Inconsistency-tolerant Query Answering in Ontology-based Data Access

Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati,
Marco Ruzzi, Domenico Fabio Savo

DIAG, Sapienza Università di Roma, Italy

Abstract

Ontology-based data access (OBDA) is receiving great attention as a new paradigm for managing information systems through semantic technologies. According to this paradigm, a Description Logic ontology provides an abstract and formal representation of the domain of interest to the information system, and is used as a sophisticated schema for accessing the data and formulating queries over them. In this paper, we address the problem of dealing with inconsistencies in OBDA. Our general goal is both to study DL semantical frameworks that are inconsistency-tolerant, and to devise techniques for answering unions of conjunctive queries under such inconsistency-tolerant semantics. Our work is inspired by the approaches to consistent query answering in databases, which are based on the idea of living with inconsistencies in the database, but trying to obtain only consistent information during query answering, by relying on the notion of database repair. We first adapt the notion of database repair to our context, and show that, according to such a notion, inconsistency-tolerant query answering is intractable, even for very simple DLs. Therefore, we propose a different repair-based semantics, with the goal of reaching a good compromise between the expressive power of the semantics and the computational complexity of inconsistency-tolerant query answering. Indeed, we show that query answering under the new semantics is first-order rewritable in OBDA, even if the ontology is expressed in one of the most expressive members of the DL-Lite family.

1. Introduction

Ontology-based data access [50] (OBDA) is receiving great attention as a new paradigm for managing information systems through semantic technologies. An OBDA system is structured according to a three-level architecture, which consists of the ontology, the data sources, and the mapping between the two. The ontology is an abstract and formal description of the domain of interest, and is used as a sophisticated schema for accessing the data and formulating queries over them. The data sources are the repositories storing the data used in the organization by the various processes and the various applications. The

mapping explicitly specifies the relationships between the domain concepts on the one hand and the data sources on the other hand. Like in data integration systems [49], the mapping is crucial for keeping the conceptual representation of the domain independent from the implementation issues, and for masking the user from all the details and the idiosyncrasies of the data sources.

In most formal approaches to OBDA, the ontology is expressed in terms of a Description Logic TBox. Description Logics (DLs) are logics specifically defined for describing knowledge bases in terms of objects, concepts, representing classes of objects, and relations between objects. represented by roles. A DL knowledge base consists of two components, called TBox and ABox. A DL TBox is a set of axioms, typically in the form of universally quantified statements, describing general properties of the concepts and the relationships that are relevant in the domain of interest. A DL ABox is a set of membership assertions stating the instances of concepts and relations. The most important service provided by an OBDA system is query answering, which amounts to computing the answers to a query posed in terms of the ontology.

In order to compute such answers, the system should reason about the ontology, the mapping, and the data at the source, with the goal of returning the tuples that satisfy the query in all the models of the system. Usually, the size of the ontology and the mapping in an OBDA system is limited with respect to the size of the data stored at the sources. For this reason, the crucial parameter for measuring the complexity of query answering in OBDA is the size of the data. Various languages for specifying the ontology in OBDA have been proposed [22, 4, 43, 17], which are designed with the goal of allowing for query answering algorithms that scale with the size of the data, and therefore are tractable with respect to *data complexity*, i.e., the complexity measured with respect to the size of the data only. Among these, the members of the *DL-Lite* family of lightweight DLs present the distinguishing feature of *first-order rewritable query answering* for unions of conjunctive queries (UCQs), which means that such a reasoning service can be realized through the evaluation of a suitable first-order query (called the rewriting of the original query, and directly translatable into SQL) over the data sources. We observe that, by virtue of its relevance in OBDA, the DL-Lite family is at the basis of OWL2 QL [58], one of the tractable profiles of OWL2.

Query rewriting is indeed the most popular approach to query answering in OBDA systems. According to this approach, query answering is divided into three steps:

1. The original query is first rewritten with respect to the ontology into a new query, again expressed over the ontology; we call this step the *ontology rewriting* of the query;

2. The query thus obtained is then rewritten into a source database query using the mapping assertions; we call this step the *mapping rewriting* of the query.

3. The query resulting from mapping rewriting is evaluated over the data

sources.

As observed in [24], when *DL-Lite* is used for expressing the ontology, first-order rewritability of query answering is guaranteed if the mapping is of type "global-as-view", i.e., each mapping assertion maps a query over the sources to a single element of the ontology. Under this assumption, it is well-known that mapping rewriting can be based on unfolding, which amounts to substituting every atom of the query with the corresponding query in the mapping[1]. In turn, it is easy to see that the whole query answering process can be based on the following strategy: (*i*) use the mapping bottom-up and compute, from the data, an ABox $\mathcal{A}$ containing all the instance assertions implied by the mapping; (*ii*) compute the answers to the query by evaluating the ontology rewriting on the computed ABox, seen simply as a relational database. This observation leads to the conclusion that, if we are not interested in optimizing the mapping rewriting step, we can simply ignore the mapping. Since the purpose of this paper is to study fundamental issues in dealing with inconsistencies in OBDA, in what follows we indeed abstract from the presence of mappings, and we consider simple DL knowledge bases consisting of a TBox and an ABox.

It is well-known that inconsistency causes severe problems in logic-based Knowledge Representation. In particular, since an inconsistent logical theory has no classical model, it logically implies every formula (*ex falso quodlibet*), and therefore query answering over an inconsistent knowledge base becomes meaningless under classical logic semantics. Similarly, when a database does not satisfy the integrity constraints defined on its schema, even the task of giving a meaning to the answers of queries becomes non-obvious.

There are various approaches for devising inconsistency-tolerant inference systems [10], originating from different areas, including Logic, Artificial Intelligence, and Databases. Roughly speaking, there are two main strategies behind such approaches. The most direct strategy is to *clean* the knowledge base of all contradictions [65, 56] so as to restore consistency. Another strategy is to leave the knowledge base unchanged, and to consider inconsistency as a natural phenomenon in realistic settings, which are to be handled by the logic used to express knowledge [61, 72, 55, 54]. An important class of such logics are called paraconsistent, and are based on the use of additional truth values standing, for example, for *underdefined* (i.e., neither true nor false), or *overdefined* (or contradictory, i.e., both true and false). Another class of such logics use the standard Boolean truth values, but tries to obtain only "meaningful" answers when evaluating queries. In the context of data and knowledge bases, this approach has been pursued first in the Database community and is commonly known as *consistent query answering* [3, 26].

In this paper, we address the problem of dealing with inconsistencies in OBDA, by starting with the consistent query answering approach, and adapting

[1] Obviously, unfolding might not be the most efficient strategy, and therefore, more sophisticated techniques for mapping rewriting have been studied (See, for example, [29]).

it to the context of DL knowledge bases. Depending on the expressive power of the underlying DL, the TBox alone might be inconsistent, or the TBox might be consistent, but the axioms in the ABox might contradict the axioms in the TBox. Here, we focus on the case where the TBox is consistent, while the instance-level information in the ABox may contradict the knowledge represented in the TBox. Indeed, in the OBDA scenario, it is appropriate to assume that the ontology is a high quality representation of the domain, designed in such a way to avoid inconsistencies in the modeling of concepts and relationships. On the contrary, as we said before, the ABox is defined through the mapping from the concrete data sources of the information system. Since such sources are often distributed, autonomous, and independent from the conceptualization represented by the TBox, they likely contain data that are not coherent with the TBox. In other words, assuming a consistent TBox and a possibly contradicting ABox is the appropriate setting for realistic OBDA applications.

In consistent query answering, the fundamental tool for obtaining consistent information from an inconsistent data or knowledge base, is the notion of repair [3, 18, 10, 26]. A *repair* of a database contradicting a set of integrity constraints is a database obtained by applying a "minimal" set of changes that restores consistency. There are several interpretations of the notion of "minimality", and different interpretations give rise to different inconsistency-tolerant semantics. Under most interpretations of minimality, there are many possible repairs for the same database, and the approach sanctions that what is consistently true is simply what is true in all possible repairs of the database. Thus, inconsistency-tolerant query answering amounts to computing the tuples that are answers to the query in all possible repairs.

In this paper we use the notion of repair in DL knowledge bases. Since we accept inconsistencies only in the ABox, we call this notion *ABox repair*, and *ABox Repair (AR) semantics* the corresponding semantics. For a DL knowledge base constituted by the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$, we define an ABox repair to be an inclusion-maximal subset of $\mathcal{A}$ that is consistent with $\mathcal{T}$. Unfortunately, we show that, even for what is considered to be the simplest logic of the DL-Lite family, inconsistency-tolerant query answering under the AR-semantics is coNP-complete with respect to data complexity, and therefore can be quite problematic in real-world applications.

To address this problem, we propose a new inconsistency-tolerant semantics for DL knowledge bases (KBs), which is the first contribution of this paper. The basic idea of the new semantics, called *Intersection ABox Repair (IAR) semantics*, is simple: instead of considering all the possible repairs of the KB as relevant for inconsistency-tolerant query answering, we consider the intersection of such repairs as the ABox to use in query answering, in the spirit of the well-known "When In Doubt, Throw It Out" principle [31]. Obviously, since all repairs are consistent, their intersection is also consistent, and query answering over the intersection can be done by resorting to the classical semantics. In other words, inconsistency-tolerant query answering in this new semantics is reduced to classical query answering over the intersection of all repairs of the original ABox. We show that the IAR-semantics enjoys several desirable properties.

In particular, we show that this semantics is a sound approximation of the AR-semantics, and that query answering under the IAR-semantics is first-order rewritable for unions of conjunctive queries, thus showing that the problem can be solved in polynomial time, in fact AC^0, in data complexity. This is the second contribution of this paper.

The first-order rewritability result concerns one of the most expressive logics in the DL-Lite family. This logic, called *DL-Lite*$_{A,id,den}$, includes the typical constructs of the DL-Lite family, and adds two distinguished features, namely identification assertions and denial assertions. *Identification assertions* are mechanisms for specifying a set of properties that can be used to identify instances of concepts [23]. Such assertions allow for sophisticated forms of object identification, which may include paths realized through the chaining of roles, their inverses, and attributes. *Denial assertions* are used to impose that the answer to a certain Boolean conjunctive query over the ontology is false, analogous to negative constraints in [16], thus forbidding that certain combinations of facts can occur in an ABox. This is particularly useful for specifying general forms of disjointness, that, like identification assertions, are not supported in traditional ontology languages. In all OBDA projects we have carried out in the last years (see, e.g., [69, 2]), the importance of these two constructs clearly emerged. Therefore, we believe that showing that these kinds of constraints can be added to DL-Lite without losing first-order rewritability of conjunctive queries even under inconsistency-tolerant semantics is crucial for promoting the adoption of OBDA in real applications.

We have implemented our algorithm and tested it over the LUBM benchmark ontology. In fact, we had to slightly modify the LUBM ontology to make it suitable for our testing aims by introducing in it denial and identification assertions, as well as some other forms of assertions possibly causing inconsistency. We made use of the LUBM data generator to obtain ABoxes of various size, ranging from around 100,000 facts to more than 2.7 million facts, and manually introduced in such ABoxes various percentages of inconsistency. We tested several queries, some taken from the LUBM benchmark, some others specifically defined by us. Our results, in terms of both rewriting and evaluation times, are encouraging and support the practical feasibility of inconsistency-tolerant query answering in an OBDA scenario.

The paper is organized as follows. In Section 2 we present the logic *DL-Lite*$_{A,id,den}$ and provide some preliminaries on query answering over DL knowledge bases (some details on first-order rewritability are deferred to Appendix 1). In Section 3 we briefly describe the algorithm for consistency checking in *DL-Lite*$_{A,id,den}$ under the classical first-order semantics. The correctness of the algorithm is proved in Appendix 2. In Section 4 we introduce the *AR*-semantics and in Section 5 we show that query answering under such semantics is intractable for both *DL-Lite*$_{A,id,den}$ and *DL-Lite*$_{core}$, i.e., the simplest *DL-Lite* logic. In Section 6 we present the *IAR*-semantics and in Section 7 we show that query answering under such semantics for UCQs is first-order rewritable in *DL-Lite*$_{A,id,den}$. In Section 8 we illustrate our experiments with the LUBM ontology, and in Section 9 we discuss related work. Section 10 concludes the paper.

The results presented in this paper appeared in preliminary form in [46, 47, 45].

2. Preliminaries

Description Logics [5] are decidable fragments of first-order logic (FOL) that can be used to represent the knowledge of a domain of interest in a structured and formally well-understood way. They model the domain of interest in terms of *objects*, i.e., individuals, *concepts*, that are abstractions for sets of objects, and *roles*, that denote binary relations between objects. In addition, some DLs distinguish concepts from *value-domains*, that denote sets of values, and roles from *attributes*, that denote binary relations between objects and values.

We consider an overall alphabet Γ, partitioned in two disjoint alphabets, namely, Γ_P, containing symbols for predicates, and Γ_C, containing symbols for individual (object and value) constants. We further partition Γ_P into four disjoint sets containing symbols denoting atomic concepts, atomic roles, attributes, and value-domains, respectively, and partition Γ_C into two disjoint sets, called Γ_O and Γ_V, which are the set of constants denoting objects, and the set of constants denoting values, respectively. In the rest of the paper, when it is clear from the context, we often implicitly refer to Γ.

Complex concept, role, and attribute expressions are constructed by applying suitable operators to atomic concepts and roles, attributes, and value-domains. Different DL languages allow for different operators in the constructs.

Given a DL language $\mathcal{L}$, an $\mathcal{L}$ *knowledge base*, or simply a DL knowledge base when $\mathcal{L}$ is clear from the context, over an alphabet Γ 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 Γ expressed in $\mathcal{L}$;
- $\mathcal{A}$, called the ABox of $\mathcal{K}$, is a finite set of extensional assertions (also called ABox assertions) over Γ expressed in $\mathcal{L}$.

Intuitively, $\mathcal{T}$ contains axioms specifying general properties of concepts, roles, and attributes, while $\mathcal{A}$ contains axioms about individual objects, thus specifying extensional knowledge. Again, different DL languages allow for different forms of TBox and ABox assertions.

The semantics of a DL knowledge base is given in terms of FOL interpretations (cf. [5]). Then, the notions of interpretation satisfying a FOL sentence, and of entailment of a sentence by a KB are given in the standard way.

In this paper, we consider *DL-Lite*$_{A,id,den}$, which is the most expressive member of the *DL-Lite* family of lightweight DLs[2] [22]. *DL-Lite*$_{A,id,den}$ has been recently introduced in [68, 45] as an extension with denial assertions of the logic *DL-Lite*$_{A,id}$, given originally in [23]. In the rest of this section, we provide

[2]Not to be confused with the set of DLs studied in [4], which is called the *extended DL-Lite* family.

syntax and semantics of *DL-Lite*$_{A,id,den}$. Since some of the hardness complexity results we give in the next sections hold already for *DL-Lite*$_{core}$, i.e., the least expressive member of the *DL-Lite* family (which can be seen as a fragment of *DL-Lite*$_{A,id,den}$), we also recall the syntax of *DL-Lite*$_{core}$. Finally, recall the basic notions related to query answering, and FO-rewritability.

2.1. DL-Lite$_{A,id,den}$

Concepts, roles, attributes, and value-domains in *DL-Lite*$_{A,id,den}$ are formed according to the following syntax:

$$\begin{array}{ll} B \longrightarrow A \mid \exists R \mid \delta(U) & E \longrightarrow \rho(U) \\ R \longrightarrow P \mid P^- & F \longrightarrow \top_D \mid T_1 \mid \cdots \mid T_n \end{array}$$

where A denotes an atomic concept, P an atomic role, P^- the *inverse* of an atomic role P, and U an attribute. B and R denote a *basic concept* and a *basic role*, respectively. The concept $\exists R$, also called unqualified existential restriction, denotes the domain of a role R, i.e., the set of objects that R relates to some object. Similarly, $\delta(U)$ denotes the *domain* of an attribute U, i.e., the set of objects that U relates to some values. Furthermore, E and F are *value-domain expressions*, $\rho(U)$ denotes the *range* of an attribute U, i.e., the set of values to which U relates some object, $T_1, \ldots, T_n$ denote unbounded pairwise disjoint predefined value-domains, and $\top_D$ is the *universal value-domain*. In the following, when R is a basic role, the expression R^- stands for P^- when R is of the form P, while R^- stands for P when R is of the form P^-. The symbols A, P, U, B, R, T_i will be used throughout the paper with the above meaning.

As stated earlier, the semantics of a *DL-Lite*$_{A,id,den}$ KB is given in terms of FOL interpretations. All such interpretations agree on the semantics assigned to each predefined value-domain T_i and to each constant in Γ_V. More precisely, we assume to have a fixed non-empty *domain of values* Δ_V, and to interpret each value-domain $T_i \in \{T_1, \ldots, T_n\}$ as the set $val(T_i)$. Furthermore, we interpret each constant $c_v \in \Gamma_V$ as one specific value, denoted $val(c_v)$, and assume that there is exactly one $T_i \in \{T_1, \ldots, T_n\}$ such that $val(c_v) \in val(T_i)$. Also, we denote by $type(c_v)$ the T_i such that $val(c_v) \in val(T_i)$. As usual in *DL-Lite*, we also assume that the set of value-domains $T_1, \ldots, T_n$ corresponds to any subset of the data types adopted by the Resource Description Framework (RDF) [42], having the following characteristics:

- (i) the extension of each T_i is unbounded;
- (ii) the extensions of value-domains are pairwise disjoint, i.e., $val(T_i) \cap val(T_j) = \emptyset$ for each $1 \leq i < j \leq n$;
- (iii) the extension of the complement in Δ_V of the extensions of all value-domains is unbounded, i.e., the set $\Delta_V \setminus \bigcup_{i=1}^{n} val(T_i)$ is unbounded.

The above assumptions allow us to treat value-domains as atomic concepts, and attributes as atomic roles when answering queries posed over *DL-Lite*$_{A,id,den}$ KBs, as we will discuss later on in this section.

$$
\begin{array}{lcl@{\qquad\qquad}rcl}
A^I & \subseteq & \Delta_O^I & (\exists R)^I & = & \{o \mid \exists o' . (o, o') \in R^I\} \\
P^I & \subseteq & \Delta_O^I \times \Delta_O^I & (\delta(U))^I & = & \{o \mid \exists v . (o, v) \in U^I\} \\
U^I & \subseteq & \Delta_O^I \times \Delta_V & (\rho(U))^I & = & \{v \mid \exists o . (o, v) \in U^I\} \\
T_i^I & = & val(T_i) & (P^-)^I & = & \{(o, o') \mid (o', o) \in P^I\} \\
\top_D^I & = & \Delta_V & & &
\end{array}
$$

Figure 1: Semantics of the *DL-Lite*$_{A,id,den}$ constructs

An *interpretation* $\mathcal{I} = (\Delta^I, \cdot^I)$ for a *DL-Lite*$_{A,id,den}$ KB over an alphabet Γ consists of an interpretation domain $\Delta^I = \Delta_O^I \cup \Delta_V$, where Δ_O^I is a non-empty set, disjoint from Δ_V, called the *domain of objects*, and an *interpretation function* $\cdot^I$. Such a function coincides with the function *val* in the interpretation of value domains and value constants, while it assigns a subset C^I of Δ_O^I to each concept C, a subset R^I of $\Delta_O^I \times \Delta_O^I$ to each role R, a subset U^I of $\Delta_O^I \times \Delta_V$ to each attribute U, and an element c^I of Δ_O^I to each constant $c \in \Gamma_O$. Each such interpretation $\mathcal{I}$ respects the *Unique Name Assumption* (UNA), i.e., $\mathcal{I}$ assigns each constant from Γ_O to a distinct element of Δ_O^I, and each constant from Γ_V to a distinct element of Δ_V.

Given an interpretation $\mathcal{I}$ of the form described above, the semantics for *DL-Lite*$_{A,id,den}$ concepts, roles, and attributes is specified in Figure 1.

We now turn to the definition of *DL-Lite*$_{A,id,den}$ TBox assertions, which have the following form[3]:

$$
\begin{array}{llll}
B_1 \sqsubseteq B_2 & R_1 \sqsubseteq R_2 & U_1 \sqsubseteq U_2 & E \sqsubseteq F \\
B_1 \sqsubseteq \neg B_2 & R_1 \sqsubseteq \neg R_2 & U_1 \sqsubseteq \neg U_2 & \\
(\mathsf{funct}\ R) & (\mathsf{funct}\ U) & & \\
\forall \vec{y} . conj(\vec{y}) \rightarrow \bot & & & \\
(id\ B\ \pi_1, \ldots, \pi_n) & & &
\end{array}
$$

Assertions of the first row are called *positive inclusion assertions* (or, simply, positive inclusions) specified, from left to right, between concepts, roles, attributes, and value-domains, respectively. Given a TBox $\mathcal{T}$, we denote with $\mathcal{T}_{inc}$ the set of concept, role, and attribute positive inclusions occurring in $\mathcal{T}$, and with $\mathcal{T}_{type}$ the set of positive inclusions between value-domains occurring in $\mathcal{T}$.

Assertions of the second row are *negative inclusion assertions* (or, simply, negative inclusions), also called disjointnesses, specified, from left to right, between concepts, roles, and attributes, respectively. The set of all negative inclusions in a TBox $\mathcal{T}$ is denoted by $\mathcal{T}_{disj}$.

[3]In fact, *DL-Lite*$_{A,id,den}$ also allows for the use of positive qualified existential restrictions [5] in the right-hand side of inclusions. This, however, can be easily encoded in *DL-Lite*$_{A,id,den}$ without qualified existential restrictions (cf. [22]).

Assertions of the third row, from left to right, are role and attribute *functionality assertions*, respectively, which state that a role or an attribute is functional. We denote with $\mathcal{T}_{funct}$ the set of the functional assertions of a TBox $\mathcal{T}$.

The assertion in the fourth row, i.e., $\forall \vec{y}.conj(\vec{y}) \rightarrow \bot$, is a *denial assertion* (or, simply, a denial). In such assertion, $conj(\vec{y})$ denotes a conjunction of atoms of the form $A(t_1)$, $P(t_1, t_2)$, or $U(t_1, t_3)$, where t_1, t_2 are either variables in $\vec{y}$ or constants in Γ_O, and t_3 is either a variable in $\vec{y}$ or a constant in Γ_V. Notice that such assertions correspond to Horn clauses having a false head. Intuitively, they allow one to specify that certain patterns of objects cannot occur as instances of the knowledge base. Interestingly, the form of these patterns is arbitrary, i.e., they are not limited to be tree-shaped, as usual in Description Logics. By means of denial assertions, we enrich DLs of the *DL-Lite* family with general forms of disjointnesses, otherwise not expressible in these languages, and with the ability of specifying irreflexivity of roles. For example, the denial assertion $\forall x.(hasFather(x, x) \rightarrow \bot)$ implies that the role $hasFather$ is irreflexive, i.e., that a person cannot be father of himself. In what follows, we denote with $\mathcal{T}_{den}$ the set of the denial assertions belonging to a TBox $\mathcal{T}$.

The last kind of assertion, i.e., $(id\ B\ \pi_1, \ldots, \pi_n)$, is an *identification assertion* (or, simply, an identification), stating that a set of properties identifies the instances of a basic concept B. In an identification assertion, π_i is a *path*, i.e., an expression built according to the following syntax:

$$\pi \longrightarrow S \ \mid \ D? \ \mid \ \pi_1 \circ \pi_2$$

where S denotes a basic role (i.e., an atomic role or the inverse of an atomic role), or an attribute, $\pi_1 \circ \pi_2$ denotes the composition of paths π_1 and π_2, and $D?$, called *test relation*, represents the identity relation on instances of D, which can be a basic concept or a value-domain expression. Test relations are used to impose that a path involves instances of a certain concept or value-domain. The length of a path π, denoted $length(\pi)$, is 0 if π has the form $D?$, is 1 if π has the form S, and is $length(\pi_1) + length(\pi_2)$ if π has the form $\pi_1 \circ \pi_2$. In our logic, identification assertions are *local*, i.e., at least one $\pi_i \in \{\pi_1, ..., \pi_n\}$ is of length 1, i.e., it is an atomic role, the inverse of an atomic role, or an attribute (possibly composed only with test relations). The term "local" emphasizes that at least one of the paths refers to a local property of B [23]. Intuitively, an identification assertion of the above form asserts that for any two different instances o, o' of B, there is at least one π_i such that o and o' differ in the set of their π_i-fillers, that is the set of objects that are reachable from o and o' by means of π_i. For example, the identification assertion $(\mathsf{id}\ Match\ homeTeam, visitorTeam)$ says that there are not two different matches with the same home team and visiting team (which is indeed what happens, for instance, in a season schedule of a football league). In what follows, we denote the set of the identification assertions belonging to a TBox $\mathcal{T}$ with $\mathcal{T}_{id}$.

We are now ready to present the definition of *DL-Lite*$_{A,id,den}$ KB.

Definition 1. *A DL-Lite*$_{A,id,den}$ *KB is a pair* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$*, such that:*

- *the TBox $\mathcal{T}$ consists of the finite sets $\mathcal{T}_{inc}$, $\mathcal{T}_{type}$, $\mathcal{T}_{disj}$, $\mathcal{T}_{funct}$, $\mathcal{T}_{den}$, and $\mathcal{T}_{id}$, as described above;*

- *the ABox $\mathcal{A}$ is a finite set of assertions of the forms $A(a), P(a,b)$ and $U(a,v)$, where a and b are constants in Γ_O, and v is a constant in Γ_V;*

- *each role or attribute that either is functional in $\mathcal{T}$, i.e., it occurs in a functionality assertion in $\mathcal{T}$, or appears (in either direct or inverse direction) in a path of an identification assertion in $\mathcal{T}$, is not specialized, i.e., it does not appear on the right-hand side of assertions of the form $R \sqsubseteq R'$ or $U \sqsubseteq U'$.*

Note that, as shown in [25], the last condition of the above definition is necessary for keeping query answering first-order rewritable (see later).

To complete the definition of the semantics of a *DL-Lite*$_{A,id,den}$ KB, we first define when an interpretation satisfies a TBox assertion, and then we extend the definition to satisfaction of ABox assertions. As usual, we denote by $\mathcal{I} \models \alpha$ the fact that $\mathcal{I}$ satisfies α, where α is either an ABox or a TBox assertion.

Concerning the satisfaction of TBox assertions, we analyze here the various cases:

- If α is a positive inclusion assertion of the form $B_1 \sqsubseteq B_2$, then $\mathcal{I} \models \alpha$ if $B_1^I \subseteq B_2^I$. The definition for the other positive inclusions is analogous.

- If α is a negative inclusion assertion of the form $B_1 \sqsubseteq \neg B_2$, then $\mathcal{I} \models \alpha$ if $B_1^I \cap B_2^I = \emptyset$. The definition for the other negative inclusions is analogous.

- If α is a role functionality assertion of the form (funct R), then $\mathcal{I} \models \alpha$ if for each $o_1, o_2, o_3 \in \Delta_O^I$ we have that $(o_1, o_2) \in R^I$ and $(o_1, o_3) \in R^I$ implies $o_2 = o_3$. The definition for attribute functionality assertions is analogous.

- If α is a denial assertion of the form $\forall y.conj(\vec{y})$, then $\mathcal{I} \models \alpha$ if the FOL sentence $\exists \vec{y}.conj(\vec{y})$ evaluates to false in $\mathcal{I}$ (i.e., is not satisfied by $\mathcal{I}$ in the standard FOL sense).

- If α is an identification assertion of the form $(id\ B\ \pi_1, \ldots, \pi_n)$, then $\mathcal{I} \models \alpha$ if for all $o, o' \in B^I$, $\pi_1^I(o) \cap \pi_1^I(o') \neq \emptyset \wedge \cdots \wedge \pi_n^I(o) \cap \pi_n^I(o') \neq \emptyset$ implies $o = o'$, where $\pi_i^I(o)$ denotes the set of π_i-fillers for o in $\mathcal{I}$, i.e., $\pi_i^I(o) = \{o' \mid (o, o') \in \pi_i^I\}$, with π^I defined as follows:
  - if $\pi = S$, then $\pi^I = S^I$,
  - if $\pi = D?$, then $\pi^I = \{(o, o) \mid o \in D^I\}$,
  - if $\pi = \pi_1 \circ \pi_2$, then $\pi^I = \pi_1^I \circ \pi_2^I$, where $\circ$ denotes the composition operator on relations.

An interpretation $\mathcal{I}$ satisfies an ABox assertion $A(a)$ if $a^I \in A^I$, an ABox assertion $P(a,b)$ if $(a^I, b^I) \in P^I$, and an ABox assertion $U(a,c)$ if $(a^I, val(c)) \in U^I$. A *model of a DL-Lite*$_{A,id,den}$ *KB* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an interpretation $\mathcal{I}$ that

satisfies all assertions in $\mathcal{T}$ and $\mathcal{A}$. We denote with $\mathcal{I} \models \mathcal{K}$ the fact that $\mathcal{I}$ is a model for $\mathcal{K}$, and with $Mod(\mathcal{K})$ the class of all models of $\mathcal{K}$. A KB is *satisfiable* if it has at least one model, i.e., $Mod(\mathcal{K}) \neq \emptyset$, *unsatisfiable* otherwise. For a KB, we also use the term *consistent* (resp. *inconsistent*) to mean satisfiable (resp. unsatisfiable). A set $\mathcal{A}'$ of ABox assertions is said to be $\mathcal{T}$*-consistent* if the KB $\langle \mathcal{T}, \mathcal{A}' \rangle$ is satisfiable, $\mathcal{T}$-inconsistent otherwise.

A KB $\mathcal{K}$ *entails* (or logically implies) a FOL sentence ϕ, and therefore a TBox or ABox assertion, written $\mathcal{K} \models \phi$, if all models of $\mathcal{K}$ are also models of ϕ. This notion naturally extends to a set of sentences. The *KB satisfiability* problem is defined as follows: given a DL KB $\mathcal{K}$, verify whether it is satisfiable.

Example 1. In this example, we present a TBox that models a (very small portion of) the network managed by a telecommunication company, extracted from an ontology we developed within a real-world experimentation [21]. In particular, our KB focuses on the connections between telecommunication devices (cf. concept *Device* in the ontology), which is realized by connecting device ports (*Port*). Each port belongs to (*of*) exactly one device. Among various kinds of ports, there are incoming ports (*PortIn*) and outgoing ports (*PortOut*), which are disjoint sets of ports. Each port is associated with a number (*number*), and there do not exist two ports of the same device with the same number. A port can be connected to (*connectedTo*) another port, according to the following rules:

(a) every port is connected to at most one other port;
(b) two ports of the same device cannot be connected to each other;
(c) there cannot exist an incoming port and an outgoing port of one device that are connected to ports of the same device.

The following *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$ captures our domain:

- the set $\mathcal{T}_{inc}$ consists of the following assertions:

  - $PortIn \sqsubseteq Port$
  - $PortOut \sqsubseteq Port$
  - $\exists connectedTo \sqsubseteq Port$
  - $\exists of \sqsubseteq Port$
  - $\exists connectedTo^{-} \sqsubseteq Port$
  - $\exists of^{-} \sqsubseteq Device$
  - $Port \sqsubseteq \delta(number)$
  - $Device \sqsubseteq \exists of^{-}$
  - $\delta(number) \sqsubseteq Port$
  - $Port \sqsubseteq \exists of$

- the set $\mathcal{T}_{type}$ consists of the following assertion:

  - $\rho(number) \sqsubseteq$ `xsd:integer`

- the set $\mathcal{T}_{disj}$ consists of the following assertions:

  - $PortIn \sqsubseteq \neg PortOut$
  - $Port \sqsubseteq \neg Device$

- the set $\mathcal{T}_{funct}$ consists of the following assertions:

– $(funct\ connectedTo)$ – $(funct\ of)$
– $(funct\ connectedTo^{-})$ – $(funct\ number)$

- the set $\mathcal{T}_{id}$ consists of the following assertion:

  – $(id\ Port\ number,\ of)$

- the set $\mathcal{T}_{den}$ consists of the following assertions:

  – $\forall x, y, z.Port(x) \wedge Port(y) \wedge of(x, z) \wedge of(y, z) \wedge connectedTo(x, y) \rightarrow \bot$
  – $\forall x, y, z, k, w, v.PortOut(x) \wedge of(x, y) \wedge connectedTo(x, z) \wedge of(z, k)$
  $\wedge\ PortIn(w) \wedge of(w, y) \wedge connectedTo(w, v) \wedge\ of(v, k) \rightarrow \bot$

Notice that the identification assertion models the fact that there do not exist two ports with the same number in the same device. Moreover, the functionality assertions on both the role *connectedTo* and its inverse encode the above rule (a), while the first denial assertion copes with the above rule (b), and the second denial assertion formalizes the above rule (c). □

2.2. DL-Lite$_{core}$

DL-Lite$_{core}$ is the least expressive member of the *DL-Lite* family, and its constructs are shared among all logics of the a family. In this sense, it can be seen as a fragment of *DL-Lite$_{A,id,den}$*, where denial, identification and functionality assertions are not allowed, and no distinction is made between roles and attributes. More precisely, concepts and roles in *DL-Lite$_{core}$* are formed according to the following syntax:

$$B \longrightarrow A \mid \exists R \qquad\qquad R \longrightarrow P \mid P^{-}$$

A *DL-Lite$_{core}$* TBox $\mathcal{T}$ is a finite set of assertions of the form

$$B_1 \sqsubseteq B_2 \qquad\qquad B_1 \sqsubseteq \neg B_2$$

that is, it is a finite set of positive and negative inclusions between basic concepts. A *DL-Lite$_{core}$* ABox has the same form of a *DL-Lite$_{A,id,den}$* ABox, and a *DL-Lite$_{core}$* knowledge base $\mathcal{K}$ is simply a pair $\langle \mathcal{T}, A \rangle$ where $\mathcal{T}$ and $\mathcal{A}$ are a TBox and ABox in *DL-Lite$_{core}$*, respectively.

The semantics of a *DL-Lite$_{core}$* KB coincides with that of *DL-Lite$_{A,id,den}$*, limited to the constructs and assertions allowed in *DL-Lite$_{core}$*. The notions of model of a KB, KB satisfiability, and entailment of a sentence by a KB are obviously the same as those given for *DL-Lite$_{A,id,den}$*.

2.3. Query Answering

A FOL query over a DL KB is a possibly open FOL formula over the KB alphabet. A *conjunctive query (CQ)* is a FOL query of the form $\exists \vec{y}.conj(\vec{x}, \vec{y})$, where $\vec{y}$ and $\vec{x}$ are disjoint sets of variables, called existential and free variables, respectively, and $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $A(z)$, $T_i(z')$, $P(z, z')$, $U(z, z')$, where z, z' are either constants or (possibly non-distinct) variables from $\vec{x}$ or $\vec{y}$. A *union of conjunctive queries (UCQ)*, is a FOL query of the form $\bigvee_{i=1}^{n} \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i})$ such that each $\exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i})$ is a conjunctive query. With a little abuse of notation, we will sometime treat a UBCQ (possibly enriched with inequalities) as a set of CQs.

A Boolean FOL query is a FOL query with no free variables, i.e., a FOL sentence. The notion of satisfaction of a Boolean query q in an interpretation $\mathcal{I}$ and the notion of entailment by a DL KB $\mathcal{K}$ are the usual ones: we write $\mathcal{I} \models q$ when q is satisfied in the interpretation $\mathcal{I}$, and $\mathcal{K} \models q$ when q is entailed by $\mathcal{K}$, i.e., q is satisfied in all models of $\mathcal{K}$. For a non-Boolean query q with free variables $\langle x_1, \ldots, x_n \rangle$, a tuple of constants $\langle a_1, \ldots, a_n \rangle$ is a *certain answer* to q over $\mathcal{K}$ if $\mathcal{K} \models q[x_1/a_1, \ldots, x_n/a_n]$, where $q[x_1/a_1, \ldots, x_n/a_n]$ is the Boolean query obtained by replacing each x_i in q with a_i.

Query answering for a non-Boolean CQ q over a KB $\mathcal{K}$ is the task of computing all certain answers to q over $\mathcal{K}$, and it is this task that we aim to study in this paper. As we have just noticed, this task can be straightforwardly reduced to entailment of Boolean conjunctive queries. Thus, for ease of exposition, and as usually done in the studies of query answering over KBs (see, e.g., [35]), we consider in the following only Boolean conjunctive queries (BCQs) and unions of Boolean conjunctive queries (UBCQs), and we define the *query answering* reasoning service as follows: given a DL KB $\mathcal{K}$ and a Boolean query q (either a BCQ or a UBCQ) over $\mathcal{K}$, verify whether $\mathcal{K} \models q$. A simplified form of query answering is *instance checking*, defined as follows: given a DL KB $\mathcal{K}$ and an ABox assertion α, verify whether $\mathcal{K} \models \alpha$. We notice that, according to what we have said above, despite the fact that we limit our investigation to Boolean queries, all the results we achieve on query answering over a DL KB can be easily extended in the standard way to the presence of free variables in queries.

In this paper we are interested in the so-called *data complexity* of query answering (and of KB satisfiability, as well), which is a notion borrowed from the database literature [73]. According to data complexity, both the TBox and the query are not considered as inputs to the query answering problem, and the complexity is therefore measured with respect to the size of the ABox only. This complexity measure is of particular interest in all those cases where the size of the intensional level of the KB (i.e., the TBox) is negligible with respect to the size of the data (i.e., the ABox), as in ontology-based data access [64].

We now introduce some notions related to query answering that will be used in the rest of the paper. In particular, the following definition provides a database-like interpretation of an ABox.

Definition 2. *Let $\mathcal{A}$ be a set of ABox assertions. The interpretation $DB(\mathcal{A}) = \langle \Delta^{DB(\mathcal{A})}, \cdot^{DB(\mathcal{A})} \rangle$ is defined as follows:*

– $\Delta^{DB(\mathcal{A})} = \Delta_O^{DB(\mathcal{A})} \cup \Delta_V$, *where* $\Delta_O^{DB(\mathcal{A})}$ *is the set of all object constants occurring in* $\mathcal{A}$,
– $T_i^{DB(\mathcal{A})} = val(T_i)$ *for each value-domain* $T_i \in \{T_1, \ldots, T_n\}$, *and* $c^{DB(\mathcal{A})} = val(c)$ *for each constant* $c \in \Gamma_V$,
– $a^{DB(\mathcal{A})} = a$, *for each object constant* a *occurring in* $\mathcal{A}$,
– $A^{DB(\mathcal{A})} = \{a \mid A(a) \in \mathcal{A}\}$, *for each atomic concept* A,
– $P^{DB(\mathcal{A})} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{A}\}$, *for each atomic role* P, *and*
– $U^{DB(\mathcal{A})} = \{(a, val(c)) \mid U(a, c) \in \mathcal{A}\}$, *for each attribute* U.

The notion of *image* of a Boolean conjunctive query in an ABox is given as follows. For a BCQ $q = \exists \vec{y}.\, conj(\vec{y})$, we denote with $conj\text{-}set(\vec{y})$ the set of atoms occurring in the conjunction $conj(\vec{y})$. Given an ABox $\mathcal{A}$, an *image of* q *in* $\mathcal{A}$ is a set $\mathcal{G} \subseteq \mathcal{A}$ such that there exists a substitution σ from the variables $\vec{y}$ in $conj\text{-}set(\vec{y})$ to constants in $\mathcal{G}$ such that the set of atoms in $\sigma(conj\text{-}set(\vec{y}))$ is equal to $\mathcal{G}$ and the formula $\sigma(conj(\vec{y}))$ evaluates to *true* in $DB(\mathcal{G})$, i.e., $DB(\mathcal{G}) \models \sigma(conj(\vec{y}))$. Obviously, we have that $DB(\mathcal{A}) \models q$ if and only if there exists an image of q in $\mathcal{A}$. We use $\mathsf{images}(q, \mathcal{A})$ to denote the set of all images of q in $\mathcal{A}$. The notion of image naturally extends to BCQ with inequalities. More precisely, if $q' = \exists \vec{y}.\, conj(\vec{y}) \wedge \phi_{\neq}(\vec{y'})$, where variables in $\vec{y'}$ occur also in $\vec{y}$ and $\phi_{\neq}(\vec{y'})$ contains all inequality predicates of q', and $\mathcal{G}$, $conj\text{-}set(\vec{y})$, and σ are as above, $\mathcal{G}$ is an image of q' in $\mathcal{A}$ if $DB(\mathcal{G}) \models \sigma(conj(\vec{y}) \wedge \phi_{\neq}(\vec{y'})))$.

2.4. The notion of FO-rewritability

In this paper we study the crucial property of *first-order rewritability (FO-rewritability)* for both KB satisfiability and query answering. Intuitively, FO-rewritability of KB satisfiability (resp., query answering) captures the property that we can reduce satisfiability checking (resp., query answering) to evaluating a FOL query over the ABox $\mathcal{A}$ considered as a relational database, i.e., over $DB(\mathcal{A})$. The definitions follow.

Definition 3. *KB satisfiability in a DL* $\mathcal{L}$ *is* FO-rewritable *if, for every TBox* $\mathcal{T}$ *expressed in* $\mathcal{L}$, *one can effectively compute a Boolean FOL query* q_s *over* $\mathcal{T}$ *such that, for every ABox* $\mathcal{A}$, *the KB* $\langle \mathcal{T}, \mathcal{A} \rangle$ *is satisfiable if and only if* $DB(\mathcal{A}) \not\models q_s$.

Definition 4. *Query answering in a DL* $\mathcal{L}$ *is* FO-rewritable, *if for every TBox* $\mathcal{T}$ *expressed in* $\mathcal{L}$ *and every query* q *over* $\mathcal{T}$, *one can effectively compute a FOL query* q_r *over* $\mathcal{T}$ *such that for every ABox* $\mathcal{A}$ *for which* $\langle \mathcal{T}, \mathcal{A} \rangle$ *is satisfiable, we have that* $\langle \mathcal{T}, \mathcal{A} \rangle \models q$ *if and only if* $DB(\mathcal{A}) \models q_r$. *We call such* q_r *a* perfect FOL rewriting *(or simply* perfect rewriting*) of* q *w.r.t.* $\mathcal{T}$.

We remark that the FOL query considered in the above definitions depends only on the TBox (and the query), but not on the ABox. Since the evaluation of a FOL query over an ABox is in AC^0 in data complexity [1], we can state that, both for KB satisfiability and query answering, the property of FO-rewritability implies that the problem is in AC^0 in data complexity.

FO-rewritability of query answering for various logics of the *DL-Lite* family has been shown in [22, 20]. In these papers, the algorithm PerfectRef has been

presented, which takes as input a *DL-Lite*$_{A,id}$ TBox $\mathcal{T}$ and a UCQs Q and returns a perfect rewriting of Q w.r.t. $\mathcal{T}$. In this paper, we consider the case in which PerfectRef takes as input a *DL-Lite*$_{A,id,den}$ TBox, and acts exactly as in [22, 20]. Then, the following proposition immediately follows from the analogous result given in [20] for *DL-Lite*$_{A,id}$.

Proposition 1. *If $\mathcal{T}$ be a DL-Lite$_{A,id,den}$ TBox, and Q is a UCQs over $\mathcal{T}$, then PerfectRef$(Q, \mathcal{T})$ is a perfect rewriting of Q w.r.t. $\mathcal{T}$.*

Instances of execution of PerfectRef can be found in Example 3, whereas the description of PerfectRef is recalled in Appendix 2.

3. Satisfiability of *DL-Lite*$_{A,id,den}$ KBs

We now deal with KB satisfiability in *DL-Lite*$_{A,id,den}$ and show that this problem is FO-rewritable (cf. Definition 3), and thus in AC^0 in data complexity.

We first notice that FO-rewritability of the satisfiability check for *DL-Lite*$_{A,id}$, i.e., *DL-Lite*$_{A,id,den}$ without denial assertions, has been already claimed in [20]. In that paper, however, the role of assertions of the set $\mathcal{T}_{type}$ has been overlooked, and therefore the algorithm for KB satisfiability given in [20] does not identify inconsistencies caused by the value-domain inclusions of the TBox. Notice that such inconsistencies may arise from the fact that the extensions of value-domains are pairwise disjoint, as shown in the following example.

Example 2. The TBox assertion $\rho(U) \sqsubseteq T_i$ implies that $\rho(U)$ is disjoint from every value-domain distinct from T_i. Thus, the ABox assertion $U(a, v)$ such that $v \in \Gamma_V$ and $val(v) \in val(T_j)$, with $T_j \neq T_i$, is inconsistent with the above TBox assertion, and the KB $\langle\{\rho(U) \sqsubseteq T_i\}, \{U(a, v)\}\rangle$ is unsatisfiable. As a further example, consider the KB $\langle\{\rho(U_1) \sqsubseteq T_i, \rho(U_2) \sqsubseteq T_j, U_1 \sqsubseteq U_2, A \sqsubseteq \delta(U_1)\}, \{A(d)\}\rangle$. The TBox implies that U_1 has an empty interpretation in every TBox model, and therefore the above KB is unsatisfiable. □

Our algorithm for KB satisfiability makes use of a function φ that associates (unions of) Boolean conjunctive queries with inequalities to assertions in $\mathcal{T}_{type} \cup \mathcal{T}_{disj} \cup \mathcal{T}_{funct} \cup \mathcal{T}_{id} \cup \mathcal{T}_{den}$. Such a function, which extends to value-domain inclusions and denial assertions an analogous function given in [20], is defined as follows.

- $\varphi((\mathsf{funct}\ P))$: $\exists x, x_1, x_2.P(x, x_1) \wedge P(x, x_2) \wedge x_1 \neq x_2$
- $\varphi((\mathsf{funct}\ P^-))$: $\exists x, x_1, x_2.P(x_1, x) \wedge P(x_2, x) \wedge x_1 \neq x_2$
- $\varphi((\mathsf{funct}\ U))$: $\exists x, x_1, x_2.U(x, x_1) \wedge U(x, x_2) \wedge x_1 \neq x_2$
- $\varphi(B_1 \sqsubseteq \neg B_2)$: $\exists x.\gamma(B_1, x) \wedge \gamma(B_2, x)$
- $\varphi(R_1 \sqsubseteq \neg R_2)$: $\exists x_1, x_2.\eta(R_1, x_1, x_2) \wedge \eta(R_2, x_1, x_2)$
- $\varphi(U_1 \sqsubseteq \neg U_2)$: $\exists x_1, x_2.U_1(x_1, x_2) \wedge U_2(x_1, x_2)$
- $\varphi(\rho(U) \sqsubseteq T_i)$: $\bigvee_{j \in \{1,\ldots,n\} \wedge j \neq i} \exists x_1, x_2.U(x_1, x_2) \wedge T_j(x_2)$
- $\varphi(\forall \vec{x}.conj(\vec{x}) \rightarrow \bot)$: $\exists \vec{x}.conj(\vec{x})$

- $\varphi((id\ B\ \pi_1, \ldots, \pi_m))\ :\ \exists x, x'.\gamma(B, x) \wedge \gamma(B, x') \wedge x \neq x' \wedge \bigwedge_{1 \leq i \leq m} \exists x_i.\xi(\pi_i, x, x_i) \wedge \xi(\pi_i, x', x_i)$

In the above definition, x, x', $x_1, x_2, \ldots, x_m$ are variables and $\vec{x}$ is a sequence of variables, and γ and ξ are two functions, that we now define. In what follows, (y_{new} denotes fresh variable symbol, i.e., a variable symbol not occurring elsewhere in the query. The function γ takes care of atoms built on basic concepts, whereas the function η takes care of atoms built on roles:

$$\gamma(B, x) = \begin{cases} A(x) & \text{if } B = A, \\ \exists y_{new}.P(x, y_{new}) & \text{if } B = \exists P, \\ \exists y_{new}.P(y_{new}, x) & \text{if } B = \exists P^-, \\ \exists y_{new}.U(x, y_{new}) & \text{if } B = \delta(U) \end{cases} \quad \text{and}$$

$$\eta(R, x_1, x_2) = \begin{cases} P(x_1, x_2) & \text{if } R = P, \\ P(x_2, x_1) & \text{if } R = P^-. \end{cases}$$

The function ξ takes care of paths π in identification assertions, and is defined inductively on the structure of π.

- if $\pi = R$, then $\xi(\pi, x_1, x_2) = \eta(R, x_1, x_2)$,
- if $\pi = U$, then $\xi(\pi, x_1, x_2) = U(x_1, x_2)$,
- if $\pi = D?$, then $\xi(\pi, x_1, x_2) = D(x_1)$,
- if $\pi = \pi_1 \circ \pi_2$, and $\pi_1 = D?$, then $\xi(\pi, x_1, x_2) = D(x_1) \wedge \xi(\pi_2, x_1, x_2)$,
- if $\pi = \pi_1 \circ \pi_2$, and π_1 is not of the form $D?$, then $\xi(\pi, x_1, x_2) = \exists y_{new}.\xi(\pi_1, x_1, y_{new}) \wedge \xi(\pi_2, y_{new}, x_2)$.

Intuitively, if α is an assertion in $\mathcal{T}_{disj} \cup \mathcal{T}_{funct} \cup \mathcal{T}_{id} \cup \mathcal{T}_{den}$, then the query $\varphi(\alpha)$ encodes the negation of α, i.e., searches for violations of α. For example, if α is the identification assertion (id *Port number, of*), then $\varphi(\alpha) = \exists x, x', x_1, x_2.\mathit{Port}(x) \wedge \mathit{Port}(x') \wedge x \neq x' \wedge \mathit{number}(x, x_1) \wedge \mathit{number}(x', x_1) \wedge \mathit{of}(x, x_2) \wedge \mathit{of}(x', x_2)$. Similarly, if $\alpha = \rho(U) \sqsubseteq T_i$, i.e., $\alpha \in \mathcal{T}_{type}$, then $\varphi(\alpha)$ encodes the negation of all disjointnesses between the range of U and each $T_j \neq T_i$. We are now ready to present the algorithm UnsatQuery (Algorithm 1), which

Input: *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$
Output: UBCQ with inequalities
begin
  return $\bigvee_{\alpha \in \mathcal{T} \setminus \mathcal{T}_{inc}} \mathsf{PerfectRefIdC}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$
end

Algorithm 1: UnsatQuery

takes as input a *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$, and returns the FOL query (more precisely the UBCQ with inequalities) that we evaluate over an ABox $\mathcal{A}$, whenever we want to verify the satisfiability of the KBs $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$.

The algorithm UnsatQuery computes the perfect rewriting with respect to the positive knowledge in $\mathcal{T}$, i.e., $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$, of each query associated by the function φ to the assertions in $\mathcal{T}_{disj} \cup \mathcal{T}_{den} \cup \mathcal{T}_{type} \cup \mathcal{T}_{funct} \cup \mathcal{T}_{id}$. By taking the union of all such perfect rewritings, UnsatQuery($\mathcal{T}$) encodes the negation of all disjointnesses, functionalities, identifications, and denials inferred by (and not only asserted in) the TBox $\mathcal{T}$. To compute perfect rewritings, UnsatQuery makes use of the algorithm PerfectRefIdC given in [20], which rewrites CQs with (limited forms of) inequalities, like those that φ associates to identification assertions and functionalities. This algorithm is a variant of PerfectRef (algorithm 6). More precisely, rewriting steps in PerfectRefIdC are exactly as in PerfectRef, with the proviso that inequality is treated as an atomic role and inequality atoms are never rewritten, since no assertions in the TBox involve the inequality predicate. Moreover, PerfectRefIdC does not unify query atoms if the variables involved in the unification occur also in inequality atoms. Obviously, if q is a CQ without inequalities, then PerfectRefIdC$(q, \mathcal{T})$ = PerfectRef$(q, \mathcal{T})$. Observe that the query returned by PerfectRefIdC is still a UBCQ with inequalities over the alphabet of $\mathcal{T}$.

Termination of the algorithm UnsatQuery follows from termination of PerfectRefIdC, which is shown in [20]. The property of UnsatQuery established by the following theorem states that satisfiability in *DL-Lite*$_{A,id,den}$ is FO-rewritable. The proof of the theorem is given in Appendix 2.

Theorem 1. *A DL-Lite*$_{A,id,den}$ *KB* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ *is unsatisfiable if and only if* $DB(\mathcal{A}) \models$ *UnsatQuery*$(\mathcal{T})$.

The following result is then a direct consequence of the above theorem and of Proposition 1.

Proposition 2. *KB satisfiability and query answering in DL-Lite*$_{A,id,den}$ *are FO-rewritable, and are therefore in* AC^0 *in data complexity.*

An easy consequence of Theorem 1 is that a *DL-Lite*$_{A,id,den}$ KB of the form $\langle \mathcal{T}, \emptyset \rangle$ is always consistent. Furthermore, a $\mathcal{T}$-inconsistent set $V \subseteq \mathcal{A}$ exists in $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ if and only if $DB(\mathcal{A}) \models$ UnsatQuery$(\mathcal{T})$. This property leads us to formalize the notion of $\mathcal{K}$-clash, which will be useful in the following.

Definition 5. *If* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ *is a DL-Lite*$_{A,id,den}$ *KB, then a set of ABox assertions* $V \subseteq \mathcal{A}$ *is a* $\mathcal{K}$-clash *if there exists a* $q \in$ *UnsatQuery*$(\mathcal{T})$ *such that* $V \in$ *images*$(q, \mathcal{A})$.

We conclude this section with an example illustrating the whole procedure for checking satisfiability of a *DL-Lite*$_{A,id,den}$ KB.

Example 3. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_{A,id,den}$-KB, where $\mathcal{T}$ is the TBox of Example 1, and $\mathcal{A}$ is the ABox formed by the following assertions:

$$PortIn(p_1), \quad PortOut(p_1), \quad connectedTo(p_1, p_2).$$

In words, $\mathcal{A}$ states that p_1 is both an incoming and an outgoing port, and that p_1 is connected to p_2. It is immediate to verify that $\mathcal{K}$ is inconsistent, since the

assertion $PortIn \sqsubseteq \neg PortOut$ in $\mathcal{T}$ is contradicted by the assertions $PortIn(p_1)$ and $PortOut(p_1)$.

For this example, let us focus only on the following two assertions in $\mathcal{T} \setminus \mathcal{T}_{inc}$.

$$\begin{aligned}\alpha_1 &= PortIn \sqsubseteq \neg PortOut;\\ \alpha_2 &= \forall x, y, z, k, w, v.PortOut(x) \wedge of(x, y) \wedge connectedTo(x, z) \wedge of(z, k)\\ &\qquad \wedge PortIn(w) \wedge of(w, y) \wedge connectedTo(w, v) \wedge\ of(v, k) \rightarrow \bot.\end{aligned}$$

The queries associated to α_1 and α_2 by the function φ are:

$$\begin{aligned}\varphi(\alpha_1) &= \exists x.PortIn(x) \wedge PortOut(x);\\ \varphi(\alpha_2) &= \exists x, y, z, k, w, v.PortOut(x) \wedge of(x, y) \wedge connectedTo(x, z) \wedge of(z, k)\\ &\qquad \wedge PortIn(w) \wedge of(w, y) \wedge connectedTo(w, v) \wedge\ of(v, k).\end{aligned}$$

$\mathsf{UnsatQuery}(\mathcal{T})$ contains, among others, the following queries:

$$\begin{aligned}q_1 &= \exists x, y, z, k, w, v.PortOut(x) \wedge of(x, y) \wedge connectedTo(x, z) \wedge\\ &\quad of(z, k) \wedge PortIn(w) \wedge of(w, y) \wedge connectedTo(w, v) \wedge of(v, k);\\ q_2 &= \exists x, y, z, w.PortOut(x) \wedge of(x, y) \wedge connectedTo(x, z) \wedge\\ &\quad PortIn(w) \wedge of(w, y) \wedge connectedTo(w, z);\\ q_3 &= \exists x, y.PortIn(x) \wedge PortOut(x) \wedge connectedTo(x, z);\\ q_4 &= \exists x.PortIn(x) \wedge PortOut(x).\end{aligned}$$

The first three queries come from the rewriting of $\varphi(\alpha_2)$. In particular,

- q_1 coincides with $\varphi(\alpha_2)$.
- q_2 is obtained from q_1 by various iterations of the algorithm $\mathsf{PerfectRef}$, which proceeds in this way[4]: (i) it first unifies atoms $of(z, k)$ and $of(v, k)$, and thus substitutes v with z throughout the query, and k with $\_$, since in the resulting atom k is unbound; (ii) then, it rewrites the atom $of(z, \_)$ through the assertions $Port \sqsubseteq \exists of$, thus obtaining the atom $Port(z)$; (iii) after, it rewrites $Port(z)$ into $connectedTo(\_, z)$, where $\_$ denotes a new unshared existentially quantified variable in the query, by applying the assertion $\exists connectedTo^- \sqsubseteq Port$; (iv) finally, it unifies $connectedTo(\_, z)$ with $connectedTo(w, z)$, thus returning q_2.
- q_3 is obtained from q_2 by first unifying atoms $of(x, y)$ and $of(w, y)$, and then applying $Port \sqsubseteq \exists of$ (as in step (ii) above), and subsequently $PortOut \sqsubseteq Port$, thus returning q_3.

The query q_4 coincides instead with $\varphi(\alpha_1)$. Such a query is not touched by the algorithm $\mathsf{PerfectRef}$, since no inclusion assertions exist in $\mathcal{T}_{inc}$ having $PortIn$ or $PortOut$ in their right-hand side, and no unifications can be performed on it.

[4] In fact, q_2 can be obtained from q_1 in various ways, and we just describe here one of such possible options.

It is easy to verify that, for the queries considered in this example, we have:

$$DB(\mathcal{A}) \not\models q_1; \qquad DB(\mathcal{A}) \not\models q_2;$$
$$DB(\mathcal{A}) \models q_3; \qquad DB(\mathcal{A}) \models q_4.$$

Hence, we conclude that the KB $\mathcal{K}$ is inconsistent since the assertions in $\mathcal{A}$ violate under $\mathcal{T}$ both the negative inclusion α_1, witnessed by $DB(\mathcal{A}) \models q_4$, and the denial α_2 witnessed by $DB(\mathcal{A}) \models q_3$. Finally, the following sets are two $\mathcal{K}$-clashes:

$\{\ \mathit{PortIn}(p_1),\ \mathit{PortOut}(p_1)\ \}$
$\{\ \mathit{PortIn}(p_1),\ \mathit{PortOut}(p_1),\ \mathit{connectedTo}(p_1, p_2)\ \}$

□

4. The ABox Repair semantics

In this section we present our first proposal of inconsistency-tolerant semantics for DL knowledge bases. Our aim here is to allow a DL KB $\mathcal{K}$ to be interpreted with a non-empty set of models even in the case where $\mathcal{K}$ is inconsistent under the classical FOL semantics.

As already said, the inconsistency-tolerant semantics we propose is based on the notion of *repair*, borrowed from the database literature [26, 8]. Intuitively, given a possibly inconsistent DL KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, a repair $\mathcal{A}_r$ for $\mathcal{K}$ is an ABox such that the KB $\langle \mathcal{T}, \mathcal{A}_r \rangle$ is consistent under the FOL semantics, and $\mathcal{A}_r$ "minimally" differs from $\mathcal{A}$. Thus a repair is a $\mathcal{T}$-consistent ABox which "is as close as possible" to $\mathcal{A}$. Different notions of "'minimality" may give rise to different inconsistency-tolerant semantics. Here, we consider as repairs of $\mathcal{K}$ the $\mathcal{T}$-consistent ABoxes that can be obtained by eliminating from $\mathcal{A}$ as few assertions as possible in order to gain consistency. In other terms, a repair is an inclusion-maximal $\mathcal{T}$-consistent subset of $\mathcal{A}$, as formalized in the following definition.

Definition 6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{A,id,den}$ KB. An* ABox repair (*AR*-repair) *of $\mathcal{K}$ is a set $\mathcal{A}'$ of ABox assertions such that:*

- $\mathcal{A}' \subseteq \mathcal{A}$,
- $Mod(\langle \mathcal{T}, \mathcal{A}' \rangle) \neq \emptyset$,
- *no $\mathcal{A}''$ exists such that $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$, and $Mod(\langle \mathcal{T}, \mathcal{A}'' \rangle) \neq \emptyset$.*

It is easy to see that more than one *AR*-repair of a KB $\mathcal{K}$ may exist. Moreover, there is always a finite number of *AR*-repairs, and each *AR*-repair is finite, since $\mathcal{A}$ is finite. In what follows, we denote by *AR-Set*($\mathcal{K}$) the set of *AR*-repairs of $\mathcal{K}$.

We now present an example illustrating the notion of *AR*-repair.

Example 4. Consider $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the TBox from Example 1, and $\mathcal{A}$ is the ABox formed by the following assertions:

$$\mathit{PortIn}(p_1), \quad \mathit{PortOut}(p_1), \quad \mathit{connectedTo}(p_1, p_2), \quad \mathit{of}(p_1, p_2), \quad \mathit{Device}(d).$$

It is easy to verify that the ABox $\mathcal{A}$ is $\mathcal{T}$-inconsistent, as it contains the following $\mathcal{K}$-clashes (cf. Definition 5):

$V_1 = \{\ \mathit{PortIn}(p_1),\ \mathit{PortOut}(p_1)\ \}$
$V_2 = \{\ \mathit{connectedTo}(p_1, p_2),\ \mathit{of}(p_1, p_2)\ \}$
$V_3 = \{\ \mathit{PortIn}(p_1),\ \mathit{PortOut}(p_1),\ \mathit{connectedTo}(p_1, p_2)\ \}$

where: V_1 and V_3 are the $\mathcal{K}$-clashes of Example 3, and V_2 is obtained from the query $\varphi(\mathit{Port} \sqsubseteq \neg \mathit{Device})$.

According to Definition 6, the set $\mathit{AR\text{-}Set}(\mathcal{K})$ consists of the following $\mathcal{T}$-consistent sets of ABox assertions:

$\mathit{AR\text{-}rep}_1 = \{\ \mathit{PortIn}(p_1), \mathit{connectedTo}(p_1, p_2), \mathit{Device}(d)\ \}$
$\mathit{AR\text{-}rep}_2 = \{\ \mathit{PortOut}(p_1), \mathit{connectedTo}(p_1, p_2), \mathit{Device}(d)\ \}$
$\mathit{AR\text{-}rep}_3 = \{\ \mathit{PortIn}(p_1), \mathit{of}(p_1, p_2), \mathit{Device}(d)\ \}$
$\mathit{AR\text{-}rep}_4 = \{\ \mathit{PortOut}(p_1), \mathit{of}(p_1, p_2), \mathit{Device}(d)\ \}$

□

The following proposition immediately follows from the fact that $\mathit{Mod}(\langle \mathcal{T}, \emptyset \rangle) \neq \emptyset$ for any *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$ (cf. Theorem 1). This implies that the set of (inclusion-maximal) $\mathcal{T}$-consistent subsets of any ABox is always non-empty.

Proposition 3. *If $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a (possibly inconsistent) DL-Lite$_{A,id,den}$ KB, then $\mathit{AR\text{-}Set}(\mathcal{K}) \neq \emptyset$.*

With the notion of AR-repair in place, we can present the *ABox Repair* semantics (AR-semantics).

Definition 7. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB. The set of* ABox Repair Models*, or simply AR-*models*, of $\mathcal{K}$, denoted $\mathit{Mod}_{AR}(\mathcal{K})$, is defined as follows:*

$$\mathit{Mod}_{AR}(\mathcal{K}) = \{\mathcal{I} \mid \mathcal{I} \in \mathit{Mod}(\langle \mathcal{T}, \mathcal{A}_i \rangle),\ \textit{for some } \mathcal{A}_i \in \mathit{AR\text{-}Set}(\mathcal{K})\}$$

The next proposition easily follows from Definition 6 and Definition 7.

Proposition 4. *If $\langle \mathcal{T}, \mathcal{A} \rangle$ is a consistent DL-Lite$_{A,id,den}$ KB, then $\mathit{Mod}_{AR}(\langle \mathcal{T}, \mathcal{A} \rangle) = \mathit{Mod}(\langle \mathcal{T}, \mathcal{A} \rangle)$.*

We notice that the AR-semantics coincides with the loosely-sound semantics studied in [18] in the context of inconsistent and incomplete databases.

The following notion of consistent entailment is the natural generalization of classical entailment to the AR-semantics.

Definition 8. *Let $\mathcal{K}$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB, and let ϕ be a first-order sentence. We say that ϕ is AR-*consistently entailed*, or simply AR-*entailed*, by $\mathcal{K}$, written $\mathcal{K} \models_{AR} \phi$, if $\mathcal{I} \models \phi$ for every $\mathcal{I} \in Mod_{AR}(\mathcal{K})$.*

In other words, we say that $\mathcal{K} \models_{AR} \phi$ if for every AR-repair $\mathcal{A}_r \in AR\text{-}Set(\mathcal{K})$, the consistent KB $\langle \mathcal{T}, \mathcal{A}_r \rangle$ entails ϕ.

Example 5. Consider the DL KB $K = \langle \mathcal{T}, \mathcal{A} \rangle$ presented in Example 4, and the following BCQs:

$$\begin{array}{ll} q_1: & \exists x \mathit{Port}(p_1) \wedge \mathit{of}(p_1, x) \wedge \mathit{Device}(x); \\ q_2: & \exists x \mathit{Port}(x) \wedge \mathit{of}(x, d) \wedge \mathit{Device}(d). \end{array}$$

q_1 asks for the existence of a device to which port p_1 belongs, and q_2 asks for the existence of a port which belongs to the device d. By looking at the set $AR\text{-}Set(\mathcal{K})$ presented in Example 4, one can easily verify that $\mathcal{K}$ AR-entails both q_1 and q_2. □

We next introduce the notion of minimal inconsistent set and provide a theorem characterizing AR-entailment.

Definition 9. *Let $\langle \mathcal{T}, V \rangle$ be a DL-Lite$_{A,id,den}$ KB. We say that V is minimal $\mathcal{T}$-inconsistent if V is $\mathcal{T}$-inconsistent, and there is no proper subset V' of V such that V' is $\mathcal{T}$-inconsistent.*

In other words, the above definition says that for each assertion α in a minimal $\mathcal{T}$-inconsistent set V, the ABox $V \setminus \{\alpha\}$ is $\mathcal{T}$-consistent. Given a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, we denote by $minIncSets(\mathcal{K})$ the set of minimal $\mathcal{T}$-inconsistent sets contained in $\mathcal{A}$.

It is worth noticing that minimal inconsistent sets correspond to justifications (also known as minAs) [39, 62, 63, 6]: in particular, a minimal $\mathcal{T}$-inconsistent sets contained in $\mathcal{A}$ corresponds to an explanation (or minimal justification) of the inconsistency of $\langle \mathcal{T}, \mathcal{A} \rangle$ at the extensional level. We will further analyze this aspect in Section 9.

Theorem 2. *If $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a possibly inconsistent DL-Lite$_{A,id,den}$ KB, and $\alpha \in \mathcal{A}$, then there exists an AR-repair $\mathcal{A}'$ of $\mathcal{K}$ such that $\alpha \notin \mathcal{A}'$ if and only if there exists $V \in minIncSets(\mathcal{K})$ such that $\alpha \in V$.*

Proof. ($\Rightarrow$) From Definition 6 it follows that $\mathcal{A}' \cup \{\alpha\}$ is $\mathcal{T}$-inconsistent. If it is also a minimal $\mathcal{T}$-inconsistent set, then the claim is directly proved. Otherwise, there must exist $V' \subset \mathcal{A}' \cup \{\alpha\}$ that is minimal $\mathcal{T}$-inconsistent. Since $\mathcal{A}'$ does not contain α and obviously does not contain any $\mathcal{T}$-inconsistent set, it follows that V' contains α, and therefore the claim is shown also in this case.

($\Leftarrow$) Towards a contradiction, suppose that every AR-repair of $\mathcal{K}$ contains α. Since $V \in minIncSets(\mathcal{K})$, we have that $V \setminus \{\alpha\}$ is $\mathcal{T}$-consistent. We have two possible cases: (*i*) $V \setminus \{\alpha\}$ is an AR-repair, but this contradicts the assumption above; or, (*ii*) there exists an AR-repair $\mathcal{A}''$ such that $V \setminus \{\alpha\} \subset \mathcal{A}''$,

but $\mathcal{A}''$ contains α by the hypothesis, which means that $\mathcal{A}'' \supset V$, which is a contradiction. ∎

We point out that, even though in the above definitions, properties, and theorems we refer to *DL-Lite*$_{A,id,den}$, the results of this section apply also to more general languages. More precisely, Definitions 6–9, Proposition 3, and Theorem 2 (limited to KBs with satisfiable TBoxes), as well as Proposition 4 apply also to KBs specified in a different DL $\mathcal{L}$, provided that the semantics for $\mathcal{L}$ adopts the unique name assumption.

5. Query answering in *DL-Lite*$_{A,id,den}$ under the *AR*-semantics

In this section we deal with the problem of answering UBCQs posed to *DL-Lite*$_{A,id,den}$ KBs under the *AR*-semantics. We first notice that from the results presented in [48, Theorem 1 and Theorem 2], we know that UBCQ entailment is coNP-complete in data complexity under the *AR*-semantics for *DL-Lite*$_R$ and *DL-Lite*$_F$, two DLs of the *DL-Lite* family which extend *DL-Lite*$_{core}$ respectively with role hierarchies and functionality assertions [22]. Both such DLs are indeed subsumed by *DL-Lite*$_{A,id,den}$, and therefore the lower bound for data complexity given in [48] applies also to *DL-Lite*$_{A,id,den}$.

Here, we strengthen this result, and show that instance checking, i.e., answering single-atom ground queries under *AR*-semantics is already coNP-hard in data complexity even if the KB is expressed in *DL-Lite*$_{core}$, i.e., the least expressive logic of the *DL-Lite* family.

Theorem 3. *Let $\mathcal{K}$ be a DL-Lite$_{core}$ KB and let α be an ABox assertion. Deciding whether $\mathcal{K} \models_{AR} \alpha$ is coNP-hard with respect to data complexity.*

Proof. We exhibit a reduction from unsatisfiability of a 3-CNF to CQ entailment in *DL-Lite*$_{core}$ under *AR*-semantics.

Let ϕ be a 3-CNF of the form $c_1 \wedge \ldots \wedge c_n$ with $c_i = l_i^1 \vee l_i^2 \vee l_i^3$, where every l_i^j is a literal from a set of propositional variables $\{x_1, \ldots, x_m\}$. Given a literal ℓ, let $s(\ell)$ denote the sign of ℓ, i.e., $s(\ell) = t$ if ℓ is a positive literal, and $s(\ell) = f$ otherwise; moreover, let $v(\ell)$ denote the propositional variable occurring in the literal ℓ.

We define the following *DL-Lite*$_{core}$ TBox $\mathcal{T}$:

$$\mathcal{T} = \{\exists R \sqsubseteq \mathit{Unsat}, \quad \exists R^- \sqsubseteq \neg\exists L_t^-, \quad \exists R^- \sqsubseteq \neg\exists L_f^-, \quad \exists L_t \sqsubseteq \neg\exists L_f\}$$

and the following ABox $\mathcal{A}$:

$$\mathcal{A} = \{R(a, c_i) \mid 1 \leq i \leq n\} \cup \{L_{s(l_i^j)}(v(l_i^j), c_i) \mid 1 \leq i \leq n,\ 1 \leq j \leq 3\}$$

We now prove that $\langle \mathcal{T}, \mathcal{A} \rangle \models_{AR} \mathit{Unsat}(a)$ if and only if ϕ is unsatisfiable.

First, if ϕ is satisfiable, then there exists an interpretation J for $\{x_1, \ldots, x_m\}$ such that J is a model for ϕ. Now consider the ABox

$$\begin{array}{rcl} \mathcal{A}' & = & \{L_t(x_j, c_i) \mid L_t(x_j, c_i) \in \mathcal{A} \text{ and } J(x_j) = \mathit{true}\} \cup \\ & & \{L_f(x_j, c_i) \mid L_f(x_j, c_i) \in \mathcal{A} \text{ and } J(x_j) = \mathit{false}\} \end{array}$$

It is immediate to see that $\langle \mathcal{T}, \mathcal{A}' \rangle$ is satisfiable. Moreover, since J is a model for ϕ, for every conjunct c_i of ϕ, there exists a propositional variable x_j such that either the literal x_j occurs positively in c_i and $J(x_j) = \mathit{true}$, or x_j occurs negatively in c_i and $J(x_j) = \mathit{false}$. This implies that, for every $1 \leq i \leq n$, $\mathcal{A}' \cup \{R(a, c_i)\}$ is $\mathcal{T}$-inconsistent. Then, due to the presence of $\exists L_t \sqsubseteq \neg \exists L_f$ in $\mathcal{T}$, it immediately follows that, for every assertion α of the form $L_t(x_j, c_i)$ or $L_f(x_j, c_i)$ such that $\alpha \in \mathcal{A} \setminus \mathcal{A}'$, $\mathcal{A}' \cup \{\alpha\}$ is $\mathcal{T}$-inconsistent. Therefore, $\mathcal{A}'$ is an inclusion-maximal $\mathcal{T}$-consistent subset of $\mathcal{A}$. And since $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \mathit{Unsat}(a)$, it follows that $\langle \mathcal{T}, \mathcal{A} \rangle \not\models_{AR} \mathit{Unsat}(a)$.

Next suppose $\langle \mathcal{T}, \mathcal{A} \rangle \not\models_{AR} \mathit{Unsat}(a)$. Then there exists $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{A}'$ is an inclusion-maximal $\mathcal{T}$-consistent subset of $\mathcal{A}$ and $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \mathit{Unsat}(a)$. Now let J be the interpretation of $\{x_1, \ldots, x_n\}$ defined as follows: $J(x_j) = \mathit{true}$ if there exists $L_t(x_j, c_i) \in \mathcal{A}'$ for some i, and $J(x_j) = \mathit{false}$ if there exists $L_f(x_j, c_i) \in \mathcal{A}'$ for some i. Now, since $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models \mathit{Unsat}(a)$, it follows that no assertion of the form $R(x_j, c_i)$ is in $\mathcal{A}'$, and since $\mathcal{A}'$ is inclusion-maximal $\mathcal{T}$-consistent, it follows that, for every $1 \leq i \leq n$, there exists an assertion of the form $L_t(x_j, c_i)$ or $L_f(x_j, c_i)$ in $\mathcal{A}'$ for some j. In turn, this immediately implies that the conjunct c_i of ϕ is satisfied in J, therefore J is a model of ϕ, which proves the claim. ∎

We notice that Theorem 3 corrects a wrong result presented in [48], which asserts tractability of instance checking under *AR*-semantics for *DL-Lite*$_F$ and *DL-Lite*$_R$, both subsuming *DL-Lite*$_{core}$. In fact, the technique presented in [48] is able to properly manage inconsistencies arising from the presence of functional assertions only, i.e., it is correct for *DL-Lite*$_{core}$ without negative inclusions, but enriched with functionalities on roles.

The following theorem provides a matching upper bound for *DL-Lite*$_{A,id,den}$, thereby establishes the exact complexity of query answering for this logic under the *AR*-semantics.

Theorem 4. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{A,id,den}$ KB, and let Q be a UBCQ. Deciding whether $\mathcal{K} \models_{AR} Q$ is coNP-complete with respect to data complexity.*

Proof. coNP-hardness follows from Theorem 3. To prove membership in coNP we provide the following algorithm

Algorithm AR-Entailment$(\mathcal{K}, Q)$
Input: *DL-Lite*$_{A,id,den}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, UBCQ Q
Output: *true* or *false*
begin
  if there exists $\mathcal{A}' \subseteq \mathcal{A}$ **such that**
    (1) $\mathcal{A}'$ is $\mathcal{T}$-consistent, and
    (2) **for each** $\alpha \in \mathcal{A} \setminus \mathcal{A}'$, $\mathcal{A}' \cup \{\alpha\}$ is $\mathcal{T}$-inconsistent, and
    (3) $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models Q$
  then return *false*
  else return *true*
end

and show that AR-Entailment$(\mathcal{K}, Q)$ returns true if and only if $\mathcal{K} \models_{AR} Q$.

By Definition 7, we know that if $\mathcal{K} \models_{AR} Q$, then $\langle \mathcal{T}, \mathcal{A}' \rangle \models Q$ for each $\mathcal{A}' \in$ *AR-Set*$(\mathcal{K})$. Since conditions (1) and (2) together check that $\mathcal{A}' \in$ *AR-Set*$(\mathcal{K})$, it is immediate to verify that, if $\mathcal{K} \models_{AR} Q$, then AR-Entailment$(\mathcal{K}, Q)$ returns true. Conversely, if AR-Entailment$(\mathcal{K}, Q)$ returns *true*, then no $\mathcal{A}' \in$ *AR-Set*$(\mathcal{K})$ exists such that $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models Q$. By Definition 7 it follows that $\mathcal{I} \models Q$ for each $\mathcal{I} \in Mod_{AR}(\mathcal{K})$, i.e., $\mathcal{K} \models_{AR} Q$.

From the algorithm AR-Entailment, it directly follows that the problem of establishing whether $\mathcal{K} \not\models_{AR} Q$, which is the complement of our problem, can be carried out by guessing an ABox $\mathcal{A}' \subseteq \mathcal{A}$, and checking conditions (1), (2), and (3). Proposition 2 implies that steps (1) and (3) can be carried out in polynomial time (in fact in AC^0), and therefore we easily conclude that the entire check is polynomial. Thus, $\mathcal{K} \not\models_{AR} Q$ can be checked by an NP algorithm, which proves the claim. ∎

The computational complexity results given in this section show that there is no hope of finding interesting cases for which conjunctive query answering is tractable in data complexity under the *AR*-semantics. Indeed, we get intractability even if we reduce to the least expressive *DL-Lite* logic, i.e., a very simple DL, and consider only instance checking, i.e., a very limited form of query answering. The only mentioned tractable case, i.e., instance checking over *DL-Lite*$_{core}$ KBs without negative inclusions, has very limited expressivity both in the KB and in the query language, thus it seems to be not suited for real-world applications.

6. The Intersection ABox Repair semantics

Towards the definition of practical solutions to the treatment of inconsistencies in DL KBs, in this section we introduce a new semantics, called *Intersection ABox Repair* (*IAR*) semantics, which is an approximation of the *AR*-semantics given in Section 4.

In a nutshell, this semantics is based on a new notion of repair obtained by following the WIDTIO (*When In Doubt Throw It Out*) principle, proposed in the area of belief revision and update [76, 31]. More precisely, it allows us to deal with a single repair, the intersection of all the *AR*-repairs of the KB, rather than the multiple repairs that may exist under the *AR*-semantics.

At the end of this section we will show that answering unions of conjunctive queries over a *DL-Lite*$_{A,id,den}$ KB under such semantics is tractable in data complexity.

Definition 10. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB. The* Intersection ABox Repair *(or IAR-repair) of $\mathcal{K}$, denoted by IAR-Repair$(\mathcal{K})$, is defined as*

$$IAR\text{-}Repair(\mathcal{K}) = \bigcap_{\mathcal{A}_i \in AR\text{-}Set(\mathcal{K})} \mathcal{A}_i$$

Then, the set of IAR-models of $\mathcal{K}$, denoted $Mod_{IAR}(\mathcal{K})$, is defined as follows:

$$Mod_{IAR}(\mathcal{K}) = Mod(\langle \mathcal{T}, \textit{IAR-Repair}(\mathcal{K}) \rangle)$$

Analogously to what we have done for the *AR*-semantics, we give below the notion of consistent entailment under the *IAR*-semantics.

Definition 11. *Let $\mathcal{K}$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB, and let ϕ be a first-order sentence. We say that ϕ is IAR-*consistently entailed*, or simply IAR-*entailed*, by $\mathcal{K}$, written $\mathcal{K} \models_{IAR} \phi$, if $\mathcal{I} \models \phi$ for every $\mathcal{I} \in Mod_{IAR}(\mathcal{K})$.*

Example 6. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be the KB presented in Example 4. According to Definition 10, and considering the set *AR-Set*($\mathcal{K}$) presented in Example 4, we have:

$$\begin{array}{rcl} \textit{IAR-Repair}(\mathcal{K}) & = & \{\ \textit{PortIn}(p_1), \textit{connectedTo}(p_1, p_2), \textit{Device}(d)\ \} \\ & \cap & \{\ \textit{PortOut}(p_1), \textit{connectedTo}(p_1, p_2), \textit{Device}(d)\ \} \\ & \cap & \{\ \textit{PortIn}(p_1), \textit{of}(p_1, p_2), \textit{Device}(d)\ \} \\ & \cap & \{\ \textit{PortOut}(p_1), \textit{of}(p_1, p_2), \textit{Device}(d)\ \} \\ & = & \{\ \textit{Device}(d)\ \} \end{array}$$

and then:

$$Mod_{IAR}(\mathcal{K}) = Mod(\langle \mathcal{T}, \textit{IAR-Repair}(\mathcal{K}) \rangle) = Mod(\langle \mathcal{T}, \{\textit{Device}(d)\} \rangle)$$

We point out that in computing *IAR-Repair*($\mathcal{K}$) all the knowledge about the fact that p_1 and p_2 are ports is lost, while we preserve the knowledge about the device d. Indeed, if we consider the BCQs q_1 and q_2 of Example 5, we have that $\mathcal{K} \not\models_{IAR} q_1$ and $\mathcal{K} \models_{IAR} q_2$. □

Next we discuss three relevant properties of the *IAR*-semantics. The first property states that the *IAR*-semantics is a sound approximation of the *AR*-semantics, in the sense that, for any knowledge base $\mathcal{K}$, every interpretation that is a model of $\mathcal{K}$ according to the *AR*-semantics is also a model of $\mathcal{K}$ according to the *IAR*-semantics.

Theorem 5. *If $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a DL-Lite$_{A,id,den}$ KB, then, $Mod_{AR}(\mathcal{K}) \subseteq Mod_{IAR}(\mathcal{K})$.*

Proof. As stated by Proposition 3, it holds that *AR-Set*($\mathcal{K}$) $\neq \emptyset$. Since *IAR-Repair*($\mathcal{K}$) is the intersection of all $\mathcal{A}_i \in$ *AR-Set*($\mathcal{K}$), clearly, for each $\mathcal{A}_i \in$ *AR-Set*($\mathcal{K}$) we have that *IAR-Repair*($\mathcal{K}$) $\subseteq \mathcal{A}_i$, and since the logic *DL-Lite*$_{A,id,den}$ is monotonic, we have that for each $\mathcal{A}_i \in$ *AR-Set*($\mathcal{K}$), $Mod(\langle \mathcal{T}, \mathcal{A}_i \rangle) \subseteq Mod(\langle \mathcal{T}, \textit{IAR-Repair}(\mathcal{K}) \rangle)$. Finally, since by definition $Mod_{IAR}(\mathcal{K}) = Mod(\langle \mathcal{T}, \textit{IAR-Repair}(\mathcal{K}) \rangle)$, and $Mod_{AR}(\langle \mathcal{T}, \mathcal{A} \rangle)$ is the union of all the models of the various *AR*-repairs, it follows that $Mod_{AR}(\langle \mathcal{T}, \mathcal{A} \rangle) \subseteq Mod_{IAR}(\langle \mathcal{T}, \mathcal{A} \rangle)$. ∎

The above theorem clearly implies that the logical consequences of $\mathcal{K}$, and thus also the answer to queries over $\mathcal{K}$, under the *IAR*-semantics are contained in

those obtained under the AR-semantics. Conversely, as Examples 5 and 6 show (cf. query q_1), there are sentences entailed by a KB $\mathcal{K}$ under the AR-semantics that are not entailed by $\mathcal{K}$ under the IAR-semantics.

The second property of the IAR-semantics that we discuss characterizes the notion of IAR-entailment, and will be used in the next sections.

Theorem 6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB, and let ϕ be a first-order sentence. Then $\mathcal{K} \models_{IAR} \phi$ if and only if there exists $\mathcal{A}' \subseteq \mathcal{A}$ such that:*

(i) $\langle \mathcal{T}, \mathcal{A}' \rangle \models \phi$;

(ii) there is no minimal $\mathcal{T}$-inconsistent set V in $\mathcal{A}$ such that $\mathcal{A}' \cap V \neq \emptyset$.

Proof.
($\Rightarrow$) Let $\mathcal{A}' = \bigcap_{\mathcal{A}_i \in AR\text{-}Set(\mathcal{K})} \mathcal{A}_i$. Suppose that $\mathcal{K} \models_{IAR} \phi$. From Definition 11 we have that $\langle \mathcal{T}, \mathcal{A}' \rangle \models \phi$. Let $\beta \in \mathcal{A}'$. Clearly, $\beta \in \mathcal{A}_i$ for each $\mathcal{A}_i \in AR\text{-}Set(\mathcal{K})$. Theorem 2 guarantees that there is no minimal $\mathcal{T}$-inconsistent set V in $\mathcal{A}$ such that $\beta \in V$. This means that $\mathcal{A}'$ is a subset of $\mathcal{A}$ for which both condition (i) and (ii) holds. ($\Leftarrow$) Let $\mathcal{A}' \subseteq \mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A}' \rangle \models \phi$ and such that for every minimal $\mathcal{T}$-inconsistent set $V \subseteq \mathcal{A}$ we have that $\mathcal{A}' \cap V = \emptyset$. Again, from Theorem 2 we have that $\mathcal{A}' \subseteq \bigcap_{\mathcal{A}_i \in AR\text{-}Set(\mathcal{K})} \mathcal{A}_i$. So, since $\langle \mathcal{T}, \mathcal{A}' \rangle \models \phi$, then $\langle \mathcal{T}, \bigcap_{\mathcal{A}_i \in AR\text{-}Set(\mathcal{K})} \mathcal{A}_i \rangle \models \phi$, which means that $\mathcal{K} \models_{IAR} \phi$. ∎

Notice that for each $\mathcal{A}'$ mentioned in Theorem 6, we have that $\mathcal{A}' \subseteq$ *IAR-Repair*($\mathcal{K}$), and condition (ii) tell us that every α belonging to the *IAR-Repair*($\mathcal{K}$) does not belong to any minimal $\mathcal{T}$-inconsistent set.

Moreover, analogously to what we have said at the end of Section 4 for the AR-semantics, we point out that the definitions of IAR-repairs and IAR-entailment can be generalized to any DL language $\mathcal{L}$, and that both Theorems 5 and 6 apply also to KBs specified in $\mathcal{L}$, provided that the semantics for $\mathcal{L}$ adopts the unique name assumption, and the KBs considered have a satisfiable TBox.

Finally, the third property of the IAR-semantics states that the IAR-repair of a *DL-Lite*$_{A,id,den}$ knowledge base can be computed in polynomial time with respect to the size of the ABox. Indeed, we conclude this section by presenting a PTIME algorithm for computing the IAR-repair of a *DL-Lite*$_{A,id,den}$ knowledge base. In order to present the algorithm, we need to introduce the notion of the *size* of a disjointness, functionality, denial or identification assertion as follows: (i) the size of every disjointness assertion is 2; (ii) the size of every functionality assertion is 2; (iii) the size of a denial assertion α is the number of atoms occurring in α; (iv) the size of an identification assertion α is $2k$, where k is the number of occurrences of atomic concepts, atomic roles, and attributes in α. Intuitively, the size of an assertion α of the above forms coincides with the number of atoms (excluding inequalities) occurring in the query $\varphi(\alpha)$ which encodes the negation of α (cf. Section 3). Given a *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$, we denote by *maxIncSize*($\mathcal{T}$) the maximum size of a disjointness, functionality, denial or identification assertion in $\mathcal{T}$. The following lemma states that *maxIncSize*($\mathcal{T}$)

bounds the cardinality of a minimal set of ABox assertions that are inconsistent with α.

Lemma 1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{A,id,den}$ KB. For every $V \in minIncSets(\mathcal{K})$ we have that $|V| \leq maxIncSize(\mathcal{T})$.*

Proof. The property can be easily proved by looking at the algorithm UnsatQuery (Algorithm 1), which makes use of the algorithm PerfectRefIdC. No conjunctive query in PerfectRefIdC$(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$ has a size which is greater than the size of the CQ with inequalities represented by $\varphi(\alpha)$, for $\alpha \in \mathcal{T} \setminus \mathcal{T}_{inc}$. It follows that the size of every disjunct of UnsatQuery$(\mathcal{T})$ is not greater than $maxIncSize(\mathcal{T})$. Consequently, by Theorem 1, it follows that the maximum size of every minimal $\mathcal{T}$-inconsistent subset of $\mathcal{A}$ is $maxIncSize(\mathcal{T})$, which proves the claim. ∎

We are now ready to present the algorithm Compute-IAR-Repair (Algorithm 2) that, given a DL-Lite$_{A,id,den}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, computes the IAR-repair of $\mathcal{K}$.

Input: DL-Lite$_{A,id,den}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$
Output: an ABox
begin
  $\mathcal{A}' \leftarrow \mathcal{A}$;
  $k = maxIncSize(\mathcal{T})$;
  foreach subset $\mathcal{S}$ of $\mathcal{A}$ such that $|\mathcal{S}| \leq k$ **do**
    if $\mathcal{S}$ is a $\mathcal{T}$-inconsistent set, and every subset of $\mathcal{S}$
      obtained by removing one fact from $\mathcal{S}$ is $\mathcal{T}$-consistent
    then $\mathcal{A}' \leftarrow \mathcal{A}' \setminus \mathcal{S}$;
  return $\mathcal{A}'$
end

Algorithm 2: Compute-IAR-Repair

Essentially, the algorithm deletes from $\mathcal{A}$ all assertions belonging to at least one minimal $\mathcal{T}$-inconsistent set. In order to single out the minimal $\mathcal{T}$-inconsistent subsets of $\mathcal{A}$, the algorithm exploits Lemma 1 and only considers the subsets of $\mathcal{A}$ whose size is not greater than $maxIncSize(\mathcal{T})$.

The following theorem establishes the termination and the correctness of Compute-IAR-Repair. The proof is omitted since it directly follows from Theorem 2 and Lemma 1.

Theorem 7. *If $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a DL-Lite$_{A,id,den}$ KB, the Compute-IAR-Repair$(\mathcal{K})$ is the IAR-repair of $\mathcal{K}$.*

As for the data complexity of computing the IAR-repair of a DL-Lite$_{A,id,den}$ KB, the following theorem states that this is a PTIME task.

Theorem 8. *If $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a DL-Lite$_{A,id,den}$ KB, then the algorithm Compute-IAR-Repair on input $\mathcal{K}$ runs in polynomial time with respect to the size of $\mathcal{A}$.*

Proof. First, observe that the maximum length k of an identification or denial assertion in $\mathcal{T}$ is of course independent of the size of the ABox. Consequently, the number of subsets of $\mathcal{A}$ that are considered by the algorithm is polynomial with respect to data complexity.

By Proposition 2, deciding whether a subset $\mathcal{S}$ of $\mathcal{A}$ is $\mathcal{T}$-consistent is in PTIME (in fact in AC^0). Moreover, deciding whether a $\mathcal{T}$-inconsistent subset $\mathcal{S}$ of $\mathcal{A}$ is a minimal $\mathcal{T}$-inconsistent set can be done by checking whether there is some $\alpha \in \mathcal{S}$ such that $\mathcal{S} \setminus \{\alpha\}$ is $\mathcal{T}$-inconsistent. Therefore, the algorithm Compute-IAR-Repair on input $\mathcal{K}$ runs in polynomial time with respect to $\mathcal{A}$. ∎

We observe that the above theorem immediately implies that query answering under the *IAR*-semantics is also in PTIME in data complexity. Indeed, if $\mathcal{A}'$ is the ABox returned by Compute-IAR-Repair$(\mathcal{T}, \mathcal{A})$, Theorem 7 tells us that, for any UBCQ Q, $\mathcal{K} \models_{IAR} Q$ iff $\langle \mathcal{T}, \mathcal{A}' \rangle \models Q$, and Theorem 8 together with Proposition 2 tells us that this can be decided in PTIME with respect to the size of $\mathcal{A}$.

In the next section, we show that we can avoid computing the *IAR*-repair in order to answer queries under the *IAR*-semantics. Indeed, we present a rewriting technique that shows that answering UBCQs under the *IAR*-semantics in *DL-Lite*$_{A,id,den}$ is in AC^0 in data complexity.

7. Query answering in *DL-Lite*$_{A,id,den}$ under the *IAR*-semantics

In this section, we show that conjunctive query answering under the *IAR*-semantics in *DL-Lite*$_{A,id,den}$ is FO-rewritable. We notice that this property is particularly interesting, since it allows us to obtain the consistent answers to a query without the need to compute the *IAR*-repair of the inconsistent KB over which the query is issued. Also, by virtue of this result, we can rely on a framework in which inconsistency-tolerant query answering in *DL-Lite*$_{A,id,den}$ has the same complexity as standard query answering under FOL semantics for this logic.

We start by providing the formal definition of FO-rewritability of query answering under the *IAR*-semantics, which is the natural generalization of the notion of FO-rewritability under classical DL semantics given in Definition 4.

Definition 12. *Query answering in a DL $\mathcal{L}$ is* FO-rewritable under *IAR*-semantics*, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$ and every query q over $\mathcal{T}$, one can effectively compute a FOL query q_r over $\mathcal{T}$ such that, for every ABox $\mathcal{A}$, $\langle \mathcal{T}, \mathcal{A} \rangle \models_{IAR} q$ if and only if $DB(\mathcal{A}) \models q_r$. We call q_r the IAR*-perfect FOL rewriting *(or simply IAR*-perfect rewriting*) of q w.r.t. $\mathcal{T}$.*

7.1. Overview of the query rewriting technique

Our technique can be summarized as follows. Given a UBCQ Q over a *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$, we first rewrite Q ignoring possible inconsistencies. To this aim, we make use of the algorithm PerfectRef, which allows us to obtain a perfect rewriting Q_r of Q w.r.t. $\mathcal{T}$ under the standard FOL semantics (cf.

Section 2). We notice that, by definition of perfect rewriting, and by definition of image of a query in an ABox (cf. Section 2.3), $\langle \mathcal{T}, \mathcal{A} \rangle \models Q$ if and only if $DB(\mathcal{A}) \models Q_r$, i.e., if and only if there is a CQ $q_r \in Q_r$ that has an image in $\mathcal{A}$, i.e., such that one of the patterns represented by q_r (ground instantiation of q_r) appears in A'. Thus, $\langle \mathcal{T}, \mathcal{A} \rangle \models_{IAR} q$ if and only if there is a CQ $q_r \in Q_r$ such that at least one of the patterns represented by q_r appears in the intersection of all AR-repairs of $\langle \mathcal{T}, \mathcal{A} \rangle$. Therefore, to properly take inconsistencies into account, the problem we have to solve is as follows: given a query $q_r \in Q_r$, does one of the patterns represented by q_r appear in the intersection of all AR-repairs of $\langle \mathcal{T}, \mathcal{A} \rangle$? To solve this problem we need to address two issues:

- How do we filter out patterns that cannot contribute to answer q_r because they do not appear in the intersection of all AR-repairs of $\langle \mathcal{T}, \mathcal{A} \rangle$?
- How do we make sure that we consider all the relevant patterns represented by a query $q_r \in Q_r$?

To address the first issue, we have to filter out those patterns that are "corrupted" by at least one inconsistency. We know from Theorem 6 that such patterns are those with a non-empty intersection with a minimal $\mathcal{T}$-inconsistent set in $\mathcal{A}$. This means that, among the patterns represented by $q_r \in Q_r$, we have to disregard those patterns $\mathcal{A}'$ such that $\mathcal{A}' \cap V \neq \emptyset$ for some $V \in minIncSets(\langle \mathcal{T}, \mathcal{A} \rangle)$. We therefore further rewrite each atom g of q_r into a FOL formula g_r such that, for any substitution σ from variables of g to constants in Γ_C such that $\sigma(g)$ is an image of g in $\mathcal{A}$, $DB(\mathcal{A}) \models \sigma(g_r)$ if only if $\sigma(g)$ does not belong to any minimal $\mathcal{T}$-inconsistent set in $\mathcal{A}$. To obtain this "inconsistency-aware component" of the rewriting, we devise an algorithm, called IncRewrIAR that, given a CQ q, adds to each atom $g \in q$ the rewriting g_r.

To address the second issue, since the check that a ground atom from q_r does not belong to a minimal $\mathcal{T}$-inconsistent set in $\mathcal{A}$ is done by checking that the atom does not appear in an image of a conjunctive query possibly with inequalities (coming from denial and identification assertions), we have to make sure that the atom does not have incomplete knowledge about the inequality of its terms. To this end, instead of computing the perfect rewriting of Q under the IAR-semantics starting from PerfectRef$(Q, \mathcal{T})$, we start from Saturate(PerfectRef$(Q, \mathcal{T})$), where Saturate is the algorithm that takes care of this issue. In particular, Saturate takes as input a UBCQ Q, and rewrites it into the union of all its possible inequality-based saturations, where a saturation of a query is obtained by choosing the pairs of variables to equate, and then adding an inequality atom $x \neq y$ for each pair of variables that have not been equated.

In the following we first describe the algorithm Saturate, then we present the algorithm IncRewrIAR and all the sub-routines it uses, and finally we illustrate the overall query rewriting algorithm.

7.1.1. The algorithm Saturate

In this subsection, we illustrate the algorithm Saturate, whose basic idea is to rewrite a UBCQ into an equivalent UBCQ containing disjuncts, in which

different variables denote different objects or values. This can be obtained by simply substituting each CQ in the original query with an equivalent UCQ with inequalities, each one obtained by equating a subset of variables in q, and imposing that the remaining variables are not equal. To formalize this process we need the following preliminary definition.

Given a Boolean query q, we say that a term t occurs in an *object position* of q if q contains an atom of the form $A(t)$, $P(t, t')$, $P(t', t)$, or $U(t, t')$, whereas we say that t occurs in a *value position* of q if q contains an atom of the form $U(t', t)$ or $T_i(t)$.

Given two different terms t_1 and t_2 occurring in a query q, we say that t_1 and t_2 are *compatible* in q if at least one of t_1 and t_2 is a variable, and one of the following conditions holds: (i) both t_1 and t_2 appear only in object positions of q or, (ii) both t_1 and t_2 appear only in value positions of q.

We now present the algorithm Saturate that takes as input a UBCQ with inequalities Q and returns a UBCQ with inequalities that we call the *inequality saturation of* Q. In the algorithm, we represent a UBCQ with inequalities as a set of BCQs with inequalities. Saturate(Q) first computes the set Q' by unifying compatible terms in each query $q \in Q$ in all possible ways; then, for any query q' in Q' and for each pair of terms t_1 and t_2 in q' that are syntactically different and compatible, it adds the inequality atom $t_1 \neq t_2$ to q'. In the algorithm $q[t_1/t_2]$ denotes the query obtained by replacing in q every occurrence of the term t_1 with the term t_2.

Input: a UBCQ with inequalities Q
Output: a UBCQ with inequalities
begin
  $Q' \leftarrow \emptyset$;
  while $Q \neq Q'$ **do**
    $Q' \leftarrow Q$;
    foreach $q \in Q$ **do**
      foreach pair of different terms t_1 and t_2 in q **do**
        if $t_1 \neq t_2$ does not occur in q **and**
          t_1 and t_2 are compatible in q
        then $Q \leftarrow Q \cup \{q[t_1/t_2]\}$;
  $Q'' \leftarrow \emptyset$;
  foreach $q \in Q'$ **do**
    foreach pair of different terms t_1 and t_2 that are compatible in q **do**
      $q \leftarrow q \wedge (t_1 \neq t_2)$;
    $Q'' \leftarrow Q'' \cup \{q\}$;
  return Q''
end

Algorithm 3: Saturate

Example 7. Let $\mathcal{T}$ be the *DL-Lite*$_{A,id,den}$ TBox presented in Example 1. Consider the following queries belonging to UnsatQuery($\mathcal{T}$).

$q_1 = \exists x, y, z.Port(x) \wedge Port(y) \wedge of(x, z) \wedge of(y, z) \wedge connectedTo(x, y)$;
$q_2 = \exists x, y, z.of(x, z) \wedge of(y, z) \wedge connectedTo(x, y)$;
$q_3 = \exists x.connectedTo(x, x)$.

It is easy to see that $\mathsf{Saturate}(q_1 \vee q_2 \vee q_3)$ is the disjunction of the following queries:

$q_1^1 = \exists x, y, z.Port(x) \wedge Port(y) \wedge of(x, z) \wedge of(y, z) \wedge connectedTo(x, y) \wedge x \neq y \wedge$
$\quad x \neq z \wedge y \neq z$;
$q_1^2 = \exists x, y.Port(x) \wedge Port(y) \wedge of(x, y) \wedge of(y, y) \wedge connectedTo(x, y) \wedge x \neq y$;
$q_1^3 = \exists x, y.Port(x) \wedge Port(y) \wedge of(x, x) \wedge of(y, x) \wedge connectedTo(x, y) \wedge x \neq y$;
$q_1^4 = \exists x, z.Port(x) \wedge of(x, z) \wedge connectedTo(x, x) \wedge x \neq z$;
$q_1^5 = \exists x.Port(x) \wedge of(x, x) \wedge connectedTo(x, x)$;
$q_2^1 = \exists x, y, z.of(x, z) \wedge of(y, z) \wedge connectedTo(x, y) \wedge x \neq y \wedge x \neq z \wedge y \neq z$;
$q_2^2 = \exists x, y.of(x, y) \wedge of(y, y) \wedge connectedTo(x, y) \wedge x \neq y$;
$q_2^3 = \exists x, y.of(x, x) \wedge of(y, x) \wedge connectedTo(x, y) \wedge x \neq y$;
$q_2^4 = \exists x, z.of(x, z) \wedge connectedTo(x, x) \wedge x \neq z$;
$q_2^5 = \exists x.of(x, x) \wedge connectedTo(x, x)$
$q_3^1 = \exists x.connectedTo(x, x)$.

□

Termination of $\mathsf{Saturate}(Q)$ is guaranteed by the fact that Q is a disjunction of a finite number of BCQs with inequalities in which there is a finite number of atoms and terms. The next lemma shows that the process applied by $\mathsf{Saturate}(Q)$ to a UBCQ Q does not affect the result of evaluating Q.

Lemma 2. *Let $\mathcal{A}$ be an ABox, and let Q be a UBCQ with inequalities. Then, $DB(\mathcal{A}) \models Q$ if and only if $DB(\mathcal{A}) \models \mathsf{Saturate}(Q)$.*

Proof. Let q be a BCQ with inequalities in Q. For each pair of t_1 and t_2 terms in Q, the following cases are conceivable:

(*i*) t_1 and t_2 are two variables compatible in q. In this case the algorithm first adds to Q a new query q' obtained from q by unifying t_1 and t_2, and then transforms the query q into $q_{\neq}$ obtained by adding the inequality atom $t_1 \neq t_2$. Since $q \equiv q' \vee q_{\neq}$, the claim follows.

(*ii*) t_1 and t_2 are not compatible in q. In this case the algorithm transforms the query q into $q_{\neq}$ by adding the inequality atom $t_1 \neq t_2$ to q. Since t_1 and t_2 are not compatible in q, it follows that they are two different constants, or one of the two occurs in an object position in q and the other one occurs in a value position in q. Since for every interpretation $\mathcal{I}$ we have that: (1) $\Delta_O^{\mathcal{I}} \cap \Delta_V = \emptyset$; (2) for every pair of different constants c_1 and c_2 in Γ_O, $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$; and (3) for every pair of different constants v_1 and v_2 in Γ_V, $val(v_1) \neq val(v_2)$, we conclude that t_1 and t_2 are always interpreted as different in $\mathcal{I}$. Hence $q \equiv q_{\neq}$, and the claim follows.

■

Note that as a consequence of Lemma 2, we have that a *DL-Lite*$_{A,id,den}$ KB $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable if and only if $DB(\mathcal{A}) \models \mathsf{Saturate}(\mathsf{UnsatQuery}(\mathcal{T}))$, and $DB(\mathcal{A}) \models \mathsf{PerfectRef}(Q, \mathcal{T})$ if and only if $DB(\mathcal{A}) \models \mathsf{Saturate}(\mathsf{PerfectRef}(Q, \mathcal{T}))$.

7.2. The algorithm IncRewrIAR

The goal of this subsection is to present the algorithm IncRewrIAR. As we said before, given a CQ q, and a TBox $\mathcal{T}$, IncRewrIAR adds to each atom $g \in q$ the FOL formula g_r that ensures that, for any ABox $\mathcal{A}$, such an atom does not belong to any minimal $\mathcal{T}$-inconsistent set in $\mathcal{A}$.

The basic building block of IncRewrIAR is the algorithm MinIncSet, that, given an atom $g \in q$, builds a FOL formula that checks whether there exists a minimal $\mathcal{T}$-inconsistent set in $\mathcal{A}$ that includes g. Obviously, IncRewrIAR will use the negation of the formula computed by MinIncSet.

In turn, the algorithm MinIncSet relies on another algorithm, called MinUnsatQuery, that computes a FOL query whose evaluation over the interpretation $DB(\mathcal{A})$ characterizes all sets of facts in the ABox $\mathcal{A}$ that form a minimal $\mathcal{T}$-inconsistent set. In the following, we first describe the algorithm MinUnsatQuery, and then we go back to the definition of both MinIncSet and IncRewrIAR.

7.2.1. The algorithm MinUnsatQuery

As shown in Section 3, a *DL-Lite*$_{A,id,den}$ KB $\langle \mathcal{T}, \mathcal{A} \rangle$ is inconsistent if and only if there is at least one query $q \in \mathsf{UnsatQuery}(\mathcal{T})$ such that $DB(\mathcal{A}) \models q$ (cf. Theorem 1). As already mentioned, we call every image of each such query q in $\mathcal{A}$ a $\mathcal{K}$-*clash*. Now, one may wonder whether a $\mathcal{K}$-clash corresponds to a minimal $\mathcal{T}$-inconsistent set (and vice-versa), so that we can directly exploit UnsatQuery to obtain the set $minIncSets(\mathcal{K})$. The following example shows that, in general, this is not the case.

Example 8. Let $\mathcal{T}$ be the *DL-Lite*$_{A,id,den}$ TBox presented in Example 1. We focus on the following denial assertion in $\mathcal{T}$.

$$\forall x, y, z.(Port(x) \wedge Port(y) \wedge of(x, z) \wedge of(y, z) \wedge connectedTo(x, y) \rightarrow \bot).$$

Consider the following ABox:

$$\begin{array}{rcll} \mathcal{A} & = & \{ & Port(p_1),\ Device(d_1),\ of(p_1, d_1),\ connectedTo(p_1, p_1), \\ & & & Port(p_2),\ of(p_2, d_1),\ Port(p_3),\ of(p_3, d_1),\ connectedTo(p_2, p_3) \quad \} \end{array}$$

It is not difficult to verify that the *DL-Lite*$_{A,id,den}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is not consistent. Indeed, the set of queries $\mathsf{UnsatQuery}(\mathcal{T})$ contains, among others, the following Boolean queries:

$q_1 = \exists x, y, z.Port(x) \wedge Port(y) \wedge of(x, z) \wedge of(y, z) \wedge connectedTo(x, y)$;
$q_2 = \exists x, y, z.of(x, z) \wedge of(y, z) \wedge connectedTo(x, y)$;
$q_3 = \exists x.connectedTo(x, x)$.

from which we obtain the following $\mathcal{K}$-clashes:

$$\begin{array}{rcll} V_1 & = & \{ & Port(p_2),\ of(p_2, d_1),\ Port(p_3),\ of(p_3, d_1),\ connectedTo(p_2, p_3) \quad \}; \\ V_2 & = & \{ & Port(p_1),\ of(p_1, d_1),\ connectedTo(p_1, p_1) \quad \}; \\ V_3 & = & \{ & of(p_2, d_1),\ of(p_3, d_1),\ connectedTo(p_2, p_3) \quad \}; \\ V_4 & = & \{ & of(p_1, d_1),\ connectedTo(p_1, p_1) \quad \}; \\ V_5 & = & \{ & connectedTo(p_1, p_1) \quad \}. \end{array}$$

It is easy to see that only the sets V_3 and V_5 are minimal $\mathcal{T}$-inconsistent sets. □

The following theorem gives the condition under which a $\mathcal{K}$-clash is a minimal $\mathcal{T}$-inconsistent in $\mathcal{A}$.

Lemma 3. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an inconsistent DL-Lite$_{A,id,den}$ KB, and let V be a $\mathcal{K}$-clash. Then $V \in minIncSets(\mathcal{K})$ if and only if for every proper subset V' of V, and every query $q \in \mathsf{UnsatQuery}(\mathcal{T})$, we have that $DB(V') \not\models q$.*

Proof. ($\Rightarrow$) Let $V \in minIncSets(\mathcal{K})$. Suppose, by way of contradiction, that there exists a query $q \in \mathsf{UnsatQuery}(\mathcal{T})$ and a proper subset V' of V, such that $DB(V') \models q$. From Theorem 1 it follows that V' is a $\mathcal{T}$-inconsistent set, but this contradicts the fact that $V \in minIncSets(\mathcal{K})$.
($\Leftarrow$) Let V be a $\mathcal{K}$-clash such that for every $V' \subset V$ and for every $q \in \mathsf{UnsatQuery}(\mathcal{T})$, $DB(V') \not\models q$. Toward a contradiction, suppose that $V \notin minIncSets(\mathcal{K})$. Since V is a $\mathcal{T}$-consistent set, we conclude that there is a proper subset V'' of V that is $\mathcal{T}$-inconsistent. From Theorem 1 it follows that there exists a query q in $\mathsf{UnsatQuery}(\mathcal{T})$ such that $DB(V'') \models q$. Hence, we have a contradiction. ∎

Based on the above results, we present, in this section, the algorithm MinUnsatQuery which, starting from the set of queries computed by the algorithm UnsatQuery, computes a new set of Boolean queries. Intuitively, our goal is to design the algorithm in such a way that the resulting UBCQ $\mathsf{MinUnsatQuery}(\mathcal{T})$ enjoys the following properties:

(P_1) For every Boolean query $q \in \mathsf{MinUnsatQuery}(\mathcal{T})$ and every ABox $\mathcal{A}$, $DB(\mathcal{A}) \models q$ if and only if there exists in $\mathsf{UnsatQuery}(\mathcal{T})$ a query q' such that $DB(\mathcal{A}) \models q'$. This guarantees that Theorem 1 also holds with $\mathsf{MinUnsatQuery}(\mathcal{T})$ in place of $\mathsf{UnsatQuery}(\mathcal{T})$.

(P_2) For every Boolean query $q \in \mathsf{MinUnsatQuery}(\mathcal{T})$ and every ABox $\mathcal{A}$, if $DB(\mathcal{A}) \models q$, then for every set of ABox assertions $V \in \mathsf{images}(q, \mathcal{A})$, and every V' such that $V' \subset V$, we have that $DB(V') \not\models q'$ for every $q' \in \mathsf{MinUnsatQuery}(\mathcal{T})$. This guarantees that if a query $q \in \mathsf{MinUnsatQuery}(\mathcal{T})$ is such that $DB(\mathcal{A}) \models q$, then every image of q in $\mathcal{A}$ is a minimal $\mathcal{T}$-inconsistent set.

Intuitively, this is achieved by basing the algorithm on the following two steps:

1) executing $\mathsf{Saturate}(\mathsf{UnsatQuery}(\mathcal{T}))$ to obtain the UBCQ $\mathcal{Q}_{str}$, where $\mathcal{Q}_{str}$ characterizes the $\mathcal{T}$-inconsistent sets;
2) modifying $\mathcal{Q}_{str}$ in such a way that it characterizes only $\mathcal{T}$-inconsistent sets that are minimal.

In the next paragraphs, we provide the details of the above step 2. For ease of exposition, we distinguish between two cases: the case of KBs without value-domain inclusions, and the case of KBs with value-domain inclusions.

KBs without value-domain inclusions. For the sake of exposition, we ignore, for the moment, inconsistencies caused by value-domain inclusions (cf. Example 2), i.e., we assume $\mathcal{T}_{type} = \emptyset$.

Firstly, we need to introduce the notion of proper syntactical subset of a query. Let q and q' be two Boolean queries. We say that q is a *proper syntactical subset* of q', written $q \prec_\varrho q'$, if there exists an injective function ϱ from the variables in q to the variables in q', such that every atom $S(\vec{t})$ occurring in $\varrho(q)$ occurs also in q', where $\varrho(q)$ denotes the query obtained by replacing every variable x in q with $\varrho(x)$, and an analogous injective function from q' to q does not exist. We denote with $\mathcal{Q}_{str}^{\prec_\varrho}$ the set of queries obtained by removing from $\mathcal{Q}_{str}$ every query q such that there exists in $\mathcal{Q}_{str}$ a query q' such that $q' \prec_\varrho q$.

The following example illustrates the role of the notion of proper syntactical subset of a query.

Example 9. Consider the set of queries $q_1^1, \ldots, q_3^1$ of Example 7, and call it $\mathcal{Q}_{str}$. It is easy to verify that the following hold:

$$\begin{array}{lll} q_2^1 \prec_\varrho q_1^1 & q_2^2 \prec_\varrho q_1^2 & q_2^3 \prec_\varrho q_1^3 \\ q_2^4 \prec_\varrho q_1^4 & q_2^5 \prec_\varrho q_1^5 & q_3^1 \prec_\varrho q_1^4 \\ q_3^1 \prec_\varrho q_1^5 & q_3^1 \prec_\varrho q_2^4 & q_3^1 \prec_\varrho q_2^5 \end{array}$$

Hence, $\mathcal{Q}_{str}^{\prec_\varrho}$ contains the following queries:

$q_2^1 = \exists x, y, z.\mathit{of}(x, z) \wedge \mathit{of}(y, z) \wedge \mathit{connectedTo}(x, y) \wedge x \neq y \wedge x \neq z \wedge y \neq z$;
$q_2^2 = \exists x, y.\mathit{of}(x, y) \wedge \mathit{of}(y, y) \wedge \mathit{connectedTo}(x, y) \wedge x \neq y$;
$q_2^3 = \exists x, y.\mathit{of}(x, x) \wedge \mathit{of}(y, x) \wedge \mathit{connectedTo}(x, y) \wedge x \neq y$;
$q_3^1 = \exists x.\mathit{connectedTo}(x, x)$.

Let $\mathcal{A}$ be the ABox of Example 8. We have that the only queries in $\mathcal{Q}_{str}^{\prec_\varrho}$ that evaluate to *true* over $DB(\mathcal{A})$ are q_2^1 and q_3^1. Their images in $\mathcal{A}$ are:

$$\begin{array}{rcl} V_2^1 & = & \{ \quad \mathit{of}(p_2, d_1),\ \mathit{of}(p_3, d_1),\ \mathit{connectedTo}(p_2, p_3) \quad \}; \\ V_3^1 & = & \{ \quad \mathit{connectedTo}(p_1, p_1) \quad \}; \end{array}$$

that coincide respectively with the V_3 and V_5 minimal $\mathcal{T}$-inconsistent sets of Example 8. □

We now provide a lemma showing that, if $\mathcal{T}$ is a TBox expressed in *DL-Lite*$_{A,id,den}$, then, for every ABox $\mathcal{A}$, we can use the set $\mathcal{Q}_{str}^{\prec_\varrho}$ for checking the satisfiability of the KB $\langle \mathcal{T}, \mathcal{A} \rangle$.

Lemma 4. *Let $\mathcal{T}$ be a DL-Lite$_{A,id,den}$ TBox, and let $\mathcal{A}$ be an ABox. Then, the KB $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable if and only if $DB(\mathcal{A}) \models \mathcal{Q}_{str}^{\prec_\varrho}$.*

Proof. ($\Rightarrow$) Since $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable, then by Theorem 1 we get $DB(\mathcal{A}) \models \mathsf{UnsatQuery}(\mathcal{T})$. Then, by Lemma 2 we get that $DB(\mathcal{A}) \models \mathcal{Q}_{str}$, where $\mathcal{Q}_{str} = \mathsf{Saturate}(\mathsf{UnsatQuery}(\mathcal{T}))$. Hence, there is a query $q \in \mathcal{Q}_{str}$ such that $DB(\mathcal{A}) \models q$. Suppose, by way of contradiction, that $DB(\mathcal{A}) \not\models \mathcal{Q}_{str}^{\prec_\varrho}$. This means that for

every query $q^{\prec}$ in $\mathcal{Q}_{str}^{\prec_\varrho}$, $DB(\mathcal{A}) \not\models q^{\prec}$. It follows that $q \notin \mathcal{Q}_{str}^{\prec_\varrho}$. This means that there exists in $\mathcal{Q}_{str}^{\prec_\varrho}$ a query q' such that $q' \prec_\varrho q$.

Since $DB(\mathcal{A}) \models q$, then there exists a substitution σ from the variables in q to constants in $\mathcal{A}$ such that the formula $\sigma(q)$ evaluates to *true* in the interpretation $DB(\mathcal{A})$. But this means that also $\sigma(\varrho(q))$ evaluates to *true* in $DB(\mathcal{A})$, then we have a contradiction.
($\Leftarrow$) If $DB(\mathcal{A}) \models \mathcal{Q}_{str}^{\prec_\varrho}$, then there is a query $q \in \mathcal{Q}_{str}^{\prec_\varrho}$ such that $DB(\mathcal{A}) \models q$. Since $\mathcal{Q}_{str}^{\prec_\varrho} \subseteq \mathcal{Q}_{str}$, then $q \in \mathcal{Q}_{str}$, and then $DB(\mathcal{A}) \models \mathcal{Q}_{str}$. Finally, from Theorem 1 and Lemma 2 it follows that $DB(\mathcal{A}) \models \mathsf{UnsatQuery}(\mathcal{T})$, which implies that $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable. ∎

The next lemma guarantees that, given a *DL-Lite*$_{A,id,den}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ with $\mathcal{T}_{type} = \emptyset$ and a query q in $\mathcal{Q}_{str}^{\prec_\varrho}$ such that $DB(\mathcal{A}) \models q$, every image V of q in $\mathcal{A}$ is a minimal $\mathcal{T}$-inconsistent set.

Lemma 5. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{A,id,den}$ KB with $\mathcal{T}_{type} = \emptyset$, and let q be a query in $\mathcal{Q}_{str}^{\prec_\varrho}$. Then, for every $V' \subset V$, where $V \in \mathsf{images}(q, \mathcal{A})$, and for every $q' \in \mathcal{Q}_{str}^{\prec_\varrho}$, $DB(V') \not\models q'$.*

Proof. Since $\mathcal{T}_{type} = \emptyset$, then every query in q belonging to $\mathsf{UnsatQuery}(\mathcal{T})$ is of the form $\exists z_1, \ldots, z_k . \bigwedge_{i=1}^{n} A_i(t_i^1) \wedge \bigwedge_{i=1}^{m} P_i(t_i^2, t_i^3) \wedge \bigwedge_{i=1}^{\ell} U_i(t_i^4, t_i^5) \wedge \bigwedge_{i=1}^{h} t_i^6 \neq t_i^7$, where every A_i, P_i, and U_i are as usual, every t_i^e is a term (i.e., either a constant or a variable), and $z_1, \ldots, z_k$ are all the variables appearing in q. In what follows, given a query q, we denote by $atoms(q)$ the set of atoms occurring in q.

If $\mathcal{K}$ is consistent, then Lemma 4 guarantees that for every query $q \in \mathcal{Q}_{str}^{\prec_\varrho}$, $DB(\mathcal{A}) \not\models q$. Hence there is no minimal $\mathcal{T}$-inconsistent set in $\mathcal{A}$. Let $\mathcal{K}$ be inconsistent. The proof proceeds by contradiction as follows. Let q be a query in $\mathcal{Q}_{str}^{\prec_\varrho}$ such that $DB(\mathcal{A}) \models q$, and let $V \in \mathsf{images}(q, \mathcal{A})$. Hence, there is a substitution σ from the variables in q to constants in $\mathcal{A}$ such that the formula $\sigma(q)$ evaluates to *true* in the interpretation $DB(V)$. Obviously, every constant occurring in q occurs also in V. Since $\mathcal{Q}_{str}^{\prec_\varrho} \subseteq \mathcal{Q}_{str}$, then we have constrained $t_1 \neq t_2$ for each pair of terms t_1 and t_2 in q. Hence, for each pair of different variables x and y in q, σ substitutes the variable x with a constant c_1 in V and the variable y with a constant c_2 in V such that $c_1 \neq c_2$. Now, let σ^{-1} be the inverse of the function σ and suppose that there is a query $q' \in \mathcal{Q}_{str}^{\prec_\varrho}$ such that $DB(V') \models q'$, where V' is a proper subset of V. Clearly, since σ^{-1} essentially performs a renaming of the constants in V without unifying any constant, we have that $DB(\sigma^{-1}(V')) \models q'$. But this means that there is a substitution σ' from the variables in q' to the terms in q such that each atom in $\sigma'(q')$ is in $atoms(q)$. Moreover, since $q' \in \mathcal{Q}_{str}^{\prec_\varrho}$, from the observations above, we have that for each pair of variables x' and y' in q', σ' substitutes the variable x' with the term t_1 in q and the variable y' with the term t_2 in q with $t_1 \neq t_2$; and since $V' \subset V$, then $\sigma'(q') \subset atoms(q)$. Hence, σ' constitutes an injective function from the variables in q' to the variables in q, such that every atom occurring in $\sigma'(q')$ occurs also in q' and an analogous function from q' to q does not exist. This means that $q' \prec_\varrho q$, which contradicts that $q \in \mathcal{Q}_{str}^{\prec_\varrho}$. ∎

KBs with value-domain inclusions. We now turn our attention to the case where $\mathcal{T}_{type}$ may be non-empty, i.e., when the TBox $\mathcal{T}$ contains value-domain inclusion assertions.

We recall that value-domains are pairwise disjoint, and inconsistency may arise because data in the ABox together with the TBox may imply a violation of such disjointnesses. The next example shows that the technique described in the previous subsection does not allow us to identify minimal $\mathcal{T}$-inconsistent sets when $\mathcal{T}_{type}$ is non-empty. Namely, Lemma 5 no longer holds.

Example 10. Let $\mathcal{T}$ be the TBox presented in Example 1. Specifically, here, we are interested in the following TBox assertions:

$$\begin{array}{lcll} \mathcal{T}_{type} & = & \{ & \rho(number) \sqsubseteq \texttt{xsd:integer} \quad \} \\ \mathcal{T}_{id} & = & \{ & (id\ Port\ number,\ of\ ) \quad \} \end{array}$$

In words, the assertion in $\mathcal{T}_{type}$ states that the range of the attribute *number* is restricted to be an integer, while the identification assertion imposes no two different ports of the same device exist having the same number. For ease of exposition, we assume to have only `xsd:integer`, `xsd:string`, and `xsd:dateTime` as value-domains.

The queries in $\mathsf{UnsatQuery}(\mathcal{T})$ originated from the assertions above are the following (see algorithm UnsatQuery in Section 3).

$q_1 = \exists x, y, d, n.Port(x) \wedge of(x, d) \wedge number(x, n) \wedge$
$\qquad Port(y) \wedge of(y, d) \wedge number(y, n) \wedge x \neq y;$
$q_2 = \exists x, y, d, n.of(x, d) \wedge number(x, n) \wedge of(y, d) \wedge number(y, n) \wedge x \neq y;$
$q_3 = \exists x, y.number(x, y) \wedge \texttt{xsd:string}(y);$
$q_4 = \exists x, y.number(x, y) \wedge \texttt{xsd:dateTime}(y).$

Now consider the following ABox:

$$\begin{array}{l} \mathcal{A} = \{Port(p_1), of(p_1, d_1), number(p_1, \text{'9XK11'}), \\ \qquad Port(p_2), of(p_2, d_1), number(p_2, \text{'9XK11'})\ \} \end{array}$$

where '9XK11' is a value of domain `xsd:string`, and therefore incoherent with respect to $\mathcal{T}_{type}$. Thus, the *DL-Lite*$_{A,id,den}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is inconsistent, and the following $\mathcal{K}$-clashes can be pinpointed.

$$\begin{array}{l} V_1 = \{Port(p_1), of(p_1, d_1), number(p_1, \text{'9XK11'}), \\ \qquad Port(p_2), of(p_2, d_1), number(p_2, \text{'9XK11'})\ \} \\ V_2 = \{of(p_1, d_1), number(p_1, \text{'9XK11'}), of(p_2, d_1), number(p_2, \text{'9XK11'})\ \} \\ V_3 = \{number(p_1, \text{'9XK11'})\ \} \\ V_4 = \{number(p_2, \text{'9XK11'})\ \} \end{array}$$

It is easy to see that only V_3 and V_4 are minimal $\mathcal{T}$-inconsistent sets. However, if we apply the technique described above, we get the following set $\mathcal{Q}_{str}^{\prec_{Rn}}$.

$q_2^1 = \exists x, y, d, n.of(x, d) \wedge number(x, n) \wedge of(y, d) \wedge number(y, n) \wedge$
$\qquad x \neq y \wedge x \neq d \wedge x \neq n \wedge y \neq d \wedge y \neq n \wedge d \neq n;$

$q_2^2 = \exists x, y, n.\mathit{of}(x, y) \wedge \mathit{number}(x, n) \wedge \mathit{of}(y, y) \wedge \mathit{number}(y, n) \wedge$
$\quad x \neq y \wedge x \neq n \wedge y \neq n;$
$q_2^3 = \exists x, y, n.\mathit{of}(x, x) \wedge \mathit{number}(x, n) \wedge \mathit{of}(y, x) \wedge \mathit{number}(y, n) \wedge$
$\quad x \neq y \wedge x \neq n \wedge y \neq n;$
$q_3^1 = \exists x, y.\mathit{number}(x, y) \wedge \texttt{xsd:string}(y);$
$q_4^1 = \exists x, y.\mathit{number}(x, y) \wedge \texttt{xsd:dateTime}(y).$

Observe that both q_2^1 and q_3^1 evaluate to *true* over $DB(\mathcal{A})$, although, as we said before, only for each $V \in \mathsf{images}(q_3^1, \mathcal{A})$ we have that V is minimal $\mathcal{T}$-inconsistent. □

Intuitively, in the above example the problem arises since in q_2^1 we do not consider the fact that the range of the attribute *number* is `xsd:integer`, and therefore we single out a violation of the identification assertion (*id Port number*, *of*) which is not minimal. The problem can be solved by simply adding to q_2^1 an atom imposing that the variable n is an integer, as shown below:

$q_2^1 = \exists x, y, d, n.\mathit{of}(x, d) \wedge \mathit{number}(x, n) \wedge \mathit{of}(y, d) \wedge \mathit{number}(y, n) \wedge$
$\quad x \neq y \wedge x \neq d \wedge x \neq n \wedge y \neq d \wedge y \neq n \wedge d \neq n \wedge \texttt{xsd:integer}(n)$

According to the above considerations, we modify our rewriting method as follows: for each query q in $\mathsf{Saturate}(\mathsf{UnsatQuery}(\mathcal{T}))$, and for each atom $U(t_1, t_2)$ in q, if $\mathcal{T} \models \rho(U) \sqsubseteq T_i$, and there exists no atom of the form $T_j(t_2)$ in q, then we add the atom $T_i(t_2)$ to q.

The definition of MinUnsatQuery. We are now ready to present the algorithm MinUnsatQuery (Algorithm 4). The algorithm proceeds as follows. First, it computes the inequality saturation of $\mathsf{UnsatQuery}(\mathcal{T})$ through the algorithm Saturate (step 1 and step 2), and calls $\mathcal{Q}_{str}$ the result. Then, in step 3 it computes the set $\mathcal{Q}'_{str}$ by substituting each query $q \in \mathcal{Q}_{str}$ with the query obtained as follows: for each atom $U(x, y)$, where U is an attribute name, if no atom $T_j(y)$ appears in q, where T_j is a value-domain, then for each $T_i \in \{T_1 \ldots T_n\}$, if $\mathcal{T} \models \rho(U) \sqsubseteq T_i$, the algorithm builds a new query by substituting the atom $U(x, y)$ with the conjunction of atoms $U(x, y) \wedge T_i(y)$. Note that checking whether $\mathcal{T} \models \rho(U) \sqsubseteq T_i$ can be done through any off-the-shelf DL reasoner. Step 4 is an optimization step in which some queries which are always false are removed from $\mathcal{Q}'_{str}$. Finally, in step 5, the algorithm removes from $\mathcal{Q}'_{str}$ every query q' such that there exists a query q in $\mathcal{Q}'_{str}$ such that $q \prec_\varrho q'$, and returns as result the set $\mathcal{Q}'_{str}$ in step 6. By analyzing each step of the algorithm, it is immediate to see that the algorithm terminates when applied to a *DL-Lite*$_{A,id,den}$ TBox.

Example 11. Let us focus on the same portion of the TBox $\mathcal{T}$ of Example 1 considered in Example 10. The set $\mathsf{MinUnsatQuery}(\mathcal{T})$ contains, among others, the following queries:

$q_2^1 = \exists x, y, d, n.\mathit{of}(x, d) \wedge \mathit{number}(x, n) \wedge \mathit{of}(y, d) \wedge \mathit{number}(y, n) \wedge$
$\quad \texttt{xsd:integer}(n) \wedge x \neq y \wedge x \neq d \wedge x \neq n \wedge y \neq d \wedge y \neq n \wedge d \neq n;$

Input: a *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$
Output: a UBCQ with inequalities
begin
  $\mathcal{Q}_{\mathcal{T}}^{unsat} \leftarrow \mathsf{UnsatQuery}(\mathcal{T})$; /* step 1 */
  $\mathcal{Q}_{str} \leftarrow \mathsf{Saturate}(\mathcal{Q}_{\mathcal{T}}^{unsat})$; /* step 2 */
  $\mathcal{Q}'_{str} \leftarrow \mathcal{Q}_{str}$;
  foreach $q \in \mathcal{Q}'_{str}$ **do** /* step 3 */
    foreach atom $U(t, t')$ in q **do**
      if there exists no atom of the form $T_j(t')$ in q **then**
        foreach value-domain T_i in $\{T_1, \ldots T_n\}$ **do**
          if $\mathcal{T} \models \rho(U) \sqsubseteq T_i$ **then**
            $\mathcal{Q}'_{str} \leftarrow (\mathcal{Q}'_{str} \setminus \{q\}) \cup \{q \wedge T_i(t')\}$;
  foreach $q \in \mathcal{Q}'_{str}$ **do** /* step 4 */
    foreach term t occurring in q **do**
      if $T_i(t)$ and $T_j(t)$ occur in q, with $i \neq j$ **then**
        $\mathcal{Q}'_{str} \leftarrow \mathcal{Q}'_{str} \setminus \{q\}$;
  foreach q and q' in $\mathcal{Q}'_{str}$ **do** /* step 5 */
    if $q \prec_{\varrho} q'$ **then** $\mathcal{Q}'_{str} \leftarrow \mathcal{Q}'_{str} \setminus \{q'\}$;
  return $\mathcal{Q}'_{str}$ /* step 6 */
end

Algorithm 4: MinUnsatQuery

$q_2^2 = \exists x, y, n.of(x, y) \wedge number(x, n) \wedge of(y, y) \wedge number(y, n) \wedge$
$\texttt{xsd:integer}(n) \wedge x \neq y \wedge x \neq n \wedge y \neq n$;
$q_2^3 = \exists x, y, n.of(x, x) \wedge number(x, n) \wedge of(y, x) \wedge number(y, n) \wedge$
$\texttt{xsd:integer}(n) \wedge x \neq y \wedge x \neq n \wedge y \neq n$;
$q_3^1 = \exists x, y.number(x, y) \wedge \texttt{xsd:string}(y)$;
$q_4^1 = \exists x, y.number(x, y) \wedge \texttt{xsd:dateTime}(y)$.

Note that the above queries are those in Example 10 to which step 3 of the algorithm MinUnsatQuery is applied. □

The following lemmas show that the algorithm MinUnsatQuery enjoys the properties (P_1) and (P_2) previously introduced in this subsection.

Lemma 6. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB. Then, $DB(\mathcal{A}) \models$ MinUnsatQuery$(\mathcal{T})$ if and only if $DB(\mathcal{A}) \models$ UnsatQuery$(\mathcal{T})$.*

Proof. The proof can be straightforwardly derived from the one of Lemma 4, by simply observing that for each constant c in Σ_V, the sentence $T_1(c) \vee \ldots \vee T_n(c)$ evaluates to *true* under every interpretation, and that $T_1(c) \wedge \ldots \wedge T_n(c)$ is a contradiction. ∎

Note that, from the lemma above, it directly follows that Theorem 1 also holds with MinUnsatQuery$(\mathcal{T})$ in place of UnsatQuery$(\mathcal{T})$.

The following crucial lemma guarantees that one can use the queries produced by $\mathsf{MinUnsatQuery}(\mathcal{T})$ in order to compute every minimal $\mathcal{T}$-inconsistent set in $\mathcal{A}$.

Lemma 7. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB, and let q be a query in $\mathsf{MinUnsatQuery}(\mathcal{T})$. If $DB(\mathcal{A}) \models q$, then every image of q in $\mathcal{A}$ is a minimal $\mathcal{T}$-inconsistent set.*

Proof. The proof can be straightforwardly derived from the one of Lemma 5, by considering Lemma 6 in place of Lemma 4 and by observing that for each constant c in Σ_V, the sentence $T_1(c) \vee \ldots \vee T_n(c)$ evaluates to *true* under every interpretation, and that $T_1(c) \wedge \ldots \wedge T_n(c)$ is a contradiction. ∎

7.2.2. The algorithm MinIncSet and the definition of IncRewrIAR

Our next goal is to describe the algorithm MinIncSet. To this aim, we introduce the notion of compatibility between atoms. Let $S(\vec{t})$, $S(\vec{t'})$ be two atoms, where S is a symbol denoting an atomic concept, an atomic role, or an attribute. We say that $S(\vec{t'})$ is *compatible* with $S(\vec{t})$ if there exists a mapping μ from the variables occurring in $S(\vec{t'})$ to the terms occurring in $S(\vec{t})$ such that $\mu(S(\vec{t'})) = S(\vec{t})$ (and in this case we denote the above mapping μ with the symbol $\mu_{S(\vec{t})/S(\vec{t'})}$). Given an atom $S(\vec{t})$ and a query q, we denote by $CompSet(S(\vec{t}), q)$ the set of atoms of q which are compatible with $S(\vec{t})$.

Then, we define MinIncSet as the algorithm that, given a *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$ and an atom $S(\vec{t})$, returns the following FOL query:

$$\mathsf{MinIncSet}(S(\vec{t}), \mathcal{T}) = \bigvee_{q \in \mathsf{MinUnsatQuery}(\mathcal{T}) \wedge CompSet(S(\vec{t}),q) \neq \emptyset} \left(\bigvee_{S(\vec{t'}) \in CompSet(S(\vec{t}),q)} \mu_{S(\vec{t})/S(\vec{t'})}(q) \right)$$

Intuitively, given a possibly inconsistent *DL-Lite*$_{A,id,den}$ KB $\langle \mathcal{T}, \mathcal{A} \rangle$, MinIncSet exploits the ability of $\mathsf{MinUnsatQuery}(\mathcal{T})$ to detect all the minimal $\mathcal{T}$-inconsistent sets in $\mathcal{A}$ so as to compute a rewriting of the atom $S(\vec{t})$ that allows for deciding if there is a minimal $\mathcal{T}$-inconsistent set V in $\mathcal{A}$ such that $DB(V) \models S(\vec{t})$. The crucial property of MinIncSet is stated in the following lemma.

Lemma 8. *Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a possibly inconsistent DL-Lite$_{A,id,den}$ KB, and let $S(\vec{c})$ be an ABox assertion. There exists a minimal $\mathcal{T}$-inconsistent set V in $\mathcal{A}$ such that $S(\vec{c}) \in V$ if and only if $DB(\mathcal{A}) \models \mathsf{MinIncSet}(S(\vec{c}), \mathcal{T})$.*

Proof. ($\Rightarrow$) Suppose that $S(\vec{c})$ belongs to a minimal $\mathcal{T}$-inconsistent set $V \subseteq \mathcal{A}$. Lemma 6 and Lemma 7 guarantee that there exists a query q in $\mathsf{MinUnsatQuery}(\mathcal{T})$ such that $DB(\mathcal{A}) \models q$ and that there is an image of q in $\mathcal{A}$ to which $S(\vec{c})$ belongs. This means that there is a query q' in the

set computed by substituting $S(\vec{c})$ to the compatible terms in q such that $DB(\mathcal{A}) \models q'$. Hence $DB(\mathcal{A}) \models \bigvee_{S(\vec{t'}) \in CompSet(S(\vec{c}),q)} (\mu_{S(\vec{c})/S(\vec{t'})}(q))$ and then $DB(\mathcal{A}) \models \mathsf{MinIncSet}(S(\vec{c}), \mathcal{T})$.
($\Leftarrow$) Let us consider $\mathsf{MinIncSet}(S(\vec{c}), \mathcal{T})$ as a set of queries. Suppose that there is query $q' \in \mathsf{MinIncSet}(S(\vec{c}), \mathcal{T})$ such that $DB(\mathcal{A}) \models q'$. Let $q \in \mathsf{MinUnsatQuery}(\mathcal{T})$ be the query from which q' is obtained from. From Lemma 7 it directly follows that there exists in $\mathcal{A}$ a minimal $\mathcal{T}$-inconsistent set that contains $S(\vec{c})$. ∎

With the algorithm MinIncSet in place, we are ready to provide the definition of algorithm IncRewrIAR.

Let $\mathcal{T}$ be a *DL-Lite*$_{A,id,den}$ TBox, and let q be a BCQ with inequalities of the form

$$\exists z_1, \ldots, z_k. \bigwedge_{i=0}^{n} A_i(t_i^1) \wedge \bigwedge_{i=0}^{m} P_i(t_i^2, t_i^3) \wedge \bigwedge_{i=0}^{\ell} U_i(t_i^4, t_i^5) \wedge \bigwedge_{i=0}^{k} T_i(t_i^6) \wedge \bigwedge_{i=0}^{h} t_i^7 \neq t_i^8 \quad (1)$$

where every t_i^e is a term (i.e., either a constant or a variable), $z_1, \ldots, z_k$ are all the variables appearing in q, such that q contains at least one atom not using inequality, and t_i^7 and t_i^8 for any i appear also in some atom not using inequality.

We denote by IncRewrIAR the algorithm that, given the above q and $\mathcal{T}$, returns the following FOL query:

$$\begin{aligned}\mathsf{IncRewrIAR}(q, \mathcal{T}) = \exists z_1, \ldots, z_k. & \textstyle\bigwedge_{i=1}^{n} A_i(t_i^1) \;\wedge\; \neg\mathsf{MinIncSet}(A_i(t_i^1), \mathcal{T}) \;\wedge \\ & \textstyle\bigwedge_{i=1}^{m} P_i(t_i^2, t_i^3) \;\wedge\; \neg\mathsf{MinIncSet}(P_i(t_i^2, t_i^3), \mathcal{T}) \;\wedge \\ & \textstyle\bigwedge_{i=1}^{\ell} U_i(t_i^4, t_i^5) \;\wedge\; \neg\mathsf{MinIncSet}(U_i(t_i^4, t_i^5), \mathcal{T}) \;\wedge \\ & \textstyle\bigwedge_{i=1}^{k} T_i(t_i^6) \;\wedge\; \bigwedge_{i=1}^{h} t_i^7 \neq t_i^8\end{aligned}$$

7.3. The overall query rewriting algorithm

We now have all the ingredients for illustrating the algorithm IAR-PerfectRef (Algorithm 5), whose goal is to compute the perfect rewriting of a UBCQ Q w.r.t. a *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$ under the *IAR*-semantics.

Input: UBCQ Q, *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$
Output: a FOL query over $\mathcal{T}$
begin
  $Q' = \mathsf{PerfectRef}(Q, \mathcal{T}_{inc} \cup \mathcal{T}_{type})$;
  $Q'' = \mathsf{Saturate}(Q')$;
  return $\bigvee_{q \in Q''} \mathsf{IncRewrIAR}(q, \mathcal{T})$
end

Algorithm 5: IAR-PerfectRef

Using PerfectRef, the algorithm first computes the perfect rewriting Q' of Q with respect to $\mathcal{T}$ (cf. Section 2). Then, the UBCQ Q' is passed to the

algorithm Saturate, producing a set Q'' of BCQs with inequalities of the form (1); as explained above, this step is indispensable for exactly identifying, for each query atom g, the queries in $\mathsf{MinUnsatQuery}(\mathcal{T})$ whose images correspond to inconsistent sets containing g. Finally, each query in the set Q'' is passed to the algorithm IncRewrIAR: the disjunction of all the FOL queries returned by IncRewrIAR is the final FOL query returned by the algorithm IAR-PerfectRef.

Termination of IAR-PerfectRef follows from the following properties: (i) the algorithm PerfectRef terminates as shown in [22, 20]; (ii) the algorithm Saturate terminates (see subsection 7.1.1); (iii) the algorithm IncRewrIAR terminates; this follows immediately from the termination of the algorithm MinUnsatQuery. Correctness of IAR-PerfectRef is proved in the following theorem.

Theorem 9. *Let $\mathcal{T}$ be a DL-Lite$_{A,id,den}$ TBox, and let Q be a UBCQ. Then, IAR-PerfectRef$(Q,\mathcal{T})$ is a IAR-perfect FOL rewriting of Q with respect to $\mathcal{T}$.*

Proof. We already observed that $\mathsf{IAR\text{-}PerfectRef}(Q,\mathcal{T})$ is a FOL query. It remains to prove that for every ABox $\mathcal{A}$, $\langle\mathcal{T},\mathcal{A}\rangle \models_{IAR} Q$ if and only if $DB(\mathcal{A}) \models \mathsf{IAR\text{-}PerfectRef}(Q,\mathcal{T})$.
($\Rightarrow$) Let Q be a UBCQ. Suppose $\langle\mathcal{T},\mathcal{A}\rangle \models_{IAR} Q$. From Theorem 6, we have that there exists a query $q \in Q$ and a $\mathcal{T}$-consistent ABox $\mathcal{A}' \subseteq \mathcal{A}$, such that: (i) $\langle\mathcal{T},\mathcal{A}'\rangle \models q$, and (ii) $\mathcal{A}' \cap V = \emptyset$ for every $V \in minIncSets(\mathcal{K})$. Since A' is $\mathcal{T}$-consistent, Proposition 1 guarantees that we can use the PerfectRef algorithm for computing its perfect FOL rewriting. Let $Q' = \mathsf{PerfectRef}(q, \mathcal{T}_{inc} \cup \mathcal{T}_{type})$. We have that $DB(\mathcal{A}') \models Q'$. Let $Q'' = \mathsf{Saturate}(Q')$. Lemma 2 guarantees that $DB(\mathcal{A}') \models Q''$. Hence, there exists a query q' in Q'', such that $DB(\mathcal{A}') \models q'$. Clearly, since $\mathcal{A}' \subseteq \mathcal{A}$, $DB(\mathcal{A}) \models q'$. Let $\mathcal{G}$ be an image of q' in $\mathcal{A}'$. Since $\mathcal{A}' \cap V = \emptyset$ for every $V \in minIncSets(\mathcal{K})$, also $\mathcal{G} \cap V = \emptyset$ for every $V \in minIncSets(\mathcal{K})$. Lemma 8 guarantees that for every assertion $\alpha \in \mathcal{G}$, $DB(\mathcal{A}) \not\models \mathsf{MinIncSet}(\alpha,\mathcal{T})$. Let $\mathsf{IncRewrIAR}(q',\mathcal{T}) = q' \wedge \bigwedge_{S\in q'} \neg\mathsf{MinIncSet}(S,\mathcal{T})$, where S is an atom in q' built over an atomic concept, atomic role, or an attribute. Since $DB(\mathcal{A}) \models q'$ and $DB(\mathcal{A}) \models \neg\mathsf{MinIncSet}(\alpha,\mathcal{T})$ $\alpha \in \mathcal{G}$, then $DB(\mathcal{A}) \models \mathsf{IncRewrIAR}(q',\mathcal{T})$. Hence, one can conclude that $DB(\mathcal{A}) \models \mathsf{IAR\text{-}PerfectRef}(Q,\mathcal{T})$.
($\Leftarrow$) Suppose that $DB(\mathcal{A}) \models \phi$, where $\phi = \mathsf{IAR\text{-}PerfectRef}(Q,\mathcal{T})$. Let q be a FOL query corresponding to a disjunct of ϕ such that $DB(\mathcal{A}) \models q$ (since $DB(\mathcal{A}) \models \phi$ such a query q exists). Let's write q as follows:

$$\exists\vec{z}.\bigwedge_{i=0}^{n} S_i(\vec{t_i}) \wedge \neg\mathsf{MinIncSet}(S_i(\vec{t_i}),\mathcal{T}) \wedge \bigwedge_{i=0}^{m} T_i(t'_i) \wedge \bigwedge_{i=0}^{h} t'_i \neq t''_i$$

Since $DB(\mathcal{A}) \models q$, then there is an image of q in $\mathcal{A}$. Let $\mathcal{G}$ be such an image. Let q' be the query in $\mathsf{Saturate}(Q')$, where $Q' = \mathsf{PerfectRef}(Q, \mathcal{T}_{inc} \cup \mathcal{T}_{type})$, such that $q = \mathsf{IncRewrIAR}(q',\mathcal{T})$. Since $DB(\mathcal{G}) \models q$, then $DB(\mathcal{G}) \models q'$. Moreover, from Proposition 1 and Lemma 2, it directly follows that $\langle\mathcal{T},\mathcal{J}\rangle \models Q$. Since $\mathcal{DB}(\mathcal{G}) \not\models \mathsf{MinIncSet}(S_i(\vec{t_i}),\mathcal{T})$ for every atom $S_i(\vec{t_i})$ of q', then from Lemma 8 it follows that $\mathcal{G} \cap V = \emptyset$ for every $V \in minIncSets(\mathcal{K})$. But this means that

$\mathcal{G}$ is a subset of $\mathcal{A}$ satisfying both conditions (i) and (ii) of Theorem 6. Hence, $\langle \mathcal{T}, \mathcal{A} \rangle \models_{IAR} Q$. ∎

Finally, the following complexity result is an immediate consequence of the termination of the algorithm IAR-PerfectRef and of Theorem 9.

Corollary 1. *Let $\mathcal{K}$ be a DL-Lite$_{A,id,den}$ KB and let Q be a UBCQ. Deciding whether $\mathcal{K} \models_{IAR} Q$ is in AC^0 in data complexity.*

Proof. The proof follows from Theorem 9 and from the fact that evaluation of FOL queries over relational databases is in AC^0 in data complexity. ∎

We conclude the section with an observation on the size of the rewriting computed by IAR-PerfectRef$(Q, \mathcal{T})$. It is well-known that the size of the perfect reformulation of a query $q \in Q$ computed by PerfectRef is polynomial with respect to $|\mathcal{T}|$, and exponential with respect to $|q|$ [22]. It is easy to see that the size of $Q'' = \mathsf{Saturate}(\mathsf{PerfectRef}(Q, \mathcal{T}_{inc} \cup \mathcal{T}_{type}))$ remains polynomial with respect to $|\mathcal{T}|$, and exponential with respect to $|Q|$. Since query IncRewrIAR$(q, \mathcal{T})$ makes use of MinIncSet$(S(\vec{t}), \mathcal{T})$ for each atom $S(\vec{t})$ in $q \in Q''$, and the size of MinIncSet$(S(\vec{t}), \mathcal{T})$ is exponential with respect to $|\mathcal{T}|$ (see Algorithm 1), we can conclude that the size of the rewriting computed by IAR-PerfectRef$(Q, \mathcal{T})$ is exponential with respect to both $|\mathcal{T}|$ and $|Q|$.

8. Experimental evaluation

In this section we illustrate the results of the experiments we carried out on the query rewriting technique for the *IAR*-semantics presented above[5]. To test the technique, we implemented IAR-PerfectRef through a JAVA program, which produces SQL encodings of the first-order rewritings computed by the algorithm. The ABoxes over which we executed such rewritings have been stored as relational databases under the DBMS PostgreSQL 9.0. Experiments have been run on a 2.6 GHz quad-core Intel Core i7 MacBook Pro laptop equipped with 8 GB ram.

We used the LUBM benchmark ontology[6] as test-bed. The LUBM TBox contains 43 atomic concepts, 25 atomic roles, 7 attributes, and about 200 assertions. Since its expressivity goes beyond *DL-Lite*$_{A,id,den}$, due to the use of conjunctions and qualified existential restrictions in the left-hand side of positive inclusion assertions, we first approximated it in *DL-Lite*, which in fact required to eliminate very few inclusion assertions from the TBox. Then, since the LUBM TBox does not contain axioms that can be contradicted by ABox assertions, we slightly modified it by adding some assertions that can cause inconsistency. Namely, we added 10 negative inclusions, 5 identifications, and 3 denials to the TBox.

[5] More details on the experiments can be found at `http://www.dis.uniroma1.it/~ruzzi/JWS/`.

[6] `http://swat.cse.lehigh.edu/projects/lubm/`

		Percentage of inconsistency			
Univ.	**Cons.**	**1**	**5**	**10**	**20**
1	90,711	91,875	94,668	99,123	107,883
5	563,095	569,612	585,824	605,876	649,664
10	1,146,997	1,160,128	1,192,564	1,232,656	1,320,244
20	2,422,559	2,449,604	2,514,476	2,594,648	2,769,836

Figure 2: Size of the ABoxes used in the experiments (number of ABox assertions)

Turning our attention to data, we initially produced 4 different ABoxes of increasing size by means of the UBA Data Generator provided by the LUBM website. Such tool can generate ABoxes of varying size according to specific input parameters. In particular, we used the generator to produce ABoxes containing data regarding 1, 5, 10, and 20 universities, respectively, which we denote with $\mathcal{A}_i$, where $1 \leq i \leq 5$. These ABoxes turned out to be consistent with the TBox of our experiments. Therefore, we modified them to introduce inconsistency, and created four different versions of each such $\mathcal{A}_i$, containing increasing percentages of inconsistency.

More precisely, we created four different inconsistent versions of each $\mathcal{A}_i$, which we denote with $\mathcal{A}_i^j$, where $j \in \{1, 5, 10, 20\}$ indicates the percentage of inconsistency in each $\mathcal{A}_i^j$. Such ABoxes are such that $\mathcal{A}_i \subset \mathcal{A}_i^1$ and $\mathcal{A}_i^j \subset A_i^{j'}$, for $j < j'$. Furthermore, let $\mathcal{K}_i^j = \langle \mathcal{T}, \mathcal{A}_i^j \rangle$ be a KB where $\mathcal{T}$ is our *DL-Lite*$_{A,id,den}$ version of the LUBM TBox, we have that all the assertions in $\mathcal{A}_i^j \setminus \mathcal{A}_i$ are involved in some $\mathcal{K}_i^j$-clash and all the assertions of $\mathcal{A}_i$ are not involved in any $\mathcal{K}_i^j$-clash. In other words, in each $\mathcal{A}_i^j$ we add only assertions that are inconsistent among them, and leave $\mathcal{A}_i$ as a consistent nucleus. We finally notice that all $\mathcal{A}_i^j$ are such that the inconsistency they generate is uniformly distributed over the axioms that we added to the original LUBM TBox.

Figure 2 shows the size (number of assertions) of all the ABoxes we used in the experiments. Every row reports the size of the ABox $\mathcal{A}_i$ and of its modified inconsistent versions $\mathcal{A}_i^j$, and is labeled with the number of universities given as input to the UBA Data Generator used to produce $\mathcal{A}_i$. For each row i, column **Cons.** shows the number of assertions in $\mathcal{A}_i$, whereas columns **1**, **5**, **10**, and **20** give the number of assertions of $\mathcal{A}_i^1$,$\mathcal{A}_i^5$, $\mathcal{A}_i^{10}$, and $\mathcal{A}_i^{20}$, respectively, i.e., modified ABox containing respectively the $1\%, 5\%, 10\%$ and 20% of assertions involved in some clash with the TBox.

We then issued 17 conjunctive queries over the test ontologies, which are taken from the LUBM benchmark query set (containing 14 queries) or adapted from it[7]. The first part of the experiments consisted in verifying the effective computability of the IAR-PerfectRef algorithm, as well as estimating the overhead it introduces with respect to the PerfectRef algorithm for query rewriting under the standard FOL semantics. Figure 3 gives us some information on the

[7]All queries are listed at `http://www.dis.uniroma1.it/~ruzzi/JWS/`

Query	Qsize	PRsize	MISsize	IARPRsize
Q_1	2	1	34	9 KB
Q_2	2	2	58	13 KB
Q_3	4	3	256	111 KB
Q_4	1	4	82	19 KB
Q_5	2	48	4,087	916 KB
Q_6	4	2	2,516	716 KB
Q_7	4	6	1,047	455 KB
Q_8	4	2	238	77 KB
Q_9	4	1	234	158 KB
Q_{10}	3	2	280	97 KB
Q_{11}	5	4	2,696	659 KB
Q_{12}	6	6	4,636	1,740 KB
Q_{13}	4	18	2,700	936 KB
Q_{14}	2	1	21	10 KB
Q_{15}	4	18	12,469	5,939 KB
Q_{16}	4	2	468	316 KB
Q_{17}	6	2	868	262 KB

Figure 3: Query rewriting sizes

size of the test queries and of their rewritings obtained through either PerfectRef or IAR-PerfectRef. More precisely, for each query Q_i the table contains the following information:

- column Qsize indicates the number of atoms of the conjunctive query;
- column PRsize reports the number of CQs contained in the rewriting returned by PerfectRef;
- column MISsize shows the number of disjuncts produced by all the executions of the MinIncSet algorithm done in the IncRewrIAR step of the rewriting, which gives us a measure of how much the processing of the query is affected by possible inconsistency. Intuitively, large values for MISsize correspond to queries whose perfect FOL rewriting returned by PerfectRef contains many atoms with predicates occurring in assertions of $\mathcal{T}_{type}$, $\mathcal{T}_{disj}$, $\mathcal{T}_{funct}$, $\mathcal{T}_{id}$, or $\mathcal{T}_{den}$;
- column IARPRsize reports the size of the binary files storing the final SQL rewriting produced by our implementation of IAR-PerfectRef.

As we can notice, the size of the rewriting produced by IAR-PerfectRef largely exceeds the one computed by PerfectRef. This is not surprising, since, as already said, the size of the rewriting that IAR-PerfectRef returns is exponential in the size of both the query and the TBox, whereas the size of the rewriting produced by PerfectRef is exponential only in the size of the query. We also notice that, as expected, the MISsize can vary a lot for queries with the same PRsize. For

Query	T1	T2	Total
Q_1	16	1	17
Q_2	15	48	63
Q_3	16	94	110
Q_4	15	16	31
Q_5	187	499	686
Q_6	78	282	360
Q_7	31	266	297
Q_8	16	82	98
Q_9	15	15	30
Q_{10}	16	63	79
Q_{11}	16	142	158
Q_{12}	47	266	313
Q_{13}	78	361	439
Q_{14}	16	1	17
Q_{15}	46	680	726
Q_{16}	16	67	83
Q_{17}	15	125	140

Figure 4: Query rewriting times (milliseconds)

example, queries Q_{13} and Q_{15} have the same PRsize, but the MISsize for Q_{13} significantly lower than the analogous value for Q_{15}. Indeed, Q_{13} is less affected than Q_{15} by the inconsistency.

Let us now consider rewriting times, which we give in Figure 4. For each query Q_i the table contains the following information:

- column T_1 indicates the time needed to run PerfectRef;
- column T_2 shows the time needed for all the executions of IncRewrIAR;
- *Total* indicates the total time that the IAR-PerfectRef algorithm takes for the input query.

Notice that we do not indicate the time needed for query saturation (execution of the algorithm Saturate) because it is negligible w.r.t. T_1 and T_2.

We recall that to compute the query MinIncSet used in the IncRewrIAR algorithm, we first need to execute the algorithm MinUnsatQuery. The result of this last algorithm is independent of the specific query at hand, and therefore we can execute it off-line, before query rewriting. Therefore, we do not include the time needed for its execution in the results shown in Figure 4. According to our experiments, the total time for executing it is 48,188 milliseconds. From the figures about rewriting times presented above, we can conclude that in our experiments the time overhead caused by the inconsistency treatment proposed in this paper with respect to the standard query rewriting technique, i.e., disregarding inconsistency, is acceptable, and does not constitute a bottleneck of the approach.

Univ.	1					5					10					20				
Perc. Inc.	0	1	5	10	20	0	1	5	10	20	0	1	5	10	20	0	1	5	10	20
Q1	0.17	0.11	0.11	0.11	0.11	0.19	0.21	0.18	0.18	0.19	0.34	0.32	0.32	0.33	0.34	0.61	0.60	0.64	0.75	0.63
Q2	0.08	0.07	0.08	0.07	0.07	0.07	0.08	0.08	0.08	0.08	0.07	0.08	0.09	0.10	0.09	0.08	0.08	0.11	0.09	0.11
Q3	1.14	1.15	1.16	1.15	1.14	1.26	1.28	1.25	1.26	1.23	1.30	1.29	1.33	1.30	1.30	1.34	1.26	1.31	1.30	1.26
Q4	4.13	5.17	10.4	17.0	32.1	170	282	408	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*
Q5	22.8	22.6	22.9	22.9	23.3	315	318	323	330	342	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*
Q6	5.82	9.61	20.4	18.6	24.2	7.03	34.05	189	340	*t/o*	7.45	117	*t/o*	*t/o*	*t/o*	8.41	472	*t/o*	*t/o*	*t/o*
Q7	1.89	1.89	1.90	1.99	2.02	2.34	2.34	2.88	2.90	2.94	2.57	2.56	2.58	2.58	4.00	2.57	2.58	2.65	2.63	2.64
Q8	0.50	0.49	0.49	0.49	0.49	0.61	0.61	0.60	0.61	0.61	0.65	0.65	0.67	0.65	0.65	0.75	0.75	0.75	0.76	0.76
Q9	1.84	1.83	2.38	2.35	2.20	1.31	1.27	1.23	1.09	1.06	1.47	1.49	1.50	1.46	1.50	2.42	2.36	2.45	2.44	2.49
Q10	0.47	0.45	0.47	0.47	0.47	0.88	0.88	0.93	0.95	0.95	1.27	1.36	1.41	1.41	1.46	2.20	2.33	2.61	2.72	2.59
Q11	4.99	4.94	4.95	4.95	5.03	5.15	5.18	5.20	5.18	5.27	5.30	5.34	5.26	5.23	5.26	5.26	5.26	5.26	5.24	5.23
Q12	25.2	23.0	22.9	22.6	22.9	24.0	24.6	24.3	24.2	24.0	25.3	26.5	25.6	25.6	26.0	29.1	28.5	27.6	28.1	28.3
Q13	3.92	3.95	3.93	3.96	3.97	9.10	9.57	11.3	13.5	18.6	10.4	11.2	15.0	20.6	41.3	17.9	19.2	24.4	45.1	77.3
Q14	0.05	0.06	0.06	0.06	0.05	0.10	0.11	0.11	0.10	0.11	0.16	0.17	0.16	0.17	0.17	0.29	0.29	0.29	0.29	0.29
Q15	129	123	127	127	128	123	123	123	124	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*	*t/o*
Q16	2.43	2.41	2.62	2.63	2.64	1.79	1.83	1.90	1.84	1.88	2.18	2.20	2.35	2.40	2.41	3.19	3.32	3.50	3.60	3.58
Q17	4.30	4.24	4.27	3.22	3.27	4.82	4.93	6.02	6.21	6.30	7.66	8.19	6.88	12.4	12.7	17.6	19.4	12.5	39.5	38.7

Figure 5: IAR-perfect rewriting evaluation times (seconds)

We now turn our attention to the evaluation of the rewriting. We computed the answers of each IAR-perfect rewriting produced by IAR-PerfectRef over all 20 test ABoxes described above. Evaluation times (in seconds) are given in Figure 5. The results are grouped according to the number of universities used to generate the ABoxes through the UBA data generator of the benchmark (supercolumns **1**, **5**, **10**, and **20**). For each such group we provide five columns, labeled with **0**, **1**, **5**, **10**, and **20**, respectively, that indicate the percentage of inconsistency of each ABox in the group (notice that for group i, column 0 contains results obtained over $\mathcal{A}_i$). We established a 10 minute timeout for every single rewriting evaluation.

To give an idea of the overhead caused by our inconsistency-tolerant query answering approach based on the IAR-semantics, with respect to standard query answering under FOL semantics, we also provide the times needed to evaluate the rewriting returned by PerfectRef over all the 20 test ABoxes. Of course, the answers we get in this case are not meaningful, since we are disregarding the inconsistency and naively query the KB as it was consistent. Results we obtained (in milliseconds) are given in Figure 6.

Each value given in Figure 5 and Figure 6 represents the average time over four evaluations of the same rewriting over the same ABox. To avoid that possible caching strategies adopted by the underlying DBMS affected our experiments, we executed four experiment rounds, and in each round we evaluated each rewriting over all the ABoxes.

Looking at the IAR-perfect rewriting evaluation times of Figure 5, we notice that we could execute successfully almost 90% of the evaluations, and that we experienced very good performances (only 3% of the successful cases took more than 4 minutes). Only 4 out of 17 rewritings reached the timeout, and only for ABoxes containing high levels of inconsistency or more than one million tuples.

The trend of the evaluation time is essentially constant or increasing within each group of ABoxes. In some cases (e.g., Q_6 or Q_{13}) it shows an increment when moving from a group containing smaller ABoxes to one containing larger

Univ.	1					5					10					20				
Perc. Inc.	0	1	5	10	20	0	1	5	10	20	0	1	5	10	20	0	1	5	10	20
Q1	23	15	15	15	14	40	41	45	42	43	78	77	78	79	79	162	154	157	157	161
Q2	5	5	5	6	7	7	7	8	10	11	8	8	11	13	15	8	8	16	15	22
Q3	10	10	10	10	10	17	18	21	12	13	23	25	14	15	19	35	37	16	19	10
Q4	31	18	18	19	20	189	180	197	193	217	362	360	382	391	431	748	746	762	790	807
Q5	94	89	91	91	94	319	332	337	341	354	615	645	652	662	679	1,039	1,098	1,115	1,124	1,167
Q6	14	14	7	7	8	7	8	8	8	80	7	7	7	151	155	8	8	8	312	316
Q7	10	11	10	11	11	18	19	19	16	16	26	26	27	22	23	46	46	49	51	57
Q8	6	7	7	7	7	7	6	7	7	7	7	7	7	7	7	7	7	7	7	7
Q9	17	17	17	17	17	63	63	63	65	66	120	119	120	123	125	248	248	248	250	255
Q10	7	7	7	7	7	64	65	119	120	122	123	123	236	239	241	247	252	488	492	502
Q11	23	22	23	22	22	24	22	22	22	22	23	23	22	22	22	23	23	23	22	22
Q12	39	39	39	38	40	48	46	46	45	45	43	44	45	43	42	44	46	45	44	44
Q13	49	49	49	50	51	2,911	2,903	2,914	2,906	2,949	15,132	14,956	15,190	14,842	15,071	17,295	17,377	17,391	17,428	17,638
Q14	8	8	8	8	7	24	24	24	24	24	42	42	42	42	42	83	83	82	83	83
Q15	78	78	79	80	80	89	92	94	97	103	98	101	106	109	125	113	117	122	134	157
Q16	26	19	27	27	27	65	68	118	121	123	121	124	235	238	241	248	256	491	491	500
Q17	40	40	40	43	44	407	423	424	442	536	890	997	905	911	1,137	2,001	1,980	2,214	2,472	2,274

Figure 6: Perfect rewriting evaluation times (milliseconds)

ABoxes, but the trend is always increasing for equal percentages of inconsistencies. For only two cases, involving queries Q_4 and Q_5, we notice a large increment in evaluation time when moving from a group to another, disregarding the percentage of inconsistency. The same behavior for these two queries can be observer also in Figure 6, which indicates that the jump in this case is somehow amplified by the PerfectRef component of the rewriting. This makes us to conclude that in general the evaluation time is more affected by the increment of inconsistency in the ABox than by the only increment of the size of the ABox. We also notice that less performing rewritings have in general a large number of disjuncts generated by MinIncSet (cf. column MISsize of Figure 3). This is however not always the case, as it can be verified looking at evaluation times for the rewriting of query Q_4, which has a small MISsize. Conversely, query Q_{12} has a quite large MISsize but its IAR-perfect rewriting is always evaluated in less than 30 seconds. This is not completely surprising, since evaluation times are also affected by the distribution of data and the features of the DBMS managing them. On the other hand, for most of the cases we observed a certain correspondence between the size of the IAR-perfect reformulation and the evaluation time of the rewriting (the larger the former the slower the latter).

Comparing the results in Figure 5 and Figure 6 we notice that using IAR-PerfectRef for query answering instead of PerfectRef makes query answering times to increase of some degree of magnitude. This was expected, because of the complex structure and large dimension of the former rewriting compared to the latter. On the other hand, an increase in the computation time is the price to pay to rely on a principled treatment of the inconsistency rather than obtaining spurious answers from an inconsistent ontology. These results, however, clearly call for optimized methods for query answering under the IAR-semantics. We further notice that no general correlation exist between the time needed to answer a query under the standard FOL semantics and what is needed under the IAR-semantics, by using our query rewriting techniques. Indeed, even though queries that we tested to be easy under the FOL semantics have been evaluated quite quickly also under the IAR-semantics, there are queries, like Q_{13}, which

turned out to be the most difficult under the FOL semantics, but definitely manageable under the IAR-semantics. In other words, we experienced that the presence of inconsistency and the way it is distributed heavily change the query answering performances.

We conclude by noticing that the rewriting technique proposed in this paper is only the starting point of our investigation on rewriting methods for inconsistency-tolerant query answering under the IAR-semantics, which has been mainly devised as a means to show FO-rewritability of this task for $\mathit{DL\text{-}Lite}_{A,id,den}$. We are currently working on optimization techniques in order to obtain more compact rewritings, thus limiting the price to pay for moving from standard query answering to inconsistency tolerant processing of queries. Nonetheless, we believe that the results we obtained in the experiments presented in this section are very encouraging, and set the stage for applicability of consistent-query answering technique in real-world OBDA scenarios.

9. Related work

Inconsistency management has been addressed in various forms in several areas, including Logic, Artificial Intelligence, and Databases. In Logic, several semantics have been proposed with the aim of providing more meaningful notions of logical entailment for classically inconsistent theories. Some notable examples come from the field of multi-valued and paraconsistent logics [61, 72, 55, 54]. Each of the proposals has advantages and drawbacks, and the choice of the paraconsistent logic depends on the requirements of the application at hand. The techniques adopted in these works are however quite different from the ones we use in the present paper, and therefore we do not discuss them further. Instead, in this section we concentrate on the approaches that are most related to OBDA, and in particular we consider the studies dealing with inconsistency handling in ontologies, belief revision, and databases.

9.1. Inconsistency handling in ontologies

Several works of the Semantic Web community focus on the issue of dealing with inconsistencies in the knowledge base. In [40], the authors present a framework for reasoning with inconsistent KBs. At the basis of such a framework is the notion of selection function, that allows for choosing some consistent sub-theory from an inconsistent KB. Standard reasoning is then applied to the selected sub-theory. An instantiation of the framework, based on a syntactic relevance-based selection function is also briefly described. In [38], a more extended framework that generalizes four approaches to inconsistency handling is presented. In particular, consistency of KB evolution, repairing inconsistency, reasoning with inconsistent KBs, and KB versioning are considered.

In [21], a tool is presented that allows for verifying whether a KB expressed in *DL-Lite* is consistent: the check is reduced to the evaluation of first-order queries over the ABox. However, the kind of support to inconsistency management provided by these approach mostly consists of identifying inconsistencies.

Furthermore, with the exception of [21], they are mainly focused on inconsistencies at the intensional level (or do not distinguish between intensional and instance level).

The form of inconsistency tolerance studied in this paper is related to the work on justification and pinpointing in Description Logic ontologies. In fact, minimal inconsistent subsets of the ABox can be seen as explanations, or minimal justifications (at the extensional level) of the inconsistency of the knowledge base. Given this connection, some computational properties of pinpointing (e.g., [62, 63, 6]) are related to the complexity of reasoning under inconsistency-tolerant semantics, although the two problems are different and are studied under different assumptions. Some works have explicitly focused on the problem of finding justifications for inconsistency. In [60], the authors present a framework for detecting and diagnosing errors in OWL ontologies. In [70], the authors discuss a number of alternative methods to explain incoherence of TBoxes, unsatisfiability of concepts and concept subsumption, in order to provide support to knowledge engineers who are building terminologies using Description Logic reasoners. Then, in [7] a visual tool is presented that allows the user to check consistency of formal KBs. In [39], the authors present an algorithm for computing justifications for inconsistent ontologies in OWL 2. To summarize, none of the above mentioned works provides concrete techniques, or complexity results for inconsistency-tolerant query answering in the framework considered in this paper.

9.2. Belief revision

The form of inconsistency tolerance considered in this work is deeply connected to the study of update in databases and belief revision/update. Consider a consistent knowledge base $\mathcal{K}$ and a new piece of information $\mathcal{N}$. Suppose that our intention is to change $\mathcal{K}$ with the insertion of $\mathcal{N}$. If $\mathcal{K} \cup \mathcal{N}$ is inconsistent, then the revision/update semantics assume that the original knowledge base $\mathcal{K}$ has to be modified in order for the result of the change to be consistent. The studies in belief revision appear very relevant for reasoning over inconsistent KBs. For instance, if $\mathcal{K}' = \langle \mathcal{T}, \mathcal{A} \rangle$ is a possibly inconsistent knowledge base, with respect to the knowledge base revision/update framework, we can consider the ABox $\mathcal{A}$ as the initial knowledge, whereas the TBox $\mathcal{T}$ represents the incoming knowledge. Based on such a correspondence, the inconsistency-tolerant semantics presented in this work are strictly related to the work presented in [32, 33, 59]. However, none of these papers provides specific techniques for DL KBs. In [65], an algorithm is proposed for handling inconsistency in DL KBs based on a revision operator. This approach allows for resolving conflicts changing the original knowledge base by weakening both ABox and TBox assertions. Similarly, in [57] inconsistency is resolved by transforming every concept inclusion assertion in the TBox into a cardinality restriction. Then, if a cardinality restriction is involved in a conflict, one weakens it by relaxing the restrictions of the number of elements it may have. To the best of our knowledge, the approach studied in this work, based on instance-level repair only, is novel for Description Logics,

and it is inspired by the work on inconsistency tolerance in Databases discussed next.

9.3. Inconsistency tolerance in databases

Traditionally, in databases, consistency is preserved by forbidding malicious update operations. Clearly, such an approach is not applicable in those scenarios where the aim is to merge information coming from different data sources, as in data integration. Data cleaning techniques provide in this case a classical procedural means for restoring consistency [15, 65, 56]. These solutions are procedural in nature and require to collect additional application-specific information, which can be a serious drawback in many contexts.

Besides traditional data cleaning, the research in databases has also pursued a declarative approach to the problem, concentrating on two main issues: restoring consistency through the computation of a new consistent database, starting from an inconsistent one, and computing meaningful answers to queries without necessarily modifying the database in order to resolve inconsistency. The second approach is known under the name of *consistent query answering*, following the seminal work of Arenas, Bertossi and Chomicki [3], and is the approach we explore in this paper in the context of OBDA.

The main notion at the basis of declarative approaches to inconsistency tolerance in databases is that of a *repair* [3]. A repair for an inconsistent database is defined as a database instance that satisfies integrity constraints and minimally differs from the original database. Various criteria of minimality have been proposed in the literature. In [28], the authors show how the notion of minimality can be interpreted in different ways, depending on the kinds of constraints that are considered. Indeed, for the large class of *denial constraints*, the only way to restore the integrity of a database is to retract a part of it. On the other hand, if the information is both incorrect and incomplete, as in the case where also inclusion dependencies are considered, both insertions and deletions of pieces of information should be considered. Alternatively, some data integration approaches give up the completeness assumption [44, 19], and therefore do not consider violations of inclusion dependencies by the underlying database as a real inconsistency. In [27] the authors study inconsistency in presence of both denial constraints and inclusion dependencies, and present a semantics in which only tuple elimination is allowed, and it is therefore used also for repairing violations of inclusions, differently from [44, 19]. In other works, repairs are given in terms of tuple updates, rather than tuple deletions or insertions (see, e.g., [74, 9]). The papers mentioned so far, as well as the present paper, adopt a minimality criterion for repairs based on set containment. An alternative approach is the one in which repairs are defined through a cardinality-based minimality criterion (see e.g., [51]).

In the last 15 years, several studies have analyzed the computational complexity of consistent query answering under various inconsistency-tolerant semantics and various classes of constraints [18, 28, 71, 75], and many algorithms to solve it have been proposed. Some of such approaches aim at rewriting the input query into a new query whose evaluation over the inconsistent database

returns the consistent answers (similar to the approach we follow in the present paper). Depending on the complexity of the problem, such rewritings are specified as (possibly optimized) logic programs allowing for the use of negation or disjunction (as in [19, 36, 30]), or as first-order logic queries (as in [3, 37, 34]). Notice, however, that this last approach can be pursued only in very limited settings, since under the semantics proposed for inconsistency tolerance in databases the problem easily becomes intractable. Thus, even if relevant for the present paper, the rewriting techniques proposed in the context of databases cannot be adapted to the setting we consider. For an overview of the work on consistent query answering, we refer the reader to [26, 8].

9.4. Instance-level inconsistency tolerance in DLs

We finally survey some recent works that, similarly to our approach, are specifically tailored to the study of "instance-level" inconsistency-tolerant semantics for DL knowledge bases.

In [48] a repair-based inconsistency-tolerant semantics for DLs is provided and data complexity of query answering under such semantics is studied for *DL-Lite*$_R$ and *DL-Lite*$_F$, two logics of the *DL-Lite* family captured by *DL-Lite*$_{A,id,den}$. As already said (cf. Section 5), the notion of repair given in [48] coincides with our notion of AR-repair, and then, their inconsistency-tolerant semantics coincides with our AR-semantics. The objective of [48] was however different from ours, since it aimed at identifying tractable cases of inconsistency-tolerant query answering under the AR-semantics, and thus focused on inconsistency-tolerant instance checking. As noticed, our complexity lower bound for *DL-Lite*$_{core}$ given in Theorem 3 corrects the results in [48], asserting tractability of instance checking under AR-semantics for *DL-Lite*$_F$ and *DL-Lite*$_R$. It can be shown that the technique presented in [48] is in fact correct only for *DL-Lite*$_{core}$ KBs without negative inclusions, but enriched with functionalities on roles.

More recently, in [11] the author carries out an investigation for *DL-Lite* KBs, with the aim of better understanding the cases in which inconsistency-tolerant query answering under AR-semantics is feasible, and in particular, can be done using query rewriting. Specifically, the author formulates some general conditions that can be used to prove that a first-order reformulation for inconsistency-tolerant query answering does or does not exist for a given *DL-Lite*$_{core}$ TBox and instance query. Subsequently, in [12, 13], the same author conducts a complexity analysis of the AR-semantics with the aim of characterizing the complexity of inconsistency-tolerant query answering based on the properties of the KB and the conjunctive query at hand. In particular, in [12], by focusing on a very simple language, which is a fragment of *DL-Lite*$_{core}$, the author identifies the number of quantified variables in the query as an important factor in determining the complexity of inconsistency-tolerant query answering. To be more precise, it is shown that inconsistency-tolerant query answering under AR-semantics: (i) is always first-order rewritable for conjunctive queries with at most one quantified variable; (ii) has polynomial data complexity when the

query has two quantified variables; (*iii*) is coNP-hard for queries having three (or more) quantified variables.

In the same paper, the author proposes a novel inconsistency-tolerant semantics that is a sound approximation of the *AR*-semantics. This semantics, named *intersection of closed repairs* (*ICR*), corresponds to closing *AR*-repairs with respect to the TBox before intersecting them. The *ICR*-semantics approximates the *AR*-semantics better than the *IAR*-semantics: however, query answering under the *ICR* semantics in *DL-Lite* logics is in general intractable. In particular, [12] shows that first-order rewritability of inconsistency-tolerant query answering under *ICR*-semantics is guaranteed only for *DL-Lite*$_{core}$ ontologies without inverse roles. For full *DL-Lite*$_{core}$, the problem is instead coNP-hard [12].

We observe that the *ICR*-semantics is similar in spirit to the *ICAR*-semantics we investigated in [46, 47]. Analogously to the *ICR*-semantics, in the *ICAR*-semantics we close the *AR*-repairs with respect to the TBox, but instead of taking their intersection, we consider the union of such closures and compute the *IAR*-repair of this union, which we call the *ICAR*-repair. We remark that the rewriting technique presented in this paper can be adapted to compute perfect rewritings under the *ICAR*-semantics, thus showing that inconsistency-tolerant query answering for UBCQs is first-order rewritable also in this case. To ease readability, we preferred to not discuss this issue in the present work. Some hints on how to modify the IAR-PerfectRef algorithm to obtain a query rewriting procedure for the *ICAR*-semantics can be found in [47], where first-order rewritability of query answering for *DL-Lite*$_A$ is shown for both the *IAR*- and the *ICAR*-semantics: we point out, however, that the technique presented in that paper is different from the one we present here, which is able to deal also with identification and denial assertions.

In [66] the author presents a computational analysis of the problems of instance checking and conjunctive query answering under *AR*- and *IAR*-semantics (and other inconsistency-tolerant semantics) for a wide spectrum of DLs, ranging from tractable ones ($\mathcal{EL}$) to very expressive ones ($\mathcal{SHIQ}$), showing that reasoning under the above semantics is inherently intractable, even for very simple DLs. In particular, these results imply that the tractability of query answering under *IAR*-semantics that we have shown for the *DL-Lite* logics does not extend to other tractable DLs like $\mathcal{EL}$.

In addition, [14] presents two parameterized inconsistency-tolerant semantics for DLs, called k-support and k-defeater semantics. The k-support semantics can be seen as a generalization of the *IAR*-semantics, in the sense that is a sound approximation of the *AR*-semantics: it converges to the *AR*-semantics for k sufficiently large and in the case when $k = 1$ it coincides with the *IAR*-semantics. The k-defeater semantics is a complete approximation of the *AR*-semantics: in the case when $k = 0$ it coincides with the "brave" version of the *AR*-semantics, and converges to the *AR*-semantics in the limit. The authors prove that, in the case of *DL-Lite* ontologies, query answering under both semantics is tractable (actually, first-order rewritable). These results in principle allow for improving the expressive abilities of the *IAR*-semantics while keeping its good computational properties. However, the applicability of this approach is still not

clear: in particular, practical query rewriting algorithms for the parameterized semantics have not been defined yet. This paper also presents a sufficient condition for the FO-rewritability of conjunctive queries under the above parameterized semantics, which includes, as a special case, query answering in the logic *DL-Lite*$_{A,id,den}$ under the *IAR*-semantics.

Then, [52] presents an approach that is also very close to the present work. In fact, it proposes an application of the *IAR*-semantics to the framework of Datalog+/- [17]. In particular, a fragment of Datalog+/- is considered that comprises linear tuple-generating dependencies (TGDs), negative constraints (NCs) and a restricted form of equality-generating dependencies (EGDs). Then, a technique for query answering under such dependencies is presented, which is technically very similar to the one presented in Section 7. In fact, the dependencies considered in [52] have a tight connection with the TBox assertions of *DL-Lite*$_{A,id,den}$, in particular: linear TGDs correspond to a generalized form of positive inclusion assertions, NCs correspond to denial assertions, and EGDs are related to identification assertions. However, there are two main differences: first, the framework of [52] does not consider value-domains, hence many of the issues dealt with by our technique are not present in the above Datalog+/- framework; moreover, although identification constraints correspond to EGDs, they are not captured by the restricted form of EGDs (called non-conflicting EGDs) considered in [52]. The work of [52] has been further extended in [53], which presents a set of complexity results about reasoning in Datalog+/- programs under both the *IAR*-semantics and the *ICR*-semantics.

Furthermore, in [67] an algorithm for materializing the IAR-repair of a *DL-Lite*$_A$ KB is defined. This work also presents an experimental comparison in *DL-Lite*$_A$ of the query rewriting approach and the materialized approach to query answering under the *IAR* semantics. A similar comparison for the framework studied here, although very interesting in principle, is outside the scope of the present paper, and is left for future work (cf. Section 10). We only note that, ABox repairs in *DL-Lite*$_A$ appear much easier to compute than in the case of *DL-Lite*$_{A,id,den}$, because minimal inconsistent subsets have a cardinality of at most 2, which implies that computing IAR-repairs in *DL-Lite*$_A$ is in PTime with respect to combined complexity. Conversely, in *DL-Lite*$_{A,id,den}$, due to the presence of denial assertions and identification assertions, the minimal inconsistent subsets of the ABox may have size larger than 2 (only bounded by the length of the above assertions). By virtue of this property, it can easily be shown that computing IAR-repairs in *DL-Lite*$_{A,id,den}$ cannot be done in polynomial time with respect to combined complexity, unless PTime = NP. For these reasons, the query rewriting techniques considered in [67] are significantly different from the one presented in this paper. Finally, as observed in [67], the ABox cleaning approach might not always be possible or easily realizable in real applications, especially in OBDA scenarios where the ABox is virtually defined through views over autonomous databases. In these cases, the OBDA system can typically only read such databases. Another reason that may prevent the applicability of the ABox cleaning approach is that several organizations do not want their data to be replicated, thus preventing from building a cleaned replica

of data. Indeed, data replication comes with the additional cost due to the need of keeping the replica up-to-date with respect to the original copy.

10. Discussion and conclusion

In this paper, we have addressed the problem of inconsistency tolerance for DLs of the *DL-Lite* family, particularly suited for ontology representation in an OBDA scenario, where data at the sources to be accessed, stored in autonomous and independent systems, typically contradict the conceptualization provided by the ontology. We have therefore analyzed inconsistency-tolerant semantics suited to deal with this situation, i.e., semantics that preserve the knowledge provided by the ontology TBox, considered a faithful representation of the domain, and repair ABox data to make them consistent with the TBox. We have first studied query answering under the *AR*-semantics, a natural adaptation to DLs of repair-based semantics proposed for incomplete and inconsistent databases, and have shown that this task is intractable in data complexity even for very simple queries and DL languages of limited expressive power. We have therefore proposed an approximation of the *AR*-semantics, called *IAR*-semantics, with the goal of reaching a compromise between the expressive power of the semantics, the ontology and the query language, and the computational complexity of inconsistency-tolerant query answering. We have then shown that query answering under such semantics is first-order rewritable for one of the most expressive logic of the *DL-Lite* family, and provided some first experiments involving ontologies with large inconsistent ABoxes, based on an implementation of our rewriting algorithm.

As for future work, we plan to continue the work on the *IAR*-semantics in different ways. First, we have implemented the query rewriting algorithm presented in Section 7 in the MASTRO system [21], with the goal of testing the algorithm in real-world scenarios. Such tests will allow us to investigate in detail suitable strategies for optimizing the rewritten query produced by the algorithm.

Also, we plan to experiment our technique in OBDA settings where the relationship between the ontology and the data sources is expressed in terms of suitable mappings, rather than a DL ABox. In this setting, the goal of devising an efficient inconsistency-tolerant query answering technique is complicated by the fact that the rewritten query produced by our algorithm should be the input to a further rewriting step that takes into account the mappings.

Finally, we would like to investigate approaches that materialize instance-level repairs in $\textit{DL-Lite}_{A,id,den}$. In particular, it would be very interesting to define techniques for ABox cleaning in $\textit{DL-Lite}_{A,id,den}$ and to extend the experimental comparison presented in [67] between virtual and materialized approaches to query answering under the *IAR*-semantics.

Acknowledgments. This research has been partially supported by the EU under FP7 project Optique – Scalable End-user Access to Big Data (grant n. FP7-318338).

References

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

[2] N. Antonioli, F. Castanò, C. Civili, S. Coletta, S. Grossi, D. Lembo, M. Lenzerini, A. Poggi, D. F. Savo, and E. Virardi. Approximate rewriting of queries using views. In *Proc. of the Industrial Track of the 25th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2013)*, volume 1017 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, pages 9–16, 2013.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

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

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2007.

[6] F. Baader and R. Peñaloza. Axiom pinpointing in general tableaux. *J. of Logic and Computation*, 20(1):5–34, 2010.

[7] K. Baclawski, M. M. Kokar, R. Waldinger, and P. A. Kogut. Consistency checking of semantic web ontologies. In *Proc. of the 1st Int. Semantic Web Conf. (ISWC 2002)*, 2002.

[8] L. E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[9] L. E. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4–5):407–434, 2008.

[10] L. E. Bertossi, A. Hunter, and T. Schaub, editors. *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*. Springer, 2005.

[11] M. Bienvenu. First-order expressibility results for queries over inconsistent *DL-Lite* knowledge bases. In *Proc. of the 24th Int. Workshop on Description Logic (DL 2011)*, volume 745 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2011.

[12] M. Bienvenu. Inconsistency-tolerant conjunctive query answering for simple ontologies. In *Proc. of the 25th Int. Workshop on Description Logic (DL 2012)*, volume 846 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2012.

[13] M. Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI 2012)*, 2012.

[14] M. Bienvenu and R. Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013)*, 2013.

[15] M. Bouzeghoub and M. Lenzerini. Introduction to the special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 26(8):535–536, 2001.

[16] A. Calì, G. Gottlob, and 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)*, pages 77–86, 2009.

[17] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. of Web Semantics*, 14:57–83, 2012.

[18] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2003)*, pages 260–271, 2003.

[19] A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 16–21, 2003.

[20] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and databases: The *DL-Lite* approach. *Reasoning Web. Semantic Technologies for Information Systems*, pages 255–356, 2009.

[21] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The Mastro system for ontology-based data access. *Semantic Web J.*, 2(1):43–53, 2011.

[22] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[23] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-based identification constraints in description logics. In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 231–241, 2008.

[24] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Conceptual modeling for data integration. In A. T. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, editors, *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*, volume 5600 of *Lecture Notes in Computer Science*, pages 173–197. Springer, 2009.

[25] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. *Artificial Intelligence*, 195:335–360, 2013.

[26] J. Chomicki. Consistent query answering: Five easy pieces. In *Proc. of the 11th Int. Conf. on Database Theory (ICDT 2007)*, volume 4353 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2007.

[27] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1–2):90–121, 2005.

[28] J. Chomicki and J. Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In L. Bertossi, A. Hunter, and T. Schaub, editors, *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 119–150. Springer, 2005.

[29] F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proc. of the 16th Int. Conf. on Extending Database Technology (EDBT 2013)*, pages 561–572, 2013.

[30] T. Eiter, M. Fink, G. Greco, and D. Lembo. Repair localization for query answering from inconsistent databases. *ACM Trans. on Database Systems*, 33(2), 2008.

[31] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[32] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *Proc. of the 2nd ACM SIGACT SIGMOD Symp. on Principles of Database Systems (PODS'83)*, pages 352–365, 1983.

[33] A. Furmann. Theory contraction through base contraction. *J. of Philosophical Logic*, 20:175–203, 1991.

[34] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. *J. of Computer and System Sciences*, 73(4):610–635, 2007.

[35] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *J. of Artificial Intelligence Research*, 31:151–198, 2008.

[36] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.

[37] L. Grieco, D. Lembo, M. Ruzzi, and R. Rosati. Consistent query answering under key and exclusion dependencies: Algorithms and experiments. In *Proc. of the 14th Int. Conf. on Information and Knowledge Management (CIKM 2005)*, pages 792–799, 2005.

[38] P. Haase, F. van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure. A framework for handling inconsistency in changing ontologies. In *Proc. of the 4th Int. Semantic Web Conf. (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 353–367. Springer, 2005.

[39] M. Horridge, B. Parsia, and U. Sattler. Explaining inconsistencies in owl ontologies. In *Proc. of the 3rd Int. Conf. on Scalable Uncertainty Management (SUM 2009)*, pages 124–137. Springer, 2009.

[40] Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 454–459, 2003.

[41] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[42] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation, World Wide Web Consortium, Feb. 2004. Available at `http://www.w3.org/TR/rdf-concepts/`.

[43] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and 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)*, pages 247–257, 2010.

[44] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *Proc. of the 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002)*, volume 54 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2002.

[45] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. Savo. Inconsistency-tolerant first-order rewritability of DL-Lite with identification and denial assertions. In *Proc. of the 25th Int. Workshop on Description Logic (DL 2012)*, volume 846 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2012.

[46] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Inconsistency-tolerant semantics for description logics. In *Proc. of the 4th Int. Conf. on Web Reasoning and Rule Systems (RR 2010)*, 2010.

[47] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Query rewriting for inconsistent *DL-Lite* ontologies. In *Proc. of the 5th Int. Conf. on Web Reasoning and Rule Systems (RR 2011)*, 2011.

[48] D. Lembo and M. Ruzzi. Consistent query answering over description logic ontologies. In *Proc. of the 1st Int. Conf. on Web Reasoning and Rule Systems (RR 2007)*, 2007.

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

[50] M. Lenzerini. Ontology-based data management: present and future. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012)*, 2012. Invited talk.

[51] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. of Cooperative Information Systems*, 7(1):55–76, 1998.

[52] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Inconsistency-tolerant query rewriting for linear datalog+/-. In *Datalog 2.0*, volume 7494 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2012.

[53] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Complexity of inconsistency-tolerant query answering in datalog+/-. In *Proc. of the 26th Int. Workshop on Description Logic (DL 2013)*, pages 791–803, 2013.

[54] Y. Ma and P. Hitzler. Paraconsistent reasoning for OWL 2. In *Proc. of the 3rd Int. Conf. on Web Reasoning and Rule Systems (RR 2009)*, pages 197–211, 2009.

[55] Y. Ma, P. Hitzler, and Z. Lin. Algorithms for paraconsistent reasoning with OWL. In *Proc. of the 4th European Semantic Web Conf. (ESWC 2007)*, pages 343–357, 1997.

[56] G. Masotti, R. Rosati, and M. Ruzzi. Practical ABoxes cleaning in *DL-Lite* (progress report). In *Proc. of the 24th Int. Workshop on Description Logic (DL 2011)*, volume 745 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2011.

[57] T. Meyer, K. Lee, and R. Booth. Knowledge integration for description logics. In *Proc. of the 7th Int. Symposium on Logical Formalizations of Commonsense Reasoning*, volume 20, pages 645–650, 2005.

[58] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language – Profiles (second edition). W3C Recommendation, World Wide Web Consortium, Dec. 2012. Available at `http://www.w3.org/TR/owl2-profiles/`.

[59] B. Nebel. Belief revision and default reasoning: syntax-based approaches. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pages 417–428, 1991.

[60] B. Parsia, E. Sirin, and A. Kalyanpur. Debugging OWL ontologies. In *Proc. of the 14th Int. World Wide Web Conf. (WWW 2005)*, pages 633–640, 2005.

[61] P. F. Patel-Schneider. Adding number restrictions to a four-valued terminological logic. In *Proc. of the 7th Nat. Conf. on Artificial Intelligence (AAAI'88)*, pages 485–490, 1988.

[62] R. Peñaloza and B. Sertkaya. Complexity of axiom pinpointing in the *DL-Lite* family. In *Proc. of the 23rd Int. Workshop on Description Logic (DL 2010)*, volume 573 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2010.

[63] R. Peñaloza and B. Sertkaya. Complexity of axiom pinpointing in the *DL-Lite* family of description logics. In *Proc. of the 19th Eur. Conf. on Artificial Intelligence (ECAI 2010)*, pages 29–34, 2010.

[64] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[65] G. Qi, W. Liu, and D. Bell. A revision-based approach to handling inconsistency in description logics. *Artificial Intelligence Review*, 26(1):115–128, 2006.

[66] R. Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*, pages 1057–1062, 2011.

[67] R. Rosati, M. Ruzzi, M. Graziosi, and G. Masotti. Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies. In *Proc. of the 11th Int. Semantic Web Conf. (ISWC 2012)*, volume 7650, pages 337–349, 2012.

[68] D. F. Savo. *Dealing with Inconsistencies and Updates in Description Logic Knowledge Bases.* PhD thesis, Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti Sapienza University of Rome, 2013. http://hdl.handle.net/10805/1986.

[69] D. F. Savo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodríguez-Muro, V. Romagnoli, M. Ruzzi, and G. Stella. MASTRO at work: Experiences on ontology-based data access. In *Proc. of the 23rd Int. Workshop on Description Logic (DL 2010)*, volume 573 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, pages 20–31, 2010.

[70] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of descritpion logic terminologies. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 355–360, 2003.

[71] S. Staworko and J. Chomicki. Consistent query answers in the presence of universal constraints. *Information Systems*, 35(1):1–22, 2010.

[72] U. Straccia. A sequent calculus for reasoning in four-valued description logics. In *Proc. of the 6th Int. Conf. on on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'97)*, pages 343–357, 1997.

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

[74] J. Wijsen. Database repairing using updates. *ACM Trans. on Database Systems*, 30(3):722–768, 2005.

[75] J. Wijsen. Charting the tractability frontier of certain conjunctive query answering. In *Proc. of the 32nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2013)*, pages 189–200, 2013.

[76] M. Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.

Appendix 1

In this appendix we recall the algorithm PerfectRef for query rewriting of UCQs over *DL-Lite* TBoxes. This algorithm has been originally presented in [22] for *DL-Lite* logics without attributes, and then used as is in [20] for *DL-Lite*$_A$. Indeed, as clarified below, by virtue of the assumptions on value-domains adopted in *DL-Lite*$_A$ (and its variants), for query rewriting it is possible to treat *DL-Lite*$_A$ TBoxes as *DL-Lite* TBoxes without attributes, provided some obvious syntactic transformations. In this paper, we assume that it takes as input a *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$ and a UCQ Q.

PerfectRef returns the perfect rewriting of Q w.r.t. $\mathcal{T}$ (cf. Proposition 1). To compute such rewriting, it exploits only the positive inclusion assertions in the TBox. In a *DL-Lite*$_{A,id,den}$ TBox, these are the assertions in $\mathcal{T}_{inc}$ and $\mathcal{T}_{type}$. In particular, PerfectRef treats predefined value-domains as atomic concepts, attributes as atomic roles, and each assertion in $\mathcal{T}_{type}$ as an inclusion between an unqualified existential restriction over a role and an atomic concept. In other words, in PerfectRef concepts of the form $\delta(U)$ are seen as concepts of the form $\exists U$, and value-domains of the form $\rho(U)$ are seen as concepts of the form $\exists U^-$. This is possible by virtue of the assumptions made on value-domains (cf. Section 2.1), through which the distinction between objects and values does not have any impact on query answering. Thus, in PerfectRef we do not need to introduce any special treatment for attributes and value-domains. We only notice that, since input queries can contain atoms of the form $T_i(t)$, with T_i value-domain in Γ_P and t a term, we also make use of inclusions of the form $\rho(U) \sqsubseteq T_i$ (interpreted as $\exists U^- \sqsubseteq T_i$) to rewrite the query.

PerfectRef (cf. Algorithm 6) consists of two main steps, applied repeatedly to each $q \in Q$ until a fixpoint is reached: step (a) uses positive inclusions as rewriting rules applied from right to left, to rewrite query atoms one by one, each time producing a new conjunctive query to be added to the final rewriting; step (b) unifies query atoms. The aim of step (b) is to make step (a) executable over atoms resulting from unification. At the end of the process, all the knowledge of the TBox that is necessary to answer q is encoded in the returned rewriting, which can be then evaluated over the underlying ABox to return the answer.

In the algorithm, $q[g/g']$ denotes the CQ obtained from a CQ q by replacing the atom g with a new atom g'. Also, $gr(g, \alpha)$ denotes the atom obtained from the atom g by applying the inclusion α. Roughly speaking, an inclusion α is applicable to an atom g if: (i) the predicate of g is equal to the predicate in the right-hand side of α; (ii) in the case when α is an inclusion assertion between concepts, the atom g has at most one bound argument, which corresponds to the object that is implicitly referred to by the inclusion α, where an argument is bound if it is a constant or a variable occurring elsewhere in the query (we refer to [22, 20] for further details). In PerfectRef, we substitute each non-bound variable with the symbol $\_$. For example, the inclusion $A \sqsubseteq \exists P^-$ is not applicable to any of the atoms in $\exists x, y.P(x, y) \wedge P(x', y) \wedge x \neq x'$, whereas the inclusion $\exists P_1 \sqsubseteq \exists P_2$ is applicable to the atom $P_2(x, z)$ in the query $\exists x.P_3(x, \_), P_2(x, \_)$. In this case, we have $gr(P_2(x, \_), \exists P_1 \sqsubseteq \exists P_2) = P_1(x, \_)$. Again, we refer

Input: UCQ Q, *DL-Lite*$_{A,id,den}$ TBox $\mathcal{T}$
Output: UCQ
begin
  $PR \leftarrow \{Q\}$;
  repeat
    $PR' \leftarrow PR$;
    for each $q \in PR'$ **do**
    (a) **for each** g in q **do**
        for each positive inclusion α in $\mathcal{T}$ **do**
          if α is applicable to g
          then $PR \leftarrow PR \cup \{\, q[g/gr(g,\alpha)] \,\}$
    (b) **for each** g_1, g_2 in q **do**
        if g_1 and g_2 unify
        then $PR \leftarrow PR \cup \{\tau(reduce(q, g_1, g_2))\}$;
  until $PR' = PR$;
  return PR;
end

Algorithm 6: The algorithm PerfectRef

to [22, 20] for the exact definition of the function *gr*. Finally, *reduce* is a function that takes as input a conjunctive query 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, whereas the function τ substitutes with the symbol _ each non-bound variable in the query it takes as argument.

Appendix 2

The goal of this appendix is to present the proof of Theorem 1. To show the only-if direction, we construct a *DL-Lite*$_R$ KB that encodes the positive knowledge specified in a *DL-Lite*$_{A,id,den}$ KB. Then we make use of the notions of chase and canonical structure for *DL-Lite*$_R$ KBs, and exploit some of their basic properties. Such notions and properties have been originally given in [22]. For the sake of completeness, we repeat them below[8].

The canonical structure for DL-Lite$_R$ *KBs*

We recall that a *DL-Lite*$_R$ KB can be seen as a particular *DL-Lite*$_{A,id,den}$ KB without attributes and value-domains, where functionalities, identifications, and denials are not allowed. We assume *DL-Lite*$_R$ KBs to be constructed over an overall alphabet of predicates $\Gamma_P^R = \Gamma_A^R \cup \Gamma_L^R$, where Γ_A^R is the alphabet of

[8] Canonical structures are called canonical interpretations in [22]. Since, as we will see in the following, they are not interpretations for *DL-Lite*$_{A,id,den}$ KBs, in this paper we prefer to use the term structure.

atomic concepts and Γ_L^R is the alphabet of atomic roles, and an alphabet of (object) constants Γ_C^R, such that with $\Gamma_A^R \cap \Gamma_L^R = \emptyset$ and $\Gamma_P^R \cap \Gamma_C^R = \emptyset$.

First of all we provide the notion of structure induced by a set of ABox assertions, which in *DL-Lite*$_R$ are only of form $A(a)$ and $P(a, b)$. Notice that this definition is a specialization to *DL-Lite*$_R$ of Definition 2, which considers *DL-Lite*$_{A,id,den}$ ABox assertions. Differently from Definition 2, below we do not provide any special interpretation of value constants, value-domains and attributes, which do not exists in *DL-Lite*$_R$. For the rest, the two definitions coincide.

Definition 13. *Let $\mathcal{S}$ be a set of DL-Lite$_R$ ABox assertions. The structure $DB_R(\mathcal{S}) = \langle \Delta^{DB_R(\mathcal{S})}, \cdot^{DB_R(\mathcal{S})} \rangle$ is defined as follows:*

- $\Delta^{DB_R(\mathcal{S})}$ *coincides with the set of constants occurring in* $\mathcal{S}$,
- $a^{DB_R(\mathcal{S})} = a$, *for each constant* a *occurring in* $\mathcal{S}$,
- $A^{DB_R(\mathcal{S})} = \{a \mid A(a) \in \mathcal{S}\}$, *for each atomic concept* $A \in \Gamma_A^R$,
- $P^{DB_R(\mathcal{S})} = \{(a_1, a_2) \mid P(a_1, a_2) \in \mathcal{S}\}$, *for each atomic role* $P \in \Gamma_L^R$.

The canonical structure of a KB in *DL-Lite*$_R$ is constructed according to the *chase* of the KB [1, 41]. We thus first recall the notion of chase, and to this aim we start by defining the notion of applicable positive inclusion assertions (PIs), which in *DL-Lite*$_R$ are only of form $B_1 \sqsubseteq B_2$ and $R_1 \sqsubseteq R_2$. In the following, similar to Section 3, we make use of the function η that takes as input a basic role and two constants and returns an ABox assertion, as specified below

$$\eta(R, a, b) = \begin{cases} R(a, b), & \text{if } R = P \\ R(b, a), & \text{if } R = P^- \end{cases}$$

Definition 14. *Let $\mathcal{S}$ be a set of DL-Lite$_R$ ABox assertions, and let $\mathcal{T}_p$ be a set of DL-Lite$_R$ PIs. Then, a PI $\alpha \in \mathcal{T}_p$ is* applicable in $\mathcal{S}$ to an ABox assertion $\beta \in \mathcal{S}$ *if*

- $\alpha = A_1 \sqsubseteq A_2$, $\beta = A_1(a)$, *and* $A_2(a) \notin \mathcal{S}$;
- $\alpha = A \sqsubseteq \exists R$, $\beta = A(a)$, *and there does not exist any constant* b *such that* $\eta(R, a, b) \in \mathcal{S}$;
- $\alpha = \exists R \sqsubseteq A$, $\beta = \eta(R, a, b)$, *and* $A(a) \notin \mathcal{S}$;
- $\alpha = \exists R_1 \sqsubseteq \exists R_2$, $\beta = \eta(R_1, a, b)$, *and there does not exist any constant* c *such that* $\eta(R_2, a, c) \in \mathcal{S}$;
- $\alpha = R_1 \sqsubseteq R_2$, $\beta = \eta(R_1, a, b)$, *and* $\eta(R_2, a, b) \notin \mathcal{S}$.

□

Applicable PIs can be used, i.e., *applied*, in order to construct the chase of a KB. In the following we assume to have an infinite set Γ_N of constant symbols disjoint from the alphabets Γ_C^R and Γ_P^R. Then, our notion of chase is defined inductively as follows.

Definition 15. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_R$ KB. We define $chase_0(\mathcal{K}) = \mathcal{A}$. Then, for every non-negative integer i, let β be an ABox assertion in $chase_i(\mathcal{K})$, let α be a PI in $\mathcal{T}$ such that α is applicable to β in $chase_i(\mathcal{K})$, and let a_{new} be a fresh constant from Γ_N, i.e., a constant not occurring in $chase_i(\mathcal{K})$, we define $chase_{i+1}(\mathcal{K}) = chase_i(\mathcal{K}) \cup \beta_{new}$, where β_{new} is an ABox assertion defined as follows*

- *if $\alpha = A_1 \sqsubseteq A_2$, $\beta = A_1(a)$ then $\beta_{new} = A_2(a)$*
- *if $\alpha = A \sqsubseteq \exists R$ and $\beta = A(a)$ then $\beta_{new} = \eta(R, a, a_{new})$*
- *if $\alpha = \exists R \sqsubseteq A$ and $\beta = \eta(R, a, b)$ then $\beta_{new} = A(a)$*
- *if $\alpha = \exists R_1 \sqsubseteq \exists R_2$ and $\beta = \eta(R_1, a, b)$ then $\beta_{new} = \eta(R_2, a, a_{new})$*
- *if $\alpha = R_1 \sqsubseteq R_2$ and $\beta = \eta(R_1, a, b)$ then $\beta_{new} = \eta(R_2, a, b)$.*

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

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

□

We notice that, given a *DL-Lite*$_R$ KB $\mathcal{K}$, a number of syntactically distinct chases for $\mathcal{K}$ can be obtained according to the above definition, depending on the order with which applicable PIs are applied in each $chase_i(\mathcal{K})$. For ease of exposition, in Definition 15 we did not explicitly indicate the ordering to follow in the application of the PIs, but assume that it is implicitly established an ordering that guarantees that if a PI α is applicable to an ABox assertion β in a certain $chase_i(\mathcal{K})$, then there exists $j > i$ such that α is no longer applicable to β in $chase_j(\mathcal{K})$. It can be shown that all chases produced adopting an ordering that guarantees the above property are unique up to renaming of constants from Γ_N (we refer the reader to [22, 41] for more details on these aspects).

With the notion of chase in place, we can define the *canonical structure* $can(\mathcal{K})$ of a *DL-Lite*$_R$ KB $\mathcal{K}$ as the strucure induced by $chase(\mathcal{K})$ according to Definition 13, i.e., $can(\mathcal{K}) = DB_R(chase(\mathcal{K}))$.

The following lemmas adapted from [22] state that $can(\mathcal{K})$ satisfies all PIs in a *DL-Lite*$_R$ KB $\mathcal{K}$ and that $can(\mathcal{K})$ is indeed a canonical model for query answering.

Lemma 9 (Lemma 7 of [22])**.** *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_R$ KB and let $\mathcal{T}_p$ be the set of PIs in $\mathcal{T}$. Then, $can(\mathcal{K})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$.*

Lemma 10 (Theorem 29 of [22])**.** *Let $\mathcal{K}$ be a satisfiable DL-Lite$_R$ KB, and let Q be a UBCQ over $\mathcal{K}$. Then, $\mathcal{K} \models Q$ if and only if $can(\mathcal{K}) \models Q$.*

Consider now *DL-Lite*$_R$ KBs equipped also with role functionality assertions, with the proviso that functional roles are not specialized (i.e., they do not occur in the right-hand side of role inclusions). Notice that KBs of this form are *DL-Lite*$_A$ KBs without attributes and value-domains (cf. the third condition in Definition 1), and we call them *DL-Lite*$_{FR}$ KBs. In such KBs, PIs and ABox assertions are exactly as in *DL-Lite*$_R$ KBs. Therefore, Definition 13 and Definition 15 apply also to *DL-Lite*$_{FR}$ KBs. The following lemma, adapted from [20], will be used in the proof of Theorem 1.

Lemma 11 (Lemma 4.6 of [20]). *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{FR}$ KB, and let $\mathcal{T}_{funct}$ be the set of functionality assertions in $\mathcal{T}$. Then, can($\mathcal{K}$) is a model of $\langle \mathcal{T}_{funct}, \mathcal{A} \rangle$ if and only if $DB_R(\mathcal{A})$ is a model of $\langle \mathcal{T}_{funct}, \mathcal{A} \rangle$.*

Similarly to what we have done before, we consider now *DL-Lite*$_R$ KBs equipped also with identification assertions, with the proviso that each role appearing (in either direct or inverse direction) in a path of an identification assertion is not specialized. Notice that KBs of this form are *DL-Lite*$_{A,id}$ KBs without attributes, value-domains and functionalities on roles (cf. the second condition in Definition 1), and we call them *DL-Lite*$_{R,id}$ KBs. Again, in such KBs, PIs and ABox assertions are exactly as in *DL-Lite*$_R$ KBs. Therefore, Definition 13 and Definition 15 apply also to *DL-Lite*$_{R,id}$ KBs. The following lemma, adapted from [20], will be used in the proof of Theorem 1.

Lemma 12 (Lemma 5.19 of [20]). *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{R,id}$ KB. Then, can($\mathcal{K}$) is a model of $\mathcal{K}$ if and only if $\mathcal{K}$ is satifsfiable.*

Proof of Theorem 1

Before attacking the proof, we first need to mention a property of the algorithm PerfectRefIdC that has been proved in [20], and that we specialize below to the case of *DL-Lite*$_{R,id}$ KBs.

Lemma 13 (Theorem 5.20 of [20]). *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_{R,id}$ KB, let $\mathcal{T}_p$ be the set of PIs in $\mathcal{T}$ and $\mathcal{T}_{id}$ be the set of identification assertions in $\mathcal{T}$, and let $q_{\mathcal{T}_{id}} = \bigcup_{\alpha \in \mathcal{T}_{id}} \{\varphi(\alpha)\}$. Then, $\mathcal{K}$ is satisfiable if and only if $DB_R(\mathcal{A}) \not\models$ PerfectRefIdC$(q_{\mathcal{T}_{id}}, \mathcal{T}_p)$.*

We have now all the ingredients needed for our proof.

Theorem 1. *A DL-Lite$_{A,id,den}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable if and only if $DB(\mathcal{A}) \models$ UnsatQuery$(\mathcal{T})$.*

Proof. ($\Rightarrow$) Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_{A,id,den}$ KB and assume that $DB(\mathcal{A}) \not\models$ UnsatQuery$(\mathcal{T})$. To prove the claim we first construct a *DL-Lite*$_R$ KB $\mathcal{K}^R = \langle \mathcal{T}^R, \mathcal{A}^R \rangle$, with the aim of encoding in $\mathcal{K}^R$ the positive knowledge specified in $\mathcal{K}$; then, based on the canonical structure of $\mathcal{K}^R$, we identify a structure $\mathcal{I}$ and prove that $\mathcal{I}$ is a model of $\mathcal{K}$.

We start by defining the *DL-Lite*$_R$ KB $\mathcal{K}^R = \langle \mathcal{T}^R, \mathcal{A}^R \rangle$. We call Γ_A, Γ_L, Γ_T, Γ_U, Γ_O, and Γ_V the pair-wise disjoint alphabets, used respectively for atomic concepts, atomic roles, value-domains, attributes, object constants, and value

constants, over which the *DL-Lite*$_{A,id,den}$ KB $\mathcal{K}$ is specified. As said, we instead call Γ_A^R, Γ_L^R, and Γ_C^R the pair-wise disjoint alphabets for atomic concepts, atomic roles, and (object) constants over which the *DL-Lite*$_R$ KB $\mathcal{K}^R$ is specified, and have that $\Gamma_A^R = \Gamma_A \cup \Gamma_T$, $\Gamma_L^R = \Gamma_L \cup \Gamma_U$, and $\Gamma_C^R = \Gamma_O \cup \Gamma_V$. In other terms, in $\mathcal{K}^R$ we consider predefined value-domains in $\mathcal{K}$ as atomic concepts and attributes in $\mathcal{K}$ as atomic roles. Consequently, inclusions involving attributes and value-domains in $\mathcal{K}$ as treated as concept and role inclusions, respectively. More precisely, $\mathcal{T}^R$ contains all the inclusions in the sets $\mathcal{T}_{inc}$ and $\mathcal{T}_{type}$ of $\mathcal{T}$ provided that each occurrence of $\delta(U)$ is replaced by $\exists U$ and each occurrence of $\rho(U)$ is replaced by $\exists U^-$. Also, $\mathcal{A}^R$ contains all the assertions in $\mathcal{A}$, plus an assertion of the form $T_i(v)$ for each value constant $v \in \Gamma_V$ and value-domain $T_i \in \Gamma_T$ such that $T_i = type(v)$.

Consider now the structure $DB_R(\mathcal{A}^R)$ induced by $\mathcal{A}^R$ according to Definition 13 and the *DL-Lite*$_{A,id,den}$ interpretation $DB(\mathcal{A})$ induced by $\mathcal{A}$ according to Definition 2. It is easy to see that $DB_R(\mathcal{A}^R)$ and $DB(\mathcal{A})$ are two isomorphic structures, since they are identical, modulo renaming objects interpreting value constants in Γ_V.

Then, we construct the chase of $\mathcal{K}^R$ according to Definition 15, and consider the canonical structure $can(\mathcal{K}^R)$ induced by the chase. We remark that $can(\mathcal{K}^R)$ is an interpretation for $\mathcal{K}^R$ but it is not an interpretation of $\mathcal{K}$, since it does not obey the assumptions on the interpretation of value-domains we impose in *DL-Lite*$_{A,id,den}$ KBs (cf. Section 2.1). We will show in the following how to construct from $can(\mathcal{K}^R)$ one such interpretation that is a model for $\mathcal{K}$, thus proving that $\mathcal{K}$ is satisfiable.

We first show that the assumption $DB(\mathcal{A}) \not\models \mathsf{UnsatQuery}(\mathcal{T})$ implies that $can(\mathcal{K}^R)$ satisfies all the assertions in $\mathcal{T}$. In doing this, we can disregard for the moment the fact that $can(\mathcal{K}^R)$ is not an interpretation for $\mathcal{K}$.

Satisfaction of assertions in $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$ by $can(\mathcal{K}^R)$ follows from Lemma 9. Indeed, such lemma states that $can(\mathcal{K}^R)$ satisfies all PIs in $\mathcal{T}^R$, which correspond to those in $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$ modulo the syntactical transformations described above. Notice that above property implies that $\mathcal{K}^R$ is satisfiable.

To show satisfaction of functionalities in $\mathcal{T}_{funct}$ we exploit the assumption that $DB(\mathcal{A}) \not\models \mathsf{UnsatQuery}(\mathcal{T})$, which implies that $DB(\mathcal{A}) \not\models \mathsf{PerfectRefIdC}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$ for each $\alpha \in \mathcal{T}_{funct}$. Since all arguments in the atoms of $\varphi(\alpha)$ are bound, and since functional roles and attributes are never specialized in $\mathcal{T}$, no PIs exist in $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$ that PerfectRefIdC can apply to atoms in $\varphi(\alpha)$ (cf. Section 3). Also, PerfectRefIdC cannot unify atoms, since such unification would involve terms occurring in an inequality atom. As a consequence, in this case $DB(\mathcal{A}) \not\models \varphi(\alpha)$. By the isomorphism between $DB(\mathcal{A})$ and $DB_R(\mathcal{A}^R)$ it follows that $DB_R(\mathcal{A}^R) \not\models \varphi(\alpha)$ for each $\alpha \in \mathcal{T}_{funct}$. This means that $DB_R(\mathcal{A}^R)$ is a model of $\langle \mathcal{T}_{funct}, \mathcal{A}^R \rangle$, and thus, since $\langle \mathcal{T}^R \cup \mathcal{T}_{funct}, \mathcal{A}^R \rangle$ is a *DL-Lite*$_{FR}$ KB, from Lemma 11 it follows that $can(\mathcal{K}^R)$ satisfies all the functionalities in $\mathcal{T}_{funct}$.

We now show that $can(\mathcal{K}^R)$ satisfies the assertions in $\mathcal{T}_{disj}$. From $DB(\mathcal{A}) \not\models \mathsf{UnsatQuery}(\mathcal{T})$, and from the fact that, for each disjointness α, $\varphi(\alpha)$ is a CQ without inequalities, it follows that $DB(\mathcal{A}) \not\models \mathsf{PerfectRef}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$

for each $\alpha \in \mathcal{T}_{disj}$. By the isomorphism between $DB(\mathcal{A})$ and $DB_R(\mathcal{A}^R)$ and the correspondence between $\mathcal{T}^R$ and $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$, we then have that $DB_R(\mathcal{A}^R) \not\models \mathsf{PerfectRef}(\varphi(\alpha), \mathcal{T}^R)$ for each $\alpha \in \mathcal{T}_{disj}$. From Proposition 1 (notice that *DL-Lite*$_R$ KBs are particular *DL-Lite*$_{A,id,den}$ KBs, and that $\mathcal{K}^R$ is satisfiable) we have that $\mathcal{K}^R \not\models \varphi(\alpha)$ for each $\alpha \in \mathcal{T}_{disj}$. From Lemma 10 it follows that $can(\mathcal{K}^R) \not\models \varphi(\alpha)$ for each $\alpha \in \mathcal{T}_{disj}$, from which it follows that $can(\mathcal{K}^R)$ satisfies all the assertions in $\mathcal{T}_{disj}$.

As for denials, which analogously to negative inclusions are associated to BCQs by the function φ, we can proceed exactly as done above for assertions in $\mathcal{T}_{disj}$, and thus show that $can(\mathcal{K}^R)$ satisfies all the assertions in $\mathcal{T}_{den}$.

We now prove that $can(\mathcal{K}^R)$ satisfies the assertions in $\mathcal{T}_{id}$. From $DB(\mathcal{A}) \not\models \mathsf{UnsatQuery}(\mathcal{T})$ it follows that $DB(\mathcal{A}) \not\models \mathsf{PerfectRefIdC}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$ for each $\alpha \in \mathcal{T}_{id}$. By the isomorphism between $DB(\mathcal{A})$ and $DB_R(\mathcal{A}^R)$ and the correspondence between $\mathcal{T}^R$ and $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$ we have that $DB_R(\mathcal{A}^R) \not\models \mathsf{PerfectRefIdC}(q_{\mathcal{T}_{id}}, \mathcal{T}^R)$, where $q_{\mathcal{T}_{id}} = \bigcup_{\alpha \in \mathcal{T}_{id}} \{\varphi(\alpha)\}$. Since $\mathcal{K}^{R,id} = \langle \mathcal{T}^R \cup \mathcal{T}_{id}, \mathcal{A}^R \rangle$ is a *DL-Lite*$_{R,id}$ KB, then Lemma 13 states that $\mathcal{K}^{R,id}$ is satisfiable. From Lemma 12 it then follows that $can(\mathcal{K}^R)$ satisfies all the assertions in $\mathcal{T}_{id}$ (notice that $can(\mathcal{K}^R) = can(\mathcal{K}^{R,id})$).

We now turn back to our construction of a model for $\mathcal{K}$. Of course, we want this model to preserve satisfaction of all assertions in $\mathcal{T}$ proved for $can(\mathcal{K}^R)$, and at the same time be compliant with the assumptions of the interpretation of value-constants and value-domains in a *DL-Lite*$_{A,id,den}$ KB. We thus construct such a model starting from $can(\mathcal{K}^R)$ (and from $chase(\mathcal{K}^R)$). Similar to Section 7.1.1, we say that a constant c occurs in an object position in $chase(\mathcal{K}^R)$ if $chase(\mathcal{K}^R)$ contains an assertion of the form $A(c)$, $P(c, a)$, $P(a, c)$, or $U(c, v)$, and we say that c occurs in a value position if $chase(\mathcal{K}^R)$ contains an assertion of the form $U(a, c)$ or $T_i(c)$. It is easy to see that, due to the syntactic restrictions on the form of both ABox assertions and TBox assertions in *DL-Lite*$_{A,id,den}$, in $chase(\mathcal{K}^R)$ no constant exists that can occur in both an object and a value position. Then, we define a structure $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ as follows:

- $\Delta^{\mathcal{I}} = \Gamma_O \cup \Gamma_N \cup \Delta_V$,
- $a_o^{\mathcal{I}} = a_o^{can(\mathcal{K}^R)} = a_o$, for each constant $a_o \in \Gamma_O$,
- $a_{no}^{\mathcal{I}} = a_{no}^{can(\mathcal{K}^R)} = a_{no}$, for each constant $a_{no} \in \Gamma_N$ occurring in an object position in $chase(\mathcal{K}^R)$,
- $a_v^{\mathcal{I}} = val(a_v)$, for each constant $a_v \in \Gamma_V$,
- $a_{nv}^{\mathcal{I}} = o_v$, where o_v belongs to $\Delta_V \setminus \bigcup_{1=1}^{n} val(T_i)$ and o_v is a fresh object, i.e., it has not been used in $\mathcal{I}$ to interpret other constants occurring in $chase(\mathcal{K}^R)$, for each constant $a_{nv} \in \Gamma_N$ occurring in a value position such that $chase(\mathcal{K}^R)$ does not contain an atom of the form $T_i(a_{nv})$ (notice that one such o_v always exists, since $\Delta_V \setminus \bigcup_{1=1}^{n} val(T_i)$ is infinite, as stated in Section 2.1),
- $a_{nv}^{\mathcal{I}} = o_v$, where o_v belongs to $val(T_i)$ and is a fresh object, i.e., it has not been used in $can(\mathcal{K}^R)$ to interpret other constants occurring in $chase(\mathcal{K}^R)$, for each constant $a_{nv} \in \Gamma_N$ occurring in a value position such

that $T_i(a_{nv}) \in \mathit{chase}(\mathcal{K}^R)$ and T_i precedes in lexicographic order every other T_j such that $T_j \neq T_i$ and $T_j(a_{nv}) \in \mathit{chase}(\mathcal{K}^R)$ (notice that one such o_v always exists, since each $\mathit{val}(T_i)$ is infinite, as stated in Section 2.1),

- $A^{\mathcal{I}} = A^{can(\mathcal{K}^R)}$, for each atomic concept A,
- $P^{\mathcal{I}} = P^{can(\mathcal{K}^R)}$, for each atomic role P,
- $U^{\mathcal{I}} = \{(a^{\mathcal{I}}, a_v^{\mathcal{I}}) \mid U(a, a_v) \in \mathit{chase}(\mathcal{K}^R)\}$, for each attribute U,
- $T_i^{\mathcal{I}} = \mathit{val}(T_i)$, for each value-domain T_i.

It is easy to see that $\mathcal{I}$ is a *DL-Lite*$_{A,id,den}$ interpretation. Indeed, its domain is partitioned into two disjoint sets $\Gamma_O \cup \Gamma_N$ and Δ_V, used to interpret object constants and value constants, respectively. Furthermore, each $a_v \in \Gamma_V$ is such that $a_v^{\mathcal{I}} = \mathit{val}(a_v)$ and each value-domain $T_i \in \Gamma_T$ is such that $T_i^{\mathcal{I}} = \mathit{val}(T_i)$. However, there can be assertions of the form $T_i(c)$ in $\mathit{chase}(\mathcal{K}^R)$ such that $c^{\mathcal{I}} \notin T_i^{\mathcal{I}}$, for some value-domain T_i. Therefore, differently from $\mathit{can}(\mathcal{K}^R)$, we cannot directly conclude that $\mathcal{I}$ satisfies all the assertions in $\mathcal{T}_{type}$. We show below that this cannot happen when $DB(\mathcal{A}) \not\models \mathsf{UnsatQuery}(\mathcal{T})$, i.e., in this case $\mathcal{I}$ is a model of a *DL-Lite*$_{A,id,den}$ KB $\langle \mathcal{T}_{type}, \mathcal{A} \rangle$. Let us assume for a contradiction that $\mathcal{I} \not\models \langle \mathcal{T}_{type}, \mathcal{A} \rangle$. This means that there exists an assertion $\alpha = \rho(U) \sqsubseteq T_i$, where U is an attribute and T_i is a value-domain, such that there exist $o_1, o_2 \in \Delta^{\mathcal{I}}$ such that $(o_1, o_2) \in U^{\mathcal{I}}$ and $o_2 \notin T_i^{\mathcal{I}}$. Since $T_i^{\mathcal{I}} = \mathit{val}(T_i)$, and the interpretations in $\mathcal{I}$ of value-domains are pairwise disjoint, this means that $o_2 \in \mathit{val}(T_j)$ with $T_i \neq T_j$. This implies, by definition of $\mathcal{I}$, that there is an assertion $U(a, b)$ in $\mathit{chase}(\mathcal{K}^R)$, with $a = o_1^{\mathcal{I}}$ and $b = o_2^{\mathcal{I}}$, such that one of the following conditions holds:

(*i*) $b \in \Gamma_V$ and $\mathit{val}(b) \in \mathit{val}(T_j)$, with $T_j \neq T_i$,
(*ii*) $b \in \Gamma_N$, $\alpha' = \rho(U) \sqsubseteq T_j$ is in $\mathcal{T}$ together with α, and T_j precedes T_i in lexicographic order. This means that $\mathit{chase}(\mathcal{K}^R)$ contains both $T_i(b)$ and $T_j(b)$ and $\mathcal{I}$ picks from $\mathit{val}(T_j)$ a fresh symbol to interpret b.

It is easy to see that in both cases $\mathit{can}(\mathcal{K}^R) \models \varphi(\alpha)$ (indeed, in both cases $\mathit{chase}(\mathcal{K}^R)$ contains both $T_i(b)$ and $T_j(b)$). Then, from Lemma 9 it follows that $DB_R(\mathcal{A}^R) \models \mathsf{PerfectRef}(\varphi(\alpha), \mathcal{T}^R)$, which in turn implies that $DB(\mathcal{A}) \models \mathsf{PerfectRefIdC}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$, which is a contradiction, thus showing that $\mathcal{I} \models \langle \mathcal{T}_{type}, \mathcal{A} \rangle$.

Since $\mathcal{I}$ is isomorphic to $\mathit{can}(\mathcal{K}^R)$ in the interpretation of atomic concepts, atomic roles, and attributes, and $\mathit{can}(\mathcal{K}^R)$ satisfies all the assertions in $\mathcal{T}$, we can then conclude that $\mathcal{I}$ is a model for $\mathcal{K}$, and thus $\mathcal{K}$ is satisfiable.

($\Leftarrow$) if $DB(\mathcal{A}) \models \mathsf{UnsatQuery}(\mathcal{T})$, then there exists at least a query $q^u = \mathsf{PerfectRefIdC}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$ in $\mathsf{UnsatQuery}(\mathcal{T})$ such $DB(\mathcal{A}) \models q^u$. Various cases are possible: (*i*) α is an assertion in $\mathcal{T}_{disj} \cup \mathcal{T}_{type} \cup \mathcal{T}_{den}$. Since $\varphi(\alpha)$ is a CQ without inequalities, in this case $q^u = \mathsf{PerfectRef}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type})$. By Proposition 1, we have that q^u is a perfect rewriting of $\varphi(\alpha)$ with respect to $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$, and therefore $DB(\mathcal{A}) \models q^u$ implies that $\langle \mathcal{T}_{inc} \cup \mathcal{T}_{type}, \mathcal{A} \rangle \models \varphi(\alpha)$. Since $\varphi(\alpha)$ encodes the negation of α, it follows that $\langle \mathcal{T}_{inc} \cup \mathcal{T}_{type} \cup \{\alpha\}, \mathcal{A} \rangle$ is unsatisfiable, and therefore $\mathcal{K}$ is unsatisfiable. (*ii*) $\alpha \in \mathcal{T}_{funct}$. As shown during the proof of the

other direction of this theorem, in this case $\mathsf{PerfectRefIdC}(\varphi(\alpha), \mathcal{T}_{inc} \cup \mathcal{T}_{type}) = \varphi(\alpha)$. Then, from $DB(\mathcal{A}) \models \varphi(\alpha)$ it immediately follows that $\langle \{\alpha\}, \mathcal{A} \rangle$ is unsatisfiable, and therefore $\mathcal{K}$ is unsatisfiable. (iii) $\alpha \in \mathcal{T}_{id}$. As done in the proof of the other direction of this theorem, we construct a *DL-Lite*$_R$ KB $\mathcal{K}^R = \langle \mathcal{T}^R, \mathcal{A}^R \rangle$ where $\mathcal{T}^R$ encodes the PI knowledge of $\mathcal{K}$, i.e., $\mathcal{T}_{inc} \cup \mathcal{T}_{type}$, and $\mathcal{A}^R$ encodes the ABox $\mathcal{A}$. From $DB(\mathcal{A}) \models q^u$ it follows that $DB_R(\mathcal{A}^R) \models q^u$. From Lemma 13 it follows that $\langle \mathcal{T}^R \cup \{\alpha\}, \mathcal{A}^R \rangle$ is unsatisfiable, which means that $\langle \mathcal{T}_{inc} \cup \mathcal{T}_{type} \cup \{\alpha\}, \mathcal{A} \rangle$ is unsatisfiable, and therefore $\mathcal{K}$ is unsatisfiable. ∎