Artificial Intelligence 195 (2013) 335–360

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Data complexity of query answering in description logics

Diego Calvanese [a], Giuseppe De Giacomo [b], Domenico Lembo [b,*], Maurizio Lenzerini [b], Riccardo Rosati [b]

[a] Faculty of Computer Science, Free University of Bozen–Bolzano, Piazza Domenicani 3, Bolzano, Italy
[b] Dipartimento di Ingegneria Informatica, Automatica e Gestionale "A. Ruberti", Sapienza Università di Roma, Via Ariosto 25, Roma, Italy

ARTICLE INFO

Article history:
Received 2 August 2011
Received in revised form 19 May 2012
Accepted 8 October 2012
Available online 15 October 2012

Keywords:
Knowledge representation
Description logics
Ontologies
Computational complexity
Conjunctive queries

ABSTRACT

In this paper we study data complexity of answering conjunctive queries over description logic (DL) knowledge bases constituted by a TBox and an ABox. In particular, we are interested in characterizing the FOL-rewritability and the polynomial tractability boundaries of conjunctive query answering, depending on the expressive power of the DL used to express the knowledge base. FOL-rewritability 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-rewritability is that, when query answering enjoys this property, we can take advantage of Relational Data Base Management System (RDBMS) 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 essentially the maximal logics allowing for 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.

1. 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 [1], data integration [2], and the semantic web [3], 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, e.g., [4]).

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, which means computing the answers to the queries that are logically implied by the whole ontology. Here, by 'complex' we mean that it does not suffice to ask for the instances of concepts, but we need at least to express conjunctive conditions on the extensional level [5–11]. Given the size of the instance repository, when measuring the computational complexity of query answering (and reasoning in general) the most

* Corresponding author.
E-mail addresses: calvanese@inf.unibz.it (D. Calvanese), degiacomo@dis.uniroma1.it (G. De Giacomo), lembo@dis.uniroma1.it (D. Lembo), lenzerini@dis.uniroma1.it (M. Lenzerini), rosati@dis.uniroma1.it (R. Rosati).

0004-3702/$ – see front matter
http://dx.doi.org/10.1016/j.artint.2012.10.003

important parameter is the size of the data. In other words, we are interested in the so-called *data complexity* of query answering [12].

In this paper we consider conjunctive queries (CQs) specified over ontologies expressed in description logics (DLs), 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 [12], on the premise that an ABox can be naturally viewed as a relational database. Recently, data complexity has attracted the interest of the DL community, first for reasoning over TBox and ABox (i.e., instance checking, which is the simplest form of query answering) [13,14], and then also for answering full conjunctive queries [15,16]. This gave rise to the study of DLs for which query answering can be done efficiently in data complexity [17–20], which is a key aspects of the present paper.

Specifically, we are interested in characterizing the FOL-rewritability 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-rewritable* in a DL $\mathcal{L}$, if for every conjunctive query q over an $\mathcal{L}$ TBox $\mathcal{T}$, one can effectively compute a first-order (FOL) query q_r such that for all ABoxes $\mathcal{A}$ the answers to q with respect to the KB $\langle \mathcal{T}, \mathcal{A} \rangle$ are the same as the answers to q_r over the database corresponding to the ABox $\mathcal{A}$. Since first-order queries can be expressed in SQL, the importance of FOL-rewritability is that, when query answering enjoys this property, we can take advantage of Relational Data Base Management System (RDBMS) 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 evaluating FOL queries over relational databases, i.e., AC^0 [21], a complexity class strictly contained in LogSpace [22].

We are also interested in knowing for which DLs we go beyond FOL. For this purpose, we single out those DLs for which query answering becomes NLogSpace-hard and PTime-hard, respectively, thus not allowing for FOL-rewritability. 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), respectively. 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-rewritable. In this class, we essentially find the languages of the *DL-Lite* [18] family.[2] Notably, the two simplest DLs of this family (namely, *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$) are rich enough to express basic ontology languages, e.g., extensions of (the DL subset of) RDFS[3] or fragments of OWL 2[4]; conceptual data models, e.g., Entity-Relationship [21]; and object-oriented formalisms, e.g., basic UML class diagrams.[5] In fact, in the present paper we consider a new DL of the *DL-Lite* family, called *DLR-Lite*$_{\mathcal{A},\sqcap}$, which generalizes both *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$ by allowing for the use of n-ary relations between (instances of) concepts, the specification of keys on relations, combined together (in a controlled way) with inclusions between (projections on) relations, and the use of conjunctions in the left-hand side of the inclusion assertions constituting the knowledge base TBox. We show that for such a DL query answering is FOL-rewritable.
- We show that minimal additions to the languages considered above make the data complexity of conjunctive query answering NLogSpace-hard or PTime-hard, thus losing the possibility of reformulating queries in first-order logic. In spite of the fact that for such languages query answering is polynomially tractable (in NLogSpace and PTime, respectively), these hardness results tell us that for 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 DLs that enjoy FOL-rewritability of conjunctive query answering and that cannot be extended with any construct typical of DLs [23] without losing this property.[6] In this sense, the DLs of the *DL-Lite* family studied here are the maximal logics that allow for answering conjunctive queries through standard database technology.

The paper is organized as follows. In Section 2 we introduce some preliminary notions on DLs and query answering, and present the DLs which we deal with in this paper, including *DLR-Lite*$_{\mathcal{A},\sqcap}$. In Section 3 we show that for this DL query answering and KB satisfiability are FOL-rewritable. Then, in Section 4 we deal with DLs for which query answering goes

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

[2] Not to be confused with the set of DLs studied in [20], which form the *DL-Lite*$_{bool}$ family.

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

[4] http://www.w3.org/TR/owl2-overview/.

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

[6] Actually, our mandatory participation and functionality constructs can be extended to unqualified number restrictions in a rather straightforward way [20,24].

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. Finally, in Section 5 we overview related work, and in Section 6 we draw some conclusions.

We point out that the present paper is an extended and revised version of [25]. In particular, the logic *DLR-Lite*$_{\mathcal{A},\sqcap}$ studied in this paper generalizes the *DL-Lite* logics considered in [25], since it allows for the use of n-ary relations rather than binary roles (this case has been only briefly commented in [25]), and for a (controlled) combination of keys on relations with inclusions between relations (which have been studied separately in [25]). Furthermore, we show here also computational complexity upper bounds for non-FOL-rewritable DLs, which were not considered in [25]. Finally, in the present paper we provide complete proofs of all the results, and a detailed related work analysis.

2. Preliminaries

Description logics (DLs) [23] are logics that represent the domain of interest in terms of *objects*, i.e., individuals, *concepts*, which are abstractions for sets of objects, and *relations* among concepts. Relations are typically binary in DLs (they are called *roles*), but in this paper we also consider n-ary relations, in the spirit of the DL *DLR* [5,9].

In the rest of the paper, we implicitly refer to a signature $\mathcal{S}$, and therefore we often omit to refer to it explicitly. The signature $\mathcal{S}$ includes symbols for constants (also called *individuals*), unary predicates (also called *atomic concepts*), binary predicates (also called *atomic roles*), n-ary (with $n \geqslant 2$) predicates (also called *atomic relations*). The arity of a relation R, denoted $ar(R)$, is the number of its arguments, also called its components.

A DL knowledge base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ over $\mathcal{S}$ is a pair formed by a set $\mathcal{T}$ of assertions, called *TBox*, and a set $\mathcal{A}$ of assertions, called *ABox*. Intuitively, $\mathcal{T}$ contains intensional assertions, i.e., axioms specifying general properties of concepts, roles, and relations, while $\mathcal{A}$ contains extensional assertions, i.e., axioms about individual objects.

Definition 2.1 *(Knowledge base).* A DL *knowledge base* over a signature $\mathcal{S}$ is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where:

- $\mathcal{T}$, called the TBox of $\mathcal{K}$, is a finite set of intensional assertions (also called TBox assertions) over $\mathcal{S}$;
- $\mathcal{A}$, called the ABox of $\mathcal{K}$, is a finite set of extensional assertions (also called ABox assertions) over $\mathcal{S}$ of the form:

$$\begin{array}{ll} A(a_1) & (\textit{concept membership assertion}), \\ P(a_1, a_2) & (\textit{role membership assertion}), \\ R(a_1, \ldots, a_n) & (\textit{relation membership assertion}), \end{array}$$

with A, P, and R denoting respectively an atomic concept symbol, an atomic role symbol, and an n-ary atomic relation symbol, for $n \geqslant 2$, and $a_1, \ldots, a_n$ denoting constant symbols. □

Informally, a concept membership assertion specifies that an object is an instance of an atomic concept. Analogously, the other types of membership assertions specify instances of atomic roles and relations.

Later in the paper, we will illustrate the form of TBox assertions. What is important to note here about such assertions is that they are specified using not only atomic concepts, roles, and relations, but also complex expressions. Complex concept expressions are constructed starting from atomic concepts by applying suitable operators. Analogously, for complex roles and complex relations. Different DLs allow for both different concept and role expressions, and different TBox intensional assertions. In other words, defining a specific DL means providing a specification of both the language for building complex expressions, and the language for specifying intensional assertions.

We start with the definition of the concept, role, and relation expressions allowed in the various DLs considered in this paper. The whole set of relevant constructs are shown in the following syntactic rules:

$$\begin{array}{lcl} C & \longrightarrow & A \mid \neg C \mid C \sqcap \cdots \sqcap C \mid C \sqcup \cdots \sqcup C \mid \exists Q \mid \exists Q.C \mid \forall Q.C \mid \exists i{:}R \\ Q & \longrightarrow & P \mid P^- \mid \neg P \mid \neg P^- \\ V & \longrightarrow & R \mid \neg R \mid R[i_1, \ldots, i_h] \mid \neg R[i_1, \ldots, i_h], \end{array}$$

where

- A denotes an atomic concept, P an atomic role, R an atomic relation, C an arbitrary (i.e., either atomic or complex) concept, Q an arbitrary role, and V an arbitrary relation. All these symbols will be used with subscripts, when needed.
- In an expression of the form $\exists i{:}R$, we have that $i \in \{1, \ldots, ar(R)\}$.
- In an expression of the form $R[i_1, \ldots, i_h]$, we have that $i_j \in \{1, \ldots, ar(R)\}$, for each $j \in \{1, \ldots, h\}$, and $i_j \neq i_\ell$, for $j, \ell \in \{1, \ldots, h\}$ with $j \neq \ell$. Such an expression denotes a relation of arity h whose components are $1, \ldots, h$, and such that the component i_j of R corresponds to component j of $R[i_1, \ldots, i_h]$. Notice that, when R has arity n, then $R[1, \ldots, n]$ coincides with R.

The semantics of a DL KB is given in terms of first-order interpretations, where an *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty interpretation domain $\Delta^{\mathcal{I}}$, and an *interpretation function* $\cdot^{\mathcal{I}}$, assigning to each atomic concept A a subset $A^{\mathcal{I}}$

Table 1
The DL constructs considered in this article with their semantics.

Construct	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Atomic role	P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Atomic relation	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$
Concept negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Concept conjunction	$C_1 \sqcap \cdots \sqcap C_n$	$C_1^{\mathcal{I}} \cap \cdots \cap C_n^{\mathcal{I}}$
Concept disjunction	$C_1 \sqcup \cdots \sqcup C_n$	$C_1^{\mathcal{I}} \cup \cdots \cup C_n^{\mathcal{I}}$
Universal quantification	$\forall Q.C$	$\{o \mid \forall o'.(o,o') \in Q^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
Unqualified existential role quantification	$\exists Q$	$\{o \mid \exists o'.(o,o') \in Q^{\mathcal{I}}\}$
Qualified existential role quantification	$\exists Q.C$	$\{o \mid \exists o'.(o,o') \in Q^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}$
Unqualified existential relation quantification	$\exists i{:}R$	$\{o' \mid \exists \vec{o} \in R^{\mathcal{I}}.\vec{o}[i] = o'\}$
Inverse role	P^-	$\{(o,o') \mid (o',o) \in P^{\mathcal{I}}\}$
Role negation	$\neg Q$	$(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus Q^{\mathcal{I}}$
Relation projection	$R[i_1,\ldots,i_h]$	$\{\vec{o}[i_1,\ldots,i_h] \mid \vec{o} \in R^{\mathcal{I}}\}$
Relation negation	$\neg V$	$(\Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}) \setminus V^{\mathcal{I}}$

Table 2
The TBox assertions considered in this article with their semantics.

TBox assertion	Syntax	Semantics
Concept inclusion	$Cl \sqsubseteq Cr$	$Cl^{\mathcal{I}} \subseteq Cr^{\mathcal{I}}$
Role inclusion	$Ql \sqsubseteq Qr$	$Ql^{\mathcal{I}} \subseteq Qr^{\mathcal{I}}$
Relation inclusion	$Vl \sqsubseteq Vr$	$Vl^{\mathcal{I}} \subseteq Vr^{\mathcal{I}}$
Role functionality assertion	(funct Q)	$\forall o_1.\forall o_2.\forall o_3.$ $(o_1,o_2) \in Q^{\mathcal{I}} \wedge (o_1,o_3) \in Q^{\mathcal{I}} \rightarrow o_2 = o_3$
Relation key assertion	(key $j_1,\ldots,j_\ell$: V)	$\forall \vec{o}_1 \in V^{\mathcal{I}}.\forall \vec{o}_2 \in V^{\mathcal{I}}.$ $\vec{o}_1[j_1,\ldots,j_\ell] = \vec{o}_2[j_1,\ldots,j_\ell] \rightarrow \vec{o}_1 = \vec{o}_2$

of $\Delta^{\mathcal{I}}$, to each atomic role P a binary relation $P^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$ (i.e., a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$), and to each n-ary relation R an n-ary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$. Also, $\cdot^{\mathcal{I}}$ assigns to each constant a an object $a^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$. All the DLs discussed in this paper follow the unique name assumption, and, therefore, if a_1 and a_2 are different constants, then the objects $a_1^{\mathcal{I}}$ and $a_2^{\mathcal{I}}$ are different as well.

In the following, we use $\vec{o}$ to denote an n-tuple of objects in $\Delta^{\mathcal{I}}$, and $\vec{o}[i]$, where $i \in \{1,\ldots,n\}$, to denote the i-th component of $\vec{o}$. Also, we will use $\vec{o}[i_1,\ldots,i_h]$, where $i_1,\ldots,i_h \in \{1,\ldots,n\}$ and $i_j \neq i_\ell$, for $j,\ell \in \{1,\ldots,h\}$ with $j \neq \ell$, as a shortcut for $(\vec{o}[i_1],\ldots,\vec{o}[i_h])$. Finally, for $\vec{a} = (a_1,\ldots,a_n)$, where $a_1,\ldots,a_n$ are constants, we use $\vec{a}^{\mathcal{I}}$ as a shortcut for $(a_1^{\mathcal{I}},\ldots,a_n^{\mathcal{I}})$.

The semantics of all the constructs that are relevant for this article is shown in Table 1. For each of the constructs, the table shows its name, its syntax, and its semantics.

We now turn to the definition of TBox assertions. In this paper, we consider three kinds of TBox assertions:

- *Inclusion assertions* between concepts, stating that all instances of one concept are also instances of another concept. Analogous assertions specify inclusions between roles, and inclusions between relations.
- *Functional assertions*, stating that a role is functional.
- *Key assertions*, stating that a set of components is a key for a relation.

Table 2 illustrates the various TBox assertions mentioned above, by describing their syntax and their semantics. In particular, the "Semantics" column of the table specifies, for each assertion, which is the condition that an interpretation $\mathcal{I}$ must obey in order to satisfy the assertion. Note that:

- In the concept inclusion assertions, Cl (resp., Cr) denotes a concept used in the left-hand side (resp., right-hand side) of the inclusion. The distinction between Cl and Cr is motivated by the fact that the constraints that the various DLs put on the form of concept expressions appearing in one side of the inclusion are often different with respect to those in the other side. Analogous observation holds for both role and relation inclusions assertions.
- In an expression of the form (key $j_1,\ldots,j_\ell$: V), we have that $j_k \in \{1,\ldots,ar(V)\}$, for each $k \in \{1,\ldots,\ell\}$, and $j_k \neq j_m$ for each $k,m \in \{1,\ldots,\ell\}$ with $k \neq m$. In particular, when R is a relation of arity n, in an expression (key $j_1,\ldots,j_\ell$: $R[i_1,\ldots,i_h]$) we have that $i_1,\ldots,i_h \in \{1,\ldots,n\}$ and $j_1,\ldots,j_\ell \in \{1,\ldots,h\}$.

We are now ready to complete the definition of the semantics of KBs. For this purpose, the basic definitions are as follows:

- An interpretation $\mathcal{I}$ is a model of a TBox assertion α if $\mathcal{I}$ satisfies α, according to what reported in Table 2.
- An interpretation $\mathcal{I}$ is a model of (or equivalently *satisfies*) a membership assertion $A(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$. It is a model of $P(a_1, a_2)$ if $(a_1, a_2) \in P^{\mathcal{I}}$, and it is a model of $R(a_1, \ldots, a_n)$ if $(a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}) \in R^{\mathcal{I}}$.

A *model of a KB* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ 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.

Example 2.2. Let us assume that our signature includes the atomic concepts *Supplier*, *Customer*, and *Product*, the ternary relation *supply*, and the binary relation *clientOf*. The following is a TBox $\mathcal{T}$:

$$\exists 1{:}\mathit{supply} \sqsubseteq \mathit{Supplier} \tag{1}$$

$$\exists 2{:}\mathit{supply} \sqsubseteq \mathit{Customer} \tag{2}$$

$$\exists 3{:}\mathit{supply} \sqsubseteq \mathit{Product} \tag{3}$$

$$\mathit{Supplier} \sqsubseteq \neg \mathit{Product} \tag{4}$$

$$\mathit{Customer} \sqsubseteq \neg \mathit{Product} \tag{5}$$

$$(\mathsf{key}\, 2, 3{:}\ \mathit{supply}) \tag{6}$$

$$\mathit{Supplier} \sqcap \mathit{Customer} \sqsubseteq \exists 1{:}\mathit{supply} \tag{7}$$

$$\mathit{Supplier} \sqcap \mathit{Customer} \sqsubseteq \exists 2{:}\mathit{supply} \tag{8}$$

$$\mathit{supply}[1, 2] \sqsubseteq \mathit{clientOf}[2, 1]. \tag{9}$$

In the above TBox, inclusions (1)–(3) specify the domain respectively of the first, second, and third component of the relation *supply*, with the intended meaning that suppliers provide customers with products. Assertions (4) and (5) impose that the set of products is disjoint from the set of customers and the set of suppliers, respectively. Assertion (6) imposes that positions 2 and 3 in *supply* constitute the key of *supply*, with the intended meaning that a customer for a certain product has only one supplier. Assertions (7) and (8) specify that those individuals that are both suppliers and customers must participate in both the first and the second component of the relation *supply*. Finally, assertion (9) says that each individual that is a supplier of a customer (for a certain product), has such a customer as a client.

As an example of ABox $\mathcal{A}$, consider

$$\mathit{Customer}(\mathsf{SmithInc})$$

$$\mathit{Supplier}(\mathsf{SmithInc})$$

$$\mathit{clientOf}(\mathsf{SmithInc}, \mathsf{SmartCompany}). \qquad \square$$

In the rest of this paper, each of the DLs that we will refer to will be characterized by the following elements:

1. the form of the concept expressions *Cl* and *Cr*,
2. the form of the role expressions *Ql* and *Qr*,
3. the form of the relation expressions *Vl* and *Vr*, and
4. the type of TBox assertions allowed in the DL.

Note that, when in the description of a DL, the second item (resp., the third item) is missing, this means simply that the DL includes only relations (resp., roles), and not roles (resp., n-ary relations).

2.1. The DL-Lite family

The *DL-Lite* family [18] is a family of DLs specifically tailored to capture knowledge representation and ontology languages, while allowing reasoning tasks to be carried out efficiently. In particular, the distinguishing feature of the DLs of this family is that query answering has the same computational complexity as in relational databases, if one measures the complexity with respect to the size of the ABox only. The goal of this subsection is to provide the definition of the DLs of the *DL-Lite* family.

Table 3 describes the basic members of the *DL-Lite* family, studied in detail in [18,26]. In all these DLs, only roles (i.e., binary relations) are allowed. Note that the symbol $(*)$ associated to the TBox assertion $(\mathsf{funct}\, Q)$ of $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ indicates that in this DL the following restriction on the use of such assertions holds:

Table 3
The basic DLs of the *DL-Lite* family.

	$DL\text{-}Lite_{core}$	$DL\text{-}Lite_{\mathcal{F}}$	$DL\text{-}Lite_{\mathcal{R}}$	$DL\text{-}Lite_{\mathcal{A}}$
Cl	$A \mid \exists P \mid \exists P^-$			
Cr	$Cl \mid \neg Cl$			
Ql	–	$P \mid P^-$		
Qr	–	$Ql \mid \neg Ql$		
Vl, Vr	–			
TBox assertions	$Cl \sqsubseteq Cr$	$Cl \sqsubseteq Cr$ (funct Ql)	$Cl \sqsubseteq Cr$ $Ql \sqsubseteq Qr$	$Cl \sqsubseteq Cr$ $Ql \sqsubseteq Qr$ (funct Ql)$^{(*)}$

Table 4
The DL $DLR\text{-}Lite_{\mathcal{A},\sqcap}$.

	$DLR\text{-}Lite_{\mathcal{A},\sqcap}$
Cl	$A \mid \exists i{:}R \mid Cl_1 \sqcap \cdots \sqcap Cl_n$
Cr	$A \mid \exists i{:}R \mid \neg A \mid \neg\exists i{:}R$
Ql, Qr	–
Vl	$R \mid R[i_1, \ldots, i_h]$
Vr	$Vl \mid \neg Vl$
TBox assertions	$Cl \sqsubseteq Cr$ $Vl \sqsubseteq Vr$ (key $j_1, \ldots, j_\ell$: Vl)$^{(*)}$

(∗) In a $DL\text{-}Lite_{\mathcal{A}}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, for each role P such that in $\mathcal{T}$ there is an assertion (funct P) or (funct P^-), in $\mathcal{T}$ there is no assertion of the form $Q \sqsubseteq P$ and no assertion of the form $Q \sqsubseteq P^-$.

In other words, functional roles cannot be specialized in the TBox. This restriction is crucial for keeping query answering efficient, as we will demonstrate in Section 4.

We observe that $DL\text{-}Lite_{\mathcal{A}}$, as defined in [18,26], is actually richer than the DL described in Table 3, because it includes constructs for modeling concept attributes, which are binary relations between concepts and value-domains. However, from the technical point of view, attributes can be considered essentially as roles, and therefore we ignore them here.

Although much of the technical work done so far on the *DL-Lite* family deals with the basic members, in this paper we consider a new member of the family, called $DLR\text{-}Lite_{\mathcal{A},\sqcap}$, which is characterized by the following features:

- it allows for modeling a domain not only in terms of concepts and roles, but also in terms of n-ary relations;
- it allows for the specification of inclusions between (projections of) n-ary relations;
- it provides the possibility of specifying conjunctions in the left-hand side of inclusions between concepts;
- it allows for the specification of key constraints on (projections of) n-ary relations.

We provide the definition of $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ in Table 4. In the following, we say that the key assertion (key $j_1, \ldots, j_\ell$: Vl) is *on relation* R if Vl is either R or $R[i_1, \ldots, i_h]$.

Analogously to the case of $DL\text{-}Lite_{\mathcal{A}}$, the symbol (∗) associated to the TBox assertion (key $j_1, \ldots, j_\ell$: Vl) of $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ indicates that in this DL the following restriction on the use of such assertions holds:

(∗) In a $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, for each relation R such that in $\mathcal{T}$ there is a key assertion on R, in $\mathcal{T}$ there is no assertion of the form $V \sqsubseteq R$ and no assertion of the form $V \sqsubseteq R[i_1, \ldots, i_h]$.

In other words, relations occurring in key assertions in $\mathcal{T}$ cannot be specialized, i.e., they cannot occur positively in the right-hand side of inclusion assertions between relations. Observe that the KB discussed in Example 2.2 is a $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ KB.

Hereinafter, we call *positive inclusions* (*PIs*) assertions of the form $Cl \sqsubseteq A$, $Cl \sqsubseteq \exists i{:}R$, $Vl \sqsubseteq R$, and $Vl \sqsubseteq R[i_1, \ldots, i_h]$. Moreover, we call *negative inclusions* (*NIs*) assertions of the form $Cl \sqsubseteq \neg A$, $Cl \sqsubseteq \neg\exists i{:}R$, $Vl \sqsubseteq \neg R$, and $Vl \sqsubseteq \neg R[i_1, \ldots, i_h]$.

Note that, analogously to $DL\text{-}Lite_{\mathcal{A}}$, $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ includes concept attributes, but we ignore them in this paper. It is immediate to verify that $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ is more expressive than all the basic members of the *DL-Lite* family. Indeed, although $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ does not include roles, they can obviously be captured by relations of arity 2. Analogously, constructs like $\exists R$ and P^- can be easily expressed in terms of the constructs of $DLR\text{-}Lite_{\mathcal{A},\sqcap}$. We observe that $DLR\text{-}Lite_{\mathcal{A},\sqcap}$ might also be enhanced with the capability of managing qualified existential quantification on the right-hand side of inclusion assertions between concepts, i.e., adding to Cr the construct

$$\exists i{:}R.A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n,$$

where R is an n-ary relation with no key constraint in $\mathcal{T}$, and $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n$ are atomic concepts [25]. The semantics of the above construct is defined as follows. Given an interpretation $\mathcal{I}$, we have that $(\exists i{:}R.A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)^{\mathcal{I}}$ is

$$\left\{\vec{o}[i] \mid \vec{o} \in R^{\mathcal{I}} \text{ and } \vec{o}[j] \in A_j^{\mathcal{I}}, \text{ for } j \in \{1, \ldots, i-1, i+1, \ldots, n\}\right\}.$$

This construct, however, can be simulated by suitably using inclusions between relations and unqualified existential quantification on relations in inclusions between concepts. More precisely, we can replace each assertion of the form $Cl \sqsubseteq \exists i{:}R.A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n$ with the assertions

$$Cl \sqsubseteq \exists i{:}\hat{R}$$
$$\hat{R} \sqsubseteq R$$
$$\exists 1{:}\hat{R} \sqsubseteq A_1$$
$$\ldots$$
$$\exists i-1{:}\hat{R} \sqsubseteq A_{i-1}$$
$$\exists i+1{:}\hat{R} \sqsubseteq A_{i+1}$$
$$\ldots$$
$$\exists n{:}\hat{R} \sqsubseteq A_n,$$

where $\hat{R}$ is a fresh n-ary relation. Therefore, in the following we do not explicitly consider qualified existential quantification. Other logics allowing for different usages of qualified existential quantification will be analyzed in the next sections.

We conclude by emphasizing that *DLR-Lite*$_{\mathcal{A},\sqcap}$ is the first DL of the *DL-Lite* family that allows for the use of (i) n-ary relations, rather than binary roles, (ii) key assertions, rather than simple functionalities, and (iii) conjunctions in the left-hand side of inclusion assertions. Nonetheless, we will show in the next section that for *DLR-Lite*$_{\mathcal{A},\sqcap}$ KBs, both query answering and KB satisfiability are FOL-rewritable, i.e., such a DL presents the distinguishing fundamental properties of the DLs of the *DL-Lite* family (cf. [18]).

2.2. Query answering

By using queries, we can extract information from the extensional level of a KB $\mathcal{K}$ expressed in a DL. We start with a general notion of queries in first-order logic, and then we move to the definition of queries over a DL KB.

A query is an open formula of first-order logic with equalities (FOL, in the following). Formally, A *FOL query* q is an expression of the form

$$\left\{\vec{x} \mid \phi(\vec{x})\right\},$$

where $\phi(\vec{x})$ is a FOL formula with free variables $\vec{x}$. We call the size of $\vec{x}$ the *arity* of q. Given an interpretation $\mathcal{I}$, $q^{\mathcal{I}}$ is the set of tuples of domain elements that, when assigned to the free variables, make the formula ϕ true in $\mathcal{I}$ [21]. A *boolean query* is a query that does not involve any free variable (i.e., ϕ is a closed formula). Given an interpretation $\mathcal{I}$, if a boolean query q is *true* in $\mathcal{I}$ then $q^{\mathcal{I}}$ consists only of the *empty tuple*, i.e., the tuple of arity 0; instead, if q is *false* in $\mathcal{I}$ then $q^{\mathcal{I}}$ is obviously empty. Finally, a *ground query* is a boolean query that does not contain any variable.

We are interested in conjunctive queries and unions of conjunctive queries. A *conjunctive query* (*CQ*) q is a query of the form

$$\left\{\vec{x} \mid \exists \vec{y}.\, conj(\vec{x}, \vec{y})\right\},$$

where $conj(\vec{x}, \vec{y})$ is a conjunction of atoms and equalities, with variables $\vec{x}$ and $\vec{y}$. A *union of conjunctive queries* (*UCQ*) Q, is a query of the form

$$\left\{\vec{x} \,\middle|\, \bigvee_{i \in \{1,\ldots,n\}} \exists \vec{y}_i.\, conj_i(\vec{x}, \vec{y}_i)\right\},$$

where each $conj_i(\vec{x}, \vec{y}_i)$ is, as before, a conjunction of atoms and equalities with free variables $\vec{x}$ and $\vec{y}_i$. Obviously, the class of unions of conjunctive queries contains the class of conjunctive queries.

For convenience, we adopt the usual Datalog notation (see e.g., [21]). Namely, a conjunctive query $q = \{\vec{x} \mid \exists \vec{y}.\, conj(\vec{x}, \vec{y})\}$ is denoted as

$$q(\vec{x}') \leftarrow conj'(\vec{x}', \vec{y}'),$$

where $conj'(\vec{x}', \vec{y}')$ is the list of atoms in $conj(\vec{x}, \vec{y})$ obtained after having equated the variables $\vec{x}, \vec{y}$ according to the equalities in $conj(\vec{x}, \vec{y})$. As a result of such equality elimination, we have that $\vec{x}'$ and $\vec{y}'$ can actually contain constants and multiple occurrences of the same variable. We call $q(\vec{x}')$ the *head* of q, denoted $head(q)$, and $conj'(\vec{x}', \vec{y}')$ the body, denoted $body(q)$. Moreover, we call the variables in $\vec{x}'$ the *distinguished variables* of q and those in $\vec{y}'$ the *non-distinguished variables*. If the query q is boolean, its Datalog notation is $q \leftarrow conj'(\vec{y}')$.

A union of conjunctive queries

$$Q = \left\{ \vec{x} \;\middle|\; \bigvee_{i \in \{1,\ldots,n\}} \exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i) \right\}$$

is denoted in Datalog notation as

$$Q = \{\alpha_1, \ldots, \alpha_n\},$$

where each α_i is the conjunctive query $\{\vec{x} \mid \exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i)\}$ expressed in Datalog notation. Notice that, for an interpretation $\mathcal{I}$, we have that $Q^{\mathcal{I}} = \bigcup_{i \in \{1,\ldots,n\}} \alpha_i^{\mathcal{I}}$.

The size of a CQ q, denoted with $size(q)$, is the number of atoms occurring in its body when q is given in Datalog notation. The size, $size(Q)$, of a UCQ Q coincides with the maximum among the sizes of the CQs contained in Q.

We can now define queries over a DL KB. We will concentrate on conjunctive queries and unions of conjunctive queries, only. A *conjunctive query over a TBox* $\mathcal{T}$ is a conjunctive query whose atoms are of the form $A(z)$ or $R(z_1, \ldots, z_n)$ where A and R are respectively an atomic concept and a relation of $\mathcal{T}$ and $z, z_1, \ldots, z_n$ are either constants or (possibly non-distinct) variables. Similarly, we define *unions of conjunctive queries over a TBox* $\mathcal{T}$. We also say that a conjunctive query q is specified over a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ if q is a conjunctive query over $\mathcal{T}$.

The reasoning service we are interested in is *(conjunctive) query answering*: given a satisfiable knowledge base $\mathcal{K}$ and a union of conjunctive queries $Q(\vec{x})$ over $\mathcal{K}$, return all tuples $\vec{a}$ of constants in $\mathcal{K}$ such that, when substituted to the variables $\vec{x}$ in $Q(\vec{x})$, denoted $q(\vec{a})$, 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 denote with $ans(Q, \mathcal{K})$ the set of such tuples. When the query Q is boolean, $ans(Q, \mathcal{K})$ contains the empty tuple if $\mathcal{K} \models Q$, i.e., if $Q^{\mathcal{I}}$ is true in every model $\mathcal{I}$ of $\mathcal{K}$, whereas $ans(Q, \mathcal{K}) = \emptyset$, otherwise.

We point out that defining query answering only for satisfiable KBs is not a simplification. Indeed, from the "ex falso quod libet" principle, it follows that the answers to a query of arity n posed over an unsatisfiable KB $\mathcal{K}$ would be trivially all possible tuples of constants in $\mathcal{K}$ whose arity is the one of the query. Since we are not interested in getting such answers, we have defined query answering only over satisfiable KBs, and we will perform query answering only after a check on the satisfiability of the KB at hand.

We observe that query answering (properly) generalizes two well-known reasoning services in DLs. The first one is *instance checking*, i.e., logical implication of an ABox assertion, which can be expressed as the problem of answering boolean ground queries whose body contains exactly one ground atom. The second one is *retrieval*, i.e., determining all individuals that are logically implied to be instances of a concept, which can be expressed as the problem of answering a unary query whose body contains exactly one unary atom.

Finally, we refer to *data complexity* of query answering, which is a notion borrowed from relational database theory [12], and in the context of DLs is defined as follows. First, we note that there is a decision problem associated with query answering: fixed a TBox $\mathcal{T}$ expressed in a DL $\mathcal{L}$, and a query q, the *recognition problem* associated to $\mathcal{T}$ and q is the decision problem of checking whether, given an ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, and a tuple $\vec{a}$ of constants, we have that $\langle \mathcal{T}, \mathcal{A} \rangle \models q(\vec{a})$. Note that neither the TBox nor the query is an input to the recognition problem.

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

2.3. The notion of FOL-rewritability

We now introduce the notion of FOL-rewritability of query answering and KB satisfiability. We start by giving the notion of $\mathcal{Q}$*-rewritability* of query answering and KB satisfiability, where $\mathcal{Q}$ is a given query language. To this purpose, given an ABox $\mathcal{A}$ (of the kind considered above), we define the interpretation $db(\mathcal{A}) = \langle \Delta^{db(\mathcal{A})}, \cdot^{db(\mathcal{A})} \rangle$ as follows:

- if $\mathcal{A} \neq \emptyset$, then $\Delta^{db(\mathcal{A})}$ is the set consisting of all constants occurring in $\mathcal{A}$, otherwise, if $\mathcal{A} = \emptyset$, then $\Delta^{db(\mathcal{A})} = \{c\}$ where c is some constant symbol,
- $a^{db(\mathcal{A})} = a$, for each constant a,
- $A^{db(\mathcal{A})} = \{a \mid A(a) \in \mathcal{A}\}$, for each atomic concept A, and
- $R^{db(\mathcal{A})} = \{(a_1, \ldots, a_n) \mid R(a_1, \ldots, a_n) \in \mathcal{A}\}$, for each n-ary atomic relation R.

Definition 2.3. Query answering in a DL $\mathcal{L}$ is $\mathcal{Q}$*-rewritable*, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$ and every (conjunctive) query q over $\mathcal{T}$, one can effectively compute a query q_r over $\mathcal{T}$, belonging to the query language $\mathcal{Q}$, such that for

every ABox $\mathcal{A}$, for which $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, and every tuple of constants $\vec{a}$ occurring in $\mathcal{A}$, $\langle \mathcal{T}, \mathcal{A} \rangle \models q(\vec{a})$ if and only if $\vec{a}^{db(\mathcal{A})} \in q_r^{db(\mathcal{A})}$. The query q_r is called the *$\mathcal{Q}$-rewriting of q w.r.t. $\mathcal{T}$*. □

In other words, $\mathcal{Q}$-rewritability of query answering captures the property that we can reduce query answering to evaluating a query belonging to the query language $\mathcal{Q}$ over the ABox $\mathcal{A}$ considered as a relational database, i.e., over $db(\mathcal{A})$.

Analogously, we can define $\mathcal{Q}$-rewritability of KB satisfiability.

Definition 2.4. KB satisfiability in a DL $\mathcal{L}$ is *$\mathcal{Q}$-rewritable*, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, one can effectively compute a boolean query q_r, over $\mathcal{T}$, belonging to the query language $\mathcal{Q}$, such that for every ABox $\mathcal{A}$, $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if q_r evaluates to false in $db(\mathcal{A})$, i.e., $q_r^{db(\mathcal{A})} = \emptyset$. □

One of the most interesting classes of queries to be considered for $\mathcal{Q}$ is that of FOL queries, since, from the practical point of view, FOL queries correspond to queries expressed in relational algebra (i.e., in SQL). In other words, (the SQL encoding of) FOL queries can be easily evaluated by an SQL engine over a simple relational database defined by ABox assertions (i.e., the relational database corresponding to the interpretation $db(\mathcal{A})$ defined above), thus taking advantage of well-established query optimization strategies supported by current industrial strength relational technology.

Observe that every FOL query can be evaluated in AC^0 with respect to data complexity (see e.g., [21]). It follows that, if query answering (or KB satisfiability) in $\mathcal{L}$ is FOL-rewritable, then query answering (resp., KB satisfiability) in $\mathcal{L}$ is in AC^0 w.r.t. data complexity. Vice-versa, if query answering (or KB satisfiability) is $\mathcal{C}$-hard w.r.t. data complexity for some complexity class $\mathcal{C}$ that strictly contains AC^0 (e.g., LogSpace, NLogSpace, PTime, coNP, etc.), then it is not FOL-rewritable.

In the following, we study FOL-rewritability of KB satisfiability only in those cases in which query answering is FOL-rewritable. Indeed, checking the satisfiability of a KB is always needed before answering a query posed over it, and establishing FOL-rewritability of both reasoning services guarantees the possibility of completely relying on relational database technology to perform them. In all the other cases, we analyze the computational complexity of query answering only.

3. FOL-rewritability in *DLR-Lite*$_{\mathcal{A},\sqcap}$

In this section we provide algorithms that reduce CQ answering and KB satisfiability in *DLR-Lite*$_{\mathcal{A},\sqcap}$ to first-order logic query evaluation, thus showing FOL-rewritability (and therefore membership in AC^0) of both such reasoning services.

We first study query answering and provide an algorithm, called PerfectRef, that takes as input a *DLR-Lite*$_{\mathcal{A},\sqcap}$ TBox $\mathcal{T}$ and a union of conjunctive queries Q and returns a union of conjunctive queries Q_r, which we show to be the FOL-rewriting of Q w.r.t. $\mathcal{T}$. In a nutshell, the algorithm compiles in Q_r both the query Q and the assertions of $\mathcal{T}$ that are relevant to compute the answers to Q. Notably, we will show that, to obtain Q_r, only the PIs explicitly asserted in $\mathcal{T}$ have to be taken into account (see Theorem 3.9).

We then deal with KB satisfiability, which we in fact reduce to query answering (of a suitable boolean query), and show that PerfectRef can be used to solve this problem through rewriting into FOL.

From now on, we assume that both PIs and NIs are transformed as described next. As for PIs, we substitute each occurrence of an atomic concept A with $A[1]$, and each occurrence of a concept of the form $\exists i{:}R$ with $R[i]$. For example, we transform the inclusion $\exists 3{:}R_1 \sqcap A \sqsubseteq \exists 2{:}R_2$ in $R_1[3] \sqcap A[1] \sqsubseteq R_2[2]$. In this way, both positive concept inclusions and positive relation inclusions in $\mathcal{T}$ are specified according to the following syntax:

$$S_1[i_{1,1}, \ldots, i_{1,k}] \sqcap \cdots \sqcap S_h[i_{h,1}, \ldots, i_{h,k}] \sqsubseteq S[i_1, \ldots, i_k], \tag{10}$$

where each of $S, S_1, \ldots, S_h$ may be an atomic concept or a relation, and $k \leqslant \min(m, m_1, \ldots, m_h)$, with $m, m_1, \ldots, m_h$ denoting the arities of $S, S_1, \ldots, S_h$, respectively. Notice that, since conjunction in the left-hand side of inclusions is allowed only in concept inclusions, we have that $k > 1$ implies $h = 1$.

We adopt an analogous transformation also for NIs, and write them according to the following syntax:

$$S_1[i_{1,1}, \ldots, i_{1,k}] \sqcap \cdots \sqcap S_h[i_{h,1}, \ldots, i_{h,k}] \sqsubseteq \neg S_0[i_{0,1}, \ldots, i_{0,k}], \tag{11}$$

where $S_0, S_1, \ldots, S_h$, $m_0, m_1, \ldots, m_h$, and k are defined as for positive inclusions. Again, $k > 1$ implies $h = 1$.

In the following, we will always use S, possibly with subscripts, to denote either an atomic concept or an atomic relation. In the former case, we always have $ar(S) = 1$.

Example 3.1. The TBox given in Example 2.2 is transformed in the following way:

$$supply[1] \sqsubseteq Supplier[1] \tag{12}$$

$$supply[2] \sqsubseteq Customer[1] \tag{13}$$

$$supply[3] \sqsubseteq Product[1] \tag{14}$$

$$Supplier[1] \sqsubseteq \neg Product[1] \tag{15}$$

Algorithm PerfectRef(Q , $\mathcal{T}$)
Input: UCQ Q of arity n and size k, *DLR-Lite*$_{\mathcal{A},\sqcap}$ TBox $\mathcal{T}$
Output: UCQ Q_r
$Q_r := Q$;
$J := \{z_1, \ldots z_k\}$, with each z_i not occurring in Q ;
repeat
  $Q'_r := Q_r$;
  for each CQ $q \in Q'_r$ **do**
  (a) **for each** g_1, g_2 in q **do**
    if g_1 and g_2 unify
    then $Q_r := Q_r \cup \{\tau(reduce(q, g_1, g_2))\}$;
  (b) **for each** g in q **do**
    for each PI I in $\mathcal{T}$ **do**
      if I is applicable to g
      then $Q_r := Q_r \cup \{atomRewrite(q, g, I, J)\}$;
until $Q'_r = Q_r$;
return Q_r

Fig. 1. The algorithm PerfectRef.

$$Customer[1] \sqsubseteq \neg Product[1] \tag{16}$$

$$(\mathsf{key}\, 2, 3\colon supply) \tag{17}$$

$$Supplier[1] \sqcap Customer[1] \sqsubseteq supply[1] \tag{18}$$

$$Supplier[1] \sqcap Customer[1] \sqsubseteq supply[2] \tag{19}$$

$$supply[1, 2] \sqsubseteq clientOf[2, 1]. \quad \square \tag{20}$$

3.1. FOL-rewritability of query answering

In the following, we illustrate PerfectRef from a technical point of view, and show its termination and its correctness. We start our discussion with some preliminary notions.

We point out that, for technical reasons, PerfectRef works on (unions of) conjunctive queries specified in Datalog syntax (see Section 2). We say that an argument of an atom in a query is *bound* if it corresponds either (i) to a distinguished variable, or (ii) to a shared variable, i.e., a variable occurring at least twice in the query body (including the case of a variable occurring more than once in a single atom of the query), or (iii) to a constant. Instead, we say that an argument of an atom is *unbound* if it corresponds to a non-distinguished non-shared variable (we use the symbol '_' to represent unbound variables).

Definition 3.2. Given a query atom $g = S(x_1, \ldots, x_n)$, where each x_i is either a bound term or a '_', and a positive inclusion $I \in \mathcal{T}$ of the form (10), we say that I *is applicable to* g (on $x_{i_1}, \ldots, x_{i_k}$) if for each $\ell \in \{1, \ldots, n\}$ such that $x_\ell \neq \_$, there exists $p \in \{1, \ldots, k\}$ such that $i_p = \ell$. We say also that the arguments $x_{i_1}, \ldots, x_{i_k}$ are *propagated* by I (or that I propagates $x_{i_1}, \ldots, x_{i_k}$). $\square$

Roughly speaking, an inclusion I is applicable to an atom g if all bound arguments of g are propagated by I. For example, the positive inclusion $R'[2, 3] \sqsubseteq R[1, 2]$, where R' is of arity 3 and R is of arity 4, is applicable to the atom $R(x_1, x_2, \_, \_)$, where x_1 and x_2 are bound terms, but it is not applicable to the atom $R(x_1, x_2, \_, x_4)$, since it does not propagate the bound term x_4. We notice that the PIs of the form $Cl \sqsubseteq A$ are always applicable to an atom of the form $A(x)$, disregarding whether x is equal to _ or not.

The algorithm PerfectRef is given in Fig. 1. It is constituted by two main steps, iteratively repeated until a fixpoint is reached, namely, the *reduce* step (Step (a)), which realizes some unifications on the query, and the *atom rewrite* step (Step (b)), which rewrites query atoms with respect to applicable PIs. Roughly speaking, in the latter step, PIs are used as rewriting rules, applied from right to left, which allow one to compile away in the reformulation the intensional knowledge (represented by $\mathcal{T}$) that is relevant for answering the query. Notice that each step produces a new CQ which is added to the set of queries returned by the algorithm.

The algorithm is structurally similar to the PerfectRef algorithm presented in [18] for computing the FOL-rewriting of a UCQ over either a *DL-Lite*$_{\mathcal{R}}$ or a *DL-Lite*$_{\mathcal{F}}$ TBox. However, differently from [18], both the reduce and the atom rewrite steps have now to deal with the presence of n-ary relations, both in the query and in the inclusions, and the atom rewrite step has to properly manage the presence of conjunctions in concept PIs. In particular, atom rewrite turns out to be much more complicated than the analogous step of the algorithm in [18]. Also, the proofs of termination and correctness of the algorithm, which we will give later on, are now much more involved due to the presence of n-ary relations and conjunctions in the left-hand side of inclusion assertions. We now describe more in detail the two steps of the algorithm.

The function *reduce* takes as input a CQ q and two atoms g_1 and g_2 occurring in the body of q, and returns a conjunctive query q' obtained by applying to q the *most general unifier* between g_1 and g_2. We point out that, in unifying g_1 and g_2, each

occurrence of the _ symbol has to be considered a different unbound variable. The most general unifier substitutes each _ symbol in g_1 with the corresponding argument in g_2, and vice-versa (obviously, if both arguments are _, the resulting argument is _). For example, given the query $q(x, w) \leftarrow R(x, y, \_, z), R(w, y, \_, z), A(y)$, the unification performed by *reduce* on the first two atoms produces the query $q(x, x) \leftarrow R(x, y, \_, z), A(y)$. Notice that, by virtue of the reduce step, variables that are bound in q may become unbound in q'. The function τ is thus used to guarantee that each unbound variable is represented by the symbol _. For instance, applied to the query in the example above, τ substitutes the unbound variable z with _ and returns the query $q(x, x) \leftarrow R(x, y, \_, \_), A(y)$. Now, it may be the case that PIs that were not applicable to atoms of the query q in the input to *reduce*, may become applicable to atoms of $\tau(q')$. For example, in our ongoing example, the PI $R'[2, 3] \sqsubseteq R[1, 2]$ is applicable to the atom $R(x, y, \_, \_)$, obtained through the reduce step, but not to the atoms $R(x, y, \_, z)$ or $R(w, y, \_, z)$.

The function *atomRewrite* takes as input a CQ q, an atom g (belonging to q), a PI I, and the set J of variables, and returns a new CQ in which the atom g has been rewritten according to the PI I. Assume that g has the form $S(x_1, \ldots, x_n)$, where S is either an atomic concept or an atomic relation symbol (in the first case, $n = 1$), and that I is specified in the form (10). The function *atomRewrite* substitutes g with

$$S_1(y_{1,1}, \ldots, y_{1,m_1}), \ldots, S_h(y_{h,1}, \ldots, y_{h,m_h}),$$

where, for each $j \in \{1, \ldots, h\}$, m_j is the arity of S_j, and for each $y_{r,s}$ different cases are possible, depending on whether I is an inclusion without conjunctions (either a role or a concept inclusion) (Case (i)), or I is a concept inclusion with conjunctions that propagates a bound term (Case (ii)), or I is as in Case (ii), but propagates an unbound term (Case (iii)). More precisely,

(i) if in Eq. (10) we have that $h = 1$ (and therefore k may be greater than 1), then for each $p \in \{1, \ldots, m_1\}$, $y_{1,p} = x_{i_r}$ if there exists r such that $i_{1,r} = p$, otherwise $y_{1,p} = \_$;
(ii) if in Eq. (10) we have that $k = 1$ and $h > 1$, and in g we have that $x_{i_1} \neq \_$, then $y_{1,i_{1,1}} = \cdots = y_{h,i_{h,1}} = x_{i_1}$, and all others $y_{r,s}$ are _;
(iii) if in Eq. (10) we have that $k = 1$ and $h > 1$ (as for Case (ii)), and in g we have that $x_{i_1} = \_$, then $y_{1,i_{1,1}} = \cdots = y_{h,i_{h,1}} = z$, where z is a symbol from J that does not occur in q, and all other $y_{r,s}$ are _.

Informally, in Case (i), *atomRewrite* substitutes the input atom g with a new atom g', over the predicate S_1, in which the terms of g propagated by the inclusion I occur as arguments of S_1 in the positions specified in the left-hand side of I, whereas the other arguments of S_1 are _. For example, if $g = R(x, y, \_, \_)$ and $I = R'[2, 3] \sqsubseteq R[1, 2]$, in the query returned by *atomRewrite* g, is substituted by $R'(\_, x, y)$ (we assume that $ar(R') = 3$).

In Case (ii), *atomRewrite* substitutes g with a conjunction of atoms over the predicates $S_1, \ldots, S_h$, where each such atom contains the term x_{i_1} in the position specified in the left-hand side of the inclusion I. Other arguments in these atoms are _. Notice that all such atoms are joined through the variable x_{i_1}. For example, if $g = R(\_, y, \_, \_)$ and $I = R'[2] \sqcap R''[1] \sqsubseteq R[2]$, in the query returned by *atomRewrite*, g is substituted by $R'(\_, y, \_), R''(y, \_)$ (we assume that $ar(R') = 3$ and $ar(R'') = 2$). R' and R'' are thus joined on y.

Case (iii) is as Case (ii), with the only difference that an unbound term is now propagated. To express the join on the new atoms introduced by *atomRewrite*, a new variable not occurring in the query q has to be used. For example, if $g = R(\_, \_, \_, \_)$ and $I = R'[2] \sqcap R''[1] \sqsubseteq R[2]$, in the query returned by *atomRewrite*, g is substituted by $R'(\_, z, \_), R''(z, \_)$, where z is a new variable not occurring in the query q (to which g belongs). The procedure picks up z from the fixed set J of variables, which contains only k variable symbols, where k is the size of the query Q given as input to PerfectRef. Using only variables from J is sufficient, since it is possible to show that PerfectRef includes in the final rewriting only CQs that may contain at most k new variable symbols, other than the variables originally occurring in Q plus the _ symbol. We will prove this property in the proof of the termination of PerfectRef.

Example 3.3. Let us consider the query $q(x) \leftarrow \mathit{supply}(x, y, z), \mathit{Product}(z)$ posed over the TBox $\mathcal{T}$ of Examples 2.2 and 3.1. The algorithm applies the PI (14) and generates the query $q(x) \leftarrow \mathit{supply}(x, y, z), \mathit{supply}(w_1, w_2, z)$.[7] Applying the *reduce* operator to the atoms contained in such a query, with unifier $\{w_1/x, w_2/y\}$, we then obtain the query $q(x) \leftarrow \mathit{supply}(x, y, z)$, to which the PI (18) can be applied, thus adding to the rewriting the query $q(x) \leftarrow \mathit{Supplier}(x), \mathit{Customer}(x)$. We notice that the use of the reduce function is necessary to generate this query. The evaluation of the last query over the ABox $\mathcal{A}$ produces the set {SmithInc}. Such a set constitutes in fact the set of answers to the input query over the KB $\langle \mathcal{T}, \mathcal{A} \rangle$. □

Lemma 3.4. *Let $\mathcal{T}$ be a DLR-Lite$_{\mathcal{A},\sqcap}$ TBox, and let Q be a union of conjunctive queries over $\mathcal{T}$. Then, the algorithm* PerfectRef$(Q, \mathcal{T})$ *terminates.*

Proof. First of all, we notice that the set of terms that occur in the conjunctive queries generated by the algorithm is equal to the set of variables and constants occurring in Q, plus the symbol _, plus the k new variables of the set J (we recall that

[7] In the example, we use new symbols for indicating unbound variables introduced by rewriting steps, rather than the symbol _.

k is the size of Q, i.e., the maximum number of atoms in a CQ contained in Q). Indeed, a new variable is introduced in the rewriting by PerfectRef only when it propagates, through *atomRewrite*, an unbound term by applying a concept inclusion with conjunctions (Case (iii) of *atomRewrite*). This may happen only if the atom to which such inclusion is applied is of the form $S(\_, \ldots, \_)$, i.e., its arguments are all unbound, and k is the maximum number of atoms of this form that may simultaneously occur in a query generated by PerfectRef that takes as input a CQ q of size k. We prove this property by induction on the size of the query taken as input by PerfectRef.

Base step: Let us assume that PerfectRef takes as input a query q of size equal to 1, i.e., with exactly one atom, and show that 1 is the maximum number of atoms of the form $S(\_, \ldots, \_)$ that can simultaneously occur in a query generated by PerfectRef. Two cases are possible: (1) there is at least a bound argument in the body of q. If such argument is a distinguished variable or a constant, there is no hope to even generate a single atom of the form $S(\_, \ldots, \_)$, since all bound arguments are always propagated by the function *atomRewrite*, and distinguished variables or constants can never be transformed into unbound arguments using the function *reduce*. If bound arguments are due to self-joins on the only atom of q, then at least two bound arguments occur in such atom, and therefore the algorithm cannot generate any query whose size is greater than 1, since PIs with conjunctions can never be applied (i.e., Case (ii) and Case (iii) of *atomRewrite*, which are the only cases that produce a query whose size is greater than the size of the input query, cannot be executed). Therefore, the claim easily follows. (2) the only atom g in the query is already of the form $S(\_, \ldots, \_)$. The algorithm can generate queries whose size is greater than 1 only by applying Case (iii) of the function *atomRewrite* to g, or to an atom obtained from g by (possibly iteratively) applying Case (i) of *atomRewrite*. It is easy to see that in each query of size greater than 1 produced by PerfectRef all atoms are of the form $S(\_, \ldots, \_, z, \_, \ldots, \_)$, where z is a variable from J. Therefore, a new atom of the form $S(\_, \ldots, \_)$ can be generated by PerfectRef only after pairwise unifications of all such atoms through reduce steps, but this means that the query that contains such atom cannot contain other atoms at all, and therefore the claim follows.

Inductive step: In the following we denote with q_j the query obtained at the j-th iteration of PerfectRef which has taken as input a CQ q. Furthermore, a conjunction c of atoms in the body of q is called an isolated component of q if there are no variables that occur both in c and in another atom of q. With a little abuse of notation, we denote with c the boolean query having c as its body, and call it a sub-query of q. We also say that a (sub-)query q_c is generated from a (sub-)query q_s if q_c is in the set of queries produced by PerfectRef taking q_s as input. Given a query q and a sub-query q_s of q, we denote with $q - q_s$ the sub-query obtained by eliminating the atoms in q_s from q.

We are now ready to face the inductive step of the proof. The inductive hypothesis establishes that if PerfectRef takes as input a query whose size is at most k, then it cannot generate a query in which more than k atoms of the form $S(\_, \ldots, \_)$ occur. Let us now assume that PerfectRef takes as input a query q of size $k+1$, and that, by contradiction, PerfectRef generates a query q_j with more than $k+1$ atoms of such form. Without loss of generality we assume that q_j contains exactly $k+2$ atoms of such form and that it is generated from q_{j-1} that contains $k+1$ atoms of such form. Therefore, $q_j = \tau(\mathit{reduce}(q_{j-1}, g_1, g_2))$, where g_1 and g_2 are atoms of q_{j-1} that are not in the form $S(\_, \ldots, \_)$, and q_j contains the atom $g = S(\_, \ldots, \_)$ which is obtained by the unification of g_1 and g_2. This means that in g_1 and g_2 no constants or distinguished variables occur, and no self-joins are possible (otherwise there should be arguments that cannot be transformed in unbound arguments after the unification), and that, for the same reason, $q_g = g_1, g_2$ constitutes an isolated component of q_{j-1}. We have that q_g is generated by an isolated component q_s of q. Indeed, at each iteration PerfectRef does not create atoms that are not obtained from the rewrite or the reduce step, and therefore each sub-query in q_{j-1} must be generated from a sub-query of q. Furthermore, q_s cannot contain constants, distinguished variables or variables occurring elsewhere in $q - q_s$, otherwise they should occur also in q_g, and q_g would no longer be an isolated component of q_{j-1}. This implies that $q_{j-1} - q_g$ is an isolated component generated by $q - q_s$. Notice that PerfectRef produces the FOL-rewriting of q if it takes as input either q or q_s and $q - q_s$ separately, and the queries in the results that it produces in this second case are then combined together in all possible ways. Indeed, the only computation that PerfectRef might execute in the former case, but not in the latter, is the reduction of atoms having all arguments unbound and belonging to the two different components, but this produces queries that are equivalent to other queries in the rewriting generated in the former case. Now, since $\mathit{size}(q - q_s) = k$ and $q_{j-1} - (g_1, g_2)$ contains $k+1$ atoms of the form $S(\_, \ldots, \_)$, we have reached a contradiction.

We can now prove termination of PerfectRef, for each q and $\mathcal{T}$ in input, which indeed follows then from the following facts:

1. Let n be proportional to the size of the input query Q and to the number of terms occurring in it. The cardinality of the set of terms that occur in the conjunctive queries generated by the algorithm is then less than or equal to $2n+1$ (as shown above).
2. As a consequence of the above point, the number of different atoms that may occur in a conjunctive query generated by the algorithm is less than or equal to $m \cdot (2n+1)^h$, where m is the number of predicate symbols (concepts or relations) that occur in the signature of the TBox, and h is the maximal arity among the arities of the relations in the signature.
3. The algorithm does not drop queries that it has generated.

Notice that the number of queries of any size that can be constructed using a fixed number of different atoms is finite. Therefore, even if the size of queries generated by the algorithm may grow, point 2 above implies that the number of distinct conjunctive queries generated by the algorithm is finite, whereas point 3 implies that the algorithm does not generate a query more than once, and therefore PerfectRef terminates. □

We now prove correctness of the algorithm PerfectRef. To this aim we need to first introduce the notion of canonical interpretation. The canonical interpretation of a *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB is an interpretation constructed according to the notion of *chase* [21]. In particular, we adapt here the notion of *restricted chase* adopted by Johnson and Klug in [27]. In the following, we assume to have an infinite set Γ_N of constant symbols not occurring in $\mathcal{A}$. We also denote with Γ_A the set of constants occurring in $\mathcal{A}$. Then, our notion of chase is as follows.

Definition 3.5. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB. Let $chase_0(\mathcal{K}) = \mathcal{A}$. For every non-negative integer i, let $chase_{i+1}(\mathcal{K})$ be the set of membership assertions obtained from $chase_i(\mathcal{K})$ by applying the following rule:

> Chase Rule. Suppose that there is a PI I in $\mathcal{T}$ of the form (10), and that there is a set of membership assertions $\mathcal{F} = \{S_1(\vec{a}_1), \ldots, S_h(\vec{a}_h)\} \subseteq chase_i(\mathcal{K})$ such that $\vec{a}_1[i_{1,1}, \ldots, i_{1,k}] = \cdots = \vec{a}_h[i_{h,1}, \ldots, i_{h,k}] = \vec{b}$. If there is no membership assertion $S(\vec{a}) \in chase_i(\mathcal{K})$ such that $\vec{a}[i_1, \ldots, i_k] = \vec{b}$, then $chase_{i+1}(\mathcal{K}) = chase_i(\mathcal{K}) \cup \{S(\vec{a}_f)\}$, where $\vec{a}_f$ is an m-tuple such that $\vec{a}_f[i_1, \ldots, i_k] = \vec{b}$, and for each $p \in \{1, \ldots, m\} \setminus \{i_1, \ldots, i_k\}$, $\vec{a}_f[p]$ is a fresh constant from Γ_N not occurring in $chase_i(\mathcal{K})$.

Then, we call *chase of* $\mathcal{K}$, denoted $chase(\mathcal{K})$, the set of membership assertions obtained as the infinite union of all $chase_i(\mathcal{K})$, i.e.,

$$chase(\mathcal{K}) = \bigcup_{i \in \mathbb{N}} chase_i(\mathcal{K}). \qquad \square$$

In the chase rule above, we also say that the PI I is applied in $chase_i(\mathcal{K})$ to the set $\mathcal{F}$ of membership assertions.

We point out that in $chase_i(\mathcal{K})$ there might be several sets of membership assertions to which a PI is applicable, and that several PIs might be applicable to a set of membership assertions. Therefore, there might be several ways of generating $chase_{i+1}(\mathcal{K})$ from $chase_i(\mathcal{K})$ via the chase rule above, and thus a number of syntactically distinct chases might result from this process. It is however possible to establish a suitable order on the application of the chase rule, in such a way that the construction process results in a unique chase. Notice that such an order must guarantee that each PI that becomes applicable at a certain step of the construction of the chase is eventually applied in the construction of the chase. In this paper, we do not discuss further this aspect, and implicitly consider a fixed ordering on the execution of the chase rules that guarantees the above properties. For more details on this aspect we refer the reader to [27,18].

With the notion of chase in place, we can introduce the notion of canonical interpretation. We define the *canonical interpretation* $can(\mathcal{K})$ as the interpretation $\langle \Delta^{can(\mathcal{K})}, \cdot^{can(\mathcal{K})} \rangle$, where:

- $\Delta^{can(\mathcal{K})} = \Gamma_A \cup \Gamma_N$,
- $a^{can(\mathcal{K})} = a$, for each constant a occurring in $chase(\mathcal{K})$,
- $S^{can(\mathcal{K})} = \{(a_1, \ldots, a_m) \mid S(a_1, \ldots, a_m) \in chase(\mathcal{K})\}$, where S as usual is an atomic concept (in this case $m = 1$) or an atomic relation of arity m.

We also define $can_i(\mathcal{K}) = \langle \Delta^{can(\mathcal{K})}, \cdot^{can_i(\mathcal{K})} \rangle$, where $\cdot^{can_i(\mathcal{K})}$ is analogous to $\cdot^{can(\mathcal{K})}$ but refers to $chase_i(\mathcal{K})$ instead of $chase(\mathcal{K})$. From the fact that $chase(\mathcal{K})$ (and $chase_i(\mathcal{K})$) is unique, it follows that also $can(\mathcal{K})$ (resp., $can_i(\mathcal{K})$) is unique. Notice also that $can_0(\mathcal{K})$ is tightly related to the interpretation $db(\mathcal{A})$. Indeed, while $\Delta^{db(\mathcal{A})} \subseteq \Delta^{can(\mathcal{K})}$, we have that $\cdot^{db(\mathcal{A})} = \cdot^{can_0(\mathcal{K})}$.

In line with similar results on the chase of TGDs, e.g., in database theory [21] and data exchange [28], the following lemma shows that there is a homomorphism from $can(\mathcal{K})$ to every model of $\mathcal{K}$ that preserves the assignment of objects to concepts and relations.

Lemma 3.6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DLR-Lite$_{\mathcal{A},\sqcap}$ KB, and let $\mathcal{M} = \langle \Delta^{\mathcal{M}}, \cdot^{\mathcal{M}} \rangle$ be a model of $\mathcal{K}$. Then, there is a function ψ from $\Delta^{can(\mathcal{K})}$ to $\Delta^{\mathcal{M}}$ such that for each predicate symbol S of arity m in $\mathcal{K}$ and each m-tuple of objects $o_1, \ldots, o_m \in \Delta^{can(\mathcal{K})}$, if $(o_1, \ldots, o_m) \in S^{can(\mathcal{K})}$ then $(\psi(o_1), \ldots, \psi(o_m)) \in S^{\mathcal{M}}$.*

Proof. The proof is by induction on the construction of $chase(\mathcal{K})$, and it is similar to the proof of Lemma 28 in [18], which states an analogous result for *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$ KBs. The main difference is here on the inductive step, in which according to Definition 3.5, applicable PIs may present conjunctions of atoms in their left-hand side for the case of concept inclusions and may involve n-ary relations. $\square$

The lemma below shows that whenever $\mathcal{K}$ is satisfiable, $can(\mathcal{K})$ is a model of $\mathcal{K}$ (and vice-versa).

Lemma 3.7. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DLR-Lite$_{\mathcal{A},\sqcap}$ KB. Then, $can(\mathcal{K})$ is a model of $\mathcal{K}$ if and only if $\mathcal{K}$ is satisfiable.*

Proof. "$\Rightarrow$" If $can(\mathcal{K})$ is a model of $\mathcal{K}$, then $\mathcal{K}$ is obviously satisfiable.

"$\Leftarrow$" We separately show that $can(\mathcal{K})$ is (i) a model of $\mathcal{A}$, (ii) a model of all PIs in $\mathcal{T}$, (iii) a model of all NIs in $\mathcal{T}$, and (iv) a model of all key assertions in $\mathcal{T}$.

Point (i) easily follows by the construction of $chase(\mathcal{K})$ (and in particular, by the fact that $\mathcal{A} \subseteq chase(\mathcal{K})$).

Point (ii) is proved by contradiction. Suppose that a PI (of the form (10)) is not satisfied in $can(\mathcal{K})$. This means that there is a set of membership assertions $\mathcal{F} = \{S_1(\vec{a}_1), \ldots, S_h(\vec{a}_h)\} \subseteq chase(\mathcal{K})$ such that $\vec{a}_1[i_{1,1}, \ldots, i_{1,k}] = \cdots = \vec{a}_h[i_{h,1}, \ldots, i_{h,k}] = \vec{b}$, and there is no membership assertion $S(\vec{a}) \in chase(\mathcal{K})$ such that $\vec{a}[i_1, \ldots, i_k] = \vec{b}$. However, this would imply that we can apply the chase rule and insert a new membership assertion $S(\vec{a}_f)$ in $chase(\mathcal{K})$, where $\vec{a}_f$ is an m-tuple such that $\vec{a}_f[i_1, \ldots, i_k] = \vec{b}$, and for each $p \in \{1, \ldots, m\}$ such that $p \notin \{i_1, \ldots, i_k\}$, $\vec{a}_f[p]$ is a fresh constant from Γ_N not occurring in $chase_i(\mathcal{K})$. Obviously, this makes the PI satisfied, thus leading to a contradiction.

Point (iii) is proved by contradiction. Suppose that a NI (of the form (11)) is not satisfied in $can(\mathcal{K})$. This means that there is a set of membership assertions $\mathcal{F} = \{S_0(\vec{a}_0), S_1(\vec{a}_1), \ldots, S_h(\vec{a}_h)\} \subseteq chase(\mathcal{K})$ such that $\vec{a}_0[i_{0,1}, \ldots, i_{0,k}] = \cdots = \vec{a}_h[i_{h,1}, \ldots, i_{h,k}]$. In other words, for each $j \in \{0, \ldots, h\}$, we have that $\vec{a}_j = (a_{j,1}, \ldots, a_{j,m_j}) \in S_j^{can(\mathcal{K})}$. By Lemma 3.6, it follows that, for each $j \in \{0, \ldots, h\}$ and for each model $\mathcal{I}$ of $\mathcal{K}$, $(\psi(a_{j,1})^{\mathcal{I}}, \ldots, \psi(a_{j,m_j})^{\mathcal{I}}) \in S_j^{\mathcal{I}}$. It is easy to see that the NI is not satisfied in $\mathcal{I}$, thus implying that no models of $\mathcal{K}$ exists and therefore contradicting the assumption, which states that $\mathcal{K}$ is satisfiable.

Point (iv) is proved by induction on the construction of the chase.

Base step: If $\mathcal{K}$ is satisfiable, then $can_0(\mathcal{K})$ satisfies all key assertions in $\mathcal{K}$. Indeed, if we assume by contradiction that $can_0(\mathcal{K})$ violates a key assertion in $\mathcal{K}$, i.e., an assertion of the form (key $j_1, \ldots, j_\ell$: V), where V is either an atomic relation R or a projection $R[i_1, \ldots, i_h]$ over R, we get that there are two membership assertions $R(\vec{a}_1)$ and $R(\vec{a}_2)$ in $\mathcal{A}$ such that $\vec{a}_1[j_1, \ldots, j_\ell] = \vec{a}_2[j_1, \ldots, j_\ell]$. Since every model has to satisfy both all ABox assertions and all key assertions in $\mathcal{K}$, this implies that no model of $\mathcal{K}$ exists (remember that we adopt the unique name assumption for the interpretation of the constants of the KB), thus contradicting the assumption that $\mathcal{K}$ is satisfiable.

Inductive step: By exploiting the inductive assumption that $can_i(\mathcal{K})$ satisfies all key assertions in $\mathcal{K}$, we show that $can_{i+1}(\mathcal{K})$ satisfies all key assertions in $\mathcal{K}$, where $can_{i+1}(\mathcal{K})$ corresponds to $chase_{i+1}(\mathcal{K})$, i.e., the chase that is obtained from $chase_i(\mathcal{K})$ by application of the chase rule (cf. Definition 3.5). This means that there exists a set of membership assertions $\mathcal{F} = \{S_1(\vec{a}_1), \ldots, S_h(\vec{a}_h)\} \subseteq chase_i(\mathcal{K})$ such that $\vec{a}_1[i_{1,1}, \ldots, i_{1,k}] = \cdots = \vec{a}_h[i_{h,1}, \ldots, i_{h,k}] = \vec{b}$, and there does not exist a membership assertion $S(\vec{a}) \in chase_i(\mathcal{K})$ such that $\vec{a}[i_1, \ldots, i_k] = \vec{b}$. Then $chase_{i+1}(\mathcal{K}) = chase_i(\mathcal{K}) \cup \{S(\vec{a}_f)\}$, where $\vec{a}_f$ is an m-tuple such that $\vec{a}_f[i_1, \ldots, i_k] = \vec{b}$, and for each $p \in \{1, \ldots, m\} \setminus \{i_1, \ldots, i_k\}$, $\vec{a}_f[p]$ is a fresh constant from Γ_N not occurring in $chase_i(\mathcal{K})$. We show below that $S(\vec{a}_f)$ cannot cause the violation of a key assertion in $can_{i+1}(\mathcal{K})$, considering all possible cases:

1. If S is an atomic concept, then no key is defined on S, and therefore the claim trivially follows;
2. If S is a relation and $k = 1$, then key assertions may be specified on S, since the PI applied in the chase rule is a concept inclusion. In the case where there are no key assertions on S, the claim trivially follows. Let us consider instead the case in which a key is specified on S. Since $k = 1$, the membership assertion $S(\vec{a}_f)$ added to $chase_{i+1}(\mathcal{K})$ is such that $\vec{a}_f$ contains only one non-fresh symbol. Let r be the position of such a symbol, i.e., $\vec{a}_f[r] \notin \Gamma_N$. This means that only a key assertion of the form (key r: V), where $V = S$ or $V = S[i_1, \ldots, i_h]$, can be violated by $can_{i+1}(\mathcal{K})$. However, a violation of this kind would imply that $chase_{i+1}(\mathcal{K})$ contains a membership assertion $S(\vec{c})$ such that $\vec{c}[r] = \vec{a}_f[r]$. Since $S(\vec{c})$ belongs also to $chase_i(\mathcal{K})$, then the chase rule would not have been applied and $S(\vec{a}_f)$ would not have been added to $chase_i(\mathcal{K})$ to obtain $chase_{i+1}(\mathcal{K})$. Hence, the claim follows also in this case.
3. If S is a relation and $k > 1$, then the PI applied in the chase rule is a relation inclusion. According to the definition of *DLR-Lite*$_{\mathcal{A},\sqcap}$ (cf. Table 4), $\mathcal{T}$ cannot contain key assertions involving S, and therefore the claim trivially follows. □

Exploiting Lemma 3.7 and Lemma 3.6, it is possible to prove the following theorem, which is in turn crucial to establish correctness of the algorithm PerfectRef.

Theorem 3.8. *Let $\mathcal{K}$ be a satisfiable DLR-Lite$_{\mathcal{A},\sqcap}$ KB, and let Q be a union of conjunctive queries over $\mathcal{K}$. Then, $ans(Q, \mathcal{K}) = Q^{can(\mathcal{K})}$.*

Proof. The proof is analogous to the proof of Theorem 29 in [18], which states an analogous result for *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$ KBs. □

We are now able to prove that for every *DLR-Lite*$_{\mathcal{A},\sqcap}$ TBox $\mathcal{T}$ and UCQ Q over $\mathcal{T}$, the algorithm PerfectRef is well suited for computing the FOL-rewriting of Q w.r.t. $\mathcal{T}$.

Theorem 3.9. *Let $\mathcal{T}$ be a DLR-Lite$_{\mathcal{A},\sqcap}$ TBox, Q a union of conjunctive queries over $\mathcal{T}$, and Q_r the union of conjunctive queries returned by* PerfectRef$(Q, \mathcal{T})$. *Then, for every DLR-Lite$_{\mathcal{A},\sqcap}$ ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, we have that $ans(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = Q_r^{db(\mathcal{A})}$.*

Proof. We first introduce the preliminary notion of witness of a tuple of constants with respect to a conjunctive query. Given a *DLR-Lite*$_{\mathcal{A},\sqcap}$ knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, a conjunctive query $q(\vec{x}) \leftarrow conj(\vec{x}, \vec{y})$ over $\mathcal{K}$, and a tuple $\vec{t}$ of constants occurring in $\mathcal{K}$, a set G of membership assertions is a *witness of $\vec{t}$ w.r.t. q* if there exists a substitution σ from the variables $\vec{y}$ in $conj(\vec{t}, \vec{y})$ to constants in G such that the set of atoms in $\sigma(conj(\vec{t}, \vec{y}))$ is equal to G. In particular, we are interested

in witnesses of a tuple $\vec{t}$ w.r.t. a query q that are contained in $chase(\mathcal{K})$. Intuitively, each such witness corresponds to a subset of $chase(\mathcal{K})$ that is sufficient in order to have that the formula $\exists \vec{y}.\, conj(\vec{t}, \vec{y})$ evaluates to true in the canonical interpretation $can(\mathcal{K})$, and therefore the tuple $\vec{t} = \vec{t}^{can(\mathcal{K})}$ belongs to $q^{can(\mathcal{K})}$. More precisely, we have that $\vec{t} \in q^{can(\mathcal{K})}$ iff there exists a witness G of $\vec{t}$ w.r.t. q such that $G \subseteq chase(\mathcal{K})$. The cardinality of a witness G, denoted by $|G|$, is the number of membership assertions in G.

Since $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, by Theorem 3.8, $ans(Q, \mathcal{K}) = Q^{can(\mathcal{K})}$. Furthermore, ${Q_r}^{db(\mathcal{A})} = \bigcup_{\hat{q} \in Q_r} \hat{q}^{db(\mathcal{A})}$, where Q_r is the union of conjunctive queries returned by PerfectRef$(Q, \mathcal{T})$. Consequently, to prove the claim it is sufficient to show that $Q^{can(\mathcal{K})} = \bigcup_{\hat{q} \in Q_r} \hat{q}^{db(\mathcal{A})}$.

"$\Leftarrow$" To prove that $\bigcup_{\hat{q} \in Q_r} \hat{q}^{db(\mathcal{A})} \subseteq Q^{can(\mathcal{K})}$, we have to prove that $\hat{q}^{db(\mathcal{A})} \subseteq Q^{can(\mathcal{K})}$, for each $\hat{q} \in Q_r$. In fact, since $\hat{q}^{db(\mathcal{A})} \subseteq \hat{q}^{can(\mathcal{K})}$, we will show that $\hat{q}^{can(\mathcal{K})} \subseteq Q^{can(\mathcal{K})}$. We proceed by induction on the construction of Q_r, which is generated by iteratively applying, as long as they are applicable, Step (a) and Step (b) of the algorithm PerfectRef (starting from the CQs constituting the query Q).

Base step: It is trivial to see that for each $\hat{q} \in Q$, it holds that $\hat{q}^{can(\mathcal{K})} \subseteq Q^{can(\mathcal{K})}$.

Inductive step: Given $q_i \in Q_r$, by inductive hypothesis we assume that $q_i^{can(\mathcal{K})} \subseteq Q^{can(\mathcal{K})}$, and show that $q_{i+1}^{can(\mathcal{K})} \subseteq Q^{can(\mathcal{K})}$ by distinguishing between (i) the case in which q_{i+1} is obtained from q_i by means of Step (a) of the algorithm PerfectRef, and (ii) the case in which q_{i+1} is obtained from q_i by means of Step (b) of the algorithm. In both cases, given a tuple $\vec{t}$ of constants occurring in $\mathcal{K}$ such that $\vec{t} \in q_{i+1}^{can(\mathcal{K})}$, we have that there exists a witness G of $\vec{t}$ w.r.t. q_{i+1} such that $G \subseteq chase(\mathcal{K})$. As for Case (i), we have that $q_{i+1} = \tau(reduce(q_i, g_1, g_2))$, where g_1, g_2 are two atoms belonging to q_i such that g_1 and g_2 unify. It is easy to see that in such a case G is also a witness of $\vec{t}$ w.r.t. q_i, and therefore $\vec{t} \in q_i^{can(\mathcal{K})}$. As for Case (ii), it is easy to see that there exists a set of membership assertions in G to which a PI is applicable (cf. Definition 3.5), which implies that there exists a witness of $\vec{t}$ w.r.t. q_i contained in $chase(\mathcal{K})$. Therefore, $\vec{t} \in q_i^{can(\mathcal{K})}$.

"$\Rightarrow$" We now prove that $Q^{can(\mathcal{K})} \subseteq \bigcup_{\hat{q} \in Q_r} \hat{q}^{db(\mathcal{A})}$, i.e., that for each tuple $\vec{t} \in Q^{can(\mathcal{K})}$ there exists $\hat{q} \in Q_r$ such that $\vec{t} \in \hat{q}^{db(\mathcal{A})}$. First, since $\vec{t} \in Q^{can(\mathcal{K})}$, it follows that there exists a CQ $q_0 \in Q$ and a finite number k such that there is a witness G_k of $\vec{t}$ w.r.t. q_0 contained in $chase_k(\mathcal{K})$. Moreover, without loss of generality, we can assume that every chase rule used in the construction of $chase_k(\mathcal{K})$ is necessary in order to generate such a witness G_k, i.e., $chase_k(\mathcal{K})$ can be seen as a (not necessarily connected) directed acyclic graph where: (i) nodes represent all membership assertions in $chase_k(\mathcal{K})$, and (ii) there is an edge from a node f_1 to a node f_2 if f_1 belongs to the set of membership assertions to which a PI is applied to produce f_2 (via the chase rule). Notice also that, in such a graph, source nodes (i.e., nodes with only outgoing edges) correspond to membership assertions in $\mathcal{A}$, whereas target nodes (i.e., nodes with only ingoing edges) correspond to membership assertions in G_k. In the following, we say that a membership assertion f is an *ancestor* of a membership assertion f' in a set $\mathcal{S}$ of membership assertions, if there exist n sets of membership assertions $\mathcal{F}_1, \ldots, \mathcal{F}_n$ such that $\mathcal{F}_i \subseteq \mathcal{S}$ for $i \in \{1, \ldots, n\}$, $f \in \mathcal{F}_1$, $\mathcal{F}_n = \{f'\}$, and for each $i \in \{2, \ldots, n\}$, one element belonging to $\mathcal{F}_i$ can be generated by applying a chase rule to $\mathcal{F}_{i-1}$, and every element in $\mathcal{F}_{i-1}$ is necessary for such a chase rule to be applicable. We also say that f' is a *successor* of f. Furthermore, for each $i \in \{0, \ldots, k\}$, we denote with G_{k-i} the *pre-witness* (of depth i) of $\vec{t}$ w.r.t. q_0 in $chase_k(\mathcal{K})$, defined as follows:

$$G_{k-i} = \big\{ f \in chase_{k-i}(\mathcal{K}) \;\big|\; \text{there exists } f' \in G_k \text{ s.t. } f \text{ is an ancestor of } f' \text{ in } chase_k(\mathcal{K}) \text{ and there exists no successor of } f \text{ in } chase_{k-i}(\mathcal{K}) \text{ that is an ancestor of } f' \text{ in } chase_k(\mathcal{K}) \big\}.$$

Now we prove by induction on i that, starting from G_k (i.e., $i = 0$), we can "go back" through the chase rule applications and find a query $\hat{q}$ in Q_r such that the pre-witness G_{k-i} of $\vec{t}$ w.r.t. q_0 in $chase_k(\mathcal{K})$ is also a witness of $\vec{t}$ w.r.t. $\hat{q}$ (such a witness is obviously in $chase_{k-i}(\mathcal{K})$). To this aim, we prove that there exists $\hat{q} \in Q_r$ such that G_{k-i} is a witness of $\vec{t}$ w.r.t. $\hat{q}$ and $size(\hat{q}) = |G_{k-i}|$. The claim then follows for $i = k$, since $chase_0(\mathcal{K}) = \mathcal{A}$, and therefore $G_0 \subseteq \mathcal{A}$.

Base step: There exists $\hat{q} \in Q_r$ such that G_k is a witness of $\vec{t}$ w.r.t. $\hat{q}$ and $size(\hat{q}) = |G_k|$. This is an immediate consequence of the fact that: (i) $q_0 \in Q_r$, and (ii) Q_r is closed with respect to Step (a) of the algorithm PerfectRef. Indeed, if $|G_k| < size(q_0)$ then there exist two atoms g_1, g_2 in Q and a membership assertion f in G_k such that f and g_1 unify and f and g_2 unify, which implies that g_1 and g_2 unify. Therefore, by Step (a) of the algorithm, it follows that there exists a query $q_1 \in Q_r$ (with $q_1 = reduce(q_0, g_1, g_2)$) such that G_k is a witness of $\vec{t}$ w.r.t. q_1 and $size(q_1) = size(q_0) - 1$. Now, if $|G_k| < size(q_1)$, we can iterate the above argument, thus we conclude that there exists $\hat{q} \in Q_r$ such that G_k is a witness of $\vec{t}$ w.r.t. $\hat{q}$ and $size(\hat{q}) = |G_k|$.

Inductive step: suppose that there exists $\hat{q} \in Q_r$ such that G_{k-i+1} is a witness of $\vec{t}$ w.r.t. $\hat{q}$ and $size(\hat{q}) = |G_{k-i+1}|$. Let I be the PI of form (10) applied to a set of membership assertions in $chase_{k-i}(\mathcal{K})$ to obtain $chase_{k-i+1}(\mathcal{K})$, i.e., $chase_{k-i+1}(\mathcal{K}) = chase_{k-i}(\mathcal{K}) \cup \{S(\vec{a_f})\}$, where $S(\vec{a_f})$ is as in Definition 3.5. Since non-propagated positions in S, i.e., positions that are outside $i_1, \ldots, i_k$, contain new constants, i.e., constants not occurring elsewhere in G_{k-i+1}, and since $size(\hat{q}) = |G_{k-i+1}|$, it follows that the variables in $\hat{q}$ that match with such constants are unbound (notice that this might not hold without the assumption $size(\hat{q}) = |G_{k-i+1}|$). Therefore, by Step (b) of the algorithm, it follows that there exists a query $q_1 \in Q_r$ such that q_1 is obtained via *atomRewrite* by substituting the atom $S(\vec{x})$ in $\hat{q}$ with its rewriting according to the applied PI, where $S(\vec{x})$ is the atom in q_0 to which I is applicable, and G_{k-i}, i.e., the pre-witness of depth i of $\vec{t}$ w.r.t. $\hat{q}$, is a witness of $\vec{t}$ w.r.t. q_1. Notice that $G_{k-i} = G_{k-i+1} \setminus \{S(\vec{a}_f)\} \cup \{S_1(\vec{a}_1), \ldots, S_h(\vec{a}_h)\}$, where each $S_i(\vec{a}_i)$ is as in Definition 3.5.

Now, there are two possible cases: either $size(q_1) = |G_{k-i}|$, and in this case the claim is immediate; or $size(q_1) > |G_{k-i}|$. This last case arises if and only if some of the membership assertions $S_1(\vec{a}_1), \ldots, S_h(\vec{a}_h)$ occur both in G_{k-i} and in G_{k-i+1}. Without loss of generality we assume that such assertions are $S_1(\vec{a}_1), \ldots, S_j(\vec{a}_j)$ (notice that $size(q_1) = |G_{k-i}| + j$), and that this implies that there exist j pairs of atoms, denoted $g_{p,1}$, $g_{p,2}$, with $p \in \{1, \ldots, j\}$, in q_1 such that $S_p(\vec{a}_p)$ and $g_{p,1}$ unify and $S_p(\vec{a}_p)$ and $g_{p,2}$ unify, hence $g_{p,1}$ and $g_{p,2}$ unify. Therefore, by Step (a) of the algorithm iteratively applied to q_1 for j times, it follows that there exists $q_2 \in Q_r$ such that G_{k-i} is a witness of $\vec{t}$ w.r.t. q_2 and $size(q_2) = |G_{k-i+1}|$, which proves the claim. □

The following corollary is an immediate consequence of the above theorem.

Corollary 3.10. *Let $\mathcal{K} = \langle \mathcal{T}_p \cup \mathcal{T}_n \cup \mathcal{T}_k, \mathcal{A} \rangle$ be a satisfiable DLR-Lite$_{\mathcal{A},\sqcap}$ KB, where $\mathcal{T}_p$, $\mathcal{T}_n$, and $\mathcal{T}_k$ respectively denote PIs, NIs, and key assertions in the TBox of $\mathcal{K}$, and let Q be a union of conjunctive queries over $\mathcal{K}$. Then $ans(Q, \mathcal{K}) = ans(Q, \langle \mathcal{T}_p, \mathcal{A} \rangle)$.*

Since the evaluation of a FOL query is in AC^0 in data complexity, the following result is an obvious consequence of Theorem 3.9.

Theorem 3.11. *Query answering in DLR-Lite$_{\mathcal{A},\sqcap}$ is in AC^0 with respect to data complexity.*

3.2. FOL-rewritability of KB satisfiability

We now consider KB satisfiability in *DLR-Lite*$_{\mathcal{A},\sqcap}$, and provide a mechanism to solve it via FOL query evaluation, thus showing its FOL-rewritability. We start by considering the special case in which no NIs and no key assertions are specified over a *DLR-Lite*$_{\mathcal{A},\sqcap}$ TBox, and get the following notable result.

Lemma 3.12. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DLR-Lite$_{\mathcal{A},\sqcap}$ KB such that $\mathcal{T}$ contains only PIs. Then, $\mathcal{K}$ is satisfiable.*

Proof. In the proof of Lemma 3.7 (Point (ii)) we have shown that $can(\mathcal{K})$ satisfies all PIs asserted in a *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB. The same proof can be used to show that for each $\mathcal{K}$ whose TBox consists of PIs only, $can(\mathcal{K})$ is a model of $\mathcal{K}$, and therefore $\mathcal{K}$ is satisfiable. □

Let us now consider generic *DLR-Lite*$_{\mathcal{A},\sqcap}$ KBs, i.e., KBs with PIs, NIs, and key assertions. In order to show that KB satisfiability for KBs expressed in such a language is FOL-rewritable, we first introduce the preliminary notions of (boolean) query associated to a key assertion and of (boolean) query associated to a NI. For ease of exposition, from now on we assume that a key assertion over a relation R of arity m is always written in the form (key $j_1, \ldots, j_\ell$: $R[1, \ldots, m]$), i.e., we exploit the fact that $R[1, \ldots, m]$ is equivalent to R to consider only key assertions over projections of atomic relations. In the following, we make use of CQs and UCQs enriched with inequalities, and express them in Datalog notation, analogously to what we have done for CQs and UCQs without inequalities.

Definition 3.13. Let $F = (\text{key}\, j_1, \ldots, j_\ell\colon\ R[i_1, \ldots, i_h])$ be a *DLR-Lite*$_{\mathcal{A},\sqcap}$ key assertion, where R is an atomic relation or arity m, $i_1, \ldots, i_h \in \{1, \ldots, m\}$, and $j_1, \ldots, j_\ell \in \{1, \ldots, h\}$ (cf. Section 2). Then, the *query associated to* F is the boolean union of conjunctive queries with inequalities $q_F = \bigcup_{k \in \{i_1, \ldots, i_h\} \setminus \{i_{j_1}, \ldots, i_{j_\ell}\}} \{q_k\}$, where each q_k is as follows

$$q_k \leftarrow R(x_1, \ldots, x_m), R(y_1, \ldots, y_m), x_k \neq y_k,$$

where $x_{i_{j_1}} = y_{i_{j_1}}, \ldots, x_{i_{j_\ell}} = y_{i_{j_\ell}}$. □

For example, given a relation R of arity 4 and the key assertion $F = (\text{key}\, 2, 3\colon\ R[2, 3, 4])$, the query associated to F is $q_F = \{q \leftarrow R(x_1, x_2, x_3, x_4), R_2(y_1, y_2, x_3, x_4), x_2 \neq y_2\}$. It is easy to see that a query associated to a key assertion F is a boolean query whose evaluation over an interpretation $\mathcal{I}$ is true if and only if $\mathcal{I}$ is not a model of F.

Definition 3.14. Given a *DLR-Lite*$_{\mathcal{A},\sqcap}$ negative inclusion N in the form (11), the *query associated to* N is a boolean conjunctive query q_N of the form

$$q_N \leftarrow S_0(z_{0,1}, \ldots, z_{0,m_0}), \ldots, S_h(z_{h,1}, \ldots, z_{h,m_h}),$$

where for $j \in \{0, \ldots, h\}$, and for $\ell \in \{1, \ldots, m_j\}$, we have that $z_{j,\ell} = y_r$ if there exists $r \in \{1, \ldots, k\}$ such that $i_{j,r} = \ell$, otherwise $z_{j,\ell}$ is a variable not occurring elsewhere in q_N. □

For example, given the negative exclusion $N = R_1[1] \sqcap R_2[2] \sqsubseteq \neg R_3[1]$, where R_1 is a binary relation and both R_2 and R_3 are ternary relations, the query associated to N is $q_N \leftarrow R_1(y_1, y_2), R_2(y_3, y_1, y_4), R_3(y_1, y_5, y_6)$. It is easy to see that a query associated to a negative inclusion N is a boolean query whose evaluation over an interpretation $\mathcal{I}$ is true if and only if $\mathcal{I}$ is not a model of N.

Example 3.15. The queries associated to the NIs (4) and (5) in Example 3.1 are $q_{N,4} \leftarrow \mathit{Supplier}(x), \mathit{Product}(x)$ and $q_{N,5} \leftarrow \mathit{Customer}(x), \mathit{Product}(x)$, whereas the query associated to the key assertion (6) is $q_{F,6} \leftarrow \mathit{supply}(x_1, x_2, x_3), \mathit{supply}(y_1, x_2, x_3), x_1 \neq y_1$.

The following lemma states that satisfiability for a *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ contains only PIs and key assertions, can be reduced to answering a union of conjunctive queries with inequalities over the ABox $\mathcal{A}$.

Lemma 3.16. *Let $\mathcal{K} = \langle \mathcal{T}_p \cup \mathcal{T}_k, \mathcal{A} \rangle$ be a DLR-Lite$_{\mathcal{A},\sqcap}$ KB where $\mathcal{T}_p$ is the set of PIs in $\mathcal{K}$ and $\mathcal{T}_k$ the set of key assertions in $\mathcal{K}$ (i.e., $\mathcal{K}$ does not contain NIs), and let $Q_k = \bigcup_{F \in \mathcal{T}_k} \{q_F\}$. Then, $\mathcal{K}$ is satisfiable if and only if $\mathcal{A} \not\models Q_k$.*

Proof. "⇐" From $\mathcal{A} \not\models Q_k$ it easily follows that $\mathcal{A} \models \mathcal{T}_k$. Then, analogously to the proof of Lemma 3.7 (Part (iv)), it can be shown that $\mathit{chase}(\langle \mathcal{T}_p, \mathcal{A} \rangle) \models \mathcal{T}_k$ and therefore $\mathcal{K} = \langle \mathcal{T}_p \cup \mathcal{T}_k, \mathcal{A} \rangle$ is satisfiable, since $\mathit{chase}(\langle \mathcal{T}_p, \mathcal{A} \rangle)$ is a model of $\mathcal{K}$.

"⇒" From $\mathcal{A} \models Q_k$ it follows that there exists a key assertion $F \in \mathcal{T}_k$ such that $\mathcal{A} \models q_F$, and therefore $\mathcal{A} \cup \{F\}$ is unsatisfiable. It is then easy to see that $\langle \mathcal{T}_p \cup \mathcal{T}_k, \mathcal{A} \rangle$ is unsatisfiable (indeed, every interpretation satisfying the ABox necessarily violates the key assertion F). □

Notably, since answering Q_k over the ABox $\mathcal{A}$ simply amounts to evaluating Q_k over $db(\mathcal{A})$, the above lemma actually says that satisfiability of a *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB without NIs is FOL-rewritable. We notice that in each such knowledge base $\mathcal{K}$ every key assertion can be processed independently, and that there is no interaction between key assertions and PIs that has to be taken into account to check if $\mathcal{K}$ is satisfiable. Actually, this holds by virtue of the controlled combination of key assertions and inclusions between relations established for *DLR-Lite*$_{\mathcal{A},\sqcap}$ (cf. Section 2).

Let us now consider the impact of NIs on the satisfiability check. The following lemma states that satisfiability of a satisfiable *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB $\mathcal{K}$ extended with a set of NIs can be reduced to query answering over $\mathcal{K}$.

Lemma 3.17. *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DLR-Lite$_{\mathcal{A},\sqcap}$ KB, let $\mathcal{T}_n$ be a set of DLR-Lite$_{\mathcal{A},\sqcap}$ NIs, and let $Q_n = \bigcup_{N \in \mathcal{T}_n} \{q_N\}$. Then, $\langle \mathcal{T} \cup \mathcal{T}_n, \mathcal{A} \rangle$ is satisfiable if and only if $\langle \mathcal{T}, \mathcal{A} \rangle \not\models Q_n$.*

Proof. "⇐" We show that if $\mathcal{K} = \langle \mathcal{T} \cup \mathcal{T}_n, \mathcal{A} \rangle$ is unsatisfiable then $\langle \mathcal{T}, \mathcal{A} \rangle \models Q_n$. Consider the FOL formula ϕ obtained as the conjunction of all the assertions in $\mathcal{K}$, each specified in FOL, i.e.,

$$\phi = \bigwedge_{\alpha \in \mathcal{T}} \alpha \wedge \bigwedge_{\beta \in \mathcal{T}_n} \beta \wedge \bigwedge_{\gamma \in \mathcal{A}} \gamma.$$

Obviously, if $\mathcal{K}$ is unsatisfiable then ϕ is unsatisfiable, and, by the deduction theorem, it follows that

$$\bigwedge_{\alpha \in \mathcal{T}} \alpha \wedge \bigwedge_{\gamma \in \mathcal{A}} \gamma \models \bigvee_{\beta \in \mathcal{T}_n} \neg\beta.$$

It is easy to see that, due to Theorem 3.8, this holds if and only if there exists a NI $N \in \mathcal{T}_n$ such that $\langle \mathcal{T}, \mathcal{A} \rangle \models q_N$ and therefore $\langle \mathcal{T}, \mathcal{A} \rangle \models Q_n$.

"⇒" We show that if $\langle \mathcal{T}, \mathcal{A} \rangle \models Q_n$ then $\langle \mathcal{T} \cup \mathcal{T}_n, \mathcal{A} \rangle$ is unsatisfiable. Again, due to Theorem 3.8, if $\langle \mathcal{T}, \mathcal{A} \rangle \models Q_n$ then there exists $N \in \mathcal{T}_n$ such that $\langle \mathcal{T}, \mathcal{A} \rangle \models q_N$. This implies that in every model of $\langle \mathcal{T}, \mathcal{A} \rangle$ (which is satisfiable by the assumption) there exist some tuples of objects that contradict the NI N, i.e., we have that $\langle \mathcal{T}, \mathcal{A} \rangle \models \neg N$. By the deduction theorem, it follows that $\langle \mathcal{T} \cup \{N\}, \mathcal{A} \rangle$ is unsatisfiable, and therefore $\langle \mathcal{T} \cup \mathcal{T}_n, \mathcal{A} \rangle$ is unsatisfiable. □

As a consequence of Lemma 3.17 and Lemma 3.12, we have that to establish satisfiability of a *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB $\mathcal{K}$ with no key assertions, it is possible to resort to query answering over satisfiable KBs. Indeed, let $\mathcal{T}_p$ be the set of PIs in $\mathcal{K}$, and $\mathcal{T}_n$ the set of NIs in $\mathcal{K}$. We have by Lemma 3.12 that $\langle \mathcal{T}_p, \mathcal{A} \rangle$ is satisfiable, and therefore by Lemma 3.17 we get that $\mathcal{K}$ is satisfiable if and only if $\langle \mathcal{T}_p, \mathcal{A} \rangle \not\models Q_n$. Notice that this means that each NI can be processed independently, and its interaction with PIs can be considered separately. Notably, by Theorem 3.9, we have that the algorithm PerfectRef can be used to establish whether $\langle \mathcal{T}_p, \mathcal{A} \rangle \not\models Q_n$. This actually implies that KB satisfiability for *DLR-Lite*$_{\mathcal{A},\sqcap}$ KBs with no key assertions is FOL-rewritable.

Now we are ready to show how to reduce satisfiability of generic *DLR-Lite*$_{\mathcal{A},\sqcap}$ KBs to FOL query evaluation, constructing on the results given in Lemma 3.12, Lemma 3.16, and Lemma 3.17. To this aim, we make use of the algorithm Consistent, described in Fig. 2. This algorithm calls the algorithm PerfectRef to compute the FOL-rewriting of the query Q_n, representing the union of all the queries associated to NIs in the KB, and then adds the query Q_k to the result of PerfectRef, where Q_k is the union of all the queries associated to key assertions in the KB. The theorem below uses the algorithm Consistent to state FOL-rewritability of KB satisfiability in *DLR-Lite*$_{\mathcal{A},\sqcap}$ KBs.

Theorem 3.18. *Let $\mathcal{K} = \langle \mathcal{T}_p \cup \mathcal{T}_n \cup \mathcal{T}_k, \mathcal{A} \rangle$ be a DLR-Lite$_{\mathcal{A},\sqcap}$ KB, where $\mathcal{T}_p$, $\mathcal{T}_n$, and $\mathcal{T}_k$ respectively denote the set of PIs, NIs, and key assertions of the TBox, and let Q_c be the UCQ with inequalities returned by the algorithm* Consistent($\mathcal{K}$). *Then, $\mathcal{K}$ is satisfiable if and only if $\mathcal{A} \not\models Q_c$, i.e., $Q_c^{db(\mathcal{A})} = \emptyset$.*

Algorithm Consistent($\mathcal{K}$)
Input: *DLR-Lite*$_{\mathcal{A},\sqcap}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, with set of PIs $\mathcal{T}_p \subseteq \mathcal{T}$
Output: UCQ Q_c
$Q_n = \emptyset$;
for each NI $N \in \mathcal{T}$ **do**
  $Q_n = Q_n \cup \{q_N\}$;
$Q_n = \mathsf{PerfectRef}(Q_n, \mathcal{T}_p)$;
$Q_k = \emptyset$;
for each key assertion $F \in \mathcal{T}$ **do**
  $Q_k = Q_k \cup \{q_F\}$;
return $Q_c = Q_n \cup Q_k$

Fig. 2. The algorithm Consistent.

Proof. "⇐" Let $Q_c = Q_k \cup Q_{NR}$, where Q_F is the union of all the queries associated to key assertions in $\mathcal{T}_f$, and Q_n^r is the output of $\mathsf{PerfectRef}(Q_n, \mathcal{T}_p)$, where Q_n is the union of all the queries associated to NIs in $\mathcal{T}_n$. From $\mathcal{A} \not\models Q_c$ it follows that $\mathcal{A} \not\models Q_k$ and $\mathcal{A} \not\models Q_n^r$. From $\mathcal{A} \not\models Q_k$ it follows that $\langle \mathcal{T}_p \cup \mathcal{T}_k, \mathcal{A} \rangle$ is satisfiable (cf. Lemma 3.16). Also, from $\mathcal{A} \not\models Q_n^r$ it follows that $\langle \mathcal{T}_p, \mathcal{A} \rangle \not\models Q_n$ (according to Lemma 3.12 stating that $\langle \mathcal{T}_p, \mathcal{A} \rangle$ is satisfiable, and to Theorem 3.9), and since $\langle \mathcal{T}_p \cup \mathcal{T}_k, \mathcal{A} \rangle$ is satisfiable, from Corollary 3.10 it follows that $\langle \mathcal{T}_p \cup \mathcal{T}_k, \mathcal{A} \rangle \not\models Q_n$. Then, from Lemma 3.17 it follows that $\mathcal{K} = \langle \mathcal{T}_p \cup \mathcal{T}_n \cup \mathcal{T}_k, \mathcal{A} \rangle$ is satisfiable.

"⇒" Suppose by contradiction that $\mathcal{A} \models Q_c$. This means that $\mathcal{A} \models Q_k$ or that $\mathcal{A} \models Q_n^r$, where Q_k and Q_n^r are as above. In the former case, from Lemma 3.16 it follows that $\langle \mathcal{T}_k, \mathcal{A} \rangle$ is unsatisfiable, thus getting a contradiction. In the latter case, from Theorem 3.9 and Lemma 3.17 it follows that $\langle \mathcal{T}_p \cup \mathcal{T}_n, \mathcal{A} \rangle$ is unsatisfiable, thus getting again a contradiction. □

As an immediate consequence of Theorem 3.18, we obtain the following result.

Theorem 3.19. *KB satisfiability in DLR-Lite*$_{\mathcal{A},\sqcap}$ *is in* AC^0 *with respect to data complexity.*

4. Going beyond FOL-rewritability

In the previous section, we have pointed out the importance of languages for which query answering and KB satisfiability are FOL-rewritable. In this section, we show that, as soon as we consider further, minimal extensions of *DLR-Lite*$_{\mathcal{A},\sqcap}$, we cross the boundary of AC^0 data complexity. Going beyond AC^0 data complexity means actually that we lose the property of FOL-rewritability 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.

We point out that the extensions of *DLR-Lite*$_{\mathcal{A},\sqcap}$ that we consider in the following present the same behavior also if we restrict relations to be binary, i.e., if we have only roles. Moreover, such extensions make query answering harder even if we apply them to the *core* of *DL-Lite*. Therefore, in the following, we consider DLs with roles rather than n-ary relations, and analyze extensions of *DL-Lite*$_{core}$, the core language of the *DL-Lite* family (cf. Section 2). Specifically, we study the computational complexity of instance checking and query answering for extensions of *DL-Lite*$_{core}$ in which the ABox of a KB is as described in Section 2, while the TBox consists of (i) concept inclusion assertions of the form $Cl \sqsubseteq Cr$, where the syntax of Cl and Cr is defined case by case, and (ii) possibly, key (actually, functionality) assertions on roles.

We remark that all lower bounds given below hold under LogSpace-reductions.

4.1. NLogSpace-hard DLs

We consider now extensions of *DL-Lite*$_{core}$ in which the concept inclusion assertions may contain forms of qualified existential quantification or universal quantification on roles, possibly combined with functionality assertions.

Theorem 4.1. *Instance checking* (*and hence query answering*) *is* NLogSpace*-hard with respect to data complexity for the cases where*

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

Proof. For Case 1, the proof is by a LogSpace reduction from reachability in directed graphs, which is NLogSpace-complete. Let $G = (N, E)$ be a directed graph, where N is a set of nodes and $E \subseteq N \times N$ is the set of edges of G, and let s, d be two nodes in N. Reachability is the problem of checking whether there is a path in G from s to d.

We define a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where the TBox $\mathcal{T}$ is constituted by a single inclusion assertion

$$\exists P.A \sqsubseteq A$$

and the ABox $\mathcal{A}$ has as constants the nodes of G, and is constituted by the membership assertion $A(d)$, and by one membership assertion $P(n, n')$ for each edge $(n, n') \in E$. It is easy to see that $\mathcal{K}$ can be constructed in LogSpace from G, s, and d. We show that there is a path in G from s to d if and only if $\mathcal{K} \models A(s)$.

"$\Leftarrow$" Suppose there is no path in G from s to d. We construct a model $\mathcal{I}$ of $\mathcal{K}$ such that $s^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = N$, $n^{\mathcal{I}} = n$ for each $n \in N$, $P^{\mathcal{I}} = E$, and $A^{\mathcal{I}} = \{n \mid \text{there is a path in} G \text{ from } n \text{ to } d\}$. We show that $\mathcal{I}$ is a model of $\mathcal{K}$. By construction, $\mathcal{I}$ satisfies all membership assertions $P(n, n')$ and the membership assertion $A(d)$. Consider an object $n \in (\exists P.A)^{\mathcal{I}}$. Then there is an object $n' \in A^{\mathcal{I}}$ such that $(n, n') \in P^{\mathcal{I}}$. Then, by definition of $\mathcal{I}$, there is a path in G from n' to d, and $(n, n') \in E$. Hence, there is also a path in G from n to d and, by definition of $\mathcal{I}$, we have that $n \in A^{\mathcal{I}}$. It follows that also the inclusion assertion $\exists P.A \sqsubseteq A$ is satisfied in $\mathcal{I}$.

"$\Rightarrow$" Suppose there is a path in G from a node n to d. We prove by induction on the length ℓ of such a path that $\mathcal{K} \models A(n)$. Base case: $\ell = 0$, then $n = d$, and the claim follows from $A(d) \in \mathcal{A}$. Inductive case: suppose there is a path in G of length $\ell - 1$ from n' to d and $(n, n') \in E$. By the inductive hypothesis, $\mathcal{K} \models A(n')$, and since by definition $P(n, n') \in \mathcal{A}$, we have that $\mathcal{K} \models \exists P.A(n)$. By the inclusion assertion in $\mathcal{T}$ it follows that $\mathcal{K} \models A(n)$.

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 $(\text{funct } P^-)$. Moreover, the graph can be encoded using only functional roles, and we can again get rid of inverse roles by inverting edges.

More precisely, let $G = (N, E)$ be a directed graph and consider the problem of reachability in G between nodes s and d. We define the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where the TBox $\mathcal{T}$ is constituted by the assertions

$$A \sqsubseteq \exists P_1.B \qquad B \sqsubseteq \exists P_1.B \qquad B \sqsubseteq \exists P_2.A \qquad (\text{funct } P_1) \qquad (\text{funct } P_2)$$

and the ABox $\mathcal{A}$ makes use of the nodes in N and the edges in E as constants. Consider a node n of G, and let $e_1, \ldots, e_k$ be all edges of G that have n as their target (i.e., such that $e_i = (n_i, n)$ for some node n_i), taken in some arbitrarily chosen order. Then the ABox $\mathcal{A}$ contains the following membership assertions:

- $P_1(n, e_1)$, and $P_1(e_i, e_{i+1})$ for $i \in \{1, \ldots, k-1\}$,
- $P_2(e_i, n_i)$, where $e_i = (n_i, n)$, for $i \in \{1, \ldots, k\}$.

Additionally, $\mathcal{A}$ contains the membership assertion $A(d)$. Notice that the assertions in the ABox do not violate the functionality assertions in the TBox. Again, it is easy to see that $\mathcal{K}$ can be constructed in LogSpace from G, s, and d. We show that there is a path in G from s to d if and only if $\mathcal{K} \models A(s)$.

"$\Leftarrow$" Suppose there is no path in G from s to d. We construct a model $\mathcal{I}$ of $\mathcal{K}$ such that $s^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = \{o\} \cup N \cup E$, and in which each constant of the ABox is interpreted as itself, $P_1^{\mathcal{I}}$ and $P_2^{\mathcal{I}}$ contain all pairs of nodes directly required by the ABox assertions, $A^{\mathcal{I}}$ contains each node n such that there is a path in G from n to d, and $B^{\mathcal{I}}$ contains all edges (i, j) such that there is a path in G from j to d. To satisfy the assertion $A \sqsubseteq \exists P_1.B$ for those objects $n \in A^{\mathcal{I}}$ that have no outgoing P_1 edge forced by the ABox (i.e., that have no incoming edge in G), we set $o \in B^{\mathcal{I}}$, $(n, o) \in P_1^{\mathcal{I}}$, and $(o, o) \in P_1^{\mathcal{I}}$. We use o in a similar way to satisfy the assertions $B \sqsubseteq \exists P_1.B$ and $B \sqsubseteq \exists P_2.A$, by setting $(o, o) \in P_2^{\mathcal{I}}$ and $o \in A^{\mathcal{I}}$. Note that in this way the functionality assertions are not violated. It is easy to see that $\mathcal{I}$ is a model of $\mathcal{K}$, and since there is no path in G from s to d, we have that $s \notin A^{\mathcal{I}}$.

"$\Rightarrow$" Suppose there is a path in G from a node n to d. We prove by induction on the length ℓ of such a path that $\mathcal{K} \models A(n)$. Base case: $\ell = 0$, then $n = d$, and the claim follows from $A(d) \in \mathcal{A}$. Inductive case: suppose there is a path in G of length $\ell - 1$ from j to d and $(n, j) \in E$. Let $n_1, \ldots, n_h$ be the nodes of G such that $(n_i, j) \in E$, up to $n_h = n$ and in the same order used in the construction of the ABox. By the inductive hypothesis, $\mathcal{K} \models A(j)$, and by the assertion $A \sqsubseteq \exists P_1.B$, functionality of P_1, and the ABox assertion $P_1(j, (n_1, j))$, we obtain that $\mathcal{K} \models B((n_1, j))$. Exploiting $B \sqsubseteq \exists P_1.B$, functionality of P_1, and the ABox assertion $P_1((n_i, j), (n_{i+1}, j))$, we obtain by induction on h that $\mathcal{K} \models B((n_h, j))$. Finally, by $B \sqsubseteq \exists P_2.A$, functionality of P_2, and the ABox assertion $P_2((n_h, j), n_h)$, we obtain that $\mathcal{K} \models A(n_h)$, i.e., $\mathcal{K} \models A(n)$. □

Note that all the above "negative" results already hold for instance checking, i.e., for the simplest possible queries. 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.

Notably, the above result says that restriction $(*)$ imposed on *DL-Lite*$_{\mathcal{A}}$ and *DLR-Lite*$_{\mathcal{A},\sqcap}$ (cf. Section 2.1) is crucial in order to guarantee FOL-rewritability of instance checking and query answering. Indeed, consider the DL *DL-Lite*$_{\mathcal{F},\mathcal{R}}$ that

is identical to *DL-Lite*$_{\mathcal{A}}$, except that restriction $(*)$ is not enforced. We observe that, using a transformation analogous to the one shown in Section 2.1 to deal with qualified existential quantification, a concept inclusion assertion of the form $A_1 \sqsubseteq \exists P.A_2$ can be rewritten equivalently into $A_1 \sqsubseteq \exists P_{A_2}$, $P_{A_2} \sqsubseteq P$, $\exists P_{A_2}^- \sqsubseteq A_2$, where P_{A_2} is a newly introduced role. Hence, the KB used in the proof of Case 3 of Theorem 4.1 can be rewritten into *DL-Lite*$_{\mathcal{F},\mathcal{R}}$, and in the resulting KB functional roles are also specialized, i.e., such a KB is not a *DL-Lite*$_{\mathcal{A}}$ KB. It follows that, if restriction $(*)$ does not hold, instance checking is NLogSpace-hard, and hence not FOL-rewritable. An analogous observation holds for *DLR-Lite*$_{\mathcal{A},\sqcap}$.

Theorem 4.2. *Conjunctive query answering is* NLogSpace*-complete with respect to data complexity for the DLs of Case* 1 *and Case* 2 *of Theorem* 4.1.

Proof. NLogSpace-hardness was already proved in Theorem 4.1. It remains to show membership in NLogSpace. For Case 1, it follows from the fact that the DL considered is a sub-logic of the DL called *DL-Lite*$^+$ in [29], and that conjunctive query answering in such a DL is NLogSpace-complete with respect to data complexity [29]. For Case 2, membership in NLogSpace immediately follows from the fact that, as already shown, this case can be reduced to Case 1. □

We conjecture that also for Case 3 of Theorem 4.1, conjunctive query answering can be done in NLogSpace, and hence the established lower-bound is tight.

4.2. PTime*-hard DLs*

Next we show that if we consider further extensions to the logics mentioned in Theorem 4.1, we get even stronger complexity results. In particular, we consider four 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.

We start by considering DLs obtained by adding conjunction in the left-hand side of inclusion assertions for the DLs considered in Theorem 4.1. Such an addition makes instance checking a PTime-hard problem.

Theorem 4.3. *Instance checking* (*and hence query answering*) *is* PTime*-complete with respect to data complexity for the cases where*

1. $Cl \longrightarrow A \mid \exists P.A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A$
   TBox assertions: $Cl \sqsubseteq Cr$.
2. $Cl \longrightarrow A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A \mid \forall P.A$
   TBox assertions: $Cl \sqsubseteq Cr$.
3. $Cl \longrightarrow A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A \mid \exists P.A$
   TBox assertions: $Cl \sqsubseteq Cr$ $\quad (\mathsf{funct}\, P)$.

Proof. We first show PTime-hardness.

For Case 1, the proof is by a LogSpace reduction from path system accessibility, which is PTime-complete [30]. An instance of path system accessibility is defined as $PS = (N, E, S, t)$, where N is a set of nodes, $E \subseteq N \times N \times N$ is an accessibility relation (we call its elements edges), $S \subseteq N$ is a set of source nodes, and $t \in N$ is a terminal node. *PS* consists in verifying whether t is *accessible*, where a node $n \in N$ is accessible if $n \in S$ or if there exist accessible nodes n_1 and n_2 such that $(n, n_1, n_2) \in E$.

We define the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where the TBox $\mathcal{T}$ is constituted by the inclusion assertions

$$\exists P_1.A \sqsubseteq B_1 \qquad \exists P_2.A \sqsubseteq B_2 \qquad B_1 \sqcap B_2 \sqsubseteq A \qquad \exists P_3.A \sqsubseteq A$$

and the ABox $\mathcal{A}$ makes use of the nodes in N and the edges in E as constants. Consider a node $n \in N$, and let $e_1, \ldots, e_\ell$ be all edges in E that have n as their first component, taken in some arbitrarily chosen order. Then the ABox $\mathcal{A}$ contains the following membership assertions:

- $P_3(n, e_1)$, and $P_3(e_i, e_{i+1})$ for $i \in \{1, \ldots, \ell - 1\}$,
- $P_1(e_i, j)$ and $P_2(e_i, k)$, where $e_i = (n, j, k)$, for $i \in \{1, \ldots, \ell\}$.

Additionally, $\mathcal{A}$ contains one membership assertion $A(n)$ for each node $n \in S$. Again, it is easy to see that $\mathcal{K}$ can be constructed in LogSpace from *PS*. We show that t is accessible in *PS* if and only if $\mathcal{K} \models A(t)$.

"$\Leftarrow$" Suppose that t is not accessible in *PS*. We construct a model $\mathcal{I}$ of $\mathcal{K}$ such that $t^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = N \cup E$, and in which each constant of the ABox is interpreted as itself, $P_1^{\mathcal{I}}$, $P_2^{\mathcal{I}}$, and $P_3^{\mathcal{I}}$ consist of all pairs of nodes directly required by the ABox assertions, $B_1^{\mathcal{I}}$ consists of all edges (i, j, k) such that j is accessible in *PS*, $B_2^{\mathcal{I}}$ consists

of all edges (i, j, k) such that k is accessible in PS, and $A^{\mathcal{I}}$ consists of all nodes n that are accessible in PS union all edges (i, j, k) such that both j and k are accessible in PS. It is easy to see that $\mathcal{I}$ is a model of $\mathcal{K}$, and since t is not accessible in PS, we have that $t \notin A^{\mathcal{I}}$.

"$\Rightarrow$" Suppose that t is accessible in PS. We prove by induction on the structure of the derivation of accessibility that if a node n is accessible, then $\mathcal{K} \models A(n)$. Base case (direct derivation): $n \in S$, hence, by definition, $\mathcal{A}$ contains the assertion $A(n)$ and $\mathcal{K} \models A(n)$. Inductive case (indirect derivation): there exists an edge $(n, j, k) \in E$ and both j and k are accessible. By the inductive hypothesis, we have that $\mathcal{K} \models A(j)$ and $\mathcal{K} \models A(k)$. Let $e_1, \ldots, e_\ell$ be the edges in E that have n as their first component, up to $e_\ell = (n, j, k)$ and in the same order used in the construction of the ABox. Then, by $P_1(e_\ell, j)$ in the ABox and the assertions $\exists P_1.A \sqsubseteq B_1$ we have that $\mathcal{K} \models B_1(e_\ell)$. Similarly, we get $\mathcal{K} \models B_2(e_\ell)$, and hence, by $B_1 \sqcap B_2 \sqsubseteq A$, we get $\mathcal{K} \models A(e_\ell)$. By exploiting the ABox assertion $P_3(e_i, e_{i+i})$ and the TBox assertions $\exists P_3.A \sqsubseteq A$, we obtain by induction on ℓ that $\mathcal{K} \models A(e_1)$. Finally, by $P_3(n, e_1)$, we obtain that $\mathcal{K} \models A(n)$.

For Cases 2 and 3, the proof of PTime-hardness follows from Case 1 and observations analogous to the ones for Theorem 4.1.

Finally, membership in PTime immediately follows from the fact that all the three DLs considered are sub-logics of the DL *Horn-$\mathcal{SHIQ}$*, and that data complexity of conjunctive query answering in *Horn-$\mathcal{SHIQ}$* is in PTime with respect to data complexity [31]. □

In the presence of inverse roles, already the use of qualified existential restriction on the left-hand side of inclusion assertions is sufficient to obtain PTime-hardness, as shown by the following theorem.

Theorem 4.4. *Instance checking* (*and hence query answering*) *is* PTime*-complete with respect to data complexity for the case where*

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

TBox assertions: $Cl \sqsubseteq Cr$.

Proof. The hardness part is a consequence of Theorem 4.3, together with the observation that an inclusion assertion of the form $A_1 \sqcap A_2 \sqsubseteq A_3$ can be encoded by introducing a fresh atomic concept A_{123} and a fresh atomic role P_{123}, and adding to the TBox the following inclusion assertions:

$$A_1 \sqsubseteq \exists P_{123} \qquad \exists P_{123}^-.A_2 \sqsubseteq A_{123} \qquad \exists P_{123}.A_{123} \sqsubseteq A_3.$$

Indeed, consider an interpretation $\mathcal{I}$ that is a model of a TBox enriched with the above assertions, and an object $o \in A_1^{\mathcal{I}} \cap A_2^{\mathcal{I}}$. By assertion $A_1 \sqsubseteq \exists P_{123}$, since $o \in A_1^{\mathcal{I}}$, there is an object $o' \in \Delta^{\mathcal{I}}$ such that $(o, o') \in P_{123}^{\mathcal{I}}$. By assertion $\exists P_{123}^-.A_2 \sqsubseteq A_{123}$, since $o \in A_2^{\mathcal{I}}$, we have that $o' \in A_{123}^{\mathcal{I}}$. Hence, by assertion $\exists P_{123}.A_{123} \sqsubseteq A_3$, we have that $o \in A_3^{\mathcal{I}}$. On the other hand, it is easy to see that if in an interpretation $\mathcal{I}$ we have that $o \notin A_1^{\mathcal{I}} \cap A_2^{\mathcal{I}}$, then the above assertions are satisfied in $\mathcal{I}$ even if $o \notin A_3^{\mathcal{I}}$.

Membership in PTime immediately follows from the fact that the considered DL is a sub-logic of the DL *Horn-$\mathcal{SHIQ}$*, and that data complexity of conjunctive query answering in *Horn-$\mathcal{SHIQ}$* is in PTime with respect to data complexity [31]. □

4.3. 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 4.5. *Query answering is coNP-complete with respect to data complexity for the cases where*

1. $\begin{array}{lcl} Cl & \rightarrow & A \\ Cr & \rightarrow & A \mid A_1 \sqcup A_2 \end{array}$
   TBox assertions: $Cl \sqsubseteq Cr$.
2. $\begin{array}{lcl} Cl & \rightarrow & A \mid \neg A \\ Cr & \rightarrow & A \end{array}$
   TBox assertions: $Cl \sqsubseteq Cr$.
3. $\begin{array}{lcl} Cl & \rightarrow & A \mid \forall P.A \\ Cr & \rightarrow & A \end{array}$
   TBox assertions: $Cl \sqsubseteq Cr$.

Proof. Let us start from coNP-hardness. In all three cases, the proof is an adaptation of the proof of coNP-hardness of instance checking for $\mathcal{ALE}$ presented in [13]. We first consider Case 1.

coNP-hardness of query answering is proved by a reduction from $2+2$-CNF unsatisfiability (which is showed to be coNP-complete in [13]). A $2+2$-CNF formula on an alphabet P is a CNF formula in which each clause has exactly four literals: two positive ones and two negative ones, where the propositional letters are elements of $P \cup \{\texttt{true}, \texttt{false}\}$. Given a $2+2$-CNF formula $F = C_1 \wedge \cdots \wedge C_n$, where $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$, we associate with it the knowledge base $\mathcal{K}_F = \langle \mathcal{T}_F, \mathcal{A}_F \rangle$ defined as follows:

$$
\begin{aligned}
\mathcal{T}_F &= \{O \sqsubseteq A_t \sqcup A_f\} \\
\mathcal{A}_F &= \big\{A_t(\texttt{true}), A_f(\texttt{false}), \\
&\qquad O\big(\ell^1_{1+}\big), O\big(\ell^1_{2+}\big), O\big(\ell^1_{1-}\big), O\big(\ell^1_{2-}\big), \\
&\qquad \cdots \\
&\qquad O\big(\ell^n_{1+}\big), O\big(\ell^n_{2+}\big), O\big(\ell^n_{1-}\big), O\big(\ell^n_{2-}\big), \\
&\qquad P_1\big(c_1, \ell^1_{1+}\big), P_2\big(c_1, \ell^1_{2+}\big), N_1\big(c_1, \ell^1_{1-}\big), N_2\big(c_1, \ell^1_{2-}\big), \\
&\qquad \cdots \\
&\qquad P_1\big(c_n, \ell^n_{1+}\big), P_2\big(c_n, \ell^n_{2+}\big), N_1\big(c_n, \ell^n_{1-}\big), N_2\big(c_n, \ell^n_{2-}\big)\big\}.
\end{aligned}
$$

Intuitively, $\mathcal{K}_F$ has one constant $\ell^i_{j\pm}$ for each literal $L^i_{j\pm}$ in F, one constant c_i for each clause C_i, and two constants $\texttt{true}$ and $\texttt{false}$ for the corresponding propositional constants. The atomic roles of $\mathcal{K}_F$ are P_1, P_2, N_1, and N_2, used to connect c_i to the constants corresponding to the two positive and to the two negative literals of C_i. The atomic concepts of $\mathcal{K}_F$ are O, A_t, and A_f, where O represents all the literals of F, while A_t and A_f represent the literals that are *true* and *false*, respectively. Note that the ABox $\mathcal{A}_F$ contains the assertions $A_t(\texttt{true})$ and $A_f(\texttt{false})$ in order to guarantee that in each model $\mathcal{I}$ of $\mathcal{K}_F$ the constants $\texttt{true}$ and $\texttt{false}$ are interpreted respectively as members of $A_t^{\mathcal{I}}$ and $A_f^{\mathcal{I}}$ (possibly of both).

Then, we consider the following boolean query q:

$$q \leftarrow P_1(x, y), A_f(y), P_2(x, z), A_f(z), N_1(x, w_1), A_t(w_1), N_2(x, w_2), A_t(w_2).$$

Intuitively, checking whether $\mathcal{K}_F \models q$ (i.e., whether the query evaluates to true in $\mathcal{K}_F$) corresponds to checking whether in every truth assignment for the formula F there exists a clause whose positive literals are interpreted as false and whose negative literals are interpreted as true, i.e., a clause that is not satisfied. Next we show that the formula F is unsatisfiable if and only if $\mathcal{K}_F \models q$.

"$\Rightarrow$" Suppose that F is unsatisfiable. Consider a model $\mathcal{I}$ of $\mathcal{K}_F$ (which always exists since $\mathcal{K}_F$ is always satisfiable), and let $\delta_{\mathcal{I}}$ be the truth assignment for F such that $\delta_{\mathcal{I}}(L) = \mathit{true}$ iff $\ell^{\mathcal{I}} \in A_t^{\mathcal{I}}$, for every literal L in F (and corresponding constant ℓ in $\mathcal{K}_F$). Since F is unsatisfiable, there exists a clause C_i that is not satisfied by $\delta_{\mathcal{I}}$, and therefore $\delta_{\mathcal{I}}(L^i_{1+}) = \mathit{false}$, $\delta_{\mathcal{I}}(L^i_{2+}) = \mathit{false}$, $\delta_{\mathcal{I}}(L^i_{1-}) = \mathit{true}$, and $\delta_{\mathcal{I}}(L^i_{2-}) = \mathit{true}$. By definition of $\delta_{\mathcal{I}}$, it follows that in $\mathcal{K}_F$ the interpretation of the constants related to c_i through the roles P_1 and P_2 is not in $A_t^{\mathcal{I}}$, and consequently is in $A_f^{\mathcal{I}}$, and the interpretation of the constants related to c_i through the roles N_1 and N_2 is in $A_t^{\mathcal{I}}$. Thus, there exists a substitution σ that assigns variables in q to constants in $\mathcal{K}_F$ in such a way that $\sigma(q)$ evaluates to true in $\mathcal{I}$ (notice that this holds even if the propositional constants $\texttt{true}$ or $\texttt{false}$ occur in F). Therefore, since this argument holds for each model $\mathcal{I}$ of $\mathcal{K}_F$, we can conclude that $\mathcal{K}_F \models q$.

"$\Leftarrow$" Suppose that F is satisfied by some truth assignment δ, and let $\mathcal{I}_\delta$ be the interpretation for $\mathcal{K}_F$ defined as follows, where we use L to denote the literal in F corresponding to constant ℓ:

- $O^{\mathcal{I}_\delta} = \{\ell^{\mathcal{I}_\delta} \mid L \text{ occurs in } F\}$,
- $A_t^{\mathcal{I}_\delta} = \{\ell^{\mathcal{I}_\delta} \mid \delta(L) = \mathit{true}\} \cup \{\texttt{true}\}$,
- $A_f^{\mathcal{I}_\delta} = \{\ell^{\mathcal{I}_\delta} \mid \delta(L) = \mathit{false}\} \cup \{\texttt{false}\}$,
- $\rho^{\mathcal{I}_\delta} = \{(a^{\mathcal{I}_\delta}, b^{\mathcal{I}_\delta}) \mid \rho(a, b) \in \mathcal{A}_F\}$, for $\rho \in \{P_1, P_2, N_1, N_2\}$.

It is easy to see that $\mathcal{I}_\delta$ is a model of $\mathcal{K}_F$. On the other hand, since F is satisfiable, for every clause in F there exists a positive literal interpreted as *true* or a negative literal interpreted as *false*. It follows that for every constant c_i, either one of the roles P_1 or P_2 relates c_i to a constant whose interpretation is in $A_t^{\mathcal{I}_\delta}$, or one of the roles N_1 or N_2 relates c_i to a constant whose interpretation is in $A_f^{\mathcal{I}_\delta}$. It follows that the query q evaluates to *false* in $\mathcal{I}_\delta$, and therefore $\mathcal{K}_F \not\models q$.

Proofs for Cases 2 and 3 are obtained by analogous reductions from $2+2$-CNF unsatisfiability. More precisely, for Case 2 the knowledge base $\mathcal{K}_F = \langle \mathcal{T}_F, \mathcal{A}_F \rangle$ has the same constants and the same atomic roles as for Case 1, and has only the atomic concepts A_t and A_f. Then, $\mathcal{T}_F = \{\neg A_t \sqsubseteq A_f\}$ and $\mathcal{A}_F$ is as for Case 1 but without the assertions involving the concept O. Finally, the query q is as for Case 1.

For Case 3, $\mathcal{K}_F$ is similar to the one for Case 2. It has the same constants, the same atomic roles plus an extra atomic role P, and the atomic concepts A and A_f. The TBox is $\mathcal{T}_F = \{\forall P.A \sqsubseteq A_f\}$. The ABox $\mathcal{A}_F$ is as for Case 2, but without the

assertion $A_t(\texttt{true})$, which is substituted by the assertion $P(\texttt{true}, d)$, where d is a new constant not occurring elsewhere in $\mathcal{K}_F$. Finally, the query q is as follows

$$q \leftarrow P_1(x, y), A_f(y), P_2(x, z), A_f(z), N_1(x, w_1), P(w_1, w_2), N_2(x, w_2), P(w_3, w_4)$$

Soundness and completeness of the above reductions can be proved as done for the reduction of Case 1.

We point out that the intuition behind the above reductions is that in all three cases it is possible to require a reasoning by case analysis, caused by set covering assertions. Indeed, whereas in Case 1 we have explicitly asserted $O \sqsubseteq A_t \sqcup A_f$, for the other cases this can be seen by considering that A_t and A_f, and $\forall P.A$ and $\exists P$ cover the entire domain in Case 2 and Case 3, respectively.

Finally, membership in coNP follows immediately from the fact that each of the DLs above considered is a sub-logic of the DL $\mathcal{ALCNR}$, and that conjunctive query answering in $\mathcal{ALCNR}$ is coNP-complete with respect to data complexity [32]. □

5. Related work

The first results on decidability of conjunctive query answering over DL knowledge bases in expressive DLs appeared in [33,5,9]. In particular, [5] proves a 2ExpTime-upper bound in combined complexity for the $\mathcal{ALCI}$ and *DLR* families of DLs. This upper bound has been extended in [11] to conjunctive query answering in $\mathcal{SHIQ}$. In [34] an analogous result is presented for the DL $\mathcal{SHOQ}$, which differs from $\mathcal{SHIQ}$ since it does not allow for inverse roles, while it allows for nominals. A 2ExpTime lower bound for conjunctive query answering has been shown for $\mathcal{ALCI}$, while for $\mathcal{ALC}$, i.e., dropping inverse roles, the problem is ExpTime-complete [10]. These results are complemented in [16], which shows that also for $\mathcal{SH}$, i.e., the DL that includes transitive roles but no inverses, conjunctive query answering is 2ExpTime-hard (and hence 2ExpTime-complete).

The first main data complexity results for DLs appeared in [13], where a coNP lower bound for data complexity of instance checking in the DL $\mathcal{ALE}$ was shown. [33] gives a coNP upper-bound with respect to data complexity for conjunctive query answering in a DL with arbitrary inclusion assertions, but lacking inverse roles. In the last decade, there has been a renewed interest in data complexity of conjunctive query answering, both for expressive and novel DLs specifically designed to have low data complexity.

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 [35,14,36]. It remains open whether such a technique can be extended to deal efficiently with conjunctive queries for expressive DLs.

In [14], a fragment of $\mathcal{SHIQ}$, called Horn-$\mathcal{SHIQ}$, which subsumes both *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, is studied and a PTime upper bound in data complexity for instance checking is shown. Our results, in particular, Theorem 4.4, 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.

Building on the techniques presented in [33], coNP-completeness in data complexity of answering conjunctive queries in $\mathcal{SHIQ}$, which includes inverse roles and number restrictions (which generalize functionality) has been shown in [15]. It is interesting to observe that the results presented here, in particular, Theorem 4.5, tell us that we get coNP-completeness already for very small fragments of $\mathcal{SHIQ}$.

Concerning less expressive DLs, [37,19] show that conjunctive query answering in the DL $\mathcal{EL}$ is tractable, i.e., PTime, with respect to data complexity. The PTime lower bound follows from Theorem 4.3, which was originally presented in [25]. Instead, conjunctive query answering in the DL $\mathcal{EL}$++ is undecidable, as independently shown in [19,38,37]. Complexity results and algorithms for query answering in the *DL-Lite* family of DLs (as described in Section 1) are presented in [18]. This investigation has been extended in [20] to DLs whose concepts are built as boolean combinations of atomic concepts and projections on roles (actually considered in the more general form of minimal number restrictions). In particular [20] studies extensions of the DLs of the *DL-Lite* family with number restrictions, role constructs such as (ir)reflexivity and (a)symmetry, different forms of concept inclusions (corresponding to Horn, Krom, and Boolean formulas), and establishes both upper and lower bounds for such logics. The FOL-rewritability results rely on a correspondence with first-order logic with unary predicates, and they cover the *DL-Lite*-variants presented in [18] (over binary relations) extended with number restrictions, (a)symmetry and (ir)reflexivity of roles, and Horn inclusions (which correspond to role conjunction as in [25]). The paper studies also DLs that are not FOL-rewritable and establishes some lower bound results that are incomparable to those presented here. We also observe that [39] advocates a new perspective on data complexity of conjunctive query answering in DLs, by considering the complexity of query answering for TBoxes with specific properties, as opposed to considering a specific TBox language. This allows one, e.g., to rephrase some of the results established in Section 4 on the border between tractability and intractability.

A technique for optimizing conjunctive query answering in *DL-Lite* is shown in [40], and in [41–44] algorithms for optimizing conjunctive query rewriting in *DL-Lite* are presented.

A recent line of research follows the idea of extending Datalog rules with existential variables in rule heads [45–49,44,50]. Among these approaches, the Datalog$^{\pm}$ family described in [45,51,52] is closely related to the present paper. Actually, Datalog$^{\pm}$ is inspired by the work on *DL-Lite*, and may be seen as an attempt to generalize the *DL-Lite* approach to more expressive ontology languages based on logical rules. In this framework, a lot of attention has been devoted to define FOL-rewritable fragments of Datalog$^{\pm}$ programs (see, e.g., [49]). In particular, [53] presents a detailed study of the relationship

Table 5
Data complexity of query answering in description logics.

Cl	*Cr*	$\mathcal{F}$	$\mathcal{R}$	Data complexity of query answering
DLR-Lite$_{\mathcal{A},\sqcap}$		$\surd$[a]	$\surd$[a]	in AC^0
$A \mid \exists P.A$	A	–	–	NLogSpace-complete
A	$A \mid \forall P.A$	–	–	NLogSpace-complete
A	$A \mid \exists P.A$	$\surd$	–	NLogSpace-hard
$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTime-complete
$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTime-complete
$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTime-complete
$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	–	–	PTime-complete
A	$A \mid A_1 \sqcup A_2$	–	–	coNP-complete
$A \mid \neg A$	A	–	–	coNP-complete
$A \mid \forall P.A$	A	–	–	coNP-complete

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

[a] With the proviso that relations involved in key assertions are not specialized.

between the Datalog$^{\pm}$ family and the *DL-Lite* family of languages. This paper shows that a particular class of Datalog$^{\pm}$ programs, called *multi-linear*, is actually able to capture the DL *DLR-Lite*$_{\mathcal{A},\sqcap}$ presented in Section 2.

Observe also that *DL-Lite*$_{\mathcal{R}}$ 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-rewritable. Finally, if we move from RDFS to DLP [54], query answering becomes PTime-hard, since DLP is a superset of the DL in Case 1 of Theorem 4.3.

As for the management of conjunctive queries in implemented DL systems: all the DLs studied in this paper are fragments of expressive DLs with assertions and inverses studied in the 90's (see [23] for an overview), which are at the base of current ontology languages such as OWL, and for which optimized automated reasoning systems such as RacerPro,[8] Pellet,[9] Hermit,[10] and Fact++,[11] have been developed. Indeed, one could use, off-the-shelf, a system like RacerPro or Pellet to perform instance checking in such DLs. However, none of these systems fully supports conjunctive query answering. Some of the above systems actually allow users to pose conjunctive queries, but such queries are evaluated under an approximation of the classical first-order semantics of conjunctive queries: in the approximated semantics, existential variables can only be assigned to explicit individuals of the knowledge base, rather than to arbitrary objects of the interpretation domain.

While the implementation of systems for answering conjunctive queries in DLs under the "real" first-order semantics has been accomplished for DLs of the *DL-Lite* family [55–57,41,40,58,59] and of the $\mathcal{EL}$ family [60,61], for more expressive DLs the current technology seems not mature yet. Unfortunately, the known reasoning algorithms for answering conjunctive queries in these DLs, which have been used to characterize computational complexity, are not tailored towards obtaining efficient implementations, and more research on this is needed.

6. Conclusions

We have presented 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-rewritability 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 Table 5.

We are currently following several directions to continue the work reported in this paper. In particular, 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. Furthermore, 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.

[8] http://www.racer-systems.com/.
[9] http://clarkparsia.com/pellet/.
[10] http://www.comlab.ox.ac.uk/projects/HermiT/.
[11] http://owl.cs.manchester.ac.uk/fact++/.

Acknowledgements

This work has been partially supported by the EU under the FP7 ICT Collaborative project ACSI (Artifact-Centric Service Interoperation), grant No. FP7-257593.

References

[1] J. Lee, K. Siau, S. Hong, Enterprise integration with ERP and EAI, Communications of the ACM 46 (2) (2003) 54–60.
[2] M. Lenzerini, Data integration: A theoretical perspective, in: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), 2002, pp. 233–246.
[3] J. Heflin, J. Hendler, A portrait of the semantic web in action, IEEE Intelligent Systems 16 (2) (2001) 54–59.
[4] A. Borgida, R.J. Brachman, D.L. McGuinness, L.A. Resnick, CLASSIC: A structural data model for objects, in: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 59–67.
[5] D. Calvanese, G. De Giacomo, M. Lenzerini, 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), 1998, pp. 149–158.
[6] I. Horrocks, S. Tessaris, A conjunctive query language for description logic ABoxes, in: Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000), 2000, pp. 399–404.
[7] R. Fikes, P. Hayes, I. Horrocks, OWL-QL: A language for deductive query answering on the semantic web, Journal of Web Semantics 2 (1) (2005) 19–29.
[8] D. Calvanese, T. Eiter, M. Ortiz, Answering regular path queries in expressive description logics: An automata-theoretic approach, in: Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI 2007), 2007, pp. 391–396.
[9] D. Calvanese, G. De Giacomo, M. Lenzerini, Conjunctive query containment and answering under description logics constraints, ACM Transactions on Computational Logic 9 (3) (2008) 22.1–22.31.
[10] C. Lutz, The complexity of conjunctive query answering in expressive description logics, in: Proc. of the 4th Int. Joint Conf. on Automated Reasoning (IJCAR 2008), in: Lecture Notes in Artificial Intelligence, vol. 5195, Springer, 2008, pp. 179–193.
[11] B. Glimm, I. Horrocks, C. Lutz, U. Sattler, Conjunctive query answering for the description logic $\mathcal{SHIQ}$, Journal of Artificial Intelligence Research 31 (2008) 151–198.
[12] M.Y. Vardi, The complexity of relational query languages, in: Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82), 1982, pp. 137–146.
[13] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, Deduction in concept languages: From subsumption to instance checking, Journal of Logic and Computation 4 (4) (1994) 423–452.
[14] U. Hustadt, B. Motik, U. Sattler, Data complexity of reasoning in very expressive description logics, in: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), 2005, pp. 466–471.
[15] M. Ortiz, D. Calvanese, T. Eiter, Data complexity of query answering in expressive description logics via tableaux, Journal of Automated Reasoning 41 (1) (2008) 61–98.
[16] T. Eiter, C. Lutz, M. Ortiz, M. Šimkus, Query answering in description logics with transitive roles, in: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009), 2009, pp. 759–764.
[17] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, *DL-Lite*: Tractable description logics for ontologies, in: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), 2005, pp. 602–607.
[18] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, Journal of Automated Reasoning 39 (3) (2007) 385–429.
[19] A. Krisnadhi, C. Lutz, Data complexity in the $\mathcal{EL}$ family of description logics, in: Proc. of the 14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2007), 2007, pp. 333–347.
[20] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyaschev, The *DL-Lite* family and relations, Journal of Artificial Intelligence Research 36 (2009) 1–69.
[21] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison Wesley Publ. Co., 1995.
[22] O. Reingold, Undirected connectivity in log-space, Journal of the ACM 55 (4) (2008) 17:1–17:24.
[23] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003.
[24] E. Botoeva, A. Artale, D. Calvanese, Query rewriting in *DL-Lite*$^{\mathcal{HN}}_{horn}$, in: Proc. of the 23rd Int. Workshop on Description Logic (DL 2010), in: CEUR Electronic Workshop Proceedings, vol. 573, 2010, pp. 267–278, http://ceur-ws.org/.
[25] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Data complexity of query answering in description logics, in: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006), 2006, pp. 260–270.
[26] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, Journal on Data Semantics X (2008) 133–173.
[27] D.S. Johnson, A.C. Klug, Testing containment of conjunctive queries under functional and inclusion dependencies, Journal of Computer and System Sciences 28 (1) (1984) 167–189.
[28] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, Data exchange: Semantics and query answering, Theoretical Computer Science 336 (1) (2005) 89–124.
[29] H. Pérez-Urbina, B. Motik, I. Horrocks, Rewriting conjunctive queries over description logic knowledge bases, in: K.-D. Schewe, B. Thalheim (Eds.), Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008), in: Lecture Notes in Computer Science, vol. 4925, Springer, 2008, pp. 199–214.
[30] M.R. Garey, D.S. Johnson, Computers and Intractability – A Guide to NP-Completeness, W.H. Freeman and Company, San Francisco, CA, 1979.
[31] T. Eiter, G. Gottlob, M. Ortiz, M. Šimkus, Query answering in the description logic Horn-$\mathcal{SHIQ}$, in: Proc. of the 11th Eur. Conference on Logics in Artificial Intelligence (JELIA 2008), 2008, pp. 166–179.
[32] A.Y. Levy, M.-C. Rousset, Verification of knowledge bases based on containment checking, Artificial Intelligence 101 (1–2) (1998) 227–250.
[33] A.Y. Levy, M.-C. Rousset, Combining Horn rules and description logics in CARIN, Artificial Intelligence 104 (1–2) (1998) 165–209.
[34] B. Glimm, I. Horrocks, U. Sattler, Unions of conjunctive queries in $\mathcal{SHOQ}$, in: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008), 2008, pp. 252–262.
[35] U. Hustadt, B. Motik, U. Sattler, Reducing $\mathcal{SHIQ}$-description logic to disjunctive Datalog programs, in: Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004), 2004, pp. 152–162.
[36] B. Motik, Reasoning in description logics using resolution and deductive databases, Ph.D. thesis, Universität Karlsruhe, Karlsruhe, Germany, January 2006.
[37] R. Rosati, The limits of querying ontologies, in: Proc. of the 11th Int. Conf. on Database Theory (ICDT 2007), in: Lecture Notes in Computer Science, vol. 4353, Springer, 2007, pp. 164–178.
[38] M. Krötzsch, S. Rudolph, Conjunctive queries for $\mathcal{EL}$ with role composition, in: Proc. of the 20th Int. Workshop on Description Logic (DL 2007), in: CEUR Electronic Workshop Proceedings, vol. 250, 2007, pp. 355–362, http://ceur-ws.org/.

[39] C. Lutz, F. Wolter, Non-uniform data complexity of query answering in description logics, in: Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012), 2012, pp. 297–307.
[40] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyaschev, The combined approach to query answering in *DL-Lite*, in: Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010), 2010, pp. 247–257.
[41] R. Rosati, A. Almatelli, Improving query answering over *DL-Lite* ontologies, in: Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010), 2010, pp. 290–300.
[42] H. Pérez-Urbina, B. Motik, I. Horrocks, Tractable query answering and rewriting under description logic constraints, Journal of Applied Logic 8 (2) (2010) 186–209.
[43] G. Gottlob, T. Schwentick, Rewriting ontological queries into small nonrecursive Datalog programs, in: Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012), 2012, pp. 254–263.
[44] G. Gottlob, G. Orsi, A. Pieris, Ontological queries: Rewriting and optimization, in: Proc. of the 27th IEEE Int. Conf. on Data Engineering (ICDE 2011), 2011, pp. 2–13.
[45] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, in: Proc. of the 28th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2009), 2009, pp. 77–86.
[46] J.-F. Baget, M. Leclère, M.-L. Mugnier, E. Salvat, On rules with existential variables: Walking the decidability line, Artificial Intelligence 175 (9–10) (2011) 1620–1654.
[47] J.-F. Baget, M.-L. Mugnier, S. Rudolph, M. Thomazo, Walking the complexity lines for generalized guarded existential rules, in: Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011), 2011, pp. 712–717.
[48] M. Krötzsch, S. Rudolph, Extending decidable existential rules by joining acyclicity and guardedness, in: Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011), 2011, pp. 963–968.
[49] A. Calì, G. Gottlob, A. Pieris, New expressive languages for ontological query answering, in: Proc. of the 25th AAAI Conf. on Artificial Intelligence (AAAI 2011), 2011, pp. 1541–1546.
[50] N. Leone, M. Manna, G. Terracina, P. Veltri, Efficiently computable Datalog$^{\exists}$ programs, in: Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012), 2012, pp. 13–23.
[51] A. Calì, G. Gottlob, A. Pieris, Advanced processing for ontological queries, Proceedings of the VLDB Endowment 3 (1) (2010) 554–565.
[52] A. Calì, G. Gottlob, A. Pieris, Query answering under non-guarded rules in Datalog+/−, in: Proc. of the 4th Int. Conf. on Web Reasoning and Rule Systems (RR 2010), 2010, pp. 1–17.
[53] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, Tech. Rep. CL-RR-10-21, Oxford University Computing Laboratory, 2010.
[54] B.N. Grosof, I. Horrocks, R. Volz, S. Decker, Description logic programs: Combining logic programs with description logic, in: Proc. of the 12th Int. World Wide Web Conf. (WWW 2003), 2003, pp. 48–57.
[55] A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, R. Rosati, QuOnto: Querying ontologies, in: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), 2005, pp. 1670–1671.
[56] M. Stocker, M. Smith, Owlgres: A scalable OWL reasoner, in: Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008), in: CEUR Electronic Workshop Proceedings, vol. 432, 2008, http://ceur-ws.org/.
[57] E. Thomas, J.Z. Pan, Y. Ren, TrOWL: Tractable OWL 2 reasoning infrastructure, in: Proc. of the 7th Extended Semantic Web Conf. (ESWC 2010), 2010, pp. 431–435.
[58] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D.F. Savo, The Mastro system for ontology-based data access, Semantic Web Journal 2 (1) (2011) 43–53.
[59] M. Rodriguez-Muro, D. Calvanese, High performance query answering over *DL-Lite* ontologies, in: Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012), 2012, pp. 308–318.
[60] F. Baader, C. Lutz, B. Suntisrivaraporn, CEL—a polynomial-time reasoner for life science ontologies, in: Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006), in: Lecture Notes in Artificial Intelligence, vol. 4130, Springer, 2006, pp. 287–291.
[61] K. Dentler, R. Cornet, A. ten Teije, N. de Keizer, Comparison of reasoners for large ontologies in the OWL 2 EL profile, Semantic Web Journal 2 (2) (2011) 71–87.