathcal{I}} \subseteq Cr^{\mathcal{I}}$. To specify the semantics of membership assertions, we extend the interpretation function to constants, by assigning to each constant a a *distinct* object $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Note that this implies that, as usual in DLs, we enforce the *unique name assumption* on constants (Baader *et al.* 2003). An interpretation $\mathcal{I}$ is a model of a membership assertion $A(a)$ (resp., $P(a_1, a_2)$) if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $(a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in P^{\mathcal{I}}$). A *model of a KB* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is an interpretation $\mathcal{I}$ that is a model of all assertions in $\mathcal{T}$ and $\mathcal{A}$. A KB is *satisfiable* if it has at least one model. A KB $\mathcal{K}$ *logically implies* (an assertion) α, written $\mathcal{K} \models \alpha$, if all models of $\mathcal{K}$ are also models of α.

We can extract information from the extensional level of a KB $\mathcal{K}$ expressed in a DL, by using queries. In particular we will concentrate on conjunctive queries: a *conjunctive query* $q(\vec{x})$ over a KB $\mathcal{K}$ is an expression of the form

$$\{\, \vec{x} \mid conj(\vec{x}, \vec{y}) \,\}$$

where $\vec{x}$ are the so-called distinguished variables (which will be bound with objects in the KB), $\vec{y}$ are the non-distinguished variables (which are existentially quantified), and $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $A(z)$ or $P(z_1, z_2)$ where A and P are respectively atomic concepts and roles of $\mathcal{K}$ and z, z_1, z_2 are either constants in $\mathcal{K}$ or variables in $\vec{x}$ or $\vec{y}$.

Given an interpretation $\mathcal{I}$, the conjunctive query $q(\vec{x}) = \{\vec{x} \mid conj(\vec{x}, \vec{y})\}$ is interpreted as the set $q^{\mathcal{I}}$ of tuples $\vec{o}$ of objects such that, when assigning $\vec{o}$ to $\vec{x}$, the first-order formula $\exists \vec{y}.conj(\vec{x}, \vec{y})$ evaluates to true in $\mathcal{I}$.

The reasoning service we are interested in is *(conjunctive) query answering*: given a knowledge base $\mathcal{K}$ and a conjunctive query $q(\vec{x})$ over $\mathcal{K}$, return all tuples $\vec{a}$ of constants in $\mathcal{K}$ such that, when substituted to the variables $\vec{x}$ in $q(\vec{x})$, we have that $\mathcal{K} \models q(\vec{a})$, i.e., such that $\vec{a}^{\mathcal{I}} \in q^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$. We observe that query answering (properly) generalizes a well known reasoning service in DLs, namely *instance checking*, i.e., logical implication of an ABox assertion. In particular, instance checking can be expressed as the problem of answering (boolean) conjunctive queries constituted by just one ground atom.

Finally, we refer to *data complexity* of query answering, which is a notion borrowed from relational database theory (Vardi 1982). First, we note that there is a recognition problem associated with query answering, which is defined as follows. We have a fixed TBox $\mathcal{T}$ expressed in a DL $\mathcal{L}$, and a fixed query q: the *recognition problem* associated to $\mathcal{T}$ and q is the decision problem of checking whether, given an ABox $\mathcal{A}$, and a tuple $\vec{a}$ of constants, we have that $(\mathcal{T}, \mathcal{A}) \models q(\vec{a})$. Note that neither the TBox nor the query is an input to the recognition problem.

Let $\mathcal{S}$ be a complexity class. When we say that query answering for a certain DL $\mathcal{L}$ *is in* $\mathcal{S}$ *with respect to data complexity*, we mean that the corresponding recognition problem is in $\mathcal{S}$. Similarly, when we say that query answering for a certain DL $\mathcal{L}$ is $\mathcal{S}$*-hard with respect to data complexity*, we mean that the corresponding recognition problem is $\mathcal{S}$-hard.

We will also use the notion of *Q-reducibility* of query answering, where Q is a given query language. To this purpose, we define $\mathcal{I}_{\mathcal{A}}$ as the interpretation defined as follows:

- $a^{\mathcal{I}_{\mathcal{A}}} = a$ for each constant a,
- $A^{\mathcal{I}_{\mathcal{A}}} = \{a \mid A(a) \in \mathcal{A}\}$ for each atomic concept A, and
- $P^{\mathcal{I}_{\mathcal{A}}} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$ for each atomic role P.

Query answering in a DL $\mathcal{L}$ is *Q-reducible* if for every (conjunctive) query q and every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a query q_1, over the same alphabet, belonging to the query language Q, such that for every ABox $\mathcal{A}$, $(\mathcal{T}, \mathcal{A}) \models q(\vec{a})$ iff $\vec{a}^{\mathcal{I}_{\mathcal{A}}} \in q_1^{\mathcal{I}_{\mathcal{A}}}$. In other words, q_1 is evaluated over the ABox $\mathcal{A}$ considered as a database. One of the most interesting classes of queries is that of FOL queries, i.e., the queries expressed in first-order logic, since, from the practical point of view, FOL queries correspond to queries expressed in relational algebra (i.e., in SQL). Observe that every FOL query can be evaluated in LOGSPACE with respect to data complexity (see e.g., (Abiteboul, Hull, & Vianu 1995)). It follows that if $\mathcal{L}$ is FOL-reducible, then query answering in $\mathcal{L}$ is in LOGSPACE wrt data complexity. Vice-versa, if query answering is $\mathcal{S}$-hard wrt data complexity for some complexity class $\mathcal{S}$ larger than LOGSPACE (e.g., NLOGSPACE, PTIME, coNP, etc.), then it is not FOL-reducible.

FOL-reducibility for the *DL-Lite* family

In this section we discuss two new DLs that extend *DL-Lite*$_{core}$, and show that in such DLs query answering is FOL-reducible (and hence is in LOGSPACE).

The first DL that we consider is *DL-Lite*$_{\mathcal{F},\sqcap}$, which extends *DL-Lite*$_{core}$ by allowing for

(a) the specification of conjunctions of concepts in the left-hand side of inclusion assertions, and

(b) the specification of functionality on a role R of the form $(funct\ R)$.

More precisely, the language for concepts, roles and TBox assertions in *DL-Lite*$_{\mathcal{F},\sqcap}$ is defined as follows:

$$\begin{array}{lcl} Cl & \longrightarrow & A \mid \exists R \mid Cl_1 \sqcap Cl_2 \\ Cr & \longrightarrow & A \mid \exists R \mid \neg A \mid \neg \exists R \\ R & \longrightarrow & P \mid P^- \end{array}$$

TBox assertions : $Cl \sqsubseteq Cr, (funct\ R)$.

Given an interpretation $\mathcal{I}$, we have that $(Cl_1 \sqcap Cl_2)^{\mathcal{I}} = Cl_1^{\mathcal{I}} \cap Cl_2^{\mathcal{I}}$. Furthermore, $\mathcal{I}$ is a model of an assertion $(funct\ \ P)$ if the binary relation $P^{\mathcal{I}}$ is a function, i.e., $(o, o_1) \in P^{\mathcal{I}}$ and $(o, o_2) \in P^{\mathcal{I}}$ implies $o_1 = o_2$. Analogously for $(funct\ P^-)$.

Notice that *DL-Lite*$_{\mathcal{F},\sqcap}$ is actually an extension of the DL presented in (Calvanese *et al.* 2005) (simply called *DL-Lite*), which essentially did not have conjunctions in Cl. In (Calvanese *et al.* 2005) we have presented an algorithm for query answering based on the idea of expanding the original query into a set (i.e., a union) of conjunctive queries that can be directly evaluated over the ABox. The expansion process takes into account only the original query and the TBox assertions, and is independent of the ABox, which can be easily managed in secondary storage by a relational DBMS. Therefore, from the results in (Calvanese *et al.* 2005) it follows that query answering in *DL-Lite* is FOL-reducible. In the following, we show that FOL-reducibility of query answering still holds in *DL-Lite*$_{\mathcal{F},\sqcap}$, as stated by the theorem below.

Theorem 1 *Query answering in DL-Lite*$_{\mathcal{F},\sqcap}$ *is FOL-reducible and therefore is in* LOGSPACE *with respect to data complexity.*

Proof (sketch). See appendix. □

We remark that *DL-Lite*$_{\mathcal{F},\sqcap}$ *does not enjoy the finite model property* (Baader *et al.* 2003): indeed, the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with $\mathcal{T} = \{A \sqsubseteq \exists P, \exists P^- \sqsubseteq A, B \sqsubseteq \neg A, (funct\ \ P^-)\}$ and $\mathcal{A} = \{B(a)\}$ admits only infinite models. We also remark that a reasoner for the description logic *DL-Lite*$_{\mathcal{F},\sqcap}$ has been implemented within the QuOnto system (Acciarri *et al.* 2005).

Notice that *DL-Lite*$_{core}$ only allows for unqualified existential quantification and inclusions between concepts. One might ask what happens to query answering if we add qualified existential quantification and inclusions between roles to the language. To this purpose, we consider a second notable extension of *DL-Lite*$_{core}$, called *DL-Lite*$_{\mathcal{R},\sqcap}$. The DL *DL-Lite*$_{\mathcal{R},\sqcap}$ extends *DL-Lite*$_{core}$ with the ability of specifying conjunctions of concepts on the left-hand side of inclusion assertions (analogously to *DL-Lite*$_{\mathcal{F},\sqcap}$), while on the right-hand side it allows also for qualified existential quantification. Furthermore, in addition to inclusion assertions between concepts, *DL-Lite*$_{\mathcal{R},\sqcap}$ allows for inclusion assertions between roles of the form:

$$R_1 \sqsubseteq R_2$$

where R_i is either an atomic role or its inverse.

More precisely, the language for concepts and roles and TBox assertions in *DL-Lite*$_{\mathcal{R},\sqcap}$ is defined as follows:

$$\begin{array}{rcl} Cl & \longrightarrow & A \mid \exists R \mid Cl_1 \sqcap Cl_2 \\ Cr & \longrightarrow & A \mid \exists R \mid \exists R.A \mid \neg A \mid \neg \exists R \\ R & \longrightarrow & P \mid P^- \end{array}$$

TBox assertions : $Cl \sqsubseteq Cr, R_1 \sqsubseteq R_2$.

For each interpretation $\mathcal{I}$, besides those seen so far, the equation $(\exists R.A)^{\mathcal{I}} = \{o \mid \exists o'.\, (o, o') \in R^{\mathcal{I}} \text{ and } o' \in A^{\mathcal{I}}\}$ holds. Furthermore, $\mathcal{I}$ is a model of an inclusion assertion of the form $R_1 \sqsubseteq R_2$, if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.

For the above DL the following result holds.

Theorem 2 *Query answering in DL-Lite*$_{\mathcal{R},\sqcap}$ *is FOL-reducible and therefore is in* LOGSPACE *with respect to data complexity.*

Proof (sketch). Again, the query answering algorithm for *DL-Lite*$_{\mathcal{R},\sqcap}$ is obtained by extending the reformulation technique of *DL-Lite* presented in (Calvanese *et al.* 2005). The extension is similar to the one devised for *DL-Lite*$_{\mathcal{F},\sqcap}$, provided that

(i) inclusion assertions with qualified existential quantification on the right-hand side are dealt with by a preprocessing step that eliminates such inclusions. In particular, each such a construct is represented through the use of unqualified existential quantification, auxiliary roles, and inclusions between roles;

(ii) the normalization step closes the TBox also with respect to a further rule which takes into account the interaction between negative inclusions and inclusion assertions between roles;

(iii) a suitable reformulation rule is added to the algorithm for taking into account inclusion assertions between roles.

□

Other logics allowing for different usages of qualified existential quantification will be analyzed in the next sections.

Extension to n-ary relations

In this section we show that we can extend the FOL-reducibility results of Theorems 1 and 2 to the case where we allow for the presence of n-ary relations, similar to the DL DLR (Calvanese, De Giacomo, & Lenzerini 1998). Given an interpretation $\mathcal{I}$, an n-ary relation R is interpreted as an n-ary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$. For each of the DLs presented in the above sections, we introduce now a corresponding variant in which we allow for n-ary relations instead of (binary) roles. In the following, R denotes an n-ary relation, and we use $\vec{o}$ to denote an n-tuple of objects, and $\vec{o}[i]$ to denote the i-th component of $\vec{o}$.

The DL *DLR-Lite*$_{core}$ is obtained from *DL-Lite*$_{core}$ by replacing both in Cl and in Cr the construct $\exists R$ with $\exists i{:}R$, where R is an n-ary relation and $i \in \{1, \ldots, n\}$, and by removing the rule $R \longrightarrow P \mid P^-$. The added construct denotes the projection of the relation denoted by R on its i-th component. More precisely, the language for concepts, roles and TBox assertions in *DLR-Lite*$_{core}$ is defined as follows:

$$\begin{array}{rcl} Cl & \longrightarrow & A \mid \exists i{:}R \\ Cr & \longrightarrow & A \mid \exists i{:}R \mid \neg A \mid \neg \exists i{:}R \end{array}$$

TBox assertions : $Cl \sqsubseteq Cr$.

Formally, for an interpretation $\mathcal{I}$, we define $(\exists i{:}R)^{\mathcal{I}} = \{\vec{o}[i] \mid \vec{o} \in R^{\mathcal{I}}\}$, and $(\neg \exists i{:}R)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus (\exists i{:}R)^{\mathcal{I}}$.

Then, *DLR-Lite*$_{\mathcal{F},\sqcap}$, analogously to *DL-Lite*$_{\mathcal{F},\sqcap}$, is obtained from *DLR-Lite*$_{core}$ by additionally allowing in the TBox for specifying conjunctions of concepts in the left-hand side of inclusion assertions, and for assertions of the form $(funct\ \ i{:}R)$, stating the functionality of the i-th component of R. More precisely, the language for concepts,

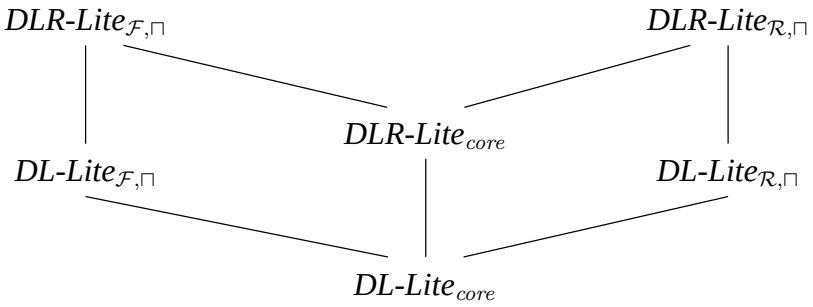

Figure 1: The *DL-Lite* family

roles and TBox assertions in *DLR-Lite*$_{\mathcal{F},\sqcap}$ is defined as follows:

$$\begin{array}{lcl} Cl & \longrightarrow & A \mid \exists i{:}R \mid Cl_1 \sqcap Cl_2 \\ Cr & \longrightarrow & A \mid \exists i{:}R \mid \neg A \mid \neg \exists i{:}R \end{array}$$
TBox assertions : $Cl \sqsubseteq Cr, (funct\ i{:}R)$.

Formally, an interpretation $\mathcal{I}$ is a model of a functionality assertion $(funct\ i{:}R)$ if $\vec{o_1}, \vec{o_2} \in R^{\mathcal{I}}$ with $\vec{o_1}[i] = \vec{o_2}[i]$ implies $\vec{o_1}[j] = \vec{o_2}[j]$ for all $j \in \{1, \ldots, n\}$.

Analogously to Theorem 1, we can show FOL-reducibility of *DLR-Lite*$_{\mathcal{F},\sqcap}$.

Theorem 3 *Query answering in DLR-Lite*$_{\mathcal{F},\sqcap}$ *is FOL-reducible, and therefore is in* LOGSPACE *with respect to data complexity.*

Proof (sketch). The thesis can be demonstrated by providing an algorithm for FOL-reduction of query answering in *DLR-Lite*$_{\mathcal{F},\sqcap}$. Such an algorithm is analogous to the one given in the proof of Theorem 1. The main modification concerns with the function PerfectRef in which the operator $gr(g, I)$, which indicates the atom obtained from an atom g by applying to it a (positive) inclusion I, is now defined as follows (notice that now a basic concept B_i can be either A or $\exists j{:}R$):

if $I = B_1 \sqcap \ldots \sqcap B_m \sqsubseteq A$
(resp., $I = B_1 \sqcap \ldots \sqcap B_m \sqsubseteq \exists k{:}R$)
and if $g = A(x)$
(resp., $g = R(\_, \ldots, \_, x, \_, \ldots, \_)$, where x occurs as the k-th argument of R and all the other arguments are unbound)
then $gr(g, I) = C_1(x) \wedge \ldots \wedge C_m(x)$,

where, for each $i \in \{1, \ldots, m\}$,

- $C_i(x) = A_i(x)$ if $B_i = A_i$, or
- $C_i(x) = \exists z_1, \ldots, z_\ell . R_i(z_1, \ldots, z_{j-1}, x, z_{j+1}, \ldots, z_\ell)$ if $B_i = \exists j{:}R_i$ and ℓ is the arity of R_i.

□

The other DL allowing for the presence of n-relations that we consider is *DLR-Lite*$_{\mathcal{R},\sqcap}$. Analogously to *DL-Lite*$_{\mathcal{R},\sqcap}$, such a new DL is obtained from *DLR-Lite*$_{core}$ by additionally allowing in the TBox for specifying conjunctions of concepts in the left-hand side of inclusion assertions and by adding to Cr the construct $\exists i{:}R.A_1, \ldots, A_n$. Such a construct denotes those objects that participate as i-th component to tuples of R in which the j-th component is an instance of A_j, for all $j \in \{1, \ldots, n\}$. Additionally, *DLR-Lite*$_{\mathcal{R},\sqcap}$ allows in the TBox for inclusion assertions between projections of relations of the forms:

$$\begin{array}{lcl} R_1[i_1, \ldots, i_k] & \sqsubseteq & R_2[j_1, \ldots, j_k] \\ R_1[i_1, \ldots, i_k] & \sqsubseteq & \neg R_2[j_1, \ldots, j_k] \end{array}$$

where R_1 is an n-ary relation, $i_1, \ldots, i_k \in \{1, \ldots, n\}$, and $i_p \neq i_q$ if $p \neq q$; R_2 is an m-ary relation, $j_1, \ldots, j_k \in \{1, \ldots, m\}$, and $j_p \neq j_q$ if $p \neq q$. The language for concepts, roles and TBox assertions in *DLR-Lite*$_{\mathcal{R},\sqcap}$ is therefore defined as follows:

$$\begin{array}{lcl} Cl & \longrightarrow & A \mid \exists i{:}R \mid Cl_1 \sqcap Cl_2 \\ Cr & \longrightarrow & A \mid \exists i{:}R \mid \neg A \mid \neg \exists i{:}R \mid \exists i{:}R.A_1, \ldots, A_n \end{array}$$
TBox assertions : $Cl \sqsubseteq Cr, (funct\ i{:}R)$,
$R_1[i_1, \ldots, i_k] \sqsubseteq R_2[j_1, \ldots, j_k]$,
$R_1[i_1, \ldots, i_k] \sqsubseteq \neg R_2[j_1, \ldots, j_k]$.

Formally, for an interpretation $\mathcal{I}$, we define $(\exists i{:}R.A_1, \ldots, A_n)^{\mathcal{I}} = \{\vec{o}[i] \mid \vec{o} \in R^{\mathcal{I}}$ with $\vec{o}[j] \in A_j^{\mathcal{I}}$, for $j \in \{1, \ldots, n\}\}$. Furthermore, given an n-ary relation R_1 and an m-ary relation R_2, $\mathcal{I}$ is a model of an assertion of the form $R_1[i_1, \ldots, i_k] \sqsubseteq R_2[j_1, \ldots, j_k]$, if for every n-tuple of objects $\vec{o_1} \in R_1^{\mathcal{I}}$ there is an m-tuple of objects $\vec{o_2} \in R_2^{\mathcal{I}}$ such that $(\vec{o_1}[i_1], \ldots, \vec{o_1}[i_k]) = (\vec{o_2}[j_1], \ldots, \vec{o_2}[j_k])$. Also, $\mathcal{I}$ is a model of an assertion of the form $R_1[i_1, \ldots, i_k] \sqsubseteq \neg R_2[j_1, \ldots, j_k]$, if there do not exist two tuples of objects $\vec{o_1} \in R_1^{\mathcal{I}}$ and $\vec{o_2} \in R_2^{\mathcal{I}}$ such that $(\vec{o_1}[i_1], \ldots, \vec{o_1}[i_k]) = (\vec{o_2}[j_1], \ldots, \vec{o_2}[j_k])$.

Analogously to Theorem 2, we can show FOL-reducibility of *DLR-Lite*$_{\mathcal{R},\sqcap}$.

Theorem 4 *Query answering in DLR-Lite*$_{\mathcal{R},\sqcap}$ *is FOL-reducible, and therefore is in* LOGSPACE *with respect to data complexity.*

Finally, we summarize the relationship between the various DLs of the *DL-Lite* family in Figure 1.

NLOGSPACE-hard DLs

In the previous section, we have pointed out the importance of languages for which query answering is FOL-reducible. In this section, we show that, as soon as we consider further, minimal extensions of *DL-Lite*$_{core}$, besides those illustrated in the above sections, we cross the boundary of

LOGSPACE data complexity. Going beyond LOGSPACE data complexity means actually that we lose the property of FOL-reducibility, and therefore query answering requires more powerful engines than those available in standard relational database technology. An immediate consequence of this fact is that we cannot take advantage anymore of data management tools and query optimization techniques of current DBMSs.

The first case of this type is when we add qualified existential quantification to Cl. The second case is when we add qualified universal quantification to Cr, and the third case is when we add qualified existential quantification to Cr, while keeping the possibility of expressing functionality constraints. This is formally stated in the following theorem.

Theorem 5 *Instance checking (and hence query answering) is* NLOGSPACE*-hard with respect to data complexity for the cases where*

1. $Cl \rightarrow A \mid \exists R.A$
   $Cr \rightarrow A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$
2. $Cl \rightarrow A$
   $Cr \rightarrow A \mid \forall R.A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$
3. $Cl \rightarrow A$
   $Cr \rightarrow A \mid \exists R.A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr, (funct\ R)$

Proof (sketch). For Case 1, the proof is by a LOGSPACE reduction from reachability in directed graphs, which is NLOGSPACE-complete. For Case 2, the proof follows from Case 1 and the observation that an assertion $\exists P.A_1 \sqsubseteq A_2$ is logically equivalent to the assertion $A_1 \sqsubseteq \forall P^-.A_2$, and that we can get rid of inverse roles by inverting the edges of the graph represented in the ABox. For Case 3, the proof is again by a LOGSPACE reduction from reachability in directed graphs, and is based on the idea that an assertion $\exists P.A_1 \sqsubseteq A_2$ can be simulated by the assertions $A_1 \sqsubseteq \exists P^-.A_2$ and $(funct\ P^-)$. Moreover, the graph can be encoded using only functional roles, and we can again get rid of inverse roles by inverting edges. □

Note that all the above "negative" results hold for instance checking already, i.e., for the simplest queries possible. Also, note that in all three cases, we are considering extensions to a minimal subset of *DL-Lite*$_{core}$ in order to get NLOGSPACE-hardness.

PTIME-hard DLs

Next we show that if we consider further extensions to the logics mentioned in Theorem 5, we get even stronger complexity results. In particular, we consider five different cases where query answering (actually, instance checking already) becomes PTIME-hard in data complexity.

Note that the PTIME-hardness result basically means that we need at least the power of full Datalog to answer queries in these cases.

Theorem 6 *Instance checking (and hence query answering) is* PTIME*-hard with respect to data complexity for the cases where*

1. $Cl \rightarrow A \mid \exists R.A$
   $Cr \rightarrow A \mid \exists P$
   $R \rightarrow P \mid P^-$
   TBox assertions: $Cl \sqsubseteq Cr$
2. $Cl \rightarrow A$
   $Cr \rightarrow A \mid \exists R.A$
   $R \rightarrow P \mid P^-$
   TBox assertions: $Cl \sqsubseteq Cr, (funct\ R)$
3. $Cl \rightarrow A \mid \exists R.A$
   $Cr \rightarrow A \mid \exists R.A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr, (funct\ R)$

Proof (sketch). For each of the three cases, the proof is by reduction from the emptiness problem of context-free grammars to query answering over such DL KBs (the only difference in the reductions is in the form of the TBox). □

Theorem 7 *Instance checking (and hence query answering) is* PTIME*-hard with respect to data complexity for the cases where*

1. $Cl \rightarrow A \mid \exists R.A \mid A_1 \sqcap A_2$
   $Cr \rightarrow A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$
2. $Cl \rightarrow A \mid A_1 \sqcap A_2$
   $Cr \rightarrow A \mid \forall R.A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$
3. $Cl \rightarrow A \mid A_1 \sqcap A_2$
   $Cr \rightarrow A \mid \exists R.A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr, (funct\ R)$

Proof (sketch). For Case 1, the proof is by a LOGSPACE reduction from Path System Accessibility, which is PTIME-complete (Garey & Johnson 1979). For Cases 2 and 3, the proof follows from Case 1 and observations analogous to the ones for Theorem 5. □

coNP-hard DLs

Finally, we show three cases where the TBox language becomes so expressive that the data complexity of query answering goes beyond PTIME (assuming PTIME $\neq$ NP).

Theorem 8 *Query answering is coNP-hard with respect to data complexity for the cases where*

1. $Cl \rightarrow A \mid \neg A$
   $Cr \rightarrow A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$
2. $Cl \rightarrow A$
   $Cr \rightarrow A \mid A_1 \sqcup A_2$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$

3. $Cl \rightarrow A \mid \forall R.A$
   $Cr \rightarrow A$
   $R \rightarrow P$
   TBox assertions: $Cl \sqsubseteq Cr$

Proof (sketch). In all three cases, the proof is an adaptation of the proof of coNP-hardness of instance checking for $\mathcal{ALE}$ presented in (Donini *et al.* 1994). The intuition is that in all three cases query answering requires reasoning by cases, caused by the presence of simple covering constraints (induced by union). Note that, whereas in case 2 covering can be explicitly specified by inclusion assertions of the form $A \sqsubseteq A_1 \sqcup A_2$, for case 1 (resp. case 3) we can exploit the fact that A and $\neg A$ (resp.$\forall R.A$ and $\exists R$) cover the entire domain. □

Related work

All the DLs studied in this paper are fragments of expressive DLs with assertions and inverses studied in the 90's (see (Baader *et al.* 2003) for an overview), which are at the base of current ontology languages such as OWL, and for which optimized automated reasoning systems such as Fact[5], Racer[6] and Pellet[7] have been developed. Indeed, one could use, off-the-shelf, a system like Racer or Pellet to perform instance checking in such DLs. Also, reasoning with conjunctive queries in these DLs has been studied (see e.g., (Calvanese, De Giacomo, & Lenzerini 1998; 2000)), although not yet implemented in systems. Unfortunately, the known reasoning algorithms for these DLs are in 2EXPTIME with respect to combined complexity, and more importantly they are not tailored towards obtaining tight complexity bounds with respect to data complexity (they are in EXPTIME). Alternative reasoning procedures that allow for clearly isolating data complexity have recently been proposed, how they will work in practice still needs to be understood. A coNP upper bound for data complexity of instance checking in the expressive DL $\mathcal{SHIQ}$ has been shown by making use of a reduction to Disjunctive Datalog and then exploiting resolution (Hustadt, Motik, & Sattler 2004; 2005). It remains open whether such a technique can be extended to deal efficiently with conjunctive queries. In (Levy & Rousset 1998), making use of an algorithm based on tableaux, a coNP, upper-bound with respect to data complexity is given for a DL with arbitrary inclusion assertions, but lacking inverse roles. Recently, building on such techniques, coNP-completeness of answering conjunctive queries for $\mathcal{SHIQ}$, which includes inverse roles, and number restrictions (that generalize functionality) has been shown (Ortiz de la Fuente *et al.* 2005). It is interesting to observe that the results in this paper (Theorem 8) tell us that we get coNP-completeness already for very small fragments of $\mathcal{SHIQ}$.

In (Hustadt, Motik, & Sattler 2005), a fragment of $\mathcal{SHIQ}$, called Horn-$\mathcal{SHIQ}$, which subsumes both *DL-Lite*$_{\mathcal{F},\sqcap}$ and *DL-Lite*$_{\mathcal{R},\sqcap}$, is studied and a PTIME upper bound in data complexity for instance checking is shown. The results in the current paper (Theorem 6) tell us that instance checking in Horn-$\mathcal{SHIQ}$ is also PTIME-hard. Indeed, Horn-$\mathcal{SHIQ}$ allows for qualified existential quantification $\exists P.A$ in both sides of inclusion assertions and (an extended form) of functionality restrictions.

DL-Lite$_{\mathcal{R},\sqcap}$ captures (the DL-subset of) RDFS extended with participation constraints (i.e., inclusion assertions with $\exists R$ on the right-hand side). Hence, query answering over an RDFS ontology, even extended with participation constraints, is FOL-reducible. Finally, if we move from RDFS to DLP (Grosof *et al.* 2003), query answering becomes PTIME-hard, since DLP is a superset of the DL in case 1 of Theorem 7.

Conclusions

We have presented first fundamental results on the data complexity (complexity with respect to the size of the ABox only) of query answering in DLs. In particular, we have concentrated on the FOL-reducibility boundary of the problem, based on the observation that, when we go above this boundary, query answering is no longer expressible as a first-order logic formula (and hence an SQL query) over the data. The results provided in this paper are summarized in Figure 2.

We are currently following several directions to continue the work reported in this paper. First, we conjecture that for all NLOGSPACE and PTIME-hardness results presented here a matching upper bound holds. Second, although here we focused on data complexity only, we are also working on characterizing the complexity of query answering with respect to the size of the TBox, with respect to the size of the query, and with respect to combined complexity. Finally, while in this paper we considered conjunctive queries, our general goal is to come up with a clear picture of how the complexity of query answering is influenced not only by different TBox languages, but also by different query languages.

Acknowledgments This work has been partially supported by the EU funded IST-2005-7603 FET Project Thinking ONtologiES (TONES), by project HYPER, funded by IBM through a Shared University Research (SUR) Award grant, and by MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts.

Acciarri, A.; Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Palmieri, M.; and Rosati, R. 2005. QUONTO: QUerying ONTOlogies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 1670–1671.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

[5] http://www.cs.man.ac.uk/~horrocks/FaCT/

[6] http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[7] http://www.mindswap.org/2003/pellet/

Cl	Cr	$\mathcal{F}$	$\mathcal{R}$	Data complexity of query answering
DL-Lite$_{\mathcal{F},\sqcap}$		$\surd$	–	in LOGSPACE
DL-Lite$_{\mathcal{R},\sqcap}$		–	$\surd$	in LOGSPACE
DLR-Lite$_{\mathcal{F},\sqcap}$		$\surd$	–	in LOGSPACE
DLR-Lite$_{\mathcal{R},\sqcap}$		–	$\surd$	in LOGSPACE
$A \mid \exists P.A$	A	–	–	NLOGSPACE-hard
A	$A \mid \forall P.A$	–	–	NLOGSPACE-hard
A	$A \mid \exists P.A$	$\surd$	–	NLOGSPACE-hard
$A \mid \exists P.A \mid \exists P^{-}.A$	$A \mid \exists P$	–	–	PTIME-hard
A	$A \mid \exists P.A \mid \exists P^{-}.A$	$\surd$	–	PTIME-hard
$A \mid \exists P.A$	$A \mid \exists P.A$	$\surd$	–	PTIME-hard
$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTIME-hard
$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTIME-hard
$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTIME-hard
$A \mid \neg A$	A	–	–	coNP-hard
A	$A \mid A_1 \sqcup A_2$	–	–	coNP-hard
$A \mid \forall P.A$	A	–	–	coNP-hard

Legenda: A (possibly with subscript) = atomic concept, P = atomic role, Cl/Cr = left/right-hand side of inclusion assertions, $\mathcal{F}$ = functionality assertions allowed, $\mathcal{R}$ = role/relationship inclusions allowed. NLOGSPACE and PTIME hardness results hold already for instance checking.

Figure 2: Data Complexity of Query Answering for various Description Logics

Borgida, A.; Brachman, R. J.; McGuinness, D. L.; and Resnick, L. A. 1989. CLASSIC: A structural data model for objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 59–67.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2005. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 602–607.

Calvanese, D.; De Giacomo, G.; and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, 149–158.

Calvanese, D.; De Giacomo, G.; and Lenzerini, M. 2000. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, 386–391.

Donini, F. M.; Lenzerini, M.; Nardi, D.; and Schaerf, A. 1994. Deduction in concept languages: From subsumption to instance checking. *J. of Logic and Computation* 4(4):423–452.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability — A guide to NP-completeness*. San Francisco (CA, USA): W. H. Freeman and Company.

Grosof, B. N.; Horrocks, I.; Volz, R.; and Decker, S. 2003. Description logic programs: Combining logic programs with description logic. In *Proc. of the 12th Int. World Wide Web Conf. (WWW 2003)*, 48–57.

Heflin, J., and Hendler, J. 2001. A portrait of the semantic web in action. *IEEE Intelligent Systems* 16(2):54–59.

Hustadt, U.; Motik, B.; and Sattler, U. 2004. Reducing $\mathcal{SHIQ}$-description logic to disjunctive datalog programs. In *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*.

Hustadt, U.; Motik, B.; and Sattler, U. 2005. Data complexity of reasoning in very expressive description logics. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*.

Lee, J.; Siau, K.; and Hong, S. 2003. Enterprise integration with erp and eai. *Communications of the ACM* 46(2):54 – 60.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, 233–246.

Levy, A. Y., and Rousset, M.-C. 1998. Combining Horn rules and description logics in CARIN. *Artificial Intelligence* 104(1–2):165–209.

Ortiz de la Fuente, M. M.; Calvanese, D.; Eiter, T.; and Franconi, E. 2005. Data complexity of answering conjunctive queries over $\mathcal{SHIQ}$ knowledge bases. Technical report, Faculty of Computer Science, Free University of Bozen-Bolzano. Also available as CORR technical report at `http://arxiv.org/abs/cs.LO/0507059/`.

Vardi, M. Y. 1982. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, 137–146.

Appendix

Theorem 1 *Query answering in DL-Lite$_{\mathcal{F},\sqcap}$ is FOL-reducible and therefore is in* LOGSPACE *with respect to data complexity.*

Proof (sketch). The query answering algorithm for *DL-Lite$_{\mathcal{F},\sqcap}$* is obtained by extending the reformulation technique of *DL-Lite* (Calvanese *et al.* 2005). In the following, we precisely describe the algorithm for FOL-reducibility of query answering in *DL-Lite$_{\mathcal{F},\sqcap}$* knowledge bases. The algorithm takes as input a *DL-Lite$_{\mathcal{R},\sqcap}$* TBox $\mathcal{T}$ and a conjunctive query q specified over $\mathcal{T}$, and returns a union of conjunctive queries (and therefore a FOL query) q_1 such that, for each ABox $\mathcal{A}$, the evaluation of q_1 over the ABox $\mathcal{A}$ considered as a database (see Section Preliminaries for details) returns the set of tuples in the answer to q over the knowledge base $(\mathcal{T}, \mathcal{A})$, i.e., returns all tuples $\vec{a}$ such that $(\mathcal{T}, \mathcal{A}) \models q(\vec{a})$.

As usual, with Cl (resp. Cr) we denote a concept used in the left-hand side (resp. right-hand side) of inclusion assertions between concepts, with A we denote an atomic concept, with P an atomic role, and with P^- its inverse (whereas R indicates either P or P^-). Furthermore, we use the symbol B (possibly with subscripts) to denote *basic concepts*, i.e., we use B to indicate either A, $\exists P$, or $\exists P^-$. Also, without loss of generality, we assume that every concept name or role name occurring in an ABox $\mathcal{A}$ also occurs in the corresponding TBox $\mathcal{T}$. Finally, we recall that a FOL query $q(\vec{x})$ over a KB $\mathcal{K}$ is an expression of the form

$$\{\, \vec{x} \mid body_q(\vec{x}, \vec{y}) \,\}$$

where $\vec{x}$ are the so-called distinguished variables (which will be bound with objects in the KB), $\vec{y}$ are the non-distinguished variables (which are existentially quantified), and $body_q(\vec{x}, \vec{y})$ is a FOL formula involving atoms of the form $A(z)$ or $P(z_1, z_2)$ where A and P are respectively atomic concepts and roles of $\mathcal{K}$ and z, z_1, z_2 are either constants in $\mathcal{K}$ or variables in $\vec{x}$ or $\vec{y}$[8].

The algorithm makes use of three main functions, namely Normalize, Consistent, and PerfectRef, which correspond to three phases called *Normalization*, *Satisfiability check* and *Query reformulation*, respectively. The first one performs some preliminary transformations on the TBox $\mathcal{T}$. The second one computes a portion of the final output query that properly deals with situations in which the ABox $\mathcal{A}$, over which the final output query is evaluated, contradicts the TBox $\mathcal{T}$, i.e., the knowledge base $(\mathcal{T}, \mathcal{A})$ is unsatisfiable. Notice that, in these cases, every n-tuple of constants of $\mathcal{A}$ is in the answer to every query of arity n over $\mathcal{T}$. Finally, the third function computes the remaining portion of the output query. Roughly speaking, PerfectRef reformulates the input query q into a FOL query in which it compiles the knowledge of the TBox $\mathcal{T}$ that is needed to answer q.

Normalization. The function Normalize takes as input the TBox $\mathcal{T}$ and closes the TBox with respect to the following inference rule:

if $B_1 \sqcap \ldots \sqcap B_n \sqsubseteq B$ occurs in $\mathcal{T}$
and $B'_1 \sqcap \ldots \sqcap B'_m \sqsubseteq \neg B$ occurs in $\mathcal{T}$
or there exists $i \in \{1, \ldots, m\}$ such that
$B'_1 \sqcap \ldots \sqcap B'_{i-1} \sqcap B \sqcap B'_{i+1} \sqcap \ldots \sqcap B'_m \sqsubseteq \neg B'_i$
occurs in $\mathcal{T}$
then add $B_1 \sqcap \ldots B_n \sqcap B'_1 \sqcap \ldots \sqcap B'_{m-1} \sqsubseteq \neg B'_m$ to $\mathcal{T}$

In the following, we denote with Normalize($\mathcal{T}$) the TBox obtained after processing $\mathcal{T}$ according to the above closure. Notice that Normalize($\mathcal{T}$) contains only assertions of the form (i) $Cl \sqsubseteq B$ and (ii) $Cl \sqsubseteq \neg B$. We call *positive inclusions (PIs)* inclusions of the forms (i) and *negative inclusions (NIs)* inclusions of form (ii). The aim of normalization is to expand the input TBox $\mathcal{T}$ by computing all (non-trivial) NIs logically implied by $\mathcal{T}$.

Indeed, it can be shown that, after normalization, for every sequence $B_1, \ldots, B_n$ of basic concepts,

$$\mathcal{T} \models (B_1 \sqcap \ldots \sqcap B_{n-1} \sqsubseteq \neg B_n)$$

iff there exists $i \in \{1, \ldots, n\}$ such that

$$(B_1 \sqcap \ldots \sqcap B_{i-1} \sqcap B_n \sqcap B_{i+1} \sqcap \ldots \sqcap B_n \sqsubseteq \neg B_i)$$

occurs in Normalize($\mathcal{T}$).

It is also possible to prove that Normalize($\mathcal{T}$) is "equivalent" to $\mathcal{T}$, in the sense that, for any ABox $\mathcal{A}$, the set of models of the knowledge base $(\mathcal{T}, \mathcal{A})$ coincides with that of the knowledge base $(\mathcal{A}, \text{Normalize}(\mathcal{T}))$.

In the following, we indicate with $\mathcal{T}_{PI}$ the TBox obtained by dropping all negative inclusions and functionality assertions from Normalize($\mathcal{T}$), and with $\mathcal{T}_{NI}$ the TBox obtained by dropping all positive inclusions from Normalize($\mathcal{T}$). It is easy to see that $\mathcal{T}_{NI}$ and $\mathcal{T}_{PI}$ are disjoint and that Normalize($\mathcal{T}$) $= \mathcal{T}_{PI} \cup \mathcal{T}_{NI}$.

Satisfiability check. The function Consistent is in charge of properly dealing with situations in which an ABox $\mathcal{A}$ contradicts NIs or functionality assertions of the TBox $\mathcal{T}$, i.e., $(\mathcal{T}_{NI}, \mathcal{A})$ is unsatisfiable. Observe that in such a case query answering is meaningless, since, according to the "ex falso quod libet" principle, every tuple is in the answer to every query (of the same arity). Therefore, with regards to this issue, the function Consistent takes as input the TBox $\mathcal{T}_{NI}$ and a query q of arity n and proceeds as follows[9]:

1. for each NI inclusion $\iota = B_1 \sqcap \ldots \sqcap B_{m-1} \sqsubseteq \neg B_m$ belonging to $\mathcal{T}_{NI}$, it computes the FOL query

$$q_\iota = \{x_1, \ldots, x_n \mid \exists y.C_1(y) \wedge \ldots \wedge C_m(y) \wedge val(x_1) \wedge \ldots \wedge val(x_n)\},$$

where

- for each $i \in \{1, \ldots, m\}$, $C_i(y) = A_i(y)$ if $B_i = A_i$, or $C_i(y) = \exists z_i.P_i(y, z_i)$ if $B_i = \exists P_i$ or $C_i(y) = \exists z_i.P_i(z_i, y)$ if $B_i = \exists P_i^-$, and
- for each $i \in \{1, \ldots, n\}$, $val(x_i) = A_1(x_i) \vee \ldots \vee A_\ell(x_i) \vee \exists w_1.R_1(x_i, w_1) \vee \ldots \vee \exists w_k.R_k(x_i, w_k) \vee \exists v_1.R_1(v_1, x_i) \vee \ldots \vee \exists v_k.R_k(v_k, x_i)$, where $A_1, \ldots, A_\ell$ and $R_1, \ldots, R_k$ are all the atomic concepts and the atomic roles over which inclusions of the TBox $\mathcal{T}$ are asserted;

[8] Obviously, for FOL queries that are conjunctive queries, $body_q(\vec{x}, \vec{y})$ is a conjunction of atoms, that can be also denoted with $\exists y.conj(\vec{x}, \vec{y})$ (see Section Preliminaries).

[9] Actually, Consistent only makes use of the arity n of q.

2. for each functionality assertion $(funct\ P)$ belonging to $\mathcal{T}_{NI}$, Consistent computes the FOL query

$$q_\phi = \{x_1, \ldots, x_n \quad | \quad \exists x, y, z.P(x, y) \wedge P(x, z) \wedge y \neq z \wedge val(x_1) \wedge \ldots \wedge val(x_n)\};$$

3. for each functionality assertion $(funct\ P^-)$ belonging to $\mathcal{T}_{NI}$, Consistent computes the FOL query

$$q_{\phi^-} = \{x_1, \ldots, x_n \quad | \quad \exists x, y, z.P(x, y) \wedge P(z, y) \wedge x \neq z \wedge val(x_1) \wedge \ldots \wedge val(x_n)\};$$

4. returns the query

$$q_c = \{\vec{x} \mid body_{q_\alpha}(\vec{x}, \vec{y_\alpha}) \vee \ldots \vee body_{q_\nu}(\vec{x}, \vec{y_\nu})\}$$

where with $body_{q_\alpha}(\vec{x}, \vec{y_\alpha}), \ldots, body_{q_\nu}(\vec{x}, \vec{y_\nu})$ we denote the bodies of queries $q_\alpha, \ldots, q_\nu$ constructed according to step 1, 2, and 3.

We remember the reader that, in order to answer the query q over a *DL-Lite*$_{\mathcal{F},\sqcap}$ knowledge base $(\mathcal{T}, \mathcal{A})$, the above query will be evaluated over the ABox $\mathcal{A}$ considered as a database, i.e., over the interpretation $\mathcal{I}_\mathcal{A}$ defined as follows: $a^{\mathcal{I}_\mathcal{A}} = a$ for each constant a occurring in $\mathcal{A}$, $A^{\mathcal{I}_\mathcal{A}} = \{a \mid A(a) \in \mathcal{A}\}$ for each atomic concept A, and $P^{\mathcal{I}_\mathcal{A}} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$ for each atomic role P. Given a knowledge base $(\mathcal{T}, \mathcal{A})$ we will call the interpretation $\mathcal{I}_\mathcal{A}$ defined above, the *minimal interpretation of* $\mathcal{A}$. Then, it can be shown that, if an ABox $\mathcal{A}$ contradicts a NI or a functionality assertion in the TBox $\mathcal{T}_{NI}$, i.e., if the knowledge base $(\mathcal{T}, \mathcal{A})$ is unsatisfiable, then, any n-tuples constructible from constants occurring in $\mathcal{A}$ is returned by the evaluation of q_c over $\mathcal{I}_\mathcal{A}$ (thanks to the use of the conjunctions of predicates of the form $val(x_1) \wedge \ldots \wedge val(x_n)$ in the bodies of the queries q_ι, q_ϕ, and q_{ϕ^-}).

Since Consistent provides the right answers to the query for all cases in which the ABox contradicts NIs of the TBox, in the following step we can focus our attention to the remaining case, i.e., the case in which the knowledge base is satisfiable.

Query reformulation. Query reformulation is achieved by means of the algorithm PerfectRef. The basic idea of our method is to reformulate the input query expressed over the TBox $\mathcal{T}$ taking into account only the PIs in $\mathcal{T}$. In particular, given a query q over a *DL-Lite*$_{\mathcal{F},\sqcap}$ knowledge base $(\mathcal{T}, \mathcal{A})$, we compile the PIs of $\mathcal{T}$ into the query itself, thus obtaining a new query q_r. The evaluation of such a new query q_r over the ABox $\mathcal{A}$ considered as a simple relational database, i.e., over the minimal interpretation $\mathcal{I}_\mathcal{A}$ of $\mathcal{A}$, returns the answer to q over $(\mathcal{T}, \mathcal{A})$ (when $(\mathcal{T}, \mathcal{A})$ is satisfiable).

In the following, we illustrate PerfectRef from a technical point of view.

We say that an argument of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant, while we say that it is *unbound* if it corresponds to a non-distinguished non-shared variable (we use the symbol $\_$ to represent non-distinguished non-shared variables).

We also say that now specify when a positive inclusion I *is applicable to a query atom* g if:

(i) g is of the form $A(x)$, and I is of the form $Cl \sqsubseteq A$, where A is an atomic concept ;

(ii) g is of the form $P_1(x_1, x_2)$, and one of the two following conditions holds

(a) $x_2 = \_$ and I is of the form $Cl \sqsubseteq \exists P_1$,
(b) $x_1 = \_$ and I is of the form $Cl \sqsubseteq \exists P_1^-$.

Roughly speaking, an inclusion I is applicable to an atom g if all bound arguments of g are propagated by I.

We indicate with $gr(g, I)$ the atom obtained from the atom g by applying the inclusion I. Let $B_1 \sqcap \ldots \sqcap B_m$ be a set of basic concepts, $gr(g, I)$ is defined as follows:

if $I = B_1 \sqcap \ldots \sqcap B_m \sqsubseteq A$
(resp., $I = B_1 \sqcap \ldots \sqcap B_m \sqsubseteq \exists P_1$
or $I = B_1 \sqcap \ldots \sqcap B_m \sqsubseteq \exists P_1^-$)
and $g = A(x)$ (resp., $g = P_1(x, \_)$ or $g = P_1(\_, x)$)
then $gr(g, I) = C_1(x) \wedge \ldots \wedge C_m(x)$,

where, for each $i \in \{1, \ldots, m\}$,

- $C_i(x) = A_i(x)$ if $B_i = A_i$, or
- $C_i(x) = \exists z_i.P_i(x, z_i)$ if $B_i = \exists P_i$ or
- $C_i(x) = \exists z_i.P_i(z_i, x)$ if $B_i = \exists P_i^-$.

We are now ready to define the algorithm PerfectRef.

Algorithm PerfectRef($q, \mathcal{T}$)
Input: conjunctive query q of arity n, *DL-Lite*$_{\mathcal{F},\sqcap}$ TBox $\mathcal{T}$
Output: FOL query q_r
P := $\{q\}$;
repeat
  P' := P;
  for each $q \in P'$ **do**
  (a) **for each** g in q **do**
    for each PI I in $\mathcal{T}$ **do**
      if I is applicable to g
      then P := $P \cup \{\, q[g/gr(g, I)] \,\}$;
  (b) **for each** g_1, g_2 in q **do**
    if g_1 and g_2 unify
    then $P := P \cup \{\tau(reduce(q, g_1, g_2))\}$;
until $P' = P$;
let q_r be a query in P
for each $q \in P$ **do**
  $body_{q_r} = body_{q_r} \vee body_q$;
return q_r
end

In the algorithm, $q[g/g']$ denotes the query obtained from q by replacing the atom g with a new atom g'.

Informally, the algorithm first reformulates the atoms of each (conjunctive) query $q \in P'$, and produces a new (conjunctive) query for each atom reformulation (step (a)). Roughly speaking, PIs are used as rewriting rules, applied from right to left, that allow to compile away in the reformulation the knowledge of $\mathcal{T}$ that is relevant for answering the query q.

At step (b), for each pair of atoms g_1, g_2 that unify, the algorithm computes the query $q' = reduce(q, g_1, g_2)$, by applying to q the *most general unifier* between g_1 and g_2. Due to the unification, variables that were bound in q may become unbound in q'. Hence, PIs that were not applicable to atoms of q, may become applicable to atoms of q' (in the

next executions of step (a)). Function τ applied to q' replaces with _ each unbound variable in q'. Then, the for cycle before the end of the algorithm transforms the set of conjunctive queries P into a FOL query (union of conjunctive queries) q_r.

Finally, we are able to illustrate the algorithm FOL-Reduction.

Algorithm FOL-Reduction($q, \mathcal{T}$)
Input: conjunctive query q of arity n, *DL-Lite*$_{\mathcal{F},\sqcap}$ TBox $\mathcal{T}$
Output: FOL query q
$\mathcal{T} = \mathsf{Normalize}(\mathcal{T})$;
let $\mathcal{T}_{NI}$ be the TBox obtained from $\mathcal{T}$ by dropping all PIs;
$\mathcal{T}_{PI} = \mathcal{T} \backslash \mathcal{T}_{NI}$;
$q_c = \mathsf{Consistent}(\mathcal{T}_{NI}, q)$;
$q_r = \mathsf{PerfectRef}(\mathcal{T}_{PI}, q)$;
$body_{q_c} = body_{q_c} \vee body_{q_r}$;
return q_c
end

Notice that in the above procedure, the algorithm PerfectRef is invoked with the knowledge base $\mathcal{T}_{PI}$ as input, since the query reformulation step is performed only according to positive inclusions, and therefore we can avoid to pass also negative inclusions and functionality assertions to PerfectRef.

Let $\mathcal{A}$ be an ABox, and $\vec{a}$ be a tuple of constants of $\mathcal{A}$, and let $q_1 = \mathsf{FOL\text{-}Reduction}(\mathcal{T}, q)$, it ca be shown that $\vec{a} \in q_1^{\mathcal{I}_\mathcal{A}}$, iff $(\mathcal{T}, \mathcal{A}) \models q(\vec{a})$. □