# Ontologies and Databases: The *DL-Lite* Approach

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2], Maurizio Lenzerini[2], Antonella Poggi[2], Mariano Rodriguez-Muro[1], and Riccardo Rosati[2]

[1] KRDB Research Centre
Free University of Bozen-Bolzano
{calvanese,rodriguez}@inf.unibz.it
[2] Dip. di Informatica e Sistemistica
SAPIENZA Università di Roma
lastname@dis.uniroma1.it

**Abstract.** Ontologies provide a conceptualization of a domain of interest. Nowadays, they are typically represented in terms of Description Logics (DLs), and are seen as the key technology used to describe the semantics of information at various sites. The idea of using ontologies as a conceptual view over data repositories is becoming more and more popular, but for it to become widespread in standard applications, it is fundamental that the conceptual layer through which the underlying data layer is accessed does not introduce a significant overhead in dealing with the data. Based on these observations, in recent years a family of DLs, called *DL-Lite*, has been proposed, which is specifically tailored to capture basic ontology and conceptual data modeling languages, while keeping low complexity of reasoning and of answering complex queries, in particular when the complexity is measured w.r.t. the size of the data. In this article, we present a detailed account of the major results that have been achieved for the *DL-Lite* family. Specifically, we concentrate on *DL-Lite*$_{\mathcal{A},id}$, an expressive member of this family, present algorithms for reasoning and query answering over *DL-Lite*$_{\mathcal{A},id}$ ontologies, and analyze their computational complexity. Such algorithms exploit the distinguishing feature of the logics in the *DL-Lite* family, namely that ontology reasoning and answering unions of conjunctive queries is first-order rewritable, i.e., it can be delegated to a relational database management system. We analyze also the effect of extending the logic with typical DL constructs, and show that for most such extensions, the nice computational properties of the *DL-Lite* family are lost. We address then the problem of accessing relational data sources through an ontology, and present a solution to the notorious impedance mismatch between the abstract objects in the ontology and the values appearing in data sources. The solution exploits suitable mappings that create the objects in the ontology from the appropriate values extracted from the data sources. Finally, we discuss the QUONTO system that implements all the above mentioned solutions and is wrapped by the DIG-QUONTO server, thus providing a standard DL reasoner for *DL-Lite*$_{\mathcal{A},id}$ with extended functionality to access external data sources.

## 1 Introduction

An *ontology* is a formalism whose purpose is to support humans or machines to share some common knowledge in a structured way. Guarino [44] distinguishes *Ontology*,

S. Tessaris et al. (Eds.): Reasoning Web 2009, LNCS 5689, pp. 255–356, 2009.

the discipline that studies the nature of being, from *ontologies* (written with lowercase initial) that are systems of categories that account for a certain view or aspect of the world. Such ontologies act as standardized reference models to support knowledge sharing and integration, and with respect to this their role is twofold: (*i*) they support human understanding and communication, and (*ii*) they facilitate content-based access, communication, and integration across different information systems; to this aim, it is important that the language used to express ontologies is formal and machine-processable. To accomplish such tasks, an ontology must focus on the explication and formalization of the *semantics* of enterprise application information resources and of the relationships among them. According to Gruber [43,42], an ontology is a formal, explicit specification of a shared conceptualization. A *conceptualization* is an abstract representation of some aspect of the world (or of a fictitious environment) which is of interest to the users of the ontology. The term *explicit* in the definition refers to the fact that constructs used in the specification must be explicitly defined and the users of the ontology, who share the information of interest and the ontology itself, must agree on them. *Formal* means that the specification is encoded in a precisely defined language whose properties are well known and understood; usally this means that the languages used for the specification of an ontology is logic-based, such as the languages used in the Knowledge Representation and Artificial Intelligence communities. *Shared* means that the ontology is meant to be shared across several people, applications, communities, and organizations. According to the W3C Ontology Working Group[1], an ontology defines a set of representational *terms* used to describe and represent an area of knowledge. The ontology can be described by giving the semantics to such terms [43]. More specifically, such terms, also called *lexical references*, are associated with (i.e., mapped to) entities in the domain of interest; formal axioms are introduced to precisely state such mappings, which are in fact the statements of a logical theory. In other words, an ontology is an explicit representation of the semantics of the domain data [65]. To sum up, though there is no precise common agreement on what an ontology is, there is a common core that underlies nearly all approaches [88]:

- a vocabulary of terms that refer to the things in the domain of interest;
- a specification of the meaning (semantics) of the terms, given (ideally) in some sort of formal logics.

Some simple ontologies consist only of a mere *taxonomy* of terms; however, usually ontologies are based on rigorous logical theories, equipped with reasoning algorithms and services. According to Gruber [43,42], knowledge in ontologies is mainly formalized using five kinds of components:

1. *concepts (or classes)*, which represent sets of objects with common properties within the domain of interest;
2. *relations*, which represent relationships among concepts by means of the notion of mathematical relation;
3. *functions*, which are functional relations;

[1] `http://www.w3c.org/2001/sw/WebOnt/`

4. *axioms (or assertions)*, which are sentences that are always true and are used in general to enforce suitable properties of classes, relations, and individuals;
5. *individuals (or instances)*, which are individual objects in the domain of interest.

Ontologies allow the key concepts and terms relevant to a given domain to be identified and defined in an unambiguous way. Moreover, ontologies facilitate the integration of different perspectives, while capturing key distinctions in a given perspective; this improves the cooperations of people or services both within a single organization and across several organizations.

### 1.1 Ontologies vs. Description Logics

An ontology, as a conceptualization of a domain of interest, provides the mechanisms for modeling the domain and reasoning upon it, and has to be represented in terms of a well-defined language. Description Logics (DLs) [7] are logics specifically designed to represent structured knowledge and to reason upon it, and as such are perfectly suited as languages for representing ontologies. Given a representation of the domain of interest, an ontology-based system should provide well-founded methods for reasoning upon it, i.e., for analyzing the representation, and drawing interesting conclusions about it. DLs, being logics, are equipped with reasoning methods, and DL-based systems provide reasoning algorithms and working implementations for them. This explains why variants of DLs are providing now the underpinning for the ontology languages promoted by the W3C, namely the standard Web Ontology Language OWL[2] and its variants (called *profiles*), which are now in the process of being standardized by the W3C in their second edition, OWL 2.

DLs stem from the effort started in the mid 80s to provide a formal basis, grounded in logic, to formalisms for the structured representation of knowledge that were popular at that time, notably Semantic Networks and Frames [67,14], that typically relied on graphical or network-like representation mechanisms. The fundamental work by Brachman and Levesque [12], initiated this effort, by showing on the one hand that the full power of First-Order Logic is not required to capture the most common representation elements, and on the other hand that the computational complexity of inference is highly sensitive to the expressive power of the KR language. Research in DLs up to our days can be seen as the systematic and exhaustive exploration of the corresponding tradeoff between expressiveness and efficiency of the various inference tasks associated to KR.

DLs are based on the idea that the knowledge in the domain to represent should be structured by grouping into classes objects of interest that have properties in common, and explicitly representing those properties through the relevant relationships holding among such classes. *Concepts* denote classes of objects, and *roles* denote (typically binary) relations between objects. Both are constructed, starting from atomic concepts and roles, by making use of various constructs, and it is precisely the set of allowed constructs that characterizes the (concept) language underlying a DL.

The domain of interest is then represented by means of a DL *knowledge base* (KB), where a separation is made between general intensional knowledge and specific knowledge about individual objects in the modeled domain. The first kind of knowledge is

[2] `http://www.w3.org/2007/OWL/`

maintained in what has been traditionally called a *TBox* (for "Terminological Box"), storing a set of universally quantified assertions that state general properties of concepts and roles. The latter kind of knowledge is represented in an *ABox* (for "Assertional Box"), constituted by assertions on individual objects, e.g., the one stating that an individual is an instance of a certain concept.

Several reasoning tasks can be carried out on a DL KB, where the basic form of reasoning involves computing the subsumption relation between two concept expressions, i.e., verifying whether one expression always denotes a subset of the objects denoted by another expression. More in general, one is interested in understanding how the various elements of a KB interact with each other in an often complex way, possibly leading to inconsistencies that need to be detected, or implying new knowledge that should be made explicit.

The above observations emphasize that a DL system is characterized by three aspects: (*i*) the set of constructs constituting the language for building the concepts and the roles used in a KB; (*ii*) the kind of assertions that may appear in the KB; (*iii*) the inference mechanisms provided for reasoning on the knowledge bases expressible in the system. The expressive power and the deductive capabilities of a DL system depend on the various choices and assumptions that the system adopts with regard to the above aspects. In the following, we present .

### 1.2 Expressive Power vs. Efficiency of Reasoning in Description Logics

The first aspect above, i.e., the language for concepts and roles, has been the subject of an intensive research work started in the late 80s. Indeed, the initial results on the computational properties of DLs have been devised in a simplified setting where both the TBox and the ABox are empty [69,84,38]. The aim was to gain a clear understanding of the properties of the language constructs and their interaction, with the goal of singling out their impact on the complexity of reasoning. Gaining this insight by understanding the combinations of language constructs that are *difficult* to deal with, and devising general methods to cope with them, is essential for the design of inference procedures. It is important to understand that in this context, the notion of "difficult" has to be understood in a precise technical sense, and the declared aim of research in this area has been to study and understand the frontier between tractability (i.e., solvable by a polynomial time algorithm) and intractability of reasoning over concept expressions. The maximal combinations of constructs (among those most commonly used) that still allowed for polynomial time inference procedures were identified, which allowed to exactly characterize the tractability frontier [38]. It should be noted that the techniques and technical tools that were used to prove such results, namely tableaux-based algorithms, are still at the basis of the modern state of the art DL reasoning systems [68], such as Fact [49], Racer [45], and Pellet [86,85].

The research on the tractability frontier for reasoning over concept expressions proved invaluable from a theoretical point of view, to precisely understand the properties and interactions of the various DL constructs, and identify practically meaningful combinations that are computationally tractable. However, from the point of view of knowledge representation, where knowledge about a domain needs to be encoded, maintained, and reasoned upon, the assumption of dealing with concept expressions

only, without considering a KB (i.e., a TBox and possibly an ABox) to which the concepts refer, is clearly unrealistic. Early successful DL KR systems, such as Classic [74], relied on a KB, but did not renounce to tractability by imposing syntactic restrictions on the use of concepts in definitions, essentially to ensure acyclicity (i.e., lack of mutual recursion). Under such an assumption, the concept definitions in a KB can be folded away, and hence reasoning over a KB can be reduced to reasoning over concept expressions only.

However, the assumption of acyclicity is strongly limiting the ability to represent real-world knowledge. These limitations became quite clear also in light of the tight connection between DLs and formalisms for the structured representation of information used in other contexts, such as databases and software engineering [31]. In the presence of cyclic KBs, reasoning becomes provably exponential (i.e, EXPTIME-complete) already when the concept language contains rather simple constructs. As a consequence of such a result, research in DLs shifted from the exploration of the tractability border to an exploration of the decidability border. The aim has been to investigate how much the expressive power of language and knowledge base constructs could be further increased while maintaining decidability of reasoning, possibly with the same, already rather high, computational complexity of inference. The techniques used to prove decidability and complexity results for expressive variants of DLs range from exploiting the correspondence with modal and dynamic logics [83,30], to automata-based techniques [92,91,26,28,18,8], to tableaux-based techniques [6,15,52,9,53]. It is worth noticing that the latter techniques, though not computationally optimal, are amenable to easier implementations, and are at the basis of the current state-of-the-art reasoners for expressive DLs [68].

### 1.3 Accessing Data through Ontologies

Current reasoners for expressive DLs perform indeed well in practice, and show that even procedures that are exponential in the size of the KB might be acceptable under suitable conditions. However, such reasoners have not specifically been tailored to deal with large amounts of data (e.g., a large ABox). This is especially critical in those settings where ontologies are used as a high-level, conceptual view over data repositories, allowing users to access data item without the need to know how the data is actually organized and where it is stored. Typical scenarios for this that are becoming more and more popular are those of Information and Data Integration Systems [63,70,29,41], the Semantic Web [47,51], and ontology-based data access [37,75,20,48]. Since the common denominator to all these scenarios, as far as this article is concerned, is the access to data through an ontology, we will refer to them together as *Ontology-Based Data Access* (OBDA).

In OBDA, data are typically very large and dominate the intentional level of the ontologies. Hence, while one could still accept reasoning that is exponential on the intentional part, it is mandatory that reasoning is polynomial (actually less – see later) in the data. If follows that, when measuring the computational complexity of reasoning, the most important parameter is the size of the data, i.e., one is interested in so-called *data complexity* [90]. Traditionally, research carried out in DLs has not paid much attention to the data complexity of reasoning, and only recently efficient management of

large amounts of data [50,33] has become a primary concern in ontology reasoning systems, and data-complexity has been studied explicitly [55,22,71,62,3,4]. Unfortunately, research on the trade-off between expressive power and computational complexity of reasoning has shown that many DLs with efficient reasoning algorithms lack the modeling power required for capturing conceptual models and basic ontology languages. On the other hand, whenever the complexity of reasoning is exponential in the size of the instances (as for example for the expressive fragments of OWL and OW2, or in [27]), there is little hope for effective instance management.

A second fundamental requirement in OBDA is the possibility to answer queries over an ontology that are more complex than the simple queries (i.e., concepts and roles) usually considered in DLs research. It turns out, however, that one cannot take the other extreme and adopt as a query language full SQL (corresponding to First-Order Logic queries), since due to the inherent incompleteness introduced by the presence of an ontology, query answering amounts to logical inference, which is undecidable for First-Order Logic. Hence, a good trade-off regarding the query language to use can be found by considering those query languages that have been advocated in databases in those settings where incompleteness of information is present [89], such as data integration [63] and data exchange [57,64]. There, the query language of choice are conjunctive queries, corresponding to the select-project-join fragment of SQL, and unions thereof, which are also the kinds of queries that are best supported by commercial database management systems.

In this paper we advocate that for OBDA, i.e., all for those contexts where ontologies are used to access large amounts of data, a suitable DL should be used, specifically tailored to capture all those constructs that are used typically in conceptual modeling, while keeping query answering efficient. Specifically, efficiency should be achieved by delegating data storage and query answering to a relational data management systems (RDBMS), which is the only technology that is currently available to deal with complex queries over large amounts of data. The chosen DL should include the main modeling features of conceptual models, which are also at the basis of most ontology languages. These features include cyclic assertions, ISA and disjointness of concepts and roles, inverses on roles, role typing, mandatory participation to roles, functional restrictions of roles, and a mechanisms for identifying instances of concepts. Also, the query language should go beyond the expressive capabilities of concept expressions in DLs, and allow for expressing conjunctive queries and unions thereof.

### 1.4 Preliminaries on Computational Complexity

In the following, we will assume that the reader is familiar with basic notions about computational complexity, as defined in standard textbooks [40,73,61]. In particular, we will refer to the following complexity classes:

$$\text{AC}^0 \subsetneq \text{LogSpace} \subseteq \text{NLogSpace} \subseteq \text{PTime} \subseteq \text{NP} \subseteq \text{ExpTime}.$$

We have depicted the known relationships between these complexity classes. In particular, it is known that $\text{AC}^0$ is strictly contained in LogSpace, while it is open whether any of the other depicted inclusions is strict. However, it is known that $\text{PTime} \subsetneq \text{ExpTime}$.

Also, we will refer to the complexity class coNP, which is the class of problems that are the complement of a problem in NP.

We only comment briefly on the complexity classes $AC^0$, LOGSPACE, and NLOGSPACE, since readers might be less familiar with them.

A (decision) problem belongs to LOGSPACE if it can be decided by a two-tape (deterministic) Turing machine that receives its input on the read-only input tape and uses a number of cells of the read/write work tape that is at most logarithmic in the length of the input. The complexity class NLOGSPACE is defined analogously, except that a nondeterministic Turing machine is used instead of a deterministic one. A typical problem that is in LOGSPACE (but not in $AC^0$) is undirected graph reachability [78]. A typical problem that is in NLOGSPACE is directed graph reachability.

A LOGSPACE *reduction* is a reduction computable by a three-tape Turing machine that receives its input on the read-only input tape, leaves its output on the write-only output tape, and uses a number of cells of the read/write work tape that is at most logarithmic in the length of the input. We observe that most reductions among decision problems presented in the computer science literature, including all reductions that we present here, are actually LOGSPACE reductions.

For the complexity class $AC^0$, we provide here only the basic intuitions, and refer to [93] for the formal definition, which is based on the circuit model. Intuitively, a problem belongs to $AC^0$ if it can be decided in constant time using a number of processors that is polynomial in the size of the input. A typical example of a problem that belongs to $AC^0$ is the evaluation of First-Order Logic (i.e., SQL) queries over relational databases, where only the database is considered to be the input, and the query is considered to be fixed [1]. This fact is of importance in the context of what discussed in this paper, since the low complexity in the size of the data of the query evaluation problem provides an intuitive justification for the ability of relational database engines to deal efficiently with very large amounts of data. Also, whenever a problem is shown to be hard for a complexity class that *strictly* contains $AC^0$ (such as LOGSPACE and all classes above it), then it cannot be reduced to the evaluation of First-Order Logic queries (cf. Section 2.6).

### 1.5 Overview of This Article

We start by presenting a family of DLs, called *DL-Lite*, that has been proposed recently [21,22,24] with the aim of addressing the above issues. Specifically, in Section 2, we present *DL-Lite*$_{\mathcal{A},id}$, a significant member of the *DL-Lite* family. One distinguishing feature of *DL-Lite*$_{\mathcal{A},id}$ is that it is tightly related to conceptual modeling formalisms and is actually able to capture their most important features, as illustrated for UML class diagrams in Section 3.

A further distinguishing feature of *DL-Lite*$_{\mathcal{A},id}$ is that query answering over an ontology can be performed as a two step process: in the first step, a query posed over the ontology is reformulated, taking into account the intensional component (the TBox) only, obtaining a union of conjunctive queries; in the second step such a union is directly evaluated over the extensional component of the ontology (the ABox). Under the assumption that the ABox is maintained by an RDBMS in secondary storage, the evaluation can be carried out by an SQL engine, taking advantage of well established query

optimization strategies. Since the first step does not depend on the data, and the second step is the evaluation of a relational query over a databases, the whole query answering process is in $AC^0$ in the size of the data [1], i.e., it has the same complexity as the plain evaluation of a conjunctive query over a relational database. In Section 4, we discuss the traditional DL reasoning services for *DL-Lite*$_{\mathcal{A},id}$ and show that they are polynomial in the size of the TBox, and in $AC^0$ in the size of the ABox (i.e., the data). Then, in Section 5 we discuss query answering and its complexity.

We show also, in Section 6, that *DL-Lite*$_{\mathcal{A},id}$ is essentially the maximal fragment exhibiting such desirable computational properties, and allowing one to ultimately delegate query answering to a relational engine [22,20,25]. Indeed, even slight extensions of *DL-Lite*$_{\mathcal{A},id}$ make query answering (actually already instance checking, i.e., answering atomic queries) at least NLOGSPACE-hard in data complexity, ruling out the possibility that query evaluation could be performed by a relational engine.

Finally, we address the issue that the TBox of the ontology provides an abstract view of the intensional level of the application domain, whereas the information about the extensional level (the instances of the ontology) reside in the data sources, which are developed independently of the conceptual layer, and are managed by traditional technologies (e.g., a relational database). Therefore, the problem arises of establishing sound mechanisms for linking existing data to the instances of the concepts and the roles in the ontology. We present, in Section 7, a recently proposed solution for this problem [75], based on a mapping mechanism to link existing data sources to an ontology expressed in *DL-Lite*$_{\mathcal{A},id}$. Such mappings allow one also to bridge the notorious *impedance mismatch problem* between values (data) stored in the sources and abstract objects that are instances of concepts and roles in the ontology [66]. Intuitively, the objects in the ontology are generated by the mappings from the data values retrieved from the data sources, by making use of suitable (designer defined) skolem functions.

All the reasoning and query answering techniques presented in the paper have been implemented in the QUONTO system [2,76], and have been wrapped in the DIG-QUONTO tool to provide a standard interface for DL reasoners according to the DIG protocol. This is discussed in Section 8, together with a Plugin for the ontology editor Protégé that provides functionalities for ontology-based data access. Finally, in Section 9, we conclude the paper.

We point out that most of the work presented in this article has been carried out within the 3-year EU FET STREP project "Thinking ONtologiES" (TONES)[3]. We also remark that large portions of the material in Sections 2, 4, and 5 are inspired or taken from [24,25], of the material in Section 6 from [22,20], and of the material in Section 7 from [75]. However, the notation and formalisms have been unified, and proofs have been revised and extended to take into account the proper features of the formalism considered here, which in part differs from the ones considered in the above mentioned works.

## 2 The Description Logic *DL-Lite*$_{\mathcal{A},id}$

In this section, we introduce formally syntax and semantics of DLs, and we do so for *DL-Lite*$_{\mathcal{A},id}$, a specific DL of the *DL-Lite* family [24,22], that is also equipped with

[3] http://www.tonesproject.org/

identification constraints [24]. We will show in the subsequent chapters that in *DL-Lite*$_{\mathcal{A},id}$ the trade-off between expressive power and computational complexity of reasoning is optimized towards the needs that arise in ontology-based data access. In other words, *DL-Lite*$_{\mathcal{A},id}$ is able to capture the most significant features of popular conceptual modeling formalisms, nevertheless query answering can be managed efficiently by relying on relational database technology.

### 2.1 *DL-Lite*$_{\mathcal{A},id}$ Expressions

As mentioned, in Description Logics [7] (DLs) the domain of interest is modeled by means of *concepts*, which denote classes of objects, and *roles* (i.e., binary relationships), which denote binary relations between objects. In addition, *DL-Lite*$_{\mathcal{A},id}$ distinguishes concepts from *value-domains*, which denote sets of (data) values, and roles from *attributes*, which denote binary relations between objects and values. We now define formally syntax and semantics of expressions in our logic.

Like in any other logic, *DL-Lite*$_{\mathcal{A},id}$ expressions are built over an alphabet. In our case, the alphabet comprises symbols for atomic concepts, value-domains, atomic roles, atomic attributes, and constants.

**Syntax.** The value-domains that we consider in *DL-Lite*$_{\mathcal{A},id}$ are those corresponding to the data types adopted by the Resource Description Framework (RDF)[4], such as `xsd:string`, `xsd:integer`, etc. Intuitively, these types represent sets of values that are pairwise disjoint. In the following, we denote such value-domains by $T_1, \ldots, T_n$. Furthermore, we denote with $\Gamma$ the alphabet for constants, which we assume partitioned into two sets, namely, $\Gamma_O$ (the set of constant symbols for *objects*), and $\Gamma_V$ (the set of constant symbols for *values*). In turn, $\Gamma_V$ is partitioned into $n$ sets $\Gamma_{V_1}, \ldots, \Gamma_{V_n}$, where each $\Gamma_{V_i}$ is the set of constants for the values in the value-domain $T_i$.

In providing the specification of our logic, we use the following notation:

1. $A$ denotes an *atomic concept*, $B$ a *basic concept*, $C$ a *general concept*, and $\top_c$ the *universal concept*. An atomic concept is a concept denoted by a name. Basic and general concepts are concept expressions whose syntax is given at point 1 below.
2. $E$ denotes a *basic value-domain*, i.e., the range of an attribute, $F$ a *value-domain expression*, and $\top_d$ the *universal value-domain*. The syntax of value-domain expressions is given at point 2 below.
3. $P$ denotes an *atomic role*, $Q$ a *basic role*, and $R$ a *general role*. An atomic role is simply a role denoted by a name. Basic and general roles are role expressions whose syntax is given at point 3 below.
4. $U$ denotes an *atomic attribute* (or simply attribute), and $V$ a *general attribute*. An atomic attribute is an attribute denoted by a name, whereas a general attribute is an attribute expression whose syntax is given at point 4 below.

[4] `http://www.w3.org/RDF/`

We are now ready to define *DL-Lite*$_{\mathcal{A},id}$ expressions[5].

1. Concept expressions are built according to the following syntax:

$$\begin{array}{lcl} B & \longrightarrow & A \mid \exists Q \mid \delta(U) \\ C & \longrightarrow & \top_c \mid B \mid \neg B \mid \exists Q.C \mid \delta_F(U) \end{array}$$

   Here, $\neg B$ denotes the *negation* of a basic concept $B$. The concept $\exists Q$, also called *unqualified existential restriction*, denotes the *domain* of a role $Q$, i.e., the set of objects that $Q$ 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 value. The concept $\exists Q.C$, also called *qualified existential restriction*, denotes the *qualified domain* of $Q$ w.r.t. $C$, i.e., the set of objects that $Q$ relates to some instance of $C$. Similarly, $\delta_F(U)$ denotes the *qualified domain* of $U$ w.r.t. a value-domain $F$, i.e., the set of objects that $U$ relates to some value in $F$.

2. Value-domain expressions are built according to the following syntax:

$$\begin{array}{lcl} E & \longrightarrow & \rho(U) \\ F & \longrightarrow & \top_d \mid T_1 \mid \cdots \mid T_n \end{array}$$

   Here, $\rho(U)$ denotes the *range* of an attribute $U$, i.e., the set of values to which $U$ relates some object. Note that the range $\rho(U)$ of $U$ is a value-domain, whereas the domain $\delta(U)$ of $U$ is a concept.

3. Role expressions are built according to the following syntax:

$$\begin{array}{lcl} Q & \longrightarrow & P \mid P^- \\ R & \longrightarrow & Q \mid \neg Q \end{array}$$

   Here, $P^-$ denotes the *inverse* of an atomic role, and $\neg Q$ denotes the *negation* of a basic role. In the following, when $Q$ is a basic role, the expression $Q^-$ stands for $P^-$ when $Q = P$, and for $P$ when $Q = P^-$.

4. Attribute expressions are built according to the following syntax:

$$V \longrightarrow U \mid \neg U$$

   Here, $\neg U$ denotes the *negation* of an atomic attribute.

As an example, consider the atomic concepts *Man* and *Woman*, and the atomic roles *HAS-HUSBAND*, representing the relationship between a woman and the man with whom she is married, and *HAS-CHILD*, representing the parent-child relationship. Then, intuitively, the inverse of *HAS-HUSBAND*, i.e., *HAS-HUSBAND*$^-$, represents the relationship between a man and his wife. Also, $\exists$*HAS-CHILD*.*Woman* represents those having a daughter.

[5] The results mentioned in this paper apply also to *DL-Lite*$_{\mathcal{A},id}$ extended with role attributes (cf. [19]), which are not considered here for the sake of simplicity.

$$
\begin{array}{rcl}
A^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \\
(\exists Q)^{\mathcal{I}} & = & \{\, o \mid \exists o' . (o, o') \in Q^{\mathcal{I}} \,\} \\
(\delta(U))^{\mathcal{I}} & = & \{\, o \mid \exists v . (o, v) \in U^{\mathcal{I}} \,\} \\
(\exists Q.C)^{\mathcal{I}} & = & \{\, o \mid \exists o' . (o, o') \in Q^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}} \,\} \\
(\delta_F(U))^{\mathcal{I}} & = & \{\, o \mid \exists v . (o, v) \in U^{\mathcal{I}} \wedge v \in F^{\mathcal{I}} \,\} \\
\top_c^{\mathcal{I}} & = & \Delta_O^{\mathcal{I}} \\
(\neg B)^{\mathcal{I}} & = & \Delta_O^{\mathcal{I}} \setminus B^{\mathcal{I}}
\end{array}
\qquad
\begin{array}{rcl}
(\rho(U))^{\mathcal{I}} & = & \{\, v \mid \exists o . (o, v) \in U^{\mathcal{I}} \,\} \\
\top_d^{\mathcal{I}} & = & \Delta_V^{\mathcal{I}} \\
T_i^{\mathcal{I}} & = & val(T_i) \\
P^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}} \\
(P^-)^{\mathcal{I}} & = & \{\, (o, o') \mid (o', o) \in P^{\mathcal{I}} \,\} \\
(\neg Q)^{\mathcal{I}} & = & (\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}} \\
U^{\mathcal{I}} & \subseteq & \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}} \\
(\neg U)^{\mathcal{I}} & = & (\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U^{\mathcal{I}}
\end{array}
$$

**Fig. 1.** Semantics of *DL-Lite*$_{\mathcal{A},id}$ expressions

**Semantics.** The meaning of *DL-Lite*$_{\mathcal{A},id}$ expressions is sanctioned by the semantics. Following the classical approach in DLs, the semantics of *DL-Lite*$_{\mathcal{A},id}$ is given in terms of First-Order Logic interpretations. All such interpretations agree on the semantics assigned to each value-domain $T_i$ and to each constant in $\Gamma_V$. In particular, each valued-domain $T_i$ is interpreted as the set $val(T_i)$ of values of the corresponding RDF data type, and each constant $c_i \in \Gamma_V$ is interpreted as one specific value, denoted $val(c_i)$, in $val(T_i)$. Note that, since the data types $T_i$ are pairwise disjoint, we have that $val(T_i) \cap val(T_j) = \emptyset$, for $i \neq j$.

Based on the above observations, we can now define the notion of interpretation in *DL-Lite*$_{\mathcal{A},id}$. An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where

- $\Delta^{\mathcal{I}}$ is the interpretation domain, which is the disjoint union of two non-empty sets: $\Delta_O^{\mathcal{I}}$, called the *domain of objects*, and $\Delta_V^{\mathcal{I}}$, called the *domain of values*. In turn, $\Delta_V^{\mathcal{I}}$ is the union of $val(T_1), \ldots, val(T_n)$.
- $\cdot^{\mathcal{I}}$ is the *interpretation function*, i.e., a function that assigns an element of $\Delta^{\mathcal{I}}$ to each constant in $\Gamma$, a subset of $\Delta^{\mathcal{I}}$ to each concept and value-domain, and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each role and attribute, in such a way that the following holds:
  - for each $c \in \Gamma_V$, $c^{\mathcal{I}} = val(c)$,
  - for each $d \in \Gamma_O$, $d^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$,
  - for each $a_1, a_2 \in \Gamma$, $a_1 \neq a_2$ implies $a_1^{\mathcal{I}} \neq a_2^{\mathcal{I}}$, and
  - the conditions shown in Figure 1 are satisfied.

Note that the above definition implies that different constants are interpreted differently in the domain, i.e., *DL-Lite*$_{\mathcal{A},id}$ adopts the so-called *unique name assumption* (UNA).

### 2.2 *DL-Lite*$_{\mathcal{A},id}$ Ontologies

Like in any DL, a *DL-Lite*$_{\mathcal{A},id}$ *ontology*, or knowledge base (KB), is constituted by two components:

- a *TBox* (where the 'T' stands for *terminological*), which is a finite set of *intensional assertions*, and
- an *ABox* (where the 'A' stands for *assertional*), which is a finite set of *extensional* (or, *membership*) *assertions*.

We now specify formally the form of a *DL-Lite*$_{\mathcal{A},id}$ TBox and ABox, and its semantics.

**Syntax.** In *DL-Lite*$_{\mathcal{A},id}$, the TBox may contain intensional assertions of three types, namely inclusion assertions, functionality assertions, and local identification assertions.

– An *inclusion assertion* has one the forms

$$B \sqsubseteq C, \qquad E \sqsubseteq F, \qquad Q \sqsubseteq R, \qquad U \sqsubseteq V,$$

denoting respectively, from left to right, inclusions between concepts, value-domains, roles, and attributes. Intuitively, an inclusion assertion states that, in every model of $\mathcal{T}$, each instance of the left-hand side expression is also an instance of the right-hand side expression.

An inclusion assertion that on the right-hand side does not contain the symbol '$\neg$' is called a *positive inclusion (PI)*, while an inclusion assertion that on the right-hand side contains the symbol '$\neg$' is called a *negative inclusion (NI)*. Hence, a negative inclusion has one of the forms $B_1 \sqsubseteq \neg B_2$, $B_1 \sqsubseteq \exists Q_1.\exists Q_2.\ldots\exists Q_k.\neg B_2$, $P_1 \sqsubseteq \neg P_2$, or $U_1 \sqsubseteq \neg U_2$.

For example, the (positive) inclusion *Parent* $\sqsubseteq$ $\exists$*HAS-CHILD* specifies that parents have a child, the inclusions $\exists$*HAS-HUSBAND* $\sqsubseteq$ $Woman$ and $\exists$*HAS-HUSBAND*$^-$ $\sqsubseteq$ $Man$ respectively specify that wifes (i.e., those who have a husband) are women and that husbands are men, and the inclusion *Person* $\sqsubseteq$ $\delta(\mathbf{hasSsn})$, where $\mathbf{hasSsn}$ is an attribute, specifies that each person has a social security number. The negative inclusion *Man* $\sqsubseteq$ $\neg$*Woman* specifies that men and women are disjoint.

– A *functionality assertion* has one of the forms

$$(\mathsf{funct}\ Q), \qquad (\mathsf{funct}\ U),$$

denoting functionality of a role and of an attribute, respectively. Intuitively, a functionality assertion states that the binary relation represented by a role (respectively, an attribute) is a function.

For example, the functionality assertion (funct *HAS-HUSBAND*$^-$) states that a person may have at most one wife, and the functionality assertion (funct $\mathbf{hasSsn}$) states that no individual may have more than one social security number.

– A *local identification assertion* (or, simply, identification assertion or identification constraint) makes use of the notion of path. A *path* is an expression built according to the following syntax,

$$\pi \longrightarrow S \mid D? \mid \pi \circ \pi \tag{1}$$

where $S$ denotes a basic role (i.e., an atomic role or the inverse of an atomic role), an atomic attribute, or the inverse of an atomic attribute, and $\pi_1 \circ \pi_2$ denotes the composition of the paths $\pi_1$ and $\pi_2$. Finally, $D$ denotes an basic concept or a (basic or arbitrary) value domain, and the expression $D?$ is called a *test relation*, which represents the identity relation on instances of $D$. Test relations are used in all those cases in which one wants to impose that a path involves instances of a certain

concept. For example, *HAS-CHILD* $\circ$ *Woman*? is the path connecting someone with his/her daughters.

A path $\pi$ denotes a complex property for the instances of concepts: given an object $o$, every object that is reachable from $o$ by means of $\pi$ is called a $\pi$-*filler* for $o$. Note that for a certain $o$ there may be several distinct $\pi$-fillers, or no $\pi$-fillers at all.

If $\pi$ is a path, the length of $\pi$, denoted $length(\pi)$, is 0 if $\pi$ has the form $D?$, is 1 if $\pi$ has the form $S$, and is $length(\pi_1) + length(\pi_2)$ if $\pi$ has the form $\pi_1 \circ \pi_2$. With the notion of path in place, we are ready for the definition of identification assertion, which is an assertion of the form

$$(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n),$$

where $B$ is a basic concept, $n \geq 1$, and $\pi_1, \ldots, \pi_n$ (called the *components* of the identifier) are paths such that $length(\pi_i) \geq 1$ for all $i \in \{1, \ldots, n\}$. Intuitively, such a constraint asserts that for any two different instances $o$, $o'$ of $B$, there is at least one $\pi_i$ such that $o$ and $o'$ differ in the set of their $\pi_i$-fillers. The identification assertion is called *local* if $length(\pi_i) = 1$ for at least one $i \in \{1, \ldots, n\}$. The term "local" emphasizes that at least one of the paths has length 1 and thus refers to a local property of $B$. In the following, we will consider only local identification assertions, and thus simply omit the 'local' qualifier.

For example, the identification assertion (id *Woman HAS-HUSBAND*) says that a woman is identified by her husband, i.e., there are not two different women with the same husband, whereas the identification assertion (id *Man HAS-CHILD*) says that a man is identified by his children, i.e., there are not two men with a child in common. We can also say that there are not two men with the same daughters by means of the identification (id *Man HAS-CHILD* $\circ$ *Woman*?).

Then, a *DL-Lite*$_{\mathcal{A},id}$ TBox is a finite sets of intensional assertions of the form above, where suitable limitations in the combination of such assertions are imposed. To precisely describe such limitations, we first introduce some preliminary notions. An atomic role $P$ (resp., an atomic attribute $U$) is called an *identifying property in a TBox* $\mathcal{T}$, if

- $\mathcal{T}$ contains a functionality assertion (funct $P$) or (funct $P^-$) (resp., (funct $U$)), or
- $P$ (resp., $U$) appears (in either direct or inverse direction) in some path of an identification assertion in $\mathcal{T}$.

We say that an atomic role $P$ (resp., an atomic attribute $U$) *appears positively* in the right-hand side of an inclusion assertion $\alpha$ if $\alpha$ has the form $Q \sqsubseteq P$ or $Q \sqsubseteq P^-$, for some basic role $Q$ (resp., $U' \sqsubseteq U$, for some atomic attribute $U'$). An atomic role $P$ (resp., an atomic attribute $U$) is called *primitive in a TBox* $\mathcal{T}$, if

- it does not appear positively in the right-hand side of an inclusion assertion of $\mathcal{T}$, and
- it does not appear in $\mathcal{T}$ in an expression of the form $\exists P.C$ or $\exists P^-.C$ (resp., $\delta_F(U)$).

With these notions in place, we are ready to define what constitutes a *DL-Lite*$_{\mathcal{A},id}$ TBox.

**Definition 2.1.** *A DL-Lite$_{\mathcal{A},id}$ TBox, $\mathcal{T}$, is a finite set of inclusion assertions, functionality assertions, and identification assertions as specified above, and such that the following conditions are satisfied:*

*(1) Each concept appearing in an identification assertion of $\mathcal{T}$ (either as the identified concept, or in some test relation of some path) is a basic concept, i.e., a concept of the form $A$, $\exists Q$, or $\delta(U)$.*
*(2) Each identifying property in $\mathcal{T}$ is primitive in $\mathcal{T}$.*

*A DL-Lite$_{\mathcal{A}}$ TBox is a DL-Lite$_{\mathcal{A},id}$ TBox that does not contain identification assertions.*

Intuitively, the condition stated at point (2) says that, in *DL-Lite$_{\mathcal{A},id}$* TBoxes, roles and attributes occurring in functionality assertions or in paths of identification constraints cannot be specialized. We will see that the above conditions ensure the tractability of reasoning in our logic.

A *DL-Lite$_{\mathcal{A},id}$* (or *DL-Lite$_{\mathcal{A}}$*) ABox consists of a set of *membership assertions*, which are used to state the instances of concepts, roles, and attributes. Such assertions have the form

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

where $A$ is an atomic concept, $P$ is an atomic role, $U$ is an atomic attribute, $a$, $a_1$, $a_2$ are constants in $\Gamma_O$, and $c$ is a constant in $\Gamma_V$.

**Definition 2.2.** *A DL-Lite$_{\mathcal{A},id}$ (resp., DL-Lite$_{\mathcal{A}}$) ontology $\mathcal{O}$ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a DL-Lite$_{\mathcal{A},id}$ (resp., DL-Lite$_{\mathcal{A}}$) TBox (cf. Definition 2.1), and $\mathcal{A}$ is a DL-Lite$_{\mathcal{A},id}$ (or DL-Lite$_{\mathcal{A}}$) ABox, all of whose atomic concepts, roles, and attributes occur in $\mathcal{T}$.*

Notice that, for an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, the requirement in Definition 2.2 that all concepts, roles, and attributes that occur in $\mathcal{A}$ occur also in $\mathcal{T}$ is not a limitation. Indeed, as will be clear from the semantics, we can deal with the general case by adding to $\mathcal{T}$ inclusion assertions $A \sqsubseteq \top_c$, $\exists P \sqsubseteq \top_c$, and $\delta(U) \sqsubseteq \top_c$, for any atomic concepts $A$, atomic role $P$, and atomic attribute $U$ occurring in $\mathcal{A}$ but not in $\mathcal{T}$, without altering the semantics of $\mathcal{O}$.

We also observe that in many DLs, functionality assertions are not explicitly present, since they can be expressed by means of number restrictions. *Number restrictions* $(\geq k\, Q)$ and $(\leq k\, Q)$, where $k$ is a positive integer and $Q$ a basic role, denotes the set of objects that are connected by means of role $Q$ respectively to at least and at most $k$ distinct objects. Hence, $(\geq k\, Q)$ generalizes existential quantification $\exists Q$, while $(\leq k\, Q)$ can be used to generalize functionality assertions. Indeed, the assertion $(\mathsf{funct}\ Q)$ is equivalent to the inclusion assertion $\exists Q \sqsubseteq (\leq 1\, Q)$, where the used number is 1, and the number restriction is expressed globally for the whole domain of $Q$. Instead, by means of an assertion $B \sqsubseteq (\leq k\, Q)$, one can impose *locally*, i.e., just for the instances of concept $B$, a numeric condition involving a number $k$ that is different from 1.

**Semantics.** We now specify the semantics of an ontology, again in terms of interpretations, by defining when an interpretation $\mathcal{I}$ *satisfies* and assertion $\alpha$ (either an intensional assertion or a membership assertion), denoted $\mathcal{I} \models \alpha$.

– An interpretation $\mathcal{I}$ satisfies a concept (resp., value-domain, role, attribute) inclusion assertion

$$\begin{array}{lll} B \sqsubseteq C, & \text{if} & B^{\mathcal{I}} \subseteq C^{\mathcal{I}}; \\ E \sqsubseteq F, & \text{if} & E^{\mathcal{I}} \subseteq F^{\mathcal{I}}; \\ Q \sqsubseteq R, & \text{if} & Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}; \\ U \sqsubseteq V, & \text{if} & U^{\mathcal{I}} \subseteq V^{\mathcal{I}}. \end{array}$$

– An interpretation $\mathcal{I}$ satisfies a role functionality assertion $(\mathsf{funct}\ Q)$, if for each $o_1, o_2, o_3 \in \Delta_O^{\mathcal{I}}$

$$(o_1, o_2) \in Q^{\mathcal{I}} \text{ and } (o_1, o_3) \in Q^{\mathcal{I}} \quad \text{implies} \quad o_2 = o_3.$$

– An interpretation $\mathcal{I}$ satisfies an attribute functionality assertion $(\mathsf{funct}\ U)$, if for each $o \in \Delta_O^{\mathcal{I}}$ and $v_1, v_2 \in \Delta_V^{\mathcal{I}}$

$$(o, v_1) \in U^{\mathcal{I}} \text{ and } (o, v_2) \in U^{\mathcal{I}} \quad \text{implies} \quad v_1 = v_2.$$

– In order to define the semantics of identification assertions, we first define the semantics of paths. The extension $\pi^{\mathcal{I}}$ of a path $\pi$ in an interpretation $\mathcal{I}$ is defined as follows:
  - if $\pi = S$, then $\pi^{\mathcal{I}} = S^{\mathcal{I}}$,
  - if $\pi = D?$, then $\pi^{\mathcal{I}} = \{(o, o) \mid o \in D^{\mathcal{I}}\}$,
  - if $\pi = \pi_1 \circ \pi_2$, then $\pi^{\mathcal{I}} = \pi_1^{\mathcal{I}} \circ \pi_2^{\mathcal{I}}$, where $\circ$ denotes the composition operator on relations.

  As a notation, we write $\pi^{\mathcal{I}}(o)$ to denote the set of $\pi$-fillers for $o$ in $\mathcal{I}$, i.e., $\pi^{\mathcal{I}}(o) = \{o' \mid (o, o') \in \pi^{\mathcal{I}}\}$.

  Then, an interpretation $\mathcal{I}$ satisfies an identification assertion $(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$ if for all $o, o' \in B^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) \cap \pi_1^{\mathcal{I}}(o') \neq \emptyset \wedge \cdots \wedge \pi_n^{\mathcal{I}}(o) \cap \pi_n^{\mathcal{I}}(o') \neq \emptyset$ implies $o = o'$. Observe that this definition is coherent with the intuitive reading of identification assertions discussed above, in particular by sanctioning that two different instances $o$, $o'$ of $B$ differ in the set of their $\pi_i$-fillers when such sets are disjoint.[6]
– An interpretation $\mathcal{I}$ satisfies a membership assertion

$$\begin{array}{lll} A(a), & \text{if} & a^{\mathcal{I}} \in A^{\mathcal{I}}; \\ P(a_1, a_2), & \text{if} & (a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in P^{\mathcal{I}}; \\ U(a, c), & \text{if} & (a^{\mathcal{I}}, c^{\mathcal{I}}) \in U^{\mathcal{I}}. \end{array}$$

An interpretation $\mathcal{I}$ is a *model* of a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$, ABox $\mathcal{A}$), or, equivalently, $\mathcal{I}$ *satisfies* $\mathcal{O}$ (resp., $\mathcal{T}$, $\mathcal{A}$), written $\mathcal{I} \models \mathcal{O}$ (resp., $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{A}$) if and only if $\mathcal{I}$ satisfies all assertions in $\mathcal{O}$ (resp., $\mathcal{T}$, $\mathcal{A}$). The semantics of a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is the set of all *models* of $\mathcal{O}$.

[6] Note that an alternative definition of the semantics of identification assertions is the one where an interpretation $\mathcal{I}$ satisfies $(\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$ if for all $o, o' \in B^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) = \pi_1^{\mathcal{I}}(o') \wedge \cdots \wedge \pi_n^{\mathcal{I}}(o) = \pi_n^{\mathcal{I}}(o')$ implies $o = o'$. This alternative semantics coincides with the one we have adopted in the case where all roles and attributes in all paths $\pi_i$ are functional, but imposes a stronger condition for identification when this is not the case. Indeed, the alternative semantics sanctions that two different instances $o$, $o'$ of $B$ differ in the set of their $\pi_i$-fillers when such sets are *different* (rather than disjoint).

With the semantics of an ontology in place, we comment briefly on the various types of assertions in a *DL-Lite*$_{\mathcal{A},id}$ TBox and relate them to constraints used in classical database theory. We remark, however, that TBox assertions have a fundamentally different role from the one of database dependencies: while the latter are typically enforced on the data in a database, this is not the case for the former, which are instead used to infer new knowledge from the asserted one.

- Inclusion assertions having a positive element in the right-hand side intuitively correspond to inclusion dependencies in databases [1]. Specifically Concept inclusions correspond to unary inclusion dependencies [34], while role inclusions correspond to binary inclusion dependencies. An inclusion assertion of the form $\exists Q \sqsubseteq A$ is in fact a *foreign key*, since objects that are instances of concept $A$ can be considered as keys for the (unary) relations denoted by $A$. Instead, an inclusion of the form $A \sqsubseteq \exists Q$ can be considered as a participation constraint.
- Inclusion assertions having a negative element in the right-hand side intuitively correspond to exclusion (or disjointness) dependencies in databases [1].
- Functionality assertions correspond to unary key dependencies in databases. Specifically $(\mathsf{funct}\ P)$ corresponds to stating that the first component of the binary relation $P$ is a key for $P$, while $(\mathsf{funct}\ P^-)$ states the same for the second component of $P$.
- Identification assertions correspond to more complex forms of key dependencies. To illustrate this correspondence, consider a concept $A$ and a set of attributes $U_1, \ldots, U_n$, where each $U_i$ is functional and has $A$ as domain (i.e., the TBox contains the functionality assertion $(\mathsf{funct}\ U_i)$ and the inclusion assertion $\delta(U_i) \sqsubseteq A$). Together, $A$ and "its attributes" can be considered as representing a single relation $R_A$ of arity $n+1$ constituted by one column for the object $A$ and one column for each of the $U_i$ attributes. Then, an identification assertion $(\mathsf{id}\ A\ U_{i_1}, \ldots, U_{i_k})$, involving a subset $U_{i_1}, \ldots, U_{i_k}$ of the attributes of $A$, resembles a key dependency on $R_A$, where the key is given by the specified subset of attributes. Indeed, due to the identification assertion, a given sequence $v_1, \ldots, v_k$ of values for $U_{i_1}, \ldots, U_{i_k}$ determines a unique instance $a$ of $A$, and since all attributes are functional and have $A$ as domain, their value is uniquely determined by $a$, and hence by $v_1, \ldots, v_k$.

For further intuitions about the meaning of the various kinds of TBox assertions we refer also to Section 3, where the relationship with conceptual models (specifically, UML class diagrams) is discussed in detail.

*Example 2.3.* We conclude this section with an example in which we present a *DL-Lite*$_{\mathcal{A},id}$ ontology modeling the annual national football[7] championships in Europe, where the championship for a specific nation is called *league* (e.g., the Spanish Liga). A league is structured in terms of a set of *rounds*. Every round contains a set of *matches*, each one characterized by one *home team* and one *host team*. We distinguish between scheduled matches, i.e., matches that have still to be played, and played matches. Obviously, a match falls in exactly one of these two categories.

[7] Football is called "soccer" in the United States.

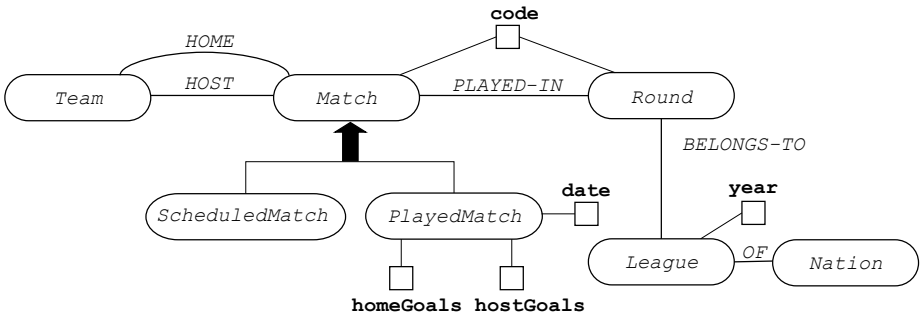



**Fig. 2.** Diagrammatic representation of the football championship ontology

| INCLUSION ASSERTIONS | | | | | |
|---|---|---|---|---|---|
| *League* | $\sqsubseteq$ | $\exists$*OF* | *PlayedMatch* | $\sqsubseteq$ | *Match* |
| $\exists$*OF* | $\sqsubseteq$ | *League* | *ScheduledMatch* | $\sqsubseteq$ | *Match* |
| $\exists$*OF*$^-$ | $\sqsubseteq$ | *Nation* | *PlayedMatch* | $\sqsubseteq$ | $\neg$*ScheduledMatch* |
| *Round* | $\sqsubseteq$ | $\exists$*BELONGS-TO* | *Match* | $\sqsubseteq$ | $\neg$*Round* |
| $\exists$*BELONGS-TO* | $\sqsubseteq$ | *Round* | *League* | $\sqsubseteq$ | $\delta(\mathbf{year})$ |
| $\exists$*BELONGS-TO*$^-$ | $\sqsubseteq$ | *League* | *Match* | $\sqsubseteq$ | $\delta(\mathbf{code})$ |
| *Match* | $\sqsubseteq$ | $\exists$*PLAYED-IN* | *Round* | $\sqsubseteq$ | $\delta(\mathbf{code})$ |
| $\exists$*PLAYED-IN* | $\sqsubseteq$ | *Match* | *PlayedMatch* | $\sqsubseteq$ | $\delta(\mathbf{date})$ |
| $\exists$*PLAYED-IN*$^-$ | $\sqsubseteq$ | *Round* | *PlayedMatch* | $\sqsubseteq$ | $\delta(\mathbf{homeGoals})$ |
| *Match* | $\sqsubseteq$ | $\exists$*HOME* | *PlayedMatch* | $\sqsubseteq$ | $\delta(\mathbf{hostGoals})$ |
| $\exists$*HOME* | $\sqsubseteq$ | *Match* | $\rho(\mathbf{date})$ | $\sqsubseteq$ | `xsd:date` |
| $\exists$*HOME*$^-$ | $\sqsubseteq$ | *Team* | $\rho(\mathbf{homeGoals})$ | $\sqsubseteq$ | `xsd:nonNegativeInteger` |
| *Match* | $\sqsubseteq$ | $\exists$*HOST* | $\rho(\mathbf{hostGoals})$ | $\sqsubseteq$ | `xsd:nonNegativeInteger` |
| $\exists$*HOST* | $\sqsubseteq$ | *Match* | $\rho(\mathbf{code})$ | $\sqsubseteq$ | `xsd:string` |
| $\exists$*HOST*$^-$ | $\sqsubseteq$ | *Team* | $\rho(\mathbf{year})$ | $\sqsubseteq$ | `xsd:positiveInteger` |

| FUNCTIONALITY ASSERTIONS | | | |
|---|---|---|---|
| (funct *OF*) | (funct *HOME*) | (funct **year**) | (funct **homeGoals**) |
| (funct *BELONGS-TO*) | (funct *HOST*) | (funct **code**) | (funct **hostGoals**) |
| (funct *PLAYED-IN*) | | (funct **date**) | |

| IDENTIFICATION ASSERTIONS | |
|---|---|
| 1. (id *League OF*, **year**) | 6. (id *PlayedMatch* **date**, *HOST*) |
| 2. (id *Round BELONGS-TO*, **code**) | 7. (id *PlayedMatch* **date**, *HOME*) |
| 3. (id *Match PLAYED-IN*, **code**) | 8. (id *League* **year**, *BELONGS-TO*$^-$ $\circ$ *PLAYED-IN*$^-$ $\circ$ *HOME*) |
| 4. (id *Match HOME*, *PLAYED-IN*) | 9. (id *League* **year**, *BELONGS-TO*$^-$ $\circ$ *PLAYED-IN*$^-$ $\circ$ *HOST*) |
| 5. (id *Match HOST*, *PLAYED-IN*) | 10. (id *Match HOME*, *HOST*, *PLAYED-IN* $\circ$ *BELONGS-TO* $\circ$ **year**) |

**Fig. 3.** The *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}_{fbc}$ for the football championship example

In Figure 2, we show a schematic representation of (a portion of) the intensional part of the ontology for the football championship domain. In this figure, the black arrow represents a partition of one concept into a set of sub-concepts. We have not represented explicitly in the figure the pairwise disjointness of the concepts *Team*, *Match*, *Round*, *League*, and *Nation*, which intuitively holds in the modeled domain. In Figure 3, a *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}_{fbc}$ is shown that captures (most of) the above aspects. In our examples, we use the *CapitalizedItalics* font to denote atomic concepts, the *ALL-CAPITALS-ITALICS* font to denote atomic roles, the `typewriter` font to denote value-domains, and the **boldface** font to denote atomic attributes. Regarding the

pairwise disjointness of the various concepts, we have represented by means of negative inclusion assertions only the disjointness between *PlayedMatch* and *ScheduledMatch* and the one between *Match* and *Round*. By virtue of the characteristics of *DL-Lite*$_{\mathcal{A},id}$, we can explicitly consider also attributes of concepts and the fact that they are used for identification. In particular, we assume that when a scheduled match takes place, it is played in a specific date, and that for every match that has been played, the number of goals scored by the home team and by the host team are given. Note that different matches scheduled for the same round can be played in different dates. Also, we want to distinguish football championships on the basis of the nation and the year in which a championship takes place (e.g., the 2009 Italian Liga). We also assume that both matches and rounds have codes. The identification assertions model the following aspects:

1. No nation has two leagues in the same year.
2. Within a league, the code associated to a round is unique.
3. Every match is identified by its code within its round.
4. A team is the home team of at most one match per round.
5. As above for the host team.
6. No home team participates in different played matches in the same date.
7. As above for the host team.
8. No home team plays in different leagues in the same year.
9. As above for the host team.
10. No pair (home team, host team) plays different matches in the same year.

Note that the *DL-Lite*$_{\mathcal{A},id}$ TBox in Figure 3 captures the ontology in Figure 2, except for the fact that the concept *Match* covers the concepts *ScheduledMatch* and *PlayedMatch*. In order to express such a condition, we would need to use disjunction in the right-hand-side of inclusion assertions, i.e.,

$$\mathit{Match} \sqsubseteq \mathit{ScheduledMatch} \sqcup \mathit{PlayedMatch}$$

where $\sqcup$ would be interpreted as set union. As we will see in Section 6, we have to renounce to the expressive power required to capture covering constraints (i.e., disjunction), if we want to preserve nice computational properties for reasoning over *DL-Lite*$_{\mathcal{A},id}$ ontologies.

An ABox, $\mathcal{A}_{fbc}$, associated to the TBox in Figure 3 is shown in Figure 4, where we have used the *slanted* font for constants in $\Gamma_O$ and the `typeface` font for constants in $\Gamma_V$. For convenience of reading, we have chosen in the example names of the constants that indicate the properties of the objects that the constants represent.

We observe that the ontology $\mathcal{O}_{fbc} = \langle \mathcal{T}_{fbc}, \mathcal{A}_{fbc} \rangle$ is satisfiable. Indeed, the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ shown in Figure 5 is a model of the ABox $\mathcal{A}_{fbc}$, where we have assumed that for each value constant $c \in \Gamma_V$, the corresponding value $val(c)$ is equal to $c$ itself, hence $c^{\mathcal{I}} = val(c) = c$. Moreover, it is easy to see that every interpretation $\mathcal{I}$ has to satisfy the conditions shown in Figure 5 in order to be a model of $\mathcal{A}_{fbc}$.

| CONCEPT AND ROLE MEMBERSHIP ASSERTIONS | | |
|---|---|---|
| *League*(*it2009*) | | *PLAYED-IN*(*m7RJ*, *r7*) |
| *Round*(*r7*) | *BELONGS-TO*(*r7*, *it2009*) | *PLAYED-IN*(*m8NT*, *r8*) |
| *Round*(*r8*) | *BELONGS-TO*(*r8*, *it2009*) | *PLAYED-IN*(*m8RM*, *r8*) |
| *PlayedMatch*(*m7RJ*) | *HOME*(*m7RJ*, *roma*) | *HOST*(*m7RJ*, *juventus*) |
| *Match*(*m8NT*) | *HOME*(*m8NT*, *napoli*) | *HOST*(*m8NT*, *torino*) |
| *Match*(*m8RM*) | *HOME*(*m8RM*, *roma*) | *HOST*(*m8RM*, *milan*) |
| *Team*(*roma*) | *Team*(*napoli*) | *Team*(*juventus*) |
| ATTRIBUTE MEMBERSHIP ASSERTIONS | | |
| $\mathbf{code}$(*r7*, `"7"`) | $\mathbf{code}$(*m7RJ*, `"RJ"`) | $\mathbf{date}$(*m7RJ*, `5/4/09`) |
| $\mathbf{code}$(*r8*, `"8"`) | $\mathbf{code}$(*m8NT*, `"NT"`) | $\mathbf{homeGoals}$(*m7RJ*, `3`) |
| | $\mathbf{code}$(*m8RM*, `"RM"`) | $\mathbf{hostGoals}$(*m7RJ*, `1`) |

**Fig. 4.** The ABox $\mathcal{A}_{fbc}$ for the football championship example

| | | |
|---|---|---|
| $(it2009^{\mathcal{I}}) \in League^{\mathcal{I}}$ | | $(m7RJ^{\mathcal{I}}, r7^{\mathcal{I}}) \in PLAYED\text{-}IN^{\mathcal{I}}$ |
| $(r7^{\mathcal{I}}) \in Round^{\mathcal{I}}$ | $(r7^{\mathcal{I}}, it2009^{\mathcal{I}}) \in BELONGS\text{-}TO^{\mathcal{I}}$ | $(m8NT^{\mathcal{I}}, r8^{\mathcal{I}}) \in PLAYED\text{-}IN^{\mathcal{I}}$ |
| $(r8^{\mathcal{I}}) \in Round^{\mathcal{I}}$ | $(r8^{\mathcal{I}}, it2009^{\mathcal{I}}) \in BELONGS\text{-}TO^{\mathcal{I}}$ | $(m8RM^{\mathcal{I}}, r8^{\mathcal{I}}) \in PLAYED\text{-}IN^{\mathcal{I}}$ |
| $(m7RJ^{\mathcal{I}}) \in PlayedMatch^{\mathcal{I}}$ | $(m7RJ^{\mathcal{I}}, roma^{\mathcal{I}}) \in HOME^{\mathcal{I}}$ | $(m7RJ^{\mathcal{I}}, juventus^{\mathcal{I}}) \in HOST^{\mathcal{I}}$ |
| $(m8NT^{\mathcal{I}}) \in Match^{\mathcal{I}}$ | $(m8NT^{\mathcal{I}}, napoli^{\mathcal{I}}) \in HOME^{\mathcal{I}}$ | $(m8NT^{\mathcal{I}}, torino^{\mathcal{I}}) \in HOST^{\mathcal{I}}$ |
| $(m8RM^{\mathcal{I}}) \in Match^{\mathcal{I}}$ | $(m8RM^{\mathcal{I}}, roma^{\mathcal{I}}) \in HOME^{\mathcal{I}}$ | $(m8RM^{\mathcal{I}}, milan^{\mathcal{I}}) \in HOST^{\mathcal{I}}$ |
| $(roma^{\mathcal{I}}) \in Team^{\mathcal{I}}$ | $(napoli^{\mathcal{I}}) \in Team^{\mathcal{I}}$ | $(juventus^{\mathcal{I}}) \in Team^{\mathcal{I}}$ |
| $(r7^{\mathcal{I}}, \texttt{"7"}) \in \mathbf{code}^{\mathcal{I}}$ | $(m7RJ^{\mathcal{I}}, \texttt{"RJ"}) \in \mathbf{code}^{\mathcal{I}}$ | $(m7RJ^{\mathcal{I}}, \texttt{5/4/09}) \in \mathbf{date}^{\mathcal{I}}$ |
| $(r8^{\mathcal{I}}, \texttt{"8"}) \in \mathbf{code}^{\mathcal{I}}$ | $(m8NT^{\mathcal{I}}, \texttt{"NT"}) \in \mathbf{code}^{\mathcal{I}}$ | $(m7RJ^{\mathcal{I}}, \texttt{3}) \in \mathbf{homeGoals}^{\mathcal{I}}$ |
| | $(m8RM^{\mathcal{I}}, \texttt{"RM"}) \in \mathbf{code}^{\mathcal{I}}$ | $(m7RJ^{\mathcal{I}}, \texttt{1}) \in \mathbf{hostGoals}^{\mathcal{I}}$ |

**Fig. 5.** A model of the ABox $\mathcal{A}_{fbc}$ for the football championship example

Furthermore, the following are necessary conditions for $\mathcal{I}$ to be also a model of the TBox $\mathcal{T}_{fbc}$, and hence of $\mathcal{O}_{fbc}$:

| | | |
|---|---|---|
| $it2009^{\mathcal{I}} \in (\exists OF)^{\mathcal{I}}$ | to satisfy | $League \sqsubseteq \exists OF$, |
| $it2009^{\mathcal{I}} \in (\delta(\mathbf{year}))^{\mathcal{I}}$ | to satisfy | $League \sqsubseteq \delta(\mathbf{year})$, |
| $m7RJ^{\mathcal{I}} \in Match^{\mathcal{I}}$ | to satisfy | $PlayedMatch \sqsubseteq Match$, |
| $torino^{\mathcal{I}} \in Team^{\mathcal{I}}$ | to satisfy | $\exists HOST^{-} \sqsubseteq Team$, |
| $milan^{\mathcal{I}} \in Team^{\mathcal{I}}$ | to satisfy | $\exists HOST^{-} \sqsubseteq Team$. |

Notice that, in order for an interpretation $\mathcal{I}$ to satisfy the condition specified in the first row above, there must be an object $o \in \Delta_O^{\mathcal{I}}$ such that $(it2009^{\mathcal{I}}, o) \in OF^{\mathcal{I}}$. According to the inclusion assertion $\exists OF^{-} \sqsubseteq Nation$, such an object $o$ must also belong to $Nation^{\mathcal{I}}$ (indeed, in our ontology, every league is of one nation). Similarly, the second row above derives from the property that every league must have a year.

We note that, besides satisfying the conditions discussed above, an interpretation $\mathcal{I}'$ may also add other elements to the interpretation of concepts, attributes, or roles specified by $\mathcal{I}$. For instance, the interpretation $\mathcal{I}'$ that adds to $\mathcal{I}$ the object

$$\mathit{italy}^{\mathcal{I}} \in \mathit{Nation}^{\mathcal{I}}$$

is still a model of the ontology $\mathcal{O}_{fbc}$.

Note, finally, that there exists no model of $\mathcal{O}_{fbc}$ such that *m7RJ* is interpreted as an instance of *ScheduledMatch*, since *m7RJ* has to be interpreted as an instance of *PlayedMatch*, and according to the inclusion assertion

$$\mathit{PlayedMatch} \sqsubseteq \neg \mathit{ScheduledMatch},$$

the sets of played matches and of scheduled matches are disjoint. ∎

The above example clearly shows the difference between a database and an ontology. From a database point of view the ontology $\mathcal{O}_{fbc}$ discussed in the example might seem incorrect: for example, while the TBox $\mathcal{T}_{fbc}$ sanctions that every league has a year, there is no explicit year for *it2009* in the ABox $\mathcal{A}_{fbc}$. However, the ontology is not incorrect: the axiom stating that every league has a year simply specifies that in every model of $\mathcal{O}_{fbc}$ there will be a year for *it2009*, even if such a year is not known.

### 2.3 *DL-Lite*$_{\mathcal{A},id}$ vs. OWL 2 QL

Having now completed the definition of the syntax and semantics of *DL-Lite*$_{\mathcal{A},id}$, we would like to point out that *DL-Lite*$_{\mathcal{A},id}$ is at the basis of OWL 2 QL, one of the three profiles of OWL 2 that are currently being standardized by the World-Wide-Web Consortium (W3C). The *OWL 2 profiles*[8] are fragments of the full OWL 2 language that have been designed and standardized for specific application requirements. According to (the current version of) the official W3C profiles document, "OWL 2 QL includes most of the main features of conceptual models such as UML class diagrams and ER diagrams. [It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems." We will substantiate all these claims in the next sections.

Here, we briefly point out the most important differences between *DL-Lite*$_{\mathcal{A},id}$ and OWL 2 QL (apart from differences in terminology and syntax, which we do not mention):

(1) The main difference is certainly the fact that OWL 2 QL does not adopt the *unique name assumption*, while such assumption holds for *DL-Lite*$_{\mathcal{A},id}$ (and the whole *DL-Lite* family, in fact). The reason for this semantic mismatch is on the one hand that OWL 2, as most DLs, does not adopt the UNA, and since the profiles are intended to be *syntactic* fragments of the full OWL 2 language, it was not desirable for a profile to change a basic semantic assumption. On the other hand, the UNA is at the basis of data management in databases, and moreover, by dropping it, *DL-Lite*$_{\mathcal{A},id}$ would lose its nice computational properties (cf. Theorem 6.6).

[8] `http://www.w3.org/TR/owl2-profiles/`

(2) OWL 2 QL does not allow for expressing *functionality* of roles or attributes, or *identification assertions*, while such constructs are present in *DL-Lite*$_{\mathcal{A},id}$. This aspect is related to Item (1), and motivated by the fact that the OWL 2 QL profile is intended to have the same nice computational properties as *DL-Lite*$_{\mathcal{A},id}$. In order to preserve such properties even in the absence of the UNA, the proof of Theorem 6.6 tells us that we need to avoid the use of functionality (and of identification assertions, since these can be used to simulate functionality). Indeed, as testified also by the complexity results in [4], in the absence of these constructs, the UNA has no impact on complexity of reasoning, and hence OWL 2 QL exhibits the same computational properties as *DL-Lite*$_{\mathcal{A},id}$.

(3) OWL 2 QL includes the possibility to assert additional role properties, such as disjointness, reflexivity, irreflexivity, symmetry, and asymmetry, that are not explicitly present in *DL-Lite*$_{\mathcal{A},id}$. It is immediate to see that disjointness between roles $Q_1$ and $Q_2$ can be expressed by means of $Q_1 \sqsubseteq \neg Q_2$, and that reflexivity of a role $P$ can be expressed by means of $P \sqsubseteq P^-$. Moreover, as shown in [4], also the addition of irreflexivity, symmetry, and asymmetry does not affect the computational complexity of inference (including query answering, see Section 2.4), and such constructs could be incorporated in the reasoning algorithms for *DL-Lite*$_{\mathcal{A},id}$ with only minor changes.

(4) OWL 2 QL inherits its specific *datatypes* (corresponding to the value domains of *DL-Lite*$_{\mathcal{A},id}$) from OWL 2, while *DL-Lite*$_{\mathcal{A},id}$ does not provide any details about datatypes. However, OWL 2 QL imposes restrictions on the allowed datatypes that ensure that no datatype has an unbounded domain, which is sufficient to guarantee that datatypes will not interfere unexpectedly in reasoning.

We remark that, due to the correspondence between OWL 2 QL and *DL-Lite*$_{\mathcal{A},id}$, all the results and techniques presented in the next sections have a direct impact on OWL 2, i.e., on a standard language for the Semantic Web, that builds on a large user base. Hence, such results are of immediate practical relevance.

### 2.4 Queries over *DL-Lite*$_{\mathcal{A},id}$ Ontologies

We are interested in queries over ontologies expressed in *DL-Lite*$_{\mathcal{A},id}$. Similarly to the case of relational databases, the basic query class that we consider is the class of unions of conjunctive queries, which is a subclass of the class of First-Order Logic queries.

**Syntax of Queries.** A First-Order Logic (FOL) *query* $q$ over a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$) is a, possibly open, FOL formula $\varphi(\boldsymbol{x})$ whose predicate symbols are atomic concepts, value-domains, roles, or attributes of $\mathcal{O}$ (resp., $\mathcal{T}$). The free variables of $\varphi(\boldsymbol{x})$ are those appearing in $\boldsymbol{x}$, which is a tuple of (pairwise distinct) variables. In other words, the atoms of $\varphi(\boldsymbol{x})$ have the form $A(x)$, $D(x)$, $P(x, y)$, $U(x, y)$, or $x = y$, where:

- $A$, $F$, $P$, and $U$ are respectively an atomic concept, a value-domain, an atomic role, and an atomic attribute in $\mathcal{O}$,
- $x$, $y$ are either variables in $\boldsymbol{x}$ or constants in $\Gamma$.

The *arity* of $q$ is the arity of $\boldsymbol{x}$. A query of arity 0 is called a *boolean query*. When we want to make the arity of a query $q$ explicit, we denote the query as $q(\boldsymbol{x})$.

A *conjunctive query* (CQ) $q(\boldsymbol{x})$ over a *DL-Lite*$_{\mathcal{A},id}$ ontology is a FOL query of the form

$$\exists \boldsymbol{y}. \, conj(\boldsymbol{x}, \boldsymbol{y}),$$

where $\boldsymbol{y}$ is a tuple of pairwise distinct variables not occurring among the free variables $\boldsymbol{x}$, and where $conj(\boldsymbol{x}, \boldsymbol{y})$ is a *conjunction* of atoms. The variables $\boldsymbol{x}$ are also called *distinguished* and the (existentially quantified) variables $\boldsymbol{y}$ are called *non-distinguished*. We will also make use of *conjunctive queries with inequalities*, which are CQs in which also atoms of the form $x \neq y$ (called *inequalities*) may appear.

A *union of conjunctive queries* (UCQ) is a FOL query that is the disjunction of a set of CQs of the same arity, i.e., it is a FOL formula of the form:

$$\exists \boldsymbol{y}_1. \, conj_1(\boldsymbol{x}, \boldsymbol{y}_1) \vee \cdots \vee \exists \boldsymbol{y}_n. \, conj_n(\boldsymbol{x}, \boldsymbol{y}_n).$$

UCQs with inequalities are obvious extensions of UCQs.

Finally, a *positive FOL query* is a FOL query $\varphi(\boldsymbol{x})$ where the formula $\varphi$ is built using only conjunction, disjunction, and existential quantification (i.e., it contains neither negation nor universal quantification).

**Datalog Notation for CQs and UCQs.** In the following, it will sometimes be convenient to consider a UCQ as a set of CQs, rather than as a disjunction of UCQs. We will also use the *Datalog* notation for CQs and UCQs. In this notation, a CQ is written as

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

and a UCQ is written as a set of CQs

$$\begin{aligned} q(\boldsymbol{x}) &\leftarrow conj'_1(\boldsymbol{x}, \boldsymbol{y}_1) \\ &\vdots \\ q(\boldsymbol{x}) &\leftarrow conj'_n(\boldsymbol{x}, \boldsymbol{y}_n) \end{aligned}$$

where $conj'(\boldsymbol{x}, \boldsymbol{y})$ and each $conj'_i(\boldsymbol{x}, \boldsymbol{y}_i)$ in a CQ are considered simply as sets of atoms (written in list notation, using a ',' as a separator). In this case, we say that $q(\boldsymbol{x})$ is the *head* of the query, and that $conj'(\boldsymbol{x}, \boldsymbol{y})$ and each $conj'_i(\boldsymbol{x}, \boldsymbol{y}_i)$ is the *body* of the corresponding CQ.

**Semantics of Queries.** Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, the FOL query $q = \varphi(\boldsymbol{x})$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\boldsymbol{o} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that the formula $\varphi$ evaluates to true in $\mathcal{I}$ under the assignment that assigns each object in $\boldsymbol{o}$ to the corresponding variable in $\boldsymbol{x}$ [1]. We call $q^{\mathcal{I}}$ the *answer* to $q$ over $\mathcal{I}$. Notice that the answer to a boolean query is either the empty tuple, "()", considered as $true$, or the empty set, considered as $false$.

We remark that a relational database (over the atomic concepts, roles, and attributes) corresponds to a finite interpretation. Hence the notion of answer to a query introduced here is the standard notion of answer to a query evaluated over a relational database.

In the case where the query is a CQ, the above definition of answer can be rephrased in terms of homomorphisms. In general, a homomorphisms between two interpretations (i.e., First-Order structures) is defined as follows.

**Definition 2.4.** *Given two interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *and* $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ *over the same set* $\mathcal{P}$ *of predicate symbols, a* homomorphism $\mu$ *from* $\mathcal{I}$ *to* $\mathcal{J}$ *is a mapping* $\mu : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{J}}$ *such that, for each predicate* $P \in \mathcal{P}$ *of arity* $n$ *and each tuple* $(o_1, \ldots, o_n) \in (\Delta^{\mathcal{I}})^n$*, if* $(o_1, \ldots, o_n) \in P^{\mathcal{I}}$*, then* $(\mu(o_1), \ldots, \mu(o_n)) \in P^{\mathcal{J}}$.

Notice that, in the case of interpretations of a *DL-Lite*$_{\mathcal{A},id}$ ontology, the set of predicate symbols in the above definition would be the set of atomic concepts, value domains, roles, and attributes of the ontology.

We can now extend the definition to consider also homomorphisms from CQs to interpretations.

**Definition 2.5.** *Given a CQ* $q(\boldsymbol{x}) = \exists \boldsymbol{y}.\ conj(\boldsymbol{x}, \boldsymbol{y})$ *over interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$*, and a tuple* $\boldsymbol{o} = (o_1, \ldots, o_n)$ *of objects of* $\Delta^{\mathcal{I}}$ *of the same arity as* $\boldsymbol{x} = (x_1, \ldots, x_n)$*, a* homomorphism *from* $q(\boldsymbol{o})$ *to* $\mathcal{I}$ *is a mapping* $\mu$ *from the variables and constants in* $q(\boldsymbol{x})$ *to* $\Delta^{\mathcal{I}}$ *such that:*

- $\mu(c) = c^{\mathcal{I}}$*, for each constant* $c$ *in* $conj(\boldsymbol{x}, \boldsymbol{y})$*,*
- $\mu(x_i) = o_i$*, for* $i \in \{1, \ldots, n\}$*, and*
- $(\mu(t_1), \ldots, \mu(t_n)) \in P^{\mathcal{I}}$*, for each atom* $P(t_1, \ldots, t_n)$ *that appears in* $conj(\boldsymbol{x}, \boldsymbol{y})$.

The following result established in [32] provides a fundamental characterization of answers to CQs in terms of homomorphism.

**Theorem 2.6 ([32]).** *Given a CQ* $q(\boldsymbol{x}) = \exists \boldsymbol{y}.\ conj(\boldsymbol{x}, \boldsymbol{y})$ *over an interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$*, and a tuple* $\boldsymbol{o} = (o_1, \ldots, o_n)$ *of objects of* $\Delta^{\mathcal{I}}$ *of the same arity as* $\boldsymbol{x} = (x_1, \ldots, x_n)$*, we have that* $\boldsymbol{o} \in q^{\mathcal{I}}$ *if and only if there is a homomorphism from* $q(\boldsymbol{o})$ *to* $\mathcal{I}$.

In fact, the notion of homomorphism is crucial in the context of the study of CQs, and most inference tasks involving CQs (including query containment [58], and tasks related to view-based query processing [46]) can be rephrased in terms of homomorphism [1].

*Example 2.7.* Consider again the ontology $\mathcal{O}_{fbc} = \langle \mathcal{T}_{fbc}, \mathcal{A}_{fbc} \rangle$ introduced in Example 2.3, and the following query asking for all matches:

$$q_1(x) \leftarrow \mathit{Match}(x).$$

If $\mathcal{I}$ is the interpretation shown in Figure 5, we have that:

$$q_1^{\mathcal{I}} = \{(\mathit{m8NT}^{\mathcal{I}}), (\mathit{m8RM}^{\mathcal{I}})\}.$$

Notice that $\mathcal{I}$ is a model of $\mathcal{A}_{fbc}$, but not of $\mathcal{T}_{fbc}$. Let instead $\mathcal{I}'$ be the interpretation analogous to $\mathcal{I}$, but extended in such a way that it becomes also a model of $\mathcal{T}_{fbc}$, and hence of $\mathcal{O}_{fbc}$, as shown in Example 2.3. Then we have that:

$$q_1^{\mathcal{I}'} = \{(\mathit{m8NT}^{\mathcal{I}}),\ (\mathit{m8RM}^{\mathcal{I}}),\ (\mathit{m7RJ}^{\mathcal{I}})\}.$$

Suppose now that we ask for teams, together with the code of the match in which they have played as home team:

$$q_2(t, c) \leftarrow \mathit{Team}(t), \mathit{HOME}(m, t), \mathit{Match}(m), \mathbf{code}(m, c).$$

Then we have that

$$q_2^{\mathcal{I}} = \{(\mathit{napoli}^{\mathcal{I}}, \texttt{"NT"}),\ (\mathit{roma}^{\mathcal{I}}, \texttt{"RM"})\},$$
$$q_2^{\mathcal{I}'} = \{(\mathit{roma}^{\mathcal{I}}, \texttt{"RJ"}),\ (\mathit{napoli}^{\mathcal{I}}, \texttt{"NT"}),\ (\mathit{roma}^{\mathcal{I}}, \texttt{"RM"})\}.$$

■

**Certain Answers.** The notion of answer to a query introduced above is not sufficient to capture the situation where a query is posed over an ontology, since in general an ontology will have many models, and we cannot single out a unique interpretation (or database) over which to answer the query. Instead, the ontology determines a set of interpretations, i.e., the set of its models, which intuitively can be considered as the set of databases that are "compatible" with the information specified in the ontology. Given a query, we are interested in those answers to this query that depend only on the information in the ontology, i.e., that are obtained by evaluating the query over a database compatible with the ontology, but independently of which is the actually chosen database. In other words, we are interested in those answers to the query that are obtained for *all* possible databases (including infinite ones) that are models of the ontology. This corresponds to the fact that the ontology conveys only incomplete information about the domain of interest, and we want to guarantee that the answers to a query that we obtain are *certain*, independently of how we complete this incomplete information. This leads us to the following definition of *certain answers* to a query over an ontology.

**Definition 2.8.** *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A},id}$ ontology and $q$ a UCQ over $\mathcal{O}$. A tuple $\mathbf{c}$ of constants appearing in $\mathcal{O}$ is a* certain answer *to $q$ over $\mathcal{O}$, written $\mathbf{c} \in cert(q, \mathcal{O})$, if for every model $\mathcal{I}$ of $\mathcal{O}$, we have that $\mathbf{c}^{\mathcal{I}} \in q^{\mathcal{I}}$.*

Answering a query $q$ posed to an ontology $\mathcal{O}$ means exactly to compute the set of certain answers to $q$ over $\mathcal{O}$.

*Example 2.9.* Consider again the ontology introduced in Example 2.3, and queries $q_1$ and $q_2$ introduced in Example 2.7. One can easily verify that

$$cert(q_1, \mathcal{O}) = \{(\mathit{m8NT}^{\mathcal{I}}),\ (\mathit{m8RM}^{\mathcal{I}}),\ (\mathit{m7RJ}^{\mathcal{I}})\},$$
$$cert(q_2, \mathcal{O}) = \{(\mathit{roma}^{\mathcal{I}}, \texttt{"RJ"}),\ (\mathit{napoli}^{\mathcal{I}}, \texttt{"NT"}),\ (\mathit{roma}^{\mathcal{I}}, \texttt{"RM"})\}.$$

■

Notice that, in the case where $\mathcal{O}$ is an unsatisfiable ontology, the set of certain answers to a (U)CQ $q$ is the finite set of all possible tuples of constants whose arity is the one of $q$. We denote such a set by $\mathit{AllTup}(q, \mathcal{O})$.

### 2.5 Reasoning Services

In studying *DL-Lite*$_{\mathcal{A},id}$, we are interested in several reasoning services, including the traditional DL reasoning services. Specifically, we consider the following problems for *DL-Lite*$_{\mathcal{A},id}$ ontologies:

- *Ontology satisfiability*, i.e., given an ontology $\mathcal{O}$, verify whether $\mathcal{O}$ admits at least one model.
- *Concept and role satisfiability*, i.e., given a TBox $\mathcal{T}$ and a concept $C$ (resp., a role $R$), verify whether $\mathcal{T}$ admits a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$ (resp., $R^{\mathcal{I}} \neq \emptyset$).
- We say that an ontology $\mathcal{O}$ (resp., a TBox $\mathcal{T}$) *logically implies* an assertion $\alpha$, denoted $\mathcal{O} \models \alpha$ (resp., $\mathcal{T} \models \alpha$, if every model of $\mathcal{O}$ (resp., $\mathcal{T}$) satisfies $\alpha$. The problem of *logical implication of assertions* consists of the following sub-problems:
  - *instance checking*, i.e., given an ontology $\mathcal{O}$, a concept $C$ and a constant $a$ (resp., a role $R$ and a pair of constants $a_1$ and $a_2$), verify whether $\mathcal{O} \models C(a)$ (resp., $\mathcal{O} \models R(a_1, a_2)$);
  - *subsumption of concepts or roles*, i.e., given a TBox $\mathcal{T}$ and two general concepts $C_1$ and $C_2$ (resp., two general roles $R_1$ and $R_2$), verify whether $\mathcal{T} \models C_1 \sqsubseteq C_2$ (resp., $\mathcal{T} \models R_1 \sqsubseteq R_2$);
  - *checking functionality*, i.e., given a TBox $\mathcal{T}$ and a basic role $Q$, verify whether $\mathcal{T} \models (\mathsf{funct}\ Q)$.
  - *checking an identification constratins*, i.e., given a TBox $\mathcal{T}$ and an identification constraint $(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$, verify whether $\mathcal{T} \models (\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$.

In addition we are interested in:

- *Query answering*, i.e., given an ontology $\mathcal{O}$ and a query $q$ (either a CQ or a UCQ) over $\mathcal{O}$, compute the set $cert(q, \mathcal{O})$.

The following decision problem, called *recognition problem*, is associated to the query answering problem: given an ontology $\mathcal{O}$, a query $q$ (either a CQ or a UCQ), and a tuple of constants $\boldsymbol{a}$ of $\mathcal{O}$, check whether $\boldsymbol{a} \in cert(q, \mathcal{O})$. When we talk about the computational complexity of query answering, in fact we implicitly refer to the associated recognition problem.

In analyzing the computational complexity of a reasoning problem over a DL ontology, we distinguish between data complexity and combined complexity [90]: *data complexity* is the complexity measured with respect to the size of the ABox only, while *combined complexity* is the complexity measured with respect to the size of all inputs to the problem, i.e., the TBox, the ABox, and the query. The data complexity measure is of interest in all those cases where the size of the intensional level of the ontology (i.e., the TBox) is negligible w.r.t. the size of the data (i.e., the ABox), as in ontology-based data access (cf. Section 1.3).

### 2.6 The Notion of FOL-Rewritability

We now introduce the notion of FOL-rewritability for both satisfiability and query answering, which will be used in the sequel.

First, given an ABox $\mathcal{A}$ (of the kind considered above), we denote by $DB(\mathcal{A}) = \langle \Delta^{DB(\mathcal{A})}, \cdot^{DB(\mathcal{A})} \rangle$ *the interpretation* defined as follows:

- $\Delta^{DB(\mathcal{A})}$ is the non-empty set consisting of the union of the set of all object constants occurring in $\mathcal{A}$ and the set $\{val(c) \mid c \text{ is a value constant that occurs in } \mathcal{A}\}$,
- $a^{DB(\mathcal{A})} = a$, for each object constant $a$,
- $A^{DB(\mathcal{A})} = \{a \mid A(a) \in \mathcal{A}\}$, 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 atomic attribute $U$.

Observe that the interpretation $DB(\mathcal{A})$ is a minimal model of the ABox $\mathcal{A}$.

Intuitively, FOL-rewritability of 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 2.10.** *Satisfiability in a DL $\mathcal{L}$ is* FOL-rewritable*, if for every TBox $\mathcal{T}$ expressed in $\mathcal{L}$, there exists a boolean FOL query $q$, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$, the ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if $q$ evaluates to false in $DB(\mathcal{A})$.*

**Definition 2.11.** *Answering UCQs in a DL $\mathcal{L}$ is* FOL-rewritable*, if for every UCQ $q$ and every TBox $\mathcal{T}$ expressed over $\mathcal{L}$, there exists a FOL query $q_1$, over the alphabet of $\mathcal{T}$, such that for every non-empty ABox $\mathcal{A}$ and every tuple of constants $\boldsymbol{a}$ occurring in $\mathcal{A}$, we have that $\boldsymbol{a} \in cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ if and only if $\boldsymbol{a}^{DB(\mathcal{A})} \in q_1^{DB(\mathcal{A})}$.*

We remark that FOL-rewritability of a reasoning problem that involves the ABox of an ontology (such as satisfiability or query answering) is tightly related to the data complexity of the problem. Indeed, since the FOL query considered in the above definitions depends only on the TBox (and the query), but not on the ABox, and since the evaluation of a First-Order Logic query (i.e., an SQL query without aggregation) over an ABox is in $AC^0$ in data complexity [1], FOL-rewritability of a problem has as an immediate consequence that the problem is in $AC^0$ in data complexity. Hence, one way of showing that for a certain DL $\mathcal{L}$ a problem is *not* FOL-rewritable, is to show that the data complexity of the problem for the DL $\mathcal{L}$ is above $AC^0$, e.g., LOGSPACE-hard, NLOGSPACE-hard, PTIME-hard, or even coNP-hard. We will provide some results of this form in Section 6 (see also [4]).

## 3 UML Class Diagrams as an Ontology Language

In this section, we discuss how UML class diagrams can be considered as an ontology language, and we show how such diagrams can be captured in *DL-Lite*$_{\mathcal{A},id}$.

Since we concentrate on class diagrams from the conceptual perspective, we do not deal with those features that are more relevant for the software engineering perspective, such as operations (methods) associated to classes, or public, protected, and private qualifiers for methods and attributes. Also, for sake of brevity and to smooth the presentation we make some simplifying assumptions that could all be lifted without changing the results presented here (we refer to [11] for further details). In particular, we will not deal explicitly with associations of arity greater than 2, and we will only deal with the following multiplicities:

- unconstrained, i.e., $0..*$,
- functional participation, i.e., $0..1$,
- mandatory participation, i.e., $1..*$, and
- one-to-one correspondence, i.e., $1..1$.

These multiplicities are particularly important since they convey meaningful semantic aspects in modeling, and thus are the most commonly used ones.

Our goal is twofold. On the one hand, we aim at showing how class diagrams can be expressed in DLs. On the other hand, we aim at understanding which is the complexity of inference over an UML class diagram. We will show that the formalization in DLs helps us in deriving complexity results both for reasoning and for query answering over an UML class diagram.

### 3.1 Classes and Attributes

A *class* in a UML class diagram denotes a *sets of objects* with common features. The specification of a class contains its *name* and its *attributes*, each denoted by a name (possibly followed by the *multiplicity*, between square brackets) and with an associated *type*, which indicates the domain of the attribute values. A UML class is represented by a DL concept. This follows naturally from the fact that both UML classes and DL concepts denote *sets of objects*.

A UML *attribute* $a$ of type $T$ for a class $C$ associates to each instance of $C$, zero, one, or more instances of type $T$. An optional *multiplicity* $[i..j]$ for $a$ specifies that $a$ associates to each instance of $C$, at least $i$ and most $j$ instances of $T$. When the multiplicity for an attribute is missing, $[1..1]$ is assumed, i.e., the attribute is *mandatory* and *single-valued*.

To formalize attributes, we have to think of an attribute $a$ of type $T$ for a class $C$ as a binary relation between instances of $C$ and instances of $T$. We capture such a binary relation by means of a DL attribute $a_C$. To specify the type of the attribute we use the DL assertions

$$\delta(a_C) \sqsubseteq C, \qquad\qquad \rho(a_C) \sqsubseteq T.$$

Such assertions specify precisely that, for each instance $(c, v)$ of the attribute $a_C$, the object $c$ is an instance of $C$, and the value $v$ is an instance of $T$. Note that the attribute name $a$ is not necessarily unique in the whole diagram, and hence two different classes, say $C$ and $C'$ could both have attribute $a$, possibly of different types. This situation is correctly captured in the DL formalization, where the attribute is contextualized to each class with a distinguished DL attribute, i.e., $a_C$ and $a_{C'}$.

To specify that the attribute is mandatory (i.e., multiplicity $[1..*]$), we add the assertion

$$C \sqsubseteq \delta(a_C),$$

which specifies that each instance of $C$ participates necessarily at least once to the DL attribute $a_C$. To specify that the attribute is single-valued (i.e., multiplicity $[0..1]$), we add the functionality assertion

$$(\mathsf{funct}\ a_C).$$

Finally, if the attribute is both mandatory and single-valued (i.e., multiplicity $[1..1]$), we use both assertions together, i.e.,

$$C \sqsubseteq \delta(a_C), \qquad (\mathsf{funct}\ a_C).$$

### 3.2 Associations

An *association* in UML is a relation between the instances of two (or more) classes. An association often has a related *association class* that describes properties of the association, such as attributes, operations, etc. A binary association $A$ between the instances of two classes $C_1$ and $C_2$ is graphically rendered as in Figure 6(a), where the *multiplicity* $m_\ell..m_u$ specifies that each instance of class $C_1$ can participate at least $m_\ell$ times and at most $m_u$ times to association $A$. The multiplicity $n_\ell..n_u$ has an analogous meaning for class $C_2$.

An association $A$ between classes $C_1$ and $C_2$ is formalized in DL by means of a role $A$ on which we enforce the assertions

$$\exists A \sqsubseteq C_1, \qquad \exists A^- \sqsubseteq C_2.$$

To express the multiplicity $m_\ell..m_u$ on the participation of instances of $C_2$ for each given instance of $C_1$, we use the assertion $C_1 \sqsubseteq \exists A$, if $m_\ell = 1$, and $(\mathsf{funct}\ A)$, if $m_u = 1$. We can use similar assertions for the multiplicity $n_\ell..n_u$ on the participation of instances of $C_1$ for each given instance of $C_2$, i.e., $C_2 \sqsubseteq \exists A^-$, if $n_\ell = 1$, and $(\mathsf{funct}\ A^-)$, if $n_u = 1$.

Next we focus on *associations* with a related *association class*, as shown in Figure 6(b), where the class $A$ is the association class related to the association, and $R_{A,1}$ and $R_{A,2}$, if present, are the *role names* of $C_1$ and $C_2$ respectively, i.e., they specify the role that each class plays within the association $A$.

We formalize in DL an association $A$ with an association class, by reifying it into a DL concept $A$ and introducing two DL roles $R_{A,1}$, $R_{A,2}$, one for each role of $A$, which intuitively connect an object representing an instance of the association respectively to the instances of $C_1$ and $C_2$ that participate to the association[9]. Then, we enforce that each instance of $A$ participates exactly once both to $R_{A,1}$ and to $R_{A,2}$, by means of the assertions

$$A \sqsubseteq \exists R_{A,1}, \qquad (\mathsf{funct}\ R_{A,1}), \qquad A \sqsubseteq \exists R_{A,2}, \qquad (\mathsf{funct}\ R_{A,2}).$$

To represent that the association $A$ is between classes $C_1$ and $C_2$, we use the assertions

$$\exists R_{A,1} \sqsubseteq A, \qquad \exists R_{A,1}^- \sqsubseteq C_1, \qquad \exists R_{A,2} \sqsubseteq A, \qquad \exists R_{A,2}^- \sqsubseteq C_2.$$

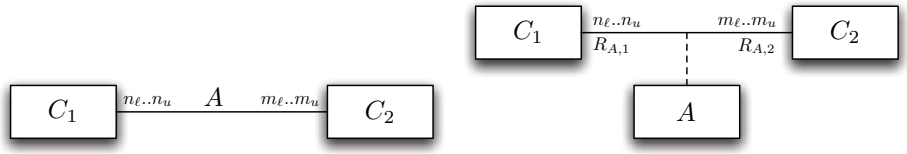
Finally, we use the assertion

$$(\mathsf{id}\ A\ R_{A,1}, R_{A,2})$$

to specify that each instance of the concept $A$ represents a *distinct* tuple in $C_1 \times C_2$.[10]

[9] If the roles of the association are not available, we may use an arbitrary DL role name.

[10] Notice that such an approach can immediately be used to represent an association of any arity: it suffices to repeat the above for every component.




(a) Without association class. (b) With association class.

**Fig. 6.** Associations in UML

We can easily represent in DL multiplicities on an association with association class, by imposing suitable assertions on the inverses of the DL roles modeling the roles of the association. For example, to say that there is a one-to-one participation of instances of $C_1$ in the association (with related association class) $A$, we assert

$$C_1 \sqsubseteq \exists R^-_{A,1}, \qquad (\mathsf{funct}\ R^-_{A,1}).$$

### 3.3 Generalizations and Class Hierarchies

In UML, one can use *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that in general do not hold for the parent class.

Generalization is naturally supported in DLs. If a UML class $C_2$ generalizes a class $C_1$, we can express this by the DL assertion

$$C_1 \sqsubseteq C_2.$$

Inheritance between DL concepts works exactly as inheritance between UML classes. This is an obvious consequence of the semantics of $\sqsubseteq$, which is based on the subset relation. As a consequence, in the formalization, each attribute of $C_2$ and each association involving $C_2$ is correctly inherited by $C_1$. Observe that the formalization in DL also captures directly multiple inheritance between classes.

In UML, one can group several generalizations into a *class hierarchy*, as shown in Figure 7. Such a hierarchy is captured in DL by a set of inclusion assertions, one between each child class and the parent class, i.e.,

$$C_i \sqsubseteq C, \qquad \text{for each } i \in \{1, \ldots, n\}.$$

Often, when defining generalizations between classes, we need to add additional assertions among the involved classes. For example, for the class hierarchy in Figure 7, an assertion may express that $C_1, \ldots, C_n$ are *mutually disjoint*. In DL, such a relationship can be expressed by the assertions

$$C_i \sqsubseteq \neg C_j, \qquad \text{for each } i, j \in \{1, \ldots, n\} \text{ with } i \neq j.$$

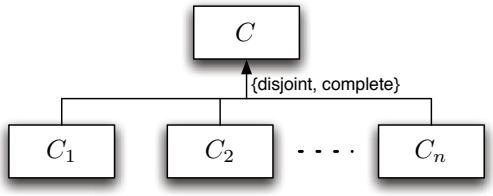



**Fig. 7.** A class hierarchy in UML

Moreover, we may want to express that a generalization hierarchy is *complete*, i.e., that the subclasses $C_1, \ldots, C_n$ are a *covering* of the superclass $C$. We can represent such a situation in DL by including the additional assertion

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_n.$$

Such an assertion models a form of *disjunctive information*: each instance of $C$ is either an instance of $C_1$, or an instance of $C_2$, ... or an instance of $C_n$. Notice, however, that the use of concept disjunction, and hence of the inclusion assertion above, is not allowed in *DL-Lite*$_{\mathcal{A},id}$.

### 3.4 Subset Assertions between Associations

Similarly to generalization between classes, UML allows one to state *subset assertions* between associations. A subset assertion between two associations $A_1$ and $A_2$ can be modeled in DL by means of the role inclusion assertion

$$A_1 \sqsubseteq A_2,$$

involving the two DL roles $A_1$ and $A_2$ representing the associations. Notice that this is allowed in *DL-Lite*$_{\mathcal{A},id}$ only if none of the maximum multiplicities of the two classes participating to $A_2$ is equal to 1.

With respect to a generalization between two association classes $A_1$ and $A_2$, we note that to correctly capture the corresponding subset assertion between the associations represented by the association classes, we would need to introduce not only inclusion assertions between the concepts representing the association classes, but also between the DL roles representing corresponding roles of $A_1$ and $A_2$. Consider, for example, the generalization between association classes depicted in Figure 8. We can correctly capture the two associations with association classes $A_1$ and $A_2$ by the following DL assertions:

$$\begin{array}{llll}
A_1 \sqsubseteq \exists R_{A1,1}, & (\mathsf{funct}\ R_{A1,1}), & A_2 \sqsubseteq \exists R_{A2,1}, & (\mathsf{funct}\ R_{A2,1}), \\
A_1 \sqsubseteq \exists R_{A1,2}, & (\mathsf{funct}\ R_{A1,2}), & A_2 \sqsubseteq \exists R_{A2,2}, & (\mathsf{funct}\ R_{A2,2}), \\
\exists R_{A1,1} \sqsubseteq A_1, & \exists R^-_{A1,1} \sqsubseteq C_{11}, & \exists R_{A2,1} \sqsubseteq A_2, & \exists R^-_{A2,1} \sqsubseteq C_{21}, \\
\exists R_{A1,2} \sqsubseteq A_1, & \exists R^-_{A1,2} \sqsubseteq C_{12}, & \exists R_{A2,2} \sqsubseteq A_2, & \exists R^-_{A2,2} \sqsubseteq C_{22}, \\
\multicolumn{2}{c}{(\mathsf{id}\ A_1\ R_{A1,1}, R_{A1,2}),} & \multicolumn{2}{c}{(\mathsf{id}\ A_2\ R_{A2,1}, R_{A2,2}).}
\end{array}$$

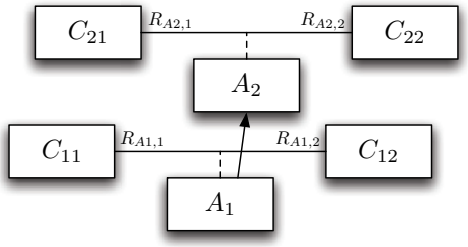



**Fig. 8.** A generalization between association classes in UML

Finally, to capture the generalization, we could use the following inclusion assertions

$$A_1 \sqsubseteq A_2, \qquad R_{A1,1} \sqsubseteq R_{A2,1}, \qquad R_{A1,2} \sqsubseteq R_{A2,2}.$$

However, since $R_{A2,1}$ and $R_{A2,2}$ are functional roles (and are also used in an identification assertion), we actually cannot specialize them, if we want to stay in *DL-Lite*$_{\mathcal{A},id}$. Hence, generalization of associations with association classes in general cannot be formalized in *DL-Lite*$_{\mathcal{A},id}$.

Finally, we observe that a formalization in *DL-Lite*$_{\mathcal{A},id}$ of generalization between association classes is possible if the sub-association does not specify new classes for the domain and range of the association with respect to the super-association. In the example of Figure 8 this would mean that $C_{11}$ coincides with $C_{21}$ and $C_{12}$ coincides with $C_{22}$. In this case, the sub-association $A_1$ is represented by simply using the same DL roles as for $A_2$ to denote its components. Hence, it is not necessary to introduce an inclusion assertion between functional DL roles to correctly capture the generalization between association classes.

### 3.5 Reasoning and Query Answering over UML Class Diagrams

The fact that UML class diagrams can be captured by DLs enables the possibility of performing sound and complete reasoning to do formal verification at design time and query answering at runtime, as will be illustrated in the next sections. Hence, one can exploit such ability to get support during the design phase of an ontology-based data access system, and to take the information in the UML class diagram fully into account during query answering.

It was shown in [11] that, unfortunately, reasoning (in particular checking the consistency of the diagram, a task to which other typical reasoning tasks of interest reduce) is EXPTIME-hard. What this result tells us is that, if the TBox is expressed in UML, then the support at design time for an ontology-based data access system may be difficult to obtain if the schema has a reasonable size.

Turning to query answering, the situation is even worse. The results in Section 6 imply that answering conjunctive queries in the presence of a UML class diagram formed by a single generalization with covering assertion is coNP-hard in the size of the instances of classes and associations. Hence, query answering over even moderately large data sets is again infeasible in practice. It is not difficult to see that this implies that, in

an ontology-based data access system where the TBox is expressed as a UML diagram, answering conjunctive queries is coNP-hard with respect to the size of the data in the accessed source.

Actually, as we will see in Section 6, one culprit of such a high complexity is the ability of expressing covering assertions, which induces reasoning by cases. A further cause of high complexity is the unrestricted interaction between multiplicities (actually, functionality) and subset constraints between associations [22,4]. Once we disallow covering and suitably restrict the simultaneous use of subset constraints between associations and multiplicities, not only the sources of exponential complexity disappear, but actually query answering becomes reducible to standard SQL evaluation over a relational database, as will be demonstrated in the following.

## 4 Reasoning over Ontologies

In this section, we study traditional DL reasoning services for ontologies. In particular, we consider the reasoning services described in Section 2.5, and we show that all such reasoning services are in PTIME w.r.t. combined complexity, and that instance checking and satisfiability (which make use of the ABox) are FOL-rewritable, and hence in $AC^0$ with respect to data complexity. We concentrate in this section on *DL-Lite*$_{\mathcal{A}}$ ontologies and address the addition of identification assertions in Section 5.6, after having discussed in Section 5 query answering based on reformulation. We deal first with ontology satisfiability, and then we tackle concept and role satisfiability, and logical implication. In fact, we will show that the latter reasoning services can be basically reduced to ontology satisfiability. Finally, we provide the complexity results mentioned above.

In the following, to ease the presentation, we make several simplifying assumptions that however do not affect the generality of the presented results:

1. Since the distinction between objects and values does not have an impact on the ontology reasoning services, we will deal only with ontologies that contain object constants, concepts, and roles only, and do not consider value constants, value domains, and attributes. Hence, we also rule out concepts of the form $\delta(U)$ and $\delta_F(U)$. With respect to the semantics, since we don't have to deal with values, we consider only interpretations $\mathcal{I}$ where the domain of values $\Delta_V^{\mathcal{I}}$ is empty, hence the interpretation domain $\Delta^{\mathcal{I}}$ coincides with the domain of objects $\Delta_O^{\mathcal{I}}$.
2. We will assume that the ontology does not contain qualified existential restrictions, i.e., concepts of the form $\exists Q.C$. Notice that in *DL-Lite*$_{\mathcal{A},id}$ such concepts may appear only in the right-hand side of inclusion assertions of the form $B \sqsubseteq \exists Q.C$, and only if the role $Q$ and its inverse do not appear in a functionality assertion. We can replace the inclusion assertion $B \sqsubseteq \exists Q.C$ by the inclusion assertions

$$\begin{aligned} B &\sqsubseteq \exists P_{new} \\ \exists P_{new}^- &\sqsubseteq C \\ P_{new} &\sqsubseteq Q \end{aligned}$$

where $P_{new}$ is a fresh atomic role. It is easy to see that the resulting ontology preserves all reasoning services over the original ontology. By repeated application

of the above transformation, once for each occurrence of a concept $\exists Q.C$ in the ontology obtained from the previous application[11], we obtain an ontology that does not contain qualified existential restrictions and that preserves all reasoning services over the original ontology.

3. Since inclusion assertions of the form $B \sqsubseteq \top_c$ do not have an impact on the semantics, we can simply discard them in reasoning.

Hence, in the following, we will consider the following simplified grammar for *DL-Lite*$_\mathcal{A}$ expressions:

$$\begin{array}{lcl@{\qquad\qquad}lcl} B & \longrightarrow & A \mid \exists Q & Q & \longrightarrow & P \mid P^- \\ C & \longrightarrow & B \mid \neg B & R & \longrightarrow & Q \mid \neg Q \end{array}$$

Our first goal is to show that ontology satisfiability is FOL-rewritable. To this aim, we resort to two main constructions, namely the canonical interpretation and the closure of the negative inclusions, which we present below.

We recall that assertions of the form $B_1 \sqsubseteq B_2$ or $Q_1 \sqsubseteq Q_2$ are called *positive inclusions (PIs)*, and assertions of the form $B_1 \sqsubseteq \neg B_2$ or $Q_1 \sqsubseteq \neg Q_2$ are called *negative inclusions (NIs)*. Notice that due to the simplified form of the grammar that we are adopting here, these are the only kinds of inclusion assertions that we need to consider.

### 4.1 Canonical Interpretation

The canonical interpretation of a *DL-Lite*$_\mathcal{A}$ ontology is an interpretation constructed according to the notion of *chase* [1]. In particular, we adapt here the notion of *restricted chase* adopted by Johnson and Klug in [56].

We start by defining the notion of applicable positive inclusion assertions (PIs), and then we exploit applicable PIs to construct the chase for a *DL-Lite*$_\mathcal{A}$ ontology. Finally, with the notion of chase in place, we give the definition of canonical interpretation.

In the following, for easiness of exposition, we make use of the following notation for a basic role $Q$ and two constants $a_1$ and $a_2$:

$$Q(a_1, a_2) \text{ denotes } \begin{cases} P(a_1, a_2), & \text{if } Q = P, \\ P(a_2, a_1), & \text{if } Q = P^-. \end{cases}$$

**Definition 4.1.** *Let $\mathcal{S}$ be a set of DL-Lite*$_\mathcal{A}$ *membership assertions. Then, a PI $\alpha$ is* applicable *in $\mathcal{S}$ to a membership assertion $\beta \in \mathcal{S}$ if*

- $\alpha = A_1 \sqsubseteq A_2$, $\beta = A_1(a)$, *and* $A_2(a) \notin \mathcal{S}$;
- $\alpha = A \sqsubseteq \exists Q$, $\beta = A(a)$, *and there does not exist any constant $a'$ such that* $Q(a, a') \in \mathcal{S}$;
- $\alpha = \exists Q \sqsubseteq A$, $\beta = Q(a, a')$, *and* $A(a) \notin \mathcal{S}$;
- $\alpha = \exists Q_1 \sqsubseteq \exists Q_2$, $\beta = Q_1(a_1, a_2)$, *and there does not exist any constant $a_2'$ such that* $Q_2(a_1, a_2') \in \mathcal{S}$;
- $\alpha = Q_1 \sqsubseteq Q_2$, $\beta = Q_1(a_1, a_2)$, *and* $Q_2(a_1, a_2) \notin \mathcal{S}$.

[11] Note that an ontology may contain concepts in which qualified existential restrictions are nested within each other.

Applicable PIs can be used, i.e., *applied*, in order to construct the chase of an ontology. Roughly speaking, the chase of a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a (possibly infinite) set of membership assertions, constructed step-by-step starting from the ABox $\mathcal{A}$. At each step of the construction, a PI $\alpha \in \mathcal{T}$ is applied to a membership assertion $\beta$ belonging to the current set $\mathcal{S}$ of membership assertions. Applying a PI means adding a new suitable membership assertion to $\mathcal{S}$, thus obtaining a new set $\mathcal{S}'$ in which $\alpha$ is not applicable to $\beta$ anymore. For example, if $\alpha = A_1 \sqsubseteq A_2$ is applicable in $\mathcal{S}$ to $\beta = A_1(a)$, the membership assertion to be added to $\mathcal{S}$ is $A_2(a)$, i.e., $\mathcal{S}' = \mathcal{S} \cup A_2(a)$. In some cases (i.e., $\alpha = A \sqsubseteq \exists Q$ or $\alpha = \exists Q_1 \sqsubseteq \exists Q_2$), to achieve an analogous aim, the new membership assertion has to make use of a new constant symbol that does not occur in $\mathcal{S}$.

Notice that such a construction process strongly depends on the order in which we select both the PI to be applied at each step and the membership assertion to which such a PI is applied, as well as on which constants we introduce at each step. Therefore, a number of syntactically distinct sets of membership assertions might result from this process. However, it is possible to show that the result is unique up to renaming of constants occurring in each such a set. Since we want our construction process to result in a unique chase of a certain ontology, along the lines of [56], we assume in the following to have a fixed infinite set of constants, whose symbols are ordered in lexicographic way, and we select PIs, membership assertions and constant symbols in lexicographic order. More precisely, given a ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we denote with $\Gamma_A$ the set of all constant symbols occurring in $\mathcal{A}$. Also, we assume to have an infinite set $\Gamma_N$ of constant symbols not occurring in $\mathcal{A}$, such that the set $\Gamma_C = \Gamma_A \cup \Gamma_N$ is totally ordered in lexicographic way. Then, our notion of chase is precisely given below.

**Definition 4.2.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology, let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$, let $n$ be the number of membership assertions in $\mathcal{A}$, and let $\Gamma_N$ be the set of constants defined above. Assume that the membership assertions in $\mathcal{A}$ are numbered from $1$ to $n$ following their lexicographic order, and consider the following definition of sets $S_j$ of membership assertions:*

- $\mathcal{S}_0 = \mathcal{A}$
- $\mathcal{S}_{j+1} = \mathcal{S}_j \cup \{\beta_{new}\}$*, where $\beta_{new}$ is a membership assertion numbered with $n + j + 1$ in $\mathcal{S}_{j+1}$ and obtained as follows:*

  **let** $\beta$ *be the first membership assertion in $\mathcal{S}_j$ such that there exists a PI $\alpha \in \mathcal{T}_p$ applicable in $\mathcal{S}_j$ to $\beta$*
  **let** $\alpha$ *be the lexicographically first PI applicable in $\mathcal{S}_j$ to $\beta$*
  **let** $a_{new}$ *be the constant of $\Gamma_N$ that follows lexicographically all constants in $\mathcal{S}_j$*
  **case** $\alpha$, $\beta$ **of**

| | | | | |
|---|---|---|---|---|
| ***(cr1)*** | $\alpha = A_1 \sqsubseteq A_2$ | *and* $\beta = A_1(a)$ | **then** | $\beta_{new} = A_2(a)$ |
| ***(cr2)*** | $\alpha = A \sqsubseteq \exists Q$ | *and* $\beta = A(a)$ | **then** | $\beta_{new} = Q(a, a_{new})$ |
| ***(cr3)*** | $\alpha = \exists Q \sqsubseteq A$ | *and* $\beta = Q(a, a')$ | **then** | $\beta_{new} = A(a)$ |
| ***(cr4)*** | $\alpha = \exists Q_1 \sqsubseteq \exists Q_2$ | *and* $\beta = Q_1(a, a')$ | **then** | $\beta_{new} = Q_2(a, a_{new})$ |
| ***(cr5)*** | $\alpha = Q_1 \sqsubseteq Q_2$ | *and* $\beta = Q_1(a, a')$ | **then** | $\beta_{new} = Q_2(a, a')$. |

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

$$chase(\mathcal{O}) = \bigcup_{j \in \mathbb{N}} \mathcal{S}_j.$$

In the above definition, **cr1**, **cr2**, **cr3**, **cr4**, and **cr5** indicate the five rules that are used for constructing the chase, each one corresponding to the application of a PI. Such rules are called *chase rules*, and we say that a chase rule is *applied to* a membership assertion $\beta$ if the corresponding PI is applied to $\beta$. Observe also that NIs and functionality assertions in $\mathcal{O}$ have no role in constructing $chase(\mathcal{O})$. Indeed $chase(\mathcal{O})$ depends only on the ABox $\mathcal{A}$ and the PIs in $\mathcal{T}$.

In the following, we will denote with $chase_i(\mathcal{O})$ the portion of the chase obtained after $i$ applications of the chase rules, selected according to the ordering established in Definition 4.2, i.e.,

$$chase_i(\mathcal{O}) = \bigcup_{j \in \{0,..,i\}} \mathcal{S}_j.$$

The following property shows that the notion of chase of an ontology is fair.

**Proposition 4.3.** *Let* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *be a DL-Lite*$_\mathcal{A}$ *ontology, and let* $\alpha$ *be a PI in* $\mathcal{T}$*. Then, if there is an* $i \in \mathbb{N}$ *such that* $\alpha$ *is applicable in* $chase_i(\mathcal{O})$ *to a membership assertion* $\beta \in chase_i(\mathcal{O})$*, then there is a* $j \geq i$ *such that* $chase_{j+1}(\mathcal{O}) = chase_j(\mathcal{O}) \cup \beta'$*, where* $\beta'$ *is the result of applying* $\alpha$ *to* $\beta$ *in* $chase_j(\mathcal{O})$*.*

*Proof.* Assume by contradiction that there is no $j \geq i$ such that $chase_{j+1}(\mathcal{O}) = chase_j(\mathcal{O}) \cup \beta'$. This would mean that either there are infinitely many membership assertions that precede $\beta$ in the ordering that we choose for membership assertions in $chase(\mathcal{O})$, or that there are infinitely many chase rules applied to some membership assertion that precedes $\beta$. However, none of these cases is possible. Indeed, $\beta$ is assigned with an ordering number $m$ such that exactly $m - 1$ membership assertions precede $\beta$. Furthermore, a PI can be applied at most once to a membership assertion (afterwards, the precondition is not satisfied and the PI is not applicable anymore), and also there exists only a finite number $\ell$ of PIs. Therefore, it is possible to apply a chase rule to some membership assertion at most $\ell$ times. We can thus conclude that the claim holds. □

With the notion of chase in place we can introduce the notion of canonical interpretation.

**Definition 4.4.** *The* canonical interpretation $can(\mathcal{O}) = \langle \Delta^{can(\mathcal{O})}, \cdot^{can(\mathcal{O})} \rangle$ *is the interpretation where:*

- $\Delta^{can(\mathcal{O})} = \Gamma_C$*,*
- $a^{can(\mathcal{O})} = a$*, for each constant* $a$ *occurring in* $chase(\mathcal{O})$*,*
- $A^{can(\mathcal{O})} = \{a \mid A(a) \in chase(\mathcal{O})\}$*, for each atomic concept* $A$*, and*
- $P^{can(\mathcal{O})} = \{(a_1, a_2) \mid P(a_1, a_2) \in chase(\mathcal{O})\}$*, for each atomic role* $P$*.*

*We also define* $can_i(\mathcal{O}) = \langle \Delta^{can(\mathcal{O})}, \cdot^{can_i(\mathcal{O})} \rangle$*, where* $\cdot^{can_i(\mathcal{O})}$ *is analogous to* $\cdot^{can(\mathcal{O})}$*, except that it refers to* $chase_i(\mathcal{O})$ *instead of* $chase(\mathcal{O})$*.*

According to the above definition, it is easy to see that $can(\mathcal{O})$ (resp., $can_i(\mathcal{O})$) is unique. Notice also that $can_0(\mathcal{O})$ is tightly related to the interpretation $DB(\mathcal{A})$. Indeed, while $\Delta^{DB(\mathcal{A})} \subseteq \Delta^{can(\mathcal{O})}$, we have that $\cdot^{DB(\mathcal{A})} = \cdot^{can_0(\mathcal{O})}$.

We point out that $chase(\mathcal{O})$ and $can(\mathcal{O})$ (resp., $chase_i(\mathcal{O})$) and $can_i(\mathcal{O})$) are strongly connected. In particular, we note that, whereas $chase_{i+1}(\mathcal{O})$ is obtained by adding a membership assertion to $chase_i(\mathcal{O})$, $can_{i+1}(\mathcal{O})$ can be seen as obtained from $can_i(\mathcal{O})$ by adding either an object to the extension of an atomic concept of $\mathcal{O}$, or a pair of objects to the extension of an atomic role of $\mathcal{O}$ (notice that the domain of interpretation is the same in each $can_i(\mathcal{O})$, and in particular in $can(\mathcal{O})$). By virtue of the strong connection discussed above, in the following we will often prove properties of $can(\mathcal{O})$ (resp., $can_i(\mathcal{O})$) by reasoning over the structure of $chase(\mathcal{O})$ (resp., $chase_i(\mathcal{O})$).

Now, we are ready to show a notable property that holds for $can(\mathcal{O})$.

**Lemma 4.5.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology and let $\mathcal{T}_p$ be the set of positive inclusion assertions in $\mathcal{T}$. Then, $can(\mathcal{O})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$.*

*Proof.* Since $\langle \mathcal{T}_p, \mathcal{A} \rangle$ does not contain NIs and functionality assertions, to prove the claim we only need to show that $can(\mathcal{O})$ satisfies all membership assertions in $\mathcal{A}$ and all PIs in $\mathcal{T}_p$. The fact that $can(\mathcal{O})$ satisfies all membership assertions in $\mathcal{A}$ follows from the fact that $\mathcal{A} \subseteq chase(\mathcal{O})$. Then, it remains to prove that $can(\mathcal{O}) \models \mathcal{T}_p$. Let us proceed by contradiction, considering all possible cases:

1. Assume by contradiction that a PI of the form $A_1 \sqsubseteq A_2 \in \mathcal{T}_p$, where $A_1$ and $A_2$ are atomic concepts, is not satisfied by $can(\mathcal{O})$. This means that there exists a constant $a \in \Gamma_C$ such that $A_1(a) \in chase(\mathcal{O})$ and $A_2(a) \notin chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr1**, since $A_1 \sqsubseteq A_2$ would be applicable to $A_1(a)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $A_2(a)$ in $chase(\mathcal{O})$. This contradicts the assumption.
2. Assume by contradiction that a PI of the form $A \sqsubseteq \exists Q \in \mathcal{T}_p$, where $A$ is an atomic concept and $Q$ is a basic role, is not satisfied by $can(\mathcal{O})$. This means that there exists a constant $a \in \Gamma_C$ such that $A(a) \in chase(\mathcal{O})$ and there does not exist a constant $a_1 \in \Gamma_C$ such that $Q(a, a_1) \in chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr2**, since $A \sqsubseteq \exists Q$ would be applicable to $A(a)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $Q(a, a_2)$ in $chase(\mathcal{O})$, where $a_2 \in \Gamma_C$ follows lexicographically all constants occurring in $chase(\mathcal{O})$ before the execution of **cr2**. This contradicts the assumption.
3. Assume by contradiction that a PI of the form $\exists Q \sqsubseteq A \in \mathcal{T}_p$, where $Q$ is a basic role and $A$ is an atomic concept, is not satisfied by $can(\mathcal{O})$. This means that there exists a pair of constants $a, a_1 \in \Gamma_C$ such that $Q(a, a_1) \in chase(\mathcal{O})$ and $A(a) \notin chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr3**, since $\exists Q \sqsubseteq A$ would be applicable to $Q(a, a_1)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $A(a)$ in $chase(\mathcal{O})$. This contradicts the assumption.
4. Assume by contradiction that a PI of the form $\exists Q_1 \sqsubseteq \exists Q_2 \in \mathcal{T}_p$, where $Q_1$ and $Q_2$ are basic roles, is not satisfied by $can(\mathcal{O})$. This means that there exists a pair

of constants $a, a_1 \in \Gamma_C$ such that $Q_1(a, a_1) \in chase(\mathcal{O})$ and there does not exist a constant $a_2 \in \Gamma_C$ such that $Q_2(a, a_2) \in chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr4** since $\exists Q_1 \sqsubseteq \exists Q_2$ would be applicable to $Q_1(a, a_1)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $Q(a, a_3)$ in $chase(\mathcal{O})$, where $a_3 \in \Gamma_C$ follows lexicographically all constants occurring in $chase(\mathcal{O})$ before the execution of **cr4**. This contradicts the assumption.

5. Assume by contradiction that a PI of the form $Q_1 \sqsubseteq Q_2 \in \mathcal{T}_p$, where $Q_1$ and $Q_2$ are basic roles, is not satisfied by $can(\mathcal{O})$. This means that there exists a pair of constants $a, a_1 \in \Gamma_C$ such that $Q_1(a, a_1) \in chase(\mathcal{O})$ and $Q_2(a, a_1) \notin chase(\mathcal{O})$. However, such a situation would trigger the chase rule **cr5**, since $Q_1 \sqsubseteq Q_2$ would be applicable to $Q_1(a, a_1)$ in $chase(\mathcal{O})$ and Proposition 4.3 ensures that such a PI would be applied at some step in the construction of the chase, thus causing the insertion of $Q_2(a, a_1)$ in $chase(\mathcal{O})$. This contradicts the assumption. □

As a consequence of Lemma 4.5, every *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ with only positive inclusions in the TBox, i.e., such that $\mathcal{T} = \mathcal{T}_p$, is always satisfiable, since we can always construct $can(\mathcal{O})$, which is a model for $\mathcal{O}$. Now, one might ask if and how $can(\mathcal{O})$ can be exploited for checking the satisfiability of an ontology with also negative inclusions and functionality assertions.

As for functionality assertions, the following lemma shows that, to establish that they are satisfied by $can(\mathcal{O})$, we have to simply verify that the interpretation $DB(\mathcal{A})$ satisfies them (and vice-versa).

**Lemma 4.6.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology, and let $\mathcal{T}_f$ be the set of functionality assertions in $\mathcal{T}$. Then, $can(\mathcal{O})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$ if and only if $DB(\mathcal{A})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$.*

*Proof.* "⇒" We show that $DB(\mathcal{A}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$ if $can(\mathcal{O}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$. This can be easily seen by observing that $\mathcal{A} \subseteq chase(\mathcal{O})$, and therefore if a membership assertion in $\mathcal{A}$ or a functionality assertion in $\mathcal{T}_f$ is satisfied by $can(\mathcal{O})$, it is also satisfied by $DB(\mathcal{A})$ (notice in particular that $\Delta^{DB(\mathcal{A})} \subseteq \Delta^{can(\mathcal{O})}$).

"⇐" We show that $can(\mathcal{O}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$ if $DB(\mathcal{A}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$. By virtue of the correspondence between $can(\mathcal{O})$ and $chase(\mathcal{O})$, we proceed by induction on the construction of $chase(\mathcal{O})$.

Base step. We have that $chase_0(\mathcal{O}) = \mathcal{A}$, and since $DB(\mathcal{A}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$, it follows that $can_0(\mathcal{O}) \models \langle \mathcal{T}_f, \mathcal{A} \rangle$.

Inductive step. Let us assume by contradiction that for some $i \geq 0$, $can_i(\mathcal{O})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$ and $can_{i+1}(\mathcal{O})$ is not. Notice that **cr2**, **cr4**, and **cr5** are the only rules that introduce new role instances, and thus may lead to a violation of a functionality assertion in $can_{i+1}(\mathcal{O})$. However, due to the restriction on the interaction between functionality and role inclusion assertions in *DL-Lite*$_\mathcal{A}$, rule **cr5** will never be applied if $\mathcal{T}$ contains a functionality assertion for $Q_2$ or its inverse. Thus, we need to consider only the rules **cr2** and **cr4**. Let us consider first rule **cr2**, and assume that $chase_{i+1}(\mathcal{O})$ is obtained by applying **cr2** to $chase_i(\mathcal{O})$. This means that a PI of the form $A \sqsubseteq \exists Q$, where $A$ is an atomic concept and $Q$ is a basic role, is applied in $chase_i(\mathcal{O})$ to a membership assertion of the form $A(a)$, such that there does not exists $a_1 \in \Gamma_C$ such that

$Q(a, a_1) \in chase_i(\mathcal{O})$. Therefore, $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup Q(a, a_{new})$, where $a_{new} \in \Gamma_C$ follows lexicographically all constants occurring in $chase_i(\mathcal{O})$. Now, if $can_{i+1}(\mathcal{O})$ is not a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$, there must exist (at least) a functionality assertion $\alpha$ that is not satisfied by $can_{i+1}(\mathcal{O})$.

- In the case where $\alpha = (\mathsf{funct}\ Q)$, for $\alpha$ to be violated, there must exist two pairs of objects $(x, y)$ and $(x, z)$ in $Q^{can_{i+1}(\mathcal{O})}$ such that $y \neq z$. However, we have that $(a, a_{new}) \in Q^{can_{i+1}(\mathcal{O})}$ and $a \notin \exists Q^{can_i(\mathcal{O})}$, since by applicability of $A \sqsubseteq \exists Q$ in $chase_i(\mathcal{O})$ it follows that there does not exist a constant $a' \in \Gamma_C$ such that $Q(a, a') \in chase_i(\mathcal{O})$. Therefore, there exists no pair $(a, a') \in Q^{can_{i+1}(\mathcal{O})}$ such that $a' \neq a_{new}$. Hence, we would conclude that $(x, y)$ and $(x, z)$ are in $Q^{can_i(\mathcal{O})}$, which would lead to a contradiction.
- In the case where $\alpha = (\mathsf{funct}\ Q^-)$, for $\alpha$ to be violated, there must exist two pairs of objects $(y, x)$ and $(z, x)$ in $Q^{can_{i+1}(\mathcal{O})}$ such that $y \neq z$. Since $a_{new}$ is a fresh constant, not occurring in $chase_i(\mathcal{O})$, we can conclude that there exists no pair $(a', a_{new})$, with $a' \neq a$, such that $Q(a', a_{new}) \in chase_i(\mathcal{O})$, and therefore, there exists no pair $(a', a_{new}) \in Q^{can_{i+1}(\mathcal{O})}$. Hence, we would conclude that $(y, x)$ and $(z, x)$ are in $Q^{can_i(\mathcal{O})}$, which would lead to a contradiction.
- In the case in which $\alpha = (\mathsf{funct}\ Q')$, with $Q' \neq Q$ and $Q' \neq Q^-$, we would conclude that $\alpha$ is not satisfied already in $can_i(\mathcal{O})$, which would lead to a contradiction.

With an almost identical argument we can prove the inductive step also in the case in which $chase_{i+1}(\mathcal{O})$ is obtained by applying **cr4** to $chase_i(\mathcal{O})$. □

### 4.2 Closure of Negative Inclusion Assertions

Let us now consider negative inclusions. In particular, we look for a property which is analogous to Lemma 4.6 for the case of NIs. Notice that, in this case, even if $DB(\mathcal{A})$ satisfies the NIs asserted in the ontology $\mathcal{O} = \langle \mathcal{T}, A \rangle$, we have that $can(\mathcal{O})$ may not satisfy $\mathcal{O}$. For example, if $\mathcal{T}$ contains the inclusion assertions $A_1 \sqsubseteq A_2$ and $A_2 \sqsubseteq \neg A_3$, and $\mathcal{A}$ contains the membership assertions $A_1(a)$ and $A_3(a)$, it is easy to see that $DB(\mathcal{A}) \models A_2 \sqsubseteq \neg A_3$, but $can(\mathcal{O}) \not\models A_2 \sqsubseteq \neg A_3$. However, as suggested by the simple example above, we get that to find the property we are looking for, we need to properly take into account the interaction between positive and negative inclusions. To this aim we construct a special TBox by closing the NIs with respect to the PIs.

**Definition 4.7.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox. We call* NI-closure of $\mathcal{T}$*, denoted by $cln(\mathcal{T})$, the TBox defined inductively as follows:*

*(1) all functionality assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;*
*(2) all negative inclusion assertions in $\mathcal{T}$ are also in $cln(\mathcal{T})$;*
*(3) if $B_1 \sqsubseteq B_2$ is in $\mathcal{T}$ and $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ is in $cln(\mathcal{T})$, then also $B_1 \sqsubseteq \neg B_3$ is in $cln(\mathcal{T})$;*
*(4) if $Q_1 \sqsubseteq Q_2$ is in $\mathcal{T}$ and $\exists Q_2 \sqsubseteq \neg B$ or $B \sqsubseteq \neg \exists Q_2$ is in $cln(\mathcal{T})$, then also $\exists Q_1 \sqsubseteq \neg B$ is in $cln(\mathcal{T})$;*

*(5) if* $Q_1 \sqsubseteq Q_2$ *is in* $\mathcal{T}$ *and* $\exists Q_2^- \sqsubseteq \neg B$ *or* $B \sqsubseteq \neg\exists Q_2^-$ *is in* $cln(\mathcal{T})$*, then also* $\exists Q_1^- \sqsubseteq \neg B$ *is in* $cln(\mathcal{T})$*;*

*(6) if* $Q_1 \sqsubseteq Q_2$ *is in* $\mathcal{T}$ *and* $Q_2 \sqsubseteq \neg Q_3$ *or* $Q_3 \sqsubseteq \neg Q_2$ *is in* $cln(\mathcal{T})$*, then also* $Q_1 \sqsubseteq \neg Q_3$ *is in* $cln(\mathcal{T})$*.*

*(7) if one of the assertions* $\exists Q \sqsubseteq \neg\exists Q$*,* $\exists Q^- \sqsubseteq \neg\exists Q^-$*, or* $Q \sqsubseteq \neg Q$ *is in* $cln(\mathcal{T})$*, then all three such assertions are in* $cln(\mathcal{T})$*.*

The following lemma shows that $cln(\mathcal{T})$ does not imply new negative inclusions or new functionality assertions not implied by $\mathcal{T}$.

**Lemma 4.8.** *Let* $\mathcal{T}$ *be a DL-Lite*$_\mathcal{A}$ *TBox, and* $\alpha$ *a negative inclusion assertion or a functionality assertion. We have that, if* $cln(\mathcal{T}) \models \alpha$*, then* $\mathcal{T} \models \alpha$*.*

*Proof.* To prove the claim it is sufficient to observe that all assertions contained in $cln(\mathcal{T})$ are logically implied by $\mathcal{T}$. □

We are now ready to show that, provided we have computed $cln(\mathcal{T})$, the analogous of Lemma 4.6 holds also for NIs.

**Lemma 4.9.** *Let* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ *be a DL-Lite*$_\mathcal{A}$ *ontology. Then,* $can(\mathcal{O})$ *is a model of* $\mathcal{O}$ *if and only if* $DB(\mathcal{A})$ *is a model of* $\langle cln(\mathcal{T}), \mathcal{A} \rangle$*.*

*Proof.* "⇒" By construction, $DB(\mathcal{A})$ cannot contradict a membership assertion in $\mathcal{A}$. Moreover, since $can(\mathcal{O})$ is a model of $\mathcal{O}$ and, by Lemma 4.8, each assertion in $cln(\mathcal{T})$ is logically implied by $\mathcal{O}$, we have that $can(\mathcal{O})$ is a model of $cln(\mathcal{T})$. Notice that $A^{DB(\mathcal{A})} = A^{can_0(\mathcal{O})} \subseteq A^{can(\mathcal{O})}$ for every atomic concept $A$ in $\mathcal{O}$, and similarly $P^{DB(\mathcal{A})} = P^{can_0(\mathcal{O})} \subseteq P^{can(\mathcal{O})}$ for every atomic role $P$ in $\mathcal{O}$. Now, considering that the structure of NIs and of functionality assertions is such that they cannot be contradicted by restricting the extension of atomic concepts and roles, we can conclude that $DB(\mathcal{A})$ is a model of $cln(\mathcal{T})$.

"⇐" We now prove that if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$, then $can(\mathcal{O})$ is a model of $\mathcal{O}$. From Lemma 4.5 it follows that $can(\mathcal{O})$ is a model of $\langle \mathcal{T}_p, \mathcal{A} \rangle$, where $\mathcal{T}_p$ is the set of PIs in $\mathcal{T}$. Moreover, since the set $\mathcal{T}_f$ of functionality assertions in $\mathcal{O}$ is contained in $cln(\mathcal{T})$, from Lemma 4.6 it follows that $can(\mathcal{O})$ is a model of $\langle \mathcal{T}_f, \mathcal{A} \rangle$. Hence, it remains to prove that $can(\mathcal{O})$ is a model of $\langle \mathcal{T} \setminus (\mathcal{T}_p \cup \mathcal{T}_f), \mathcal{A} \rangle$. We show this by proving that $can(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}) \setminus \mathcal{T}_f, \mathcal{A} \rangle$ (notice that $\mathcal{T} \setminus \mathcal{T}_p$ is contained in $cln(\mathcal{T})$). The proof is by induction on the construction of $chase(\mathcal{O})$.

Base step. By construction, $chase_0(\mathcal{O}) = \mathcal{A}$, and therefore $A^{can_0(\mathcal{O})} = A^{DB(\mathcal{A})}$ for every atomic concept $A$ in $\mathcal{O}$, and $P^{can_0(\mathcal{O})} = P^{DB(\mathcal{A})}$ for every atomic role $P$ in $\mathcal{O}$. Hence, by the assumption that $DB(\mathcal{A}) \models \langle cln(\mathcal{T}), \mathcal{A} \rangle$, it follows that $can_0(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.

Inductive step. Let us assume by contradiction that $can_i(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$ and $can_{i+1}(\mathcal{O})$ is not, and that $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by execution of the rule **cr1**. According to **cr1**, a PI of the form $A_1 \sqsubseteq A_2$, where $A_1$ and $A_2$ are atomic concepts in $\mathcal{T}$, is applied in $chase_i(\mathcal{O})$ to a membership assertion of the form $A_1(a)$, such that $A_2(a) \notin chase_i(\mathcal{O})$. Therefore $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{A_2(a)\}$ (notice that this means that $a \in A_2^{can_{i+1}(\mathcal{O})}$). Now, if $can_{i+1}(\mathcal{O})$

is not a model of $cln(\mathcal{T})$, there must exist a NI in $cln(\mathcal{T})$ of the form $A_2 \sqsubseteq \neg A_3$ or $A_3 \sqsubseteq \neg A_2$, where $A_3$ is an atomic concept, (or $A_2 \sqsubseteq \neg\exists Q$ or $\exists Q \sqsubseteq \neg A_2$, where $Q$ is a basic role) such that $A_3(a) \in chase_i(\mathcal{O})$ (resp., there exists a constant $a'$ such that $Q(a, a') \in chase_i(\mathcal{O})$). Notice that this means that $a \in A_3^{can_i(\mathcal{O})}$ (resp., $a \in \exists Q^{can_i(\mathcal{O})}$). It is easy to see that, if such a NI exists, then also $A_1 \sqsubseteq \neg A_3$ (resp., $A_1 \sqsubseteq \neg\exists Q$) belongs to $cln(\mathcal{T})$, according to NI-closure rule 3 in Definition 4.7. Since $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{A_2(a)\}$, then $A_1 \sqsubseteq \neg A_3$ (resp., $A_1 \sqsubseteq \neg\exists Q$) is not satisfied already by $can_i(\mathcal{O})$, if $A_3 \neq A_2$. If $A_3 = A_2$, we need to consider again NI-closure rule 3, according to which, from the fact that $A_1 \sqsubseteq A_2$ in $\mathcal{T}_p$, and $A_1 \sqsubseteq \neg A_2$ in $cln(\mathcal{T})$, it follows that $A_1 \sqsubseteq \neg A_1$ is in $cln(\mathcal{T})$, and therefore $A_1(a)$ is not satisfied already by $can_i(\mathcal{O})$. In both cases, we have thus contradicted the assumption that $can_i(\mathcal{O})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A}\rangle$. With an almost identical argument we can prove the inductive step also in those cases in which $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by executing rule **cr3** or rule **cr5** (in this last case, in particular, we need to use in the proof NI-closure rules 4, 5 and 6). As for the cases in which $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by applying rule **cr2**, we proceed as follows (for rule **cr4** the proof is analogous). According to **cr2**, a PI of the form $A \sqsubseteq \exists Q$, where $A$ is an atomic concept in $\mathcal{T}$, and $Q$ is a basic role in $\mathcal{T}$, is applied in $chase_i(\mathcal{O})$ to a membership assertion $A(a)$ such that there does not exist $a_1 \in \Gamma_C$ such that $Q(a, a_1) \in chase_i(\mathcal{O})$. Therefore $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{Q(a, a_2)\}$, where $a_2$ follows lexicographically all constants appearing in $chase_i(\mathcal{O})$ (notice that this means that $a \in \exists Q^{can_{i+1}(\mathcal{O})}$). Now, if $can_{i+1}(\mathcal{O})$ is not a model of $cln(\mathcal{T})$, there must exist a NI in $cln(\mathcal{T})$ of the form $\exists Q \sqsubseteq \neg B$, where $B$ is a basic concept, or of the form $\exists Q^- \sqsubseteq \neg\exists Q^-$, or of the form $Q \sqsubseteq \neg Q$. As for the first form of NI, we can reach a contradiction as done above for the case of execution of chase rule **cr1**. As for the last two forms of NIs, according to NI-closure rule 7, we have that if (at least) one of these NIs is in $cln(\mathcal{T})$, then also $\exists Q \sqsubseteq \neg\exists Q$ is in $cln(\mathcal{T})$, and thus we can again reason on a NI of the first form to reach a contradiction. □

The following corollary is an interesting consequence of the lemma above.

**Corollary 4.10.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and $\alpha$ a negative inclusion assertion or a functionality assertion. We have that, if $\mathcal{T} \models \alpha$, then $cln(\mathcal{T}) \models \alpha$.*

*Proof.* We first consider the case in which $\alpha$ is a NI. We prove the claim by contradiction. Let us assume that $\mathcal{T} \models \alpha$ and $cln(\mathcal{T}) \not\models \alpha$. We show that from $cln(\mathcal{T}) \not\models \alpha$ one can construct a model of $\mathcal{T}$ which does not satisfy $\alpha$, thus obtaining a contradiction.

Let us assume that $\alpha = A_1 \sqsubseteq \neg A_2$, where $A_1$ and $A_2$ are atomic concepts in $\mathcal{T}$, and consider the *DL-Lite$_\mathcal{A}$* ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{A} = \{A_1(a), A_2(a)\}$. We show that $can(\mathcal{O})$ is the model we are looking for, i.e., $can(\mathcal{O}) \models \mathcal{T}$ but $can(\mathcal{O}) \not\models \alpha$. The last property follows trivially by the form of $\mathcal{A}$. Hence, in the following we concentrate on proving that $can(\mathcal{O}) \models \mathcal{T}$.

We recall that $DB(\mathcal{A})$ is such that $A_1^{DB(\mathcal{A})} = \{a\}$, $A_2^{DB(\mathcal{A})} = \{a\}$, $A^{DB(\mathcal{A})} = \emptyset$ for each atomic concept $A \in \mathcal{T}$ such that $A \neq A_1$ and $A \neq A_2$, and $P^{DB(\mathcal{A})} = \emptyset$ for each atomic role $P \in \mathcal{T}$. Therefore, the only NIs that can be violated by $DB(\mathcal{A})$ are $A_1 \sqsubseteq \neg A_2$, $A_2 \sqsubseteq \neg A_1$, $A_1 \sqsubseteq \neg A_1$, and $A_2 \sqsubseteq \neg A_2$. By assumption, we have that $cln(\mathcal{T}) \not\models$

$A_1 \sqsubseteq \neg A_2$, and therefore also $cln(\mathcal{T}) \not\models A_2 \sqsubseteq \neg A_1$. From this, it follows also that $cln(\mathcal{T}) \not\models A_1 \sqsubseteq \neg A_1$ and $cln(\mathcal{T}) \not\models A_2 \sqsubseteq \neg A_2$, since either $A_1 \sqsubseteq \neg A_1$ or $A_2 \sqsubseteq \neg A_2$ logically implies $A_1 \sqsubseteq \neg A_2$. Moreover, being $\mathcal{A} = \{A_1(a), A_2(a)\}$, $DB(\mathcal{A})$ cannot violate functionality assertions. Therefore, we can conclude that $DB(\mathcal{A}) \models cln(\mathcal{T})$ and hence $DB(\mathcal{A}) \models \langle cln(\mathcal{T}), \mathcal{A}\rangle$. Then, from Lemma 4.9 it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$.

Proceeding analogously as done above, we can easily prove the claim in those cases in which $\alpha$ is one of $A \sqsubseteq \neg\exists Q$, $\exists Q \sqsubseteq \neg A$, $\exists Q_1 \sqsubseteq \neg\exists Q_2$, or $Q_1 \sqsubseteq \neg Q_2$.

The proof for the case in which $\alpha$ is a functionality assertion of the form $(\mathsf{funct}\ Q)$ can be obtained in an analogous way, by constructing the canonical interpretation starting from an ABox with the assertions $Q(a, a_1)$ and $Q(a, a_2)$. □

### 4.3 FOL-Rewritability of Ontology Satisfiability

Before providing the main results of this subsection, we need also the following crucial property, which asserts that to establish satisfiability of an ontology, we can resort to constructing the canonical interpretation.

**Lemma 4.11.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL-Lite$_\mathcal{A}$ ontology. Then, $can(\mathcal{O})$ is a model of $\mathcal{O}$ if and only if $\mathcal{O}$ is satisfiable.*

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

"⇐" We prove this direction by showing that if $can(\mathcal{O})$ is not a model of $\mathcal{O}$, then $\mathcal{O}$ is unsatisfiable. By Lemma 4.9 ("if" direction), it follows that $DB(\mathcal{A})$ is not a model of $\langle cln(\mathcal{T}), \mathcal{A}\rangle$, and since $DB(\mathcal{A}) \models \mathcal{A}$ by construction, $DB(\mathcal{A}) \not\models cln(\mathcal{T})$. This means that there exists a NI or functionality assertion $\alpha$ such that $DB(\mathcal{A}) \not\models \alpha$ and $cln(\mathcal{T}) \models \alpha$, and hence by Lemma 4.8 $\mathcal{T} \not\models \alpha$.

Consider the case where $\alpha$ if of the form $B_1 \sqsubseteq \neg B_2$, where $B_1$ and $B_2$ are basic concepts (resp., $\alpha$ is of the form $Q_1 \sqsubseteq \neg Q_2$, where $Q_1$ and $Q_2$ are basic roles). Then, there exists $a_1 \in \Delta^{DB(\mathcal{A})}$ such that $a_1 \in B_1^{DB(\mathcal{A})}$ and $a_1 \in B_2^{DB(\mathcal{A})}$ (resp., there exist $a_1, a_2 \in \Delta^{DB(\mathcal{A})}$ such that $(a_1, a_2) \in Q_1^{DB(\mathcal{A})}$ and $(a_1, a_2) \in Q_2^{DB(\mathcal{A})}$). Let us assume by contradiction that a model $\mathcal{M} = \langle \Delta^\mathcal{M}, \cdot^\mathcal{M}\rangle$ of $\mathcal{O}$ exists. For each model $\mathcal{M}$, we can construct a homomorphism $\psi$ from $\Delta^{DB(\mathcal{A})}$ to $\Delta^\mathcal{M}$ such that $\psi(a) = a^\mathcal{M}$ for each constant $a$ occurring in $\mathcal{A}$ (notice that $\mathcal{M}$ assigns a distinct object to each such constant, since $\mathcal{M} \models \mathcal{A}$). From the fact that $\mathcal{M}$ satisfies the membership assertions in $\mathcal{A}$, it easily follows that $\psi(a_1) \in B_1^\mathcal{M}$ and $\psi(a_1) \in B_2^\mathcal{M}$ (resp., $(\psi(a_1), \psi(a_2)) \in Q_1^\mathcal{M}$ and $(\psi(a_1), \psi(a_2)) \in Q_2^\mathcal{M}$). But this makes the NI $B_1 \sqsubseteq \neg B_2$ (resp., $Q_1 \sqsubseteq \neg Q_2$) be violated also in $\mathcal{M}$, and since a model cannot violate a NI that is logically implied by $\mathcal{T}$, it contradicts the fact that $\mathcal{M}$ is a model of $\mathcal{O}$.

The proof for the case where $\alpha$ is a functionality assertion of the form $(\mathsf{funct}\ Q)$ can be obtained in an analogous way, considering that there exist $a_1, a_2, a_3 \in \Delta^{DB(\mathcal{A})}$ such that $(a_1, a_2) \in Q^{DB(\mathcal{A})}$ and $(a_1, a_3) \in Q^{DB(\mathcal{A})}$. □

Notice that, the construction of $can(\mathcal{O})$ is in general neither convenient nor possible, since $can(\mathcal{O})$ may be infinite. However, by simply combining Lemma 4.9 and Lemma 4.11, we obtain the notable result that to check satisfiability of an ontology, it

**Algorithm** Satisfiable($\mathcal{O}$)
**Input:** *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$
**Output:** $true$ if $\mathcal{O}$ is satisfiable, $false$ otherwise
**begin**
  $q_{unsat(\mathcal{T})} := \{\bot\}$;
  **for each** $\alpha \in cln(\mathcal{T})$ **do** $q_{unsat(\mathcal{T})} := q_{unsat(\mathcal{T})} \cup \{\delta(\alpha)\}$;
  **if** $q_{unsat(\mathcal{T})}^{DB(\mathcal{A})} = \emptyset$ **then return** $true$; **else return** $false$;
**end**

**Fig. 9.** The algorithm Satisfiable that checks satisfiability of a *DL-Lite*$_\mathcal{A}$ ontology

is sufficient (and necessary) to look at $DB(\mathcal{A})$ (provided we have computed $cln(\mathcal{T})$). More precisely, the next theorem shows that a contradiction on a *DL-Lite*$_\mathcal{A}$ ontology may hold only if a membership assertion in the ABox contradicts a functionality assertion or a NI implied by the closure $cln(\mathcal{T})$.

**Theorem 4.12.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology. Then, $\mathcal{O}$ is satisfiable if and only if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.*

*Proof.* "$\Rightarrow$" If $\mathcal{O}$ is satisfiable, from Lemma 4.11 ("only-if" direction), it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$, and therefore, from Lemma 4.9 ("only-if" direction), it follows that $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$.

"$\Leftarrow$" If $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$, from Lemma 4.9 ("if" direction), it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$, and therefore $\mathcal{O}$ is satisfiable. □

At this point, it is not difficult to show that verifying whether $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$ can be done by simply evaluating a suitable boolean FOL query, in fact a boolean UCQ with inequalities, over $DB(\mathcal{A})$ itself. In particular we define a translation function $\delta$ from assertions in $cln(\mathcal{T})$ to boolean CQs with inequalities, as follows:

$$\begin{aligned}
\delta((\mathsf{funct}\ P)) &= \exists x, y_1, y_2 . P(x, y_1) \wedge P(x, y_2) \wedge y_1 \neq y_2 \\
\delta((\mathsf{funct}\ P^-)) &= \exists x_1, x_2, y . P(x_1, y) \wedge P(x_2, y) \wedge x_1 \neq x_2 \\
\delta(B_1 \sqsubseteq \neg B_2) &= \exists x . \gamma_1(B_1, x) \wedge \gamma_2(B_2, x) \\
\delta(Q_1 \sqsubseteq \neg Q_2) &= \exists x, y . \rho(Q_1, x, y) \wedge \rho(Q_2, x, y)
\end{aligned}$$

where in the last two equations

$$\gamma_i(B, x) = \begin{cases} A(x), & \text{if } B = A, \\ \exists y_i . P(x, y_i), & \text{if } B = \exists P, \\ \exists y_i . P(y_i, x), & \text{if } B = \exists P^-, \end{cases} \qquad \rho(Q, x, y) = \begin{cases} P(x, y), & \text{if } Q = P, \\ P(y, x), & \text{if } Q = P^-. \end{cases}$$

The algorithm Satisfiable, shown in Figure 9, takes as input a *DL-Lite*$_\mathcal{A}$ ontology, computes $DB(\mathcal{A})$ and $cln(\mathcal{T})$, and evaluates over $DB(\mathcal{A})$ the boolean FOL query $q_{unsat(\mathcal{T})}$ obtained by taking the union of all FOL formulas returned by the application of the above function $\delta$ to every assertion in $cln(\mathcal{T})$. In the algorithm, the symbol $\bot$ indicates a predicate whose evaluation is $false$ in every interpretation. Notice that in the case in which neither functionality assertions nor negative inclusion assertions occur in $\mathcal{T}$, $q_{unsat(\mathcal{T})} = \bot$, and therefore $q_{unsat(\mathcal{T})}^{DB(\mathcal{A})} = \bot^{DB(\mathcal{A})} = \emptyset$ and Satisfiable($\mathcal{O}$) returns $true$.

| TBox $\mathcal{T}'_{fbc}$ | | | | | |
|---|---|---|---|---|---|
| *League* | $\sqsubseteq$ | $\exists$*OF* | *Match* | $\sqsubseteq$ | $\exists$*HOME* |
| $\exists$*OF* | $\sqsubseteq$ | *League* | $\exists$*HOME* | $\sqsubseteq$ | *Match* |
| $\exists$*OF*$^-$ | $\sqsubseteq$ | *Nation* | $\exists$*HOME*$^-$ | $\sqsubseteq$ | *Team* |
| *Round* | $\sqsubseteq$ | $\exists$*BELONGS-TO* | *Match* | $\sqsubseteq$ | $\exists$*HOST* |
| $\exists$*BELONGS-TO* | $\sqsubseteq$ | *Round* | $\exists$*HOST* | $\sqsubseteq$ | *Match* |
| $\exists$*BELONGS-TO*$^-$ | $\sqsubseteq$ | *League* | $\exists$*HOST*$^-$ | $\sqsubseteq$ | *Team* |
| *Match* | $\sqsubseteq$ | $\exists$*PLAYED-IN* | *Match* | $\sqsubseteq$ | $\neg$*Round* |
| $\exists$*PLAYED-IN* | $\sqsubseteq$ | *Match* | | | |
| $\exists$*PLAYED-IN*$^-$ | $\sqsubseteq$ | *Round* | (funct *OF*) | | |
| *PlayedMatch* | $\sqsubseteq$ | *Match* | (funct *BELONGS-TO*) | | |
| *ScheduledMatch* | $\sqsubseteq$ | *Match* | (funct *HOME*) | | |
| *PlayedMatch* | $\sqsubseteq$ | $\neg$*ScheduledMatch* | (funct *HOST*) | | |

| ABox $\mathcal{A}'_{fbc}$ | | |
|---|---|---|
| *League*(*it2009*) | | *PLAYED-IN*(*m7RJ*, *r7*) |
| *Round*(*r7*) | *BELONGS-TO*(*r7*, *it2009*) | *PLAYED-IN*(*m8NT*, *r8*) |
| *Round*(*r8*) | *BELONGS-TO*(*r8*, *it2009*) | *PLAYED-IN*(*m8RM*, *r8*) |
| *PlayedMatch*(*m7RJ*) | *HOME*(*m7RJ*, *roma*) | *HOST*(*m7RJ*, *juventus*) |
| *Match*(*m8NT*) | *HOME*(*m8NT*, *napoli*) | *HOST*(*m8NT*, *torino*) |
| *Match*(*m8RM*) | *HOME*(*m8RM*, *roma*) | *HOST*(*m8RM*, *milan*) |
| *Team*(*roma*) | *Team*(*napoli*) | *Team*(*juventus*) |

**Fig. 10.** The simplified *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}'_{fbc} = \langle \mathcal{T}'_{fbc}, \mathcal{A}'_{fbc} \rangle$ for the football championship example

**Lemma 4.13.** *Let $\mathcal{O}$ be a DL-Lite*$_\mathcal{A}$ *ontology. Then, the algorithm* Satisfiable($\mathcal{O}$) *terminates, and $\mathcal{O}$ is satisfiable if and only if* Satisfiable($\mathcal{O}$) $= true$.

*Proof.* Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. Since $cln(\mathcal{T})$ is a finite set of membership and functionality assertions, the algorithm terminates. By Theorem 4.12, we have that $DB(\mathcal{A})$ is a model of all assertions in $cln(\mathcal{T})$ if and only if $\mathcal{O}$ is satisfiable. The query $q_{unsat(\mathcal{T})}$ verifies whether there exists an assertion $\alpha$ that is violated in $DB(\mathcal{A})$, by expressing its negation as a FOL formula $\delta(\alpha)$ and evaluating it over $DB(\mathcal{A})$. □

As a direct consequence of Lemma 4.13, we get:

**Theorem 4.14.** *In DL-Lite*$_\mathcal{A}$*, ontology satisfiability is FOL-rewritable.*

*Example 4.15.* We refer again to Example 2.3 about the football championship domain. Let $\mathcal{O}'_{fbc} = \langle \mathcal{T}'_{fbc}, \mathcal{A}'_{fbc} \rangle$ be the *DL-Lite*$_\mathcal{A}$ ontology shown in Figure 10, which is a simplified version of the ontology $\mathcal{O}_{fbc}$ used in Example 2.3. Specifically, the TBox $\mathcal{T}'_{fbc}$ is obtained from the TBox $\mathcal{T}_{fbc}$ shown in Figure 3 by ignoring identification constraints, all assertions involving value domains or attributes, and the functionality assertion on *PLAYED-IN*. The ABox $\mathcal{A}'_{fbc}$ is obtained from the ABox $\mathcal{A}_{fbc}$ shown in Figure 4 by considering only the membership assertions involving concepts and roles (and ignoring those for attributes). To check satisfiability of $\mathcal{O}'_{fbc}$, we first compute $cln(\mathcal{T}'_{fbc})$, which, besides the functionality assertions shown in Figure 10 contains the NIs shown in Figure 11.

| | |
|---|---|
| $\mathit{PlayedMatch} \sqsubseteq \neg \mathit{ScheduledMatch}$ | |
| $\mathit{Match} \sqsubseteq \neg \mathit{Round}$ | $\exists \mathit{PLAYED\text{-}IN} \sqsubseteq \neg \mathit{Round}$ |
| $\mathit{PlayedMatch} \sqsubseteq \neg \mathit{Round}$ | $\exists \mathit{HOME} \sqsubseteq \neg \mathit{Round}$ |
| $\mathit{ScheduledMatch} \sqsubseteq \neg \mathit{Round}$ | $\exists \mathit{HOST} \sqsubseteq \neg \mathit{Round}$ |
| $\mathit{Match} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-$ | $\exists \mathit{PLAYED\text{-}IN} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-$ |
| $\mathit{PlayedMatch} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-$ | $\exists \mathit{HOME} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-$ |
| $\mathit{ScheduledMatch} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-$ | $\exists \mathit{HOST} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-$ |
| $\mathit{Match} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}$ | $\exists \mathit{PLAYED\text{-}IN} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}$ |
| $\mathit{PlayedMatch} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}$ | $\exists \mathit{HOME} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}$ |
| $\mathit{ScheduledMatch} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}$ | $\exists \mathit{HOST} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}$ |

**Fig. 11.** The negative inclusions in $cln(\mathcal{T}'_{fbc})$

We show some of the boolean queries obtained by applying the translation function $\delta$ to the NIs in Figure 11:

$$\begin{aligned}
\delta(\mathit{PlayedMatch} \sqsubseteq \neg \mathit{ScheduledMatch}) &= \exists x.\, \mathit{PlayedMatch}(x) \wedge \mathit{ScheduledMatch}(x) \\
\delta(\exists \mathit{PLAYED\text{-}IN} \sqsubseteq \neg \mathit{Round}) &= \exists x.\, (\exists y.\, \mathit{PLAYED\text{-}IN}(x, y)) \wedge \mathit{Round}(x) \\
\delta(\mathit{Match} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-) &= \exists x.\, \mathit{Match}(x) \wedge (\exists y.\, \mathit{PLAYED\text{-}IN}(y, x)) \\
\delta(\exists \mathit{HOME} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-) &= \exists x.\, (\exists y_1.\, \mathit{HOME}(x, y_1)) \wedge (\exists y_2.\, \mathit{PLAYED\text{-}IN}(y_2, x))
\end{aligned}$$

We also show one of the boolean queries obtained by applying the translation function $\delta$ to the functionality assertions in $\mathcal{T}'_{fbc}$ (the other queries are defined analogously):

$$\delta((\mathsf{funct}\ \mathit{OF})) = \exists x, y_1, y_2.\, \mathit{OF}(x, y_1) \wedge \mathit{OF}(x, y_2) \wedge y_1 \neq y_2.$$

The union of the boolean queries for all the NIs and for all the functionality assertions is $q_{unsat(\mathcal{T}'_{fbc})}$. Such a query, when evaluated over $DB(\mathcal{A}'_{fbc})$, returns $false$, thus showing that $\mathcal{O}'_{fbc}$ is satisfiable.

As a further example, consider now the TBox $\mathcal{T}''_{fbc}$ obtained from $\mathcal{T}'_{fbc}$ by introducing a new role *NEXT* and adding the role inclusion assertion $\mathit{NEXT} \sqsubseteq \mathit{PLAYED\text{-}IN}$. In this case $cln(\mathcal{T}''_{fbc})$ consists of $cln(\mathcal{T}'_{fbc})$ plus the following NIs:

| | | |
|---|---|---|
| $\exists \mathit{NEXT} \sqsubseteq \neg \mathit{Round}$ | $\exists \mathit{NEXT}^- \sqsubseteq \neg \mathit{Match}$ | $\exists \mathit{NEXT}^- \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}$ |
| $\exists \mathit{NEXT} \sqsubseteq \neg \exists \mathit{PLAYED\text{-}IN}^-$ | $\exists \mathit{NEXT}^- \sqsubseteq \neg \mathit{PlayedMatch}$ | $\exists \mathit{NEXT}^- \sqsubseteq \neg \exists \mathit{HOME}$ |
| $\exists \mathit{NEXT} \sqsubseteq \neg \exists \mathit{BELONGS\text{-}TO}$ | $\exists \mathit{NEXT}^- \sqsubseteq \neg \mathit{ScheduledMatch}$ | $\exists \mathit{NEXT}^- \sqsubseteq \neg \exists \mathit{HOST}$ |

So $q_{unsat(\mathcal{T}''_{fbc})}$ includes the disjuncts of $q_{unsat(\mathcal{T}'_{fbc})}$ plus those obtained from the above NIs. Since $q_{unsat(\mathcal{T}''_{fbc})}$, when evaluated over $DB(\mathcal{A}'_{fbc})$, returns $false$, we conclude that $\mathcal{O}''_{fbc} = \langle \mathcal{T}''_{fbc}, \mathcal{A}'_{fbc} \rangle$ is satisfiable.

If we instead add to $\mathcal{T}'_{fbc}$ the functionality assertion $(\mathsf{funct}\ \mathit{PLAYED\text{-}IN}^-)$, we obtain a TBox $\mathcal{T}'''_{fbc}$ whose NI-closure is $cln(\mathcal{T}'''_{fbc}) = cln(\mathcal{T}'_{fbc}) \cup \{(\mathsf{funct}\ \mathit{PLAYED\text{-}IN}^-)\}$. In this case, $q_{unsat(\mathcal{T}'''_{fbc})}$ includes the disjuncts of $q_{unsat(\mathcal{T}'_{fbc})}$ plus the query

$$\delta((\mathsf{funct}\ \mathit{PLAYED\text{-}IN}^-)) = \exists x_1, x_2, y.\, \mathit{PLAYED\text{-}IN}(x_1, y) \wedge \mathit{PLAYED\text{-}IN}(x_2, y) \wedge x_1 \neq x_2.$$

Now, $q^{DB(\mathcal{A}'_{fbc})}_{unsat(\mathcal{T}'''_{fbc})}$ is $true$, and hence $\mathcal{O}'''_{fbc} = \langle \mathcal{T}'''_{fbc}, \mathcal{A}'_{fbc} \rangle$ is unsatisfiable. ∎

### 4.4 Concept and Role Satisfiability and Logical Implication

We start by showing that concept and role satisfiability with respect to a TBox (or an ontology) can be reduced to ontology satisfiability.

**Theorem 4.16.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, $C$ a general concept, and $Q$ a basic role. Then the following holds:*

*(1) $C$ is satisfiable w.r.t. $\mathcal{T}$ if and only if the ontology*

$$\mathcal{O}_{\mathcal{T},C} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq C\},\ \{A_{new}(a)\}\rangle$$

*is satisfiable, where $A_{new}$ is an atomic concept not appearing in $\mathcal{T}$, and $a$ is a fresh constant.*

*(2) $Q$ is satisfiable w.r.t. $\mathcal{T}$ if and only if the ontology*

$$\mathcal{O}_{\mathcal{T},Q} = \langle \mathcal{T},\ \{Q(a_1, a_2)\}\rangle$$

*is satisfiable, where $a_1$ and $a_2$ are two fresh constants.*

*(3) $\neg Q$ is satisfiable w.r.t. $\mathcal{T}$ if and only if the ontology*

$$\mathcal{O}_{\mathcal{T},\neg Q} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q\},\ \{P_{new}(a_1, a_2)\}\rangle$$

*is satisfiable, where $P_{new}$ is an atomic role not appearing in $\mathcal{T}$, and $a_1$ and $a_2$ are two fresh constants.*

*Proof.* We observe that for roles we have distinguished Case 2 from Case 3, since we need to ensure that the ontology that we obtain in the reduction is a valid *DL-Lite*$_\mathcal{A}$ ontology, and hence does not introduce a positive role inclusion assertion on a possibly functional role. We then give only the proof for Case 1, i.e., for concepts, since Case 2 is immediate, and the proof for Case 3 is analogous to that for Case 1.

"$\Leftarrow$" If $\mathcal{O}_{\mathcal{T},C}$ is satisfiable, there exists a model $\mathcal{M}$ of $\mathcal{T}$ such that $A_{new}^{\mathcal{M}} \subseteq C^{\mathcal{M}}$ and $a^{\mathcal{M}} \in A_{new}^{\mathcal{M}}$. Hence $C^{\mathcal{M}} \neq \emptyset$, and $C$ is satisfiable w.r.t. $\mathcal{T}$.

"$\Rightarrow$" If $C$ is satisfiable w.r.t. $\mathcal{T}$, there exists a model $\mathcal{M}$ of $\mathcal{T}$ and an object $o \in \Delta^{\mathcal{M}}$ such that $o \in C^{\mathcal{M}}$. We can extend $\mathcal{M}$ by defining $a^{\mathcal{I}} = o$ and $A_{new}^{\mathcal{I}} = \{o\}$, and obtain a model of $\mathcal{O}_{\mathcal{T},C}$. □

Next, we show that both instance checking and subsumption can be reduced to ontology satisfiability. We first consider the problem of instance checking for concept expressions.

**Theorem 4.17.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL-Lite$_\mathcal{A}$ ontology, $C$ a general concept, and $a$ a constant appearing in $\mathcal{O}$. Then $\mathcal{O} \models C(a)$ if and only if the ontology*

$$\mathcal{O}_{C(a)} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq \neg C\},\ \mathcal{A} \cup \{A_{new}(a)\}\rangle$$

*is unsatisfiable, where $A_{new}$ is an atomic concept not appearing in $\mathcal{O}$.*

*Proof.* "⇒" Suppose that $\mathcal{O} \models C(a)$, but there exists a model $\mathcal{M}'$ of $\mathcal{O}_{C(a)}$. Then $\mathcal{M}' \models A_{new}(a)$ and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C$. But then $\mathcal{M}' \models \neg C(a)$. Observe that $\mathcal{M}'$ is a model of $\mathcal{O}$, hence we get a contradiction.

"⇐" Suppose that $\mathcal{O}_{C(a)}$ is unsatisfiable, but there exists a model $\mathcal{M}$ of $\mathcal{O}$ such that $\mathcal{M} \models \neg C(d)$. Then we can define an interpretation $\mathcal{M}'$ of $\mathcal{O}_{C(a)}$ that interprets all constants, concepts, and roles in $\mathcal{O}$ as before, and assigns to $A_{new}$ (which does not appear in $\mathcal{O}$) the extension $A_{new}^{\mathcal{M}'} = \{a^{\mathcal{M}}\}$. Now, $\mathcal{M}'$ is still a model of $\mathcal{O}$, and moreover we have that $\mathcal{M}' \models A_{new}(a)$ and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C$, hence $\mathcal{M}'$ is a model of $\mathcal{O}_{C(a)}$. Thus we get a contradiction. □

The analogous of the above theorem holds for the problem of instance checking for role expressions.

**Theorem 4.18.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology, $Q$ a basic role, and $a_1$ and $a_2$ two constants appearing in $\mathcal{O}$. Then*

*(1) $\mathcal{O} \models Q(a_1, a_2)$ if and only if the ontology*

$$\mathcal{O}_{Q(a_1,a_2)} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q\},\ \mathcal{A} \cup \{P_{new}(a_1, a_2)\} \rangle$$

*is unsatisfiable, where $P_{new}$ is an atomic role not appearing in $\mathcal{O}$.*

*(2) $\mathcal{O} \models \neg Q(a_1, a_2)$ if and only if the ontology*

$$\mathcal{O}_{\neg Q(a_1,a_2)} = \langle \mathcal{T},\ \mathcal{A} \cup \{Q(a_1, a_2)\} \rangle$$

*is unsatisfiable.*

*Proof.* We observe that we need again to distinguish the two cases to ensure that the ontology that we obtain in the reduction is a valid *DL-Lite*$_\mathcal{A}$ ontology. Then, the proof of Case 1 is similar to the proof of Theorem 4.17, while Case 2 is obvious. □

We now address the subsumption problem and provide different reductions of this problem to the problem of ontology satisfiability. The case of subsumption between concepts is dealt with by the following theorem, and the case of subsumption between roles, is considered in the two subsequent theorems.

**Theorem 4.19.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, and $C_1$ and $C_2$ two general concepts. Then, $\mathcal{T} \models C_1 \sqsubseteq C_2$ if and only if the ontology*

$$\mathcal{O}_{C_1 \sqsubseteq C_2} = \langle \mathcal{T} \cup \{A_{new} \sqsubseteq C_1, A_{new} \sqsubseteq \neg C_2\},\ \{A_{new}(a)\} \rangle,$$

*is unsatisfiable, where $A_{new}$ is an atomic concept not appearing in $\mathcal{T}$, and $a$ is a fresh constant.*

*Proof.* "⇒" Suppose that $\mathcal{T} \models C_1 \sqsubseteq C_2$, but there exists a model $\mathcal{M}'$ of $\mathcal{O}_{C_1 \sqsubseteq C_2}$. Then $\mathcal{M}' \models A_{new}(a)$, $\mathcal{M}' \models A_{new} \sqsubseteq C_1$, and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C_2$. But then $\mathcal{M}' \models C_1(a)$ and $\mathcal{M}' \models \neg C_2(a)$. Observe that $\mathcal{M}'$ is a model of $\mathcal{T}$, hence we get a contradiction.

"⇐" Suppose that $\mathcal{O}_{C_1 \sqsubseteq C_2}$ is unsatisfiable, but there exists a model $\mathcal{M}$ of $\mathcal{T}$ such that $o \in C_1^{\mathcal{M}}$ and $o \notin C_2^{\mathcal{M}}$ for some object $o$ in the domain of $\mathcal{M}$. Then we can define

an interpretation $\mathcal{M}'$ of $\mathcal{O}_{C_1 \sqsubseteq C_2}$ that interprets all concepts and roles in $\mathcal{T}$ as before, and assigns to $a$ the extensions $a^{\mathcal{M}'} = o$, and to $A_{new}$ (which does not appear in $\mathcal{T}$) the extension $A_{new}^{\mathcal{M}'} = \{o\}$. Now, $\mathcal{M}'$ is still a model of $\mathcal{T}$, and moreover we have that $\mathcal{M}' \models A_{new}(a)$, $\mathcal{M}' \models A_{new} \sqsubseteq C_1$, and $\mathcal{M}' \models A_{new} \sqsubseteq \neg C_2$. Hence $\mathcal{M}'$ is a model of $\mathcal{O}_{C_1 \sqsubseteq C_2}$, and we get a contradiction. □

**Theorem 4.20.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, and $Q_1$ and $Q_2$ two basic roles. Then,*

*(1) $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ if and only if the ontology*

$$\mathcal{O}_{Q_1 \sqsubseteq Q_2} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q_2\},\ \{Q_1(a_1, a_2), P_{new}(a_1, a_2)\}\rangle$$

*is unsatisfiable, where $P_{new}$ is an atomic role not appearing in $\mathcal{T}$, and $a_1$, $a_2$ are two fresh constants.*

*(2) $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ if and only if the ontology*

$$\mathcal{O}_{\neg Q_1 \sqsubseteq Q_2} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q_1, P_{new} \sqsubseteq \neg Q_2\},\ \{P_{new}(a_1, a_2)\}\rangle$$

*is unsatisfiable, where $P_{new}$ is an atomic role not appearing in $\mathcal{T}$, and $a_1$, $a_2$ are two fresh constants.*

*(3) $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ if and only if the ontology*

$$\mathcal{O}_{Q_1 \sqsubseteq \neg Q_2} = \langle \mathcal{T},\ \{Q_1(a_1, a_2), Q_2(a_1, a_2)\}\rangle$$

*is unsatisfiable, where $a_1$, $a_2$ are two fresh constants.*

*(4) $\mathcal{T} \models \neg Q_1 \sqsubseteq \neg Q_2$ if and only if the ontology*

$$\mathcal{O}_{\neg Q_1 \sqsubseteq \neg Q_2} = \langle \mathcal{T} \cup \{P_{new} \sqsubseteq \neg Q_1\},\ \{Q_2(a_1, a_2), P_{new}(a_1, a_2)\}\rangle$$

*is unsatisfiable, where $P_{new}$ is an atomic role not appearing in $\mathcal{T}$, and $a_1$, $a_2$ are two fresh constants.*

*Proof.* Let $R_i$, for $i \in \{1, 2\}$, denote either $Q_i$ or $\neg Q_i$, depending on the case we are considering. First of all, we observe that in all four cases, the ontology $\mathcal{O}_{R_1 \sqsubseteq R_2}$ constructed in the reduction is a valid *DL-Lite*$_\mathcal{A}$ ontology.

"⇒" Suppose that $\mathcal{T} \models R_1 \sqsubseteq R_2$, but there exists a model $\mathcal{M}$ of $\mathcal{O}_{R_1 \sqsubseteq R_2}$. In Case 1, since $\mathcal{M} \models Q_1(a_1, a_2)$ and $\mathcal{M} \models P_{new}(a_1, a_2)$, and since $P_{new}^{\mathcal{M}} \subseteq (\neg Q_2)^{\mathcal{M}}$, we have that $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \in Q_1^{\mathcal{M}}$ and $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \notin Q_2^{\mathcal{M}}$. Since $\mathcal{M}$ is a model of $\mathcal{T}$, we get a contradiction. In Case 2, since $\mathcal{M} \models P_{new}(a_1, a_2)$, and since $P_{new}^{\mathcal{M}} \subseteq (\neg Q_1)^{\mathcal{M}}$ and $P_{new}^{\mathcal{M}} \subseteq (\neg Q_2)^{\mathcal{M}}$, we have that $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \notin Q_1^{\mathcal{M}}$ and $(a_1^{\mathcal{M}}, a_2^{\mathcal{M}}) \notin Q_2^{\mathcal{M}}$. Since $\mathcal{M}$ is a model of $\mathcal{T}$, we get a contradiction. In Case 3, since $\mathcal{M} \models Q_1(a_1, a_2)$ and $\mathcal{M} \models Q_2(a_1, a_2)$, and since $\mathcal{M}$ is a model of $\mathcal{T}$, we get a contradiction. Case 4 is analogous to Case 1, since $\mathcal{T} \models \neg Q_1 \sqsubseteq \neg Q_2$ iff $\mathcal{T} \models Q_2 \sqsubseteq Q_1$.

"⇐" Suppose that $\mathcal{O}_{R_1 \sqsubseteq R_2}$ is unsatisfiable, but there exists a model $\mathcal{M}$ of $\mathcal{T}$ such that $(o_a, o_b) \in R_1^{\mathcal{M}}$ and $(o_a, o_b) \notin R_2^{\mathcal{M}}$ for some pair of objects in the domain of $\mathcal{M}$. We first show that we can assume w.l.o.g. that $o_a$ and $o_b$ are distinct objects. Indeed, if $o_a = o_b$, we can construct a new model $\mathcal{M}_d$ of $\mathcal{T}$ as follows:

$\Delta^{\mathcal{M}_d} = \Delta^{\mathcal{M}} \times \{1,2\}$, $A^{\mathcal{M}_d} = A^{\mathcal{M}} \times \{1,2\}$ for each atomic concept $A$, and $P^{\mathcal{M}_d} = (\{((o,1),(o',1)),((o,2),(o',2)) \mid (o,o') \in P^{\mathcal{M}}\} \cup U) \setminus V$, where

$$U = \begin{cases} \emptyset, & \text{if } (o_a,o_a) \notin P^{\mathcal{M}} \\ \{((o_a,1),(o_a,2)),((o_a,2),(o_a,1))\}, & \text{if } (o_a,o_a) \in P^{\mathcal{M}} \end{cases}$$

$$V = \begin{cases} \emptyset, & \text{if } (o_a,o_a) \notin P^{\mathcal{M}} \\ \{((o_a,1),(o_a,1)),((o_a,2),(o_a,2))\}, & \text{if } (o_a,o_a) \in P^{\mathcal{M}} \end{cases}$$

for each atomic role $P$. It is immediate to see that $\mathcal{M}_d$ is still a model of $\mathcal{T}$ containing a pair of distinct objects in $R_1^{\mathcal{M}_d}$ and not in $R_2^{\mathcal{M}_d}$.

Now, given that we can assume that $o_a \neq o_b$, we can define an interpretation $\mathcal{M}'$ of $\mathcal{O}_{R_1 \sqsubseteq R_2}$ that interprets all concepts and roles in $\mathcal{T}$ as before, and assigns to $a_1$ and $a_2$ respectively the extensions $a_1^{\mathcal{M}'} = o_a$ and $a_2^{\mathcal{M}'} = o_b$, and to $P_{new}$ (which does not appear in $\mathcal{T}$), if present in $\mathcal{O}_{R_1 \sqsubseteq R_2}$, the extension $P_{new}^{\mathcal{M}'} = \{(o_a,o_b)\}$. We have that $\mathcal{M}'$ is still a model of $\mathcal{T}$, and moreover it is easy to see that in all four cases $\mathcal{M}'$ is a model of $\mathcal{O}_{R_1 \sqsubseteq R_2}$. Thus we get a contradiction. □

We remark that in the previous theorem the answer in Case 2 will always be false, since a *DL-Lite*$_{\mathcal{A}}$ TBox cannot imply an inclusion with a negated role on the left-hand side. We have nevertheless included this case in the theorem statement to cover explicitly all possibilities.

The following theorem characterizes logical implication of a functionality assertion in *DL-Lite*$_{\mathcal{A}}$, in terms of subsumption between roles.

**Theorem 4.21.** *Let $\mathcal{T}$ be a DL-Lite$_{\mathcal{A}}$ TBox and $Q$ a basic role. Then, $\mathcal{T} \models (\mathsf{funct}\ Q)$ if and only if either $(\mathsf{funct}\ Q) \in \mathcal{T}$ or $\mathcal{T} \models Q \sqsubseteq \neg Q$.*

*Proof.* "⇐" The case in which $(\mathsf{funct}\ Q) \in \mathcal{T}$ is trivial. Instead, if $\mathcal{T} \models Q \sqsubseteq \neg Q$, then $Q^{\mathcal{I}} = \emptyset$ and hence $\mathcal{I} \models (\mathsf{funct}\ Q)$, for every model $\mathcal{I}$ of $\mathcal{T}$.

"⇒" We assume that neither $(\mathsf{funct}\ Q) \in \mathcal{T}$ nor $\mathcal{T} \models Q \sqsubseteq \neg Q$, and we construct a model of $\mathcal{T}$ that is not a model of $(\mathsf{funct}\ Q)$. First of all, notice that, since $\mathcal{T}$ does not imply $Q \sqsubseteq \neg Q$, it also does not imply $\exists Q \sqsubseteq \neg \exists Q$ and $\exists Q^- \sqsubseteq \neg \exists Q^-$. Now, consider the ABox $\mathcal{A} = \{Q(a,a_1), Q(a,a_2)\}$, where $a$, $a_1$, and $a_2$ are pairwise distinct objects, and the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. According to Theorem 4.12, $\mathcal{O}$ is satisfiable if and only if $DB(\mathcal{A})$ is a model of $\langle cln(\mathcal{T}), \mathcal{A} \rangle$. Since $Q(a,a_1)$ and $Q(a,a_2)$ are the only membership assertions in $\mathcal{A}$, the only assertions that $DB(\mathcal{A})$ can violate are (*i*) the NIs $Q \sqsubseteq \neg Q$, $\exists Q \sqsubseteq \neg \exists Q$, and $\exists Q^- \sqsubseteq \neg \exists Q^-$, and (*ii*) the functionality assertion $(\mathsf{funct}\ Q)$. But, by assumption, $\mathcal{T}$ does not imply any of such assertions, and therefore $DB(\mathcal{A})$ satisfies $cln(\mathcal{T})$. In particular, by Lemma 4.11, it follows that $can(\mathcal{O})$ is a model of $\mathcal{O}$, and therefore a model of $\mathcal{T}$. However, by construction of $\mathcal{A}$, $(\mathsf{funct}\ Q)$ is not satisfied in $DB(\mathcal{A})$, and hence also not in $can(\mathcal{O})$, which means that $can(\mathcal{O})$ is not a model of $(\mathsf{funct}\ Q)$. □

Notice that the role inclusion assertion we are using in Theorem 4.21 is of the form $\mathcal{T} \models Q \sqsubseteq \neg Q$, and thus expresses the fact that role $Q$ has an empty extension in every model of $\mathcal{T}$. Also, by Theorem 4.20, logical implication of role inclusion assertions can in turn be reduced to ontology satisfiability.

Hence, with the above results in place, in the following we can concentrate on ontology satisfiability only.

### 4.5 Computational Complexity

From the results in the previous subsections we can establish the computational complexity characterization for the classical DL reasoning problems for *DL-Lite*$_{\mathcal{A}}$.

**Theorem 4.22.** *In DL-Lite*$_{\mathcal{A}}$*, ontology satisfiability is in* AC$^0$ *in the size of the ABox (data complexity) and in* PTIME *in the size of the whole ontology (combined complexity).*

*Proof.* First, AC$^0$ data complexity follows directly from FOL-rewritability, since evaluating FOL queries/formulas over a model is in AC$^0$ in the size of the model [90,1]. As for the combined complexity, we have that $cln(\mathcal{T})$ is polynomially related to the size of the TBox $\mathcal{T}$ and hence $q_{unsat(\mathcal{T})}$ defined in algorithm Satisfiable is formed by a number of disjuncts that is polynomial in $\mathcal{T}$. Each disjunct can be evaluated separately and contains either 2 or 3 variables. Now, each disjunct can be evaluated by checking the formula under each of the $n^3$ possible assignments, where $n$ is the size of the domain of $DB(\mathcal{A})$ [90]. Finally, once an assignment is fixed, the evaluation of the formula can be done in AC$^0$. As a result, we get the PTIME bound. □

Taking into account the reductions in Theorems 4.16, 4.17, 4.18, 4.19, 4.20, and 4.21, as a consequence of Theorem 4.22, we get the following result.

**Theorem 4.23.** *In DL-Lite*$_{\mathcal{A}}$*, (concept and role) satisfiability and subsumption and logical implication of functionality assertions are in* PTIME *in the size of the TBox, and (concept and role) instance checking is in* AC$^0$ *in the size of the ABox and in* PTIME *in the size of the whole ontology.*

## 5 Query Answering over Ontologies

We study now query answering in *DL-Lite*$_{\mathcal{A},id}$. In a nutshell, our query answering method strongly separates the intensional and the extensional level of the *DL-Lite*$_{\mathcal{A},id}$ ontology: the query is first processed and reformulated based on the TBox axioms; then, the TBox is discarded and the reformulated query is evaluated over the ABox, as if the ABox was a simple relational database (cf. Section 2.6). More precisely, given a query $q$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compile the assertions of $\mathcal{T}$ (in fact, the PIs in $\mathcal{T}$) into the query itself, thus obtaining a new query $q'$. Such a new query $q'$ is then evaluated over $DB(\mathcal{A})$, thus essentially reducing query answering to query evaluation over a database instance. Since the size of $q'$ does not depend on the ABox, the data complexity of the whole query answering algorithm is the same as the data complexity of evaluating $q'$. We show that, in the case where $q$ is a CQ or a UCQ, the query $q'$ is a UCQ. Hence, the data complexity of the whole query answering algorithm is AC$^0$.

As done in the previous section for ontology reasoning, we deal first with query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies. To this end, we establish some preliminary properties of *DL-Lite*$_{\mathcal{A}}$. Then we define an algorithm for the reformulation of CQs. Based

on this algorithm we describe a technique for answering UCQs in *DL-Lite*$_{\mathcal{A}}$, prove its correctness, and analyze its computational complexity. Finally, we discuss the addition of identification assertions, and discuss query answering over *DL-Lite*$_{\mathcal{A},id}$ ontologies.

### 5.1 Preliminary Properties

First, we recall that, in the case where $\mathcal{O}$ is an unsatisfiable ontology, the answer to a UCQ $q$ is the finite set of tuples *AllTup*$(q, \mathcal{O})$. Therefore, we focus for the moment on the case where $\mathcal{O}$ is satisfiable.

We start by showing a central property of the canonical interpretation $can(\mathcal{O})$. In particular, the following lemma shows that, for every model $\mathcal{M}$ of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, there is a homomorphism (cf. Definition 2.4) from $can(\mathcal{O})$ to $\mathcal{M}$.

**Lemma 5.1.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_{\mathcal{A}}$ ontology, and let $\mathcal{M} = (\Delta^{\mathcal{M}}, \cdot^{\mathcal{M}})$ be a model of $\mathcal{O}$. Then, there is a homomorphism from $can(\mathcal{O})$ to $\mathcal{M}$.*

*Proof.* We define a function $\psi$ from $\Delta^{can(\mathcal{O})}$ to $\Delta^{\mathcal{M}}$ by induction on the construction of $chase(\mathcal{O})$, and simultaneously show that the following properties hold:

- *(i)* for each atomic concept $A$ in $\mathcal{O}$ and each object $o \in \Delta^{can(\mathcal{O})}$, if $o \in A^{can(\mathcal{O})}$ then $\psi(o) \in A^{\mathcal{M}}$, and
- *(ii)* for each atomic role $P$ in $\mathcal{O}$ and each pair of objects $o, o' \in \Delta^{can(\mathcal{O})}$, if $(o, o') \in P^{can(\mathcal{O})}$ then $(\psi(o), \psi(o')) \in P^{\mathcal{M}}$.

Hence, $\psi$ is the desired homomorphism.

Base Step. For each constant $d$ occurring in $\mathcal{A}$, we set $\psi(d^{can(\mathcal{O})}) = d^{\mathcal{M}}$ (notice that each model $\mathcal{M}$ interprets each such constant with an element in $\Delta^{\mathcal{M}}$). We remind that $chase_0(\mathcal{O}) = \mathcal{A}$, $\Delta^{can_0(\mathcal{O})} = \Delta^{can(\mathcal{O})} = \Gamma_C$, and that, for each constant $d$ occurring in $\mathcal{A}$, $d^{can_0(\mathcal{O})} = d$. Then, it is easy to see that for each object $o \in \Delta^{can_0(\mathcal{O})}$ (resp., each pair of objects $o_1, o_2 \in \Delta^{can_0(\mathcal{O})}$) such that $o \in A^{can_0(\mathcal{O})}$, where $A$ is an atomic concept in $\mathcal{O}$ (resp., $(o_1, o_2) \in P^{can(\mathcal{O})}$, where $P$ is an atomic role in $\mathcal{O}$), we have that $A(o) \in chase_0(\mathcal{O})$ (resp., $P(o_1, o_2) \in chase_0(\mathcal{O})$). Since $\mathcal{M}$ satisfies all membership assertions in $\mathcal{A}$, we also have that $\psi(o) \in A^{\mathcal{M}}$ (resp., $(\psi(o_1), \psi(o_2)) \in P^{\mathcal{M}}$).

Inductive Step. Let us assume that $chase_{i+1}(\mathcal{O})$ is obtained from $chase_i(\mathcal{O})$ by applying rule **cr2**. This means that a PI of the form $A \sqsubseteq \exists Q$, where $A$ is an atomic concept in $\mathcal{T}$, and $Q$ is a basic role in $\mathcal{T}$, is applied in $chase_i(\mathcal{O})$ to a membership assertion of the form $A(a)$, such that there does not exist a constant $a'' \in \Gamma_C$ such that $Q(a, a'') \in chase_i(\mathcal{O})$. Therefore $chase_{i+1}(\mathcal{O}) = chase_i(\mathcal{O}) \cup \{Q(a, a')\}$, where $a'$ follows lexicographically all constants appearing in $chase_i(\mathcal{O})$ (notice that this means that $(a, a') \in Q^{can_{i+1}(\mathcal{O})}$). By induction hypothesis, there exists $o_m \in \Delta^{\mathcal{M}}$ such that $\psi(a) = o_m$ and $o_m \in A^{\mathcal{M}}$. Because of the presence of the PI $A \sqsubseteq \exists Q$ in $\mathcal{T}$, and because $\mathcal{M}$ is a model of $\mathcal{O}$, there is at least one object $o'_m \in \Delta^{\mathcal{M}}$ such that $(o_m, o'_m) \in Q^{\mathcal{M}}$. Then, we set $\psi(a') = o'_m$, and we can conclude that $(\psi(a), \psi(a')) \in Q^{\mathcal{M}}$.

With a similar argument we can prove the inductive step also in those cases in which $can_{i+1}(\mathcal{O})$ is obtained from $can_i(\mathcal{O})$ by applying one of the rules **cr1**, **cr3**, **cr4**, or **cr5**. □

Based on the above property, we now prove that the canonical interpretation $can(\mathcal{O})$ of a satisfiable ontology $\mathcal{O}$ is able to represent all models of $\mathcal{O}$ with respect to UCQs.

**Theorem 5.2.** *Let $\mathcal{O}$ be a satisfiable DL-Lite$_\mathcal{A}$ ontology, and let $q$ be a UCQ over $\mathcal{O}$. Then, $cert(q, \mathcal{O}) = q^{can(\mathcal{O})}$.*

*Proof.* We first recall that $\Delta^{can(\mathcal{O})} = \Gamma_C$ and that, for each constant $a$ occurring in $\mathcal{O}$, we have that $a^{can(\mathcal{O})} = a$. Therefore, given a tuple $\boldsymbol{t}$ of constants occurring in $\mathcal{O}$, we have that $\boldsymbol{t}^{can(\mathcal{O})} = \boldsymbol{t}$. We can hence rephrase the claim as $\boldsymbol{t} \in cert(q, \mathcal{O})$ iff $\boldsymbol{t} \in q^{can(\mathcal{O})}$.

"$\Rightarrow$" Suppose that $\boldsymbol{t} \in cert(q, \mathcal{O})$. Then, since $can(\mathcal{O})$ is a model of $\mathcal{O}$, we have that $\boldsymbol{t}^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$.

"$\Leftarrow$" Suppose that $\boldsymbol{t}^{can(\mathcal{O})} \in q^{can(\mathcal{O})}$. Let $q$ be the UCQ $q = \{q_1, \ldots, q_k\}$ with $q_i$ defined as $q_i(\boldsymbol{x}_i) \leftarrow conj_i(\boldsymbol{x}_i, \boldsymbol{y}_i)$, for each $i \in \{1, \ldots, k\}$. Then, by Theorem 2.6, there exists $i \in \{1, \ldots, k\}$ such that there is a homomorphism $\mu$ from $conj_i(\boldsymbol{t}, \boldsymbol{y}_i)$ to $can(\mathcal{O})$ (cf. Definition 2.5).

Now let $\mathcal{M}$ be a model of $\mathcal{O}$. By Lemma 5.1, there is a homomorphism $\psi$ from $can(\mathcal{O})$ to $\mathcal{M}$. Since homomorphisms are closed under composition, the function obtained by composing $\mu$ and $\psi$ is a homomorphisms from $conj_i(\boldsymbol{t}, \boldsymbol{y}_i)$ to $\mathcal{M}$, and by Theorem 2.6, we have that $\boldsymbol{t}^{\mathcal{M}} \in q^{\mathcal{M}}$. Since $\mathcal{M}$ was an arbitrary model, this implies that $\boldsymbol{t} \in cert(q, \mathcal{O})$. □

The above property shows that the canonical interpretation $can(\mathcal{O})$ is a correct representative of all the models of a *DL-Lite$_\mathcal{A}$* ontology with respect to the problem of answering UCQs. In other words, for every UCQ $q$, the answers to $q$ over $\mathcal{O}$ correspond to the evaluation of $q$ in $can(\mathcal{O})$.

In fact, this property holds for all positive FOL queries, but not in general. Consider for example the *DL-Lite$_\mathcal{A}$* ontology $\mathcal{O} = \langle \emptyset, \{A_1(a)\} \rangle$, and the FOL boolean query $q = \exists x.\, A_1(x) \wedge \neg A_2(x)$. We have that $chase(\mathcal{O}) = \{A_1(a)\}$, and therefore $q$ is $true$ in $can(\mathcal{O})$, but the answer to $q$ over $\mathcal{O}$ is $false$, since there exists a model $\mathcal{M}$ of $\mathcal{O}$ such that $q$ is $false$ in $\mathcal{M}$. Assume, for instance, that $\mathcal{M}$ has the same interpretation domain as $can(\mathcal{O})$, and that $a^{\mathcal{M}} = a$, $A_1^{\mathcal{M}} = \{a\}$, and $A_2^{\mathcal{M}} = \{a\}$. It is easy to see that $\mathcal{M}$ is a model of $\mathcal{O}$ and that $q$ is $false$ in $\mathcal{M}$.

Theorem 5.2, together with the fact that the canonical interpretation depends only on the positive inclusions, and not on the negative inclusions or the functionality assertions, has an interesting consequence for *satisfiable* ontologies, namely that the certain answers to a UCQ depend only on the set of positive inclusions and the ABox, but are not affected by the negative inclusions and the functionality assertions.

**Corollary 5.3.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_\mathcal{A}$ ontology, and let $q$ be a UCQ over $\mathcal{O}$. Then, $cert(q, \mathcal{O}) = cert(q, \langle \mathcal{T}_p, \mathcal{A} \rangle)$, where $\mathcal{T}_p$ is the set of positive inclusions in $\mathcal{T}$.*

We point out that the canonical interpretation is in general infinite, consequently it cannot be effectively computed in order to solve the query answering problem in *DL-Lite$_\mathcal{A}$*.

Now, given the limited expressive power of *DL-Lite$_\mathcal{A}$* TBoxes, it might seem that, in order to answer a query over an ontology $\mathcal{O}$, we could simply build a *finite* interpretation $\mathcal{I}_\mathcal{O}$ that allows for reducing answering *every* UCQ (or even every single CQ) over $\mathcal{O}$ to evaluating the query in $\mathcal{I}_\mathcal{O}$. The following theorem shows that this is *not* the case.

**Theorem 5.4.** *There exists a DL-Lite$_\mathcal{A}$ ontology $\mathcal{O}$ for which no finite interpretation $\mathcal{I}_\mathcal{O}$ exists such that, for every CQ $q$ over $\mathcal{O}$, $cert(q, \mathcal{O}) = q^{\mathcal{I}_\mathcal{O}}$.*

*Proof.* Let $\mathcal{O}$ be the *DL-Lite*$_\mathcal{A}$ ontology whose TBox consists of the cyclic concept inclusion $\exists P^- \sqsubseteq \exists P$ and whose ABox consists of the assertion $P(a, b)$.

Let $\mathcal{I}_\mathcal{O}$ be a finite interpretation. There are two possible cases:

(1) There is no cycle on the relation $P$ in $\mathcal{I}_\mathcal{O}$, i.e., the maximum path on the relation $P^{\mathcal{I}_\mathcal{O}}$ has a finite length $n$. In this case, consider the boolean conjunctive query $q() \leftarrow P(x_0, x_1), P(x_1, x_2), \ldots, P(x_n, x_{n+1})$ that represents the existence of a path of length $n + 1$ in $P$. It is immediate to verify that the query $q$ is $\mathit{false}$ in $\mathcal{I}_\mathcal{O}$, i.e., $q^{\mathcal{I}_\mathcal{O}} = \emptyset$, while the answer to $q$ over $\mathcal{O}$ is $\mathit{true}$, i.e., $\mathit{cert}(q, \mathcal{O}) = () \neq \emptyset$. This last property can be seen easily by noticing that $q^{\mathit{can}(\mathcal{O})}$ is $\mathit{true}$.

(2) $\mathcal{I}_\mathcal{O}$ satisfies the TBox cycle, so it has a finite cycle. More precisely, let us assume that $\mathcal{I}_\mathcal{O}$ is such that $\{(o_1, o_2), (o_2, o_3), \ldots, (o_n, o_1)\} \subseteq P^{\mathcal{I}_\mathcal{O}}$. In this case, consider the boolean CQ $q() \leftarrow P(x_1, x_2), \ldots, P(x_n, x_1)$. It is immediate to verify that such a query is $\mathit{true}$ in $\mathcal{I}_\mathcal{O}$, while the answer to $q$ over $\mathcal{O}$ is $\mathit{false}$. This last property can be seen easily by noticing that $q^{\mathit{can}(\mathcal{O})}$ is $\mathit{false}$, since $\mathit{chase}(\mathcal{O})$ does not contain a set of facts $P(a_1, a_2), P(a_2, a_3), \ldots, P(a_n, a_1)$, for any $n$, and therefore in $\mathit{can}(\mathcal{O})$ there does not exist any cycle on the relation $P$.

Consequently, in both cases $\mathit{cert}(q, O) \neq q^{\mathcal{I}_\mathcal{O}}$. □

Notice that in the proof of the above result we are using neither functionality assertions nor role inclusion assertions, hence the results holds already for the fragment of *DL-Lite*$_\mathcal{A}$ called *DL-Lite*$_{\mathit{core}}$ [24]. The above property demonstrates that answering queries in *DL-Lite*$_\mathcal{A}$ (or even in *DL-Lite*$_{\mathit{core}}$) goes beyond both propositional logic and relational databases.

Finally, we prove a property that relates answering UCQs to answering CQs.

**Theorem 5.5.** *Let $\mathcal{O}$ be a DL-Lite$_\mathcal{A}$ ontology, and let $q$ be a UCQ over $\mathcal{O}$. Then,*

$$\mathit{cert}(q, \mathcal{O}) = \bigcup_{q_i \in q} \mathit{cert}(q_i, \mathcal{O}).$$

*Proof.* The proof that $\bigcup_{q_i \in q} \mathit{cert}(q_i, \mathcal{O}) \subseteq \mathit{cert}(q, \mathcal{O})$ is immediate. To prove that $\mathit{cert}(q, \mathcal{O}) \subseteq \bigcup_{q_i \in Q} \mathit{cert}(q_i, \mathcal{O})$, we distinguish two possible cases:

(1) $\mathcal{O}$ is unsatisfiable. Then, it immediately follows that $\bigcup_{q_i \in q} \mathit{cert}(q_i, \mathcal{O})$ and $\mathit{cert}(q, \mathcal{O})$ are equal and coincide with the set *AllTup*$(q, \mathcal{O})$;

(2) $\mathcal{O}$ is satisfiable. Suppose that every $q_i \in q$ is of the form $q_i(\boldsymbol{x}) \leftarrow \mathit{conj}_i(\boldsymbol{x}, \boldsymbol{y}_i)$, and consider a tuple $\boldsymbol{t} \in \mathit{cert}(q, \mathcal{O})$. Then, by Theorem 5.2, $\boldsymbol{t}^{\mathit{can}(\mathcal{O})} \in q^{\mathit{can}(\mathcal{O})}$, which implies that there exists $i \in \{1, \ldots, k\}$ such that $\boldsymbol{t}^{\mathit{can}(\mathcal{O})} \in \mathit{conj}_i(\boldsymbol{t}, \boldsymbol{y}_i)^{\mathit{can}(\mathcal{O})}$. Hence, from Theorem 5.2, it follows that $\boldsymbol{t} \in \mathit{cert}(q_i, \mathcal{O})$. □

Informally, the above property states that the set of answers to a UCQ $q$ in *DL-Lite*$_\mathcal{A}$ corresponds to the union of the answers to the various CQs in $q$.

### 5.2 Query Reformulation

Based on the properties shown above, we define now an algorithm for answering UCQs in *DL-Lite*$_\mathcal{A}$, and analyze then its computational complexity. We need some preliminary definitions.

| Atom $g$ | Positive inclusion $\alpha$ | $gr(g, \alpha)$ |
|---|---|---|
| $A(x)$ | $A_1 \sqsubseteq A$ | $A_1(x)$ |
| $A(x)$ | $\exists P \sqsubseteq A$ | $P(x, \_)$ |
| $A(x)$ | $\exists P^- \sqsubseteq A$ | $P(\_, x)$ |
| $P(x, \_)$ | $A \sqsubseteq \exists P$ | $A(x)$ |
| $P(x, \_)$ | $\exists P_1 \sqsubseteq \exists P$ | $P_1(x, \_)$ |
| $P(x, \_)$ | $\exists P_1^- \sqsubseteq \exists P$ | $P_1(\_, x)$ |
| $P(\_, x)$ | $A \sqsubseteq \exists P^-$ | $A(x)$ |
| $P(\_, x)$ | $\exists P_1 \sqsubseteq \exists P^-$ | $P_1(x, \_)$ |
| $P(\_, x)$ | $\exists P_1^- \sqsubseteq \exists P^-$ | $P_1(\_, x)$ |
| $P(x_1, x_2)$ | $P_1 \sqsubseteq P$ or $P_1^- \sqsubseteq P^-$ | $P_1(x_1, x_2)$ |
| $P(x_1, x_2)$ | $P_1 \sqsubseteq P^-$ or $P_1^- \sqsubseteq P$ | $P_1(x_2, x_1)$ |

**Fig. 12.** The result $gr(g, \alpha)$ of applying a positive inclusion $\alpha$ to an atom $g$

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

We define first when a PI is *applicable to an atom*:

- A PI $\alpha$ is applicable to an atom $A(x)$, if $\alpha$ has $A$ in its right-hand side.
- A PI $\alpha$ is applicable to an atom $P(x_1, x_2)$, if one of the following conditions holds:
  - *(i)* $x_2 = \_$ and the right-hand side of $\alpha$ is $\exists P$; or
  - *(ii)* $x_1 = \_$ and the right-hand side of $\alpha$ is $\exists P^-$; or
  - *(iii)* $\alpha$ is a role inclusion assertion and its right-hand side is either $P$ or $P^-$.

Roughly speaking, a PI $\alpha$ is applicable to an atom $g$ if the predicate of $g$ is equal to the predicate in the right-hand side of $\alpha$ and, in the case when $\alpha$ is an inclusion assertion between concepts, if $g$ has at most one bound argument that corresponds to the object that is implicitly referred to by the inclusion $\alpha$.

We indicate with $gr(g, \alpha)$ the atom obtained from the atom $g$ by applying the applicable inclusion $\alpha$. Formally:

**Definition 5.6.** *Let $\alpha$ be an PI that is applicable to the atom $g$. Then, $gr(g, \alpha)$ is the atom obtained from $g$ and $\alpha$ as defined in Figure 12.*

In Figure 13, we provide the algorithm PerfectRef, which reformulates a UCQ (considered as a set of CQs) by taking into account the PIs of a TBox $\mathcal{T}$. In the algorithm, $q'[g/g']$ denotes the CQ obtained from a CQ $q'$ by replacing the atom $g$ with a new atom $g'$. Furthermore, *anon* is a function that takes as input a CQ $q'$ and returns a new CQ obtained by replacing each occurrence of an unbound variable in $q'$ with the symbol $\_$. Finally, *reduce* is a function that takes as input a CQ $q'$ and two atoms $g_1$ and $g_2$ occurring in the body of $q'$, and returns a CQ $q''$ obtained by applying to $q'$ the *most general unifier* between $g_1$ and $g_2$. We point out that, in unifying $g_1$ and $g_2$, each occurrence of the $\_$ symbol has to be considered a different unbound variable. The most

**Algorithm** PerfectRef$(q, \mathcal{T})$
**Input:** UCQ $q$, *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$
**Output:** UCQ $pr$
$pr := q$;
**repeat**
  $pr' := pr$;
  **for each** CQ $q' \in pr'$ **do**
  (a) **for each** atom $g$ in $q'$ **do**
      **for each** PI $\alpha$ in $\mathcal{T}$ **do**
        **if** $\alpha$ is applicable to $g$
        **then** $pr := pr \cup \{\, q'[g/gr(g, \alpha)] \,\}$;
  (b) **for each** pair of atoms $g_1, g_2$ in $q'$ **do**
      **if** $g_1$ and $g_2$ unify
      **then** $pr := pr \cup \{anon(reduce(q', g_1, g_2))\}$;
**until** $pr' = pr$;
**return** $pr$;

**Fig. 13.** The algorithm PerfectRef that computes the perfect reformulation of a CQ w.r.t. a *DL-Lite*$_\mathcal{A}$ TBox

general unifier substitutes each _ symbol in $g_1$ with the corresponding argument in $g_2$, and vice-versa (obviously, if both arguments are _, the resulting argument is _).

Informally, the algorithm first reformulates the atoms of each CQ $q' \in pr'$, and produces a new query for each atom reformulation (step (a)). Roughly speaking, PIs are used as rewriting rules, applied from right to left, which allow one to compile away in the reformulation the intensional knowledge (represented by $\mathcal{T}$) that is relevant for answering $q$. At step (b), for each pair of atoms $g_1$, $g_2$ that unify and occur in the body of a query $q'$, the algorithm computes the CQ $q'' = reduce(q, g_1, g_2)$. Thanks to the unification performed by *reduce*, variables that are bound in $q'$ may become unbound in $q''$. Hence, PIs that were not applicable to atoms of $q'$, may become applicable to atoms of $q''$ (in the next executions of step (a)). Notice that the use of *anon* is necessary in order to guarantee that each unbound variable is represented by the symbol _.

We observe that the reformulation of a UCQ $q$ w.r.t. a TBox $\mathcal{T}$ computed by PerfectRef depends only on the set of PIs in $\mathcal{T}$, and that NIs and functionality assertions do not play any role in such a process. Indeed, as demonstrated below by the proof of correctness of answering (U)CQs over *DL-Lite*$_\mathcal{A}$ ontologies based on the perfect reformulation, NIs and functionality assertions have to be considered only when verifying the satisfiability of the ontology. Once the satisfiability is established, they can be ignored in the query reformulation phase.

*Example 5.7.* Consider the following *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}_u$

| | |
|---|---|
| *Professor* $\sqsubseteq \neg$*Student* | $\exists$*HAS-TUTOR*$^-$ $\sqsubseteq$ *Professor* |
| *Professor* $\sqsubseteq \exists$*TEACHES-TO* | $\exists$*TEACHES-TO*$^-$ $\sqsubseteq$ *Student* |
| *Student* $\sqsubseteq \exists$*HAS-TUTOR* | (funct *HAS-TUTOR*) |

making use of the atomic concepts *Professor* and *Student*, and of the atomic roles *TEACHES-TO* and *HAS-TUTOR*. Such a TBox states that no student is also a professor

(and vice-versa), that professors do teach to students, that students have a tutor, who is also a professor, and that everyone has at most one tutor.

Consider the CQ over $\mathcal{T}_u$

$$q(x) \leftarrow \textit{TEACHES-TO}(x, y), \textit{TEACHES-TO}(\_, y).$$

In such a query, the atoms $\textit{TEACHES-TO}(x, y)$ and $\textit{TEACHES-TO}(\_, y)$ unify, and by executing $\textit{reduce}(q, \textit{TEACHES-TO}(x, y), \textit{TEACHES-TO}(\_, y))$, we obtain the atom $\textit{TEACHES-TO}(x, y)$. The variable $y$ is unbound, and therefore the function *anon* replaces it with $\_$. Now, the PI $\textit{Professor} \sqsubseteq \exists \textit{TEACHES-TO}$ can be applied to $\textit{TEACHES-TO}(x, \_)$, whereas, before the reduction step, it could not be applied to any atom of the query. ■

The following lemma shows that the algorithm PerfectRef terminates, when applied to a UCQ and a *DL-Lite*$_\mathcal{A}$ TBox.

**Lemma 5.8.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox and $q$ a UCQ over $\mathcal{T}$. Then, the algorithm* $\mathsf{PerfectRef}(q, \mathcal{T})$ *terminates.*

*Proof.* The termination of PerfectRef, for each $q$ and $\mathcal{T}$ given as inputs, immediately follows from the following facts:

(1) The maximum number of atoms in the body of a CQ generated by the algorithm is equal to the maximum length of the CQs in the input UCQ $q$. Indeed, in each iteration, a query atom is either replaced with another one, or the number of atoms in the query is reduced; hence, the number of atoms is bounded by the number of atoms in each input CQ. The length of the query is less than or equal to $n$, where $n$ is the input query size, i.e., $n$ is proportional to the number of atoms and the number of terms occurring in the input UCQ.
(2) The set of terms that occur in the CQs generated by the algorithm is equal to the set of variables and constants occurring in $q$ plus the symbol $\_$, hence such a set has cardinality less than or equal to $n + 1$, where $n$ is the query size.
(3) As a consequence of the above point, the number of different atoms that may occur in a CQ generated by the algorithm is less than or equal to $m \cdot (n + 1)^2$, where $m$ is the number of predicate symbols (concept or role names) that occur either in the TBox or in the query.
(4) The algorithm does not drop queries that it has generated.

The above points 1 and 3 imply that the number of distinct CQs generated by the algorithm is finite, whereas point 4 implies that the algorithm does not generate a query more than once, and therefore PerfectRef terminates. More precisely, the number of distinct CQs generated by the algorithm is less than or equal to $(m \cdot (n + 1)^2)^n$, which corresponds to the maximum number of executions of the repeat-until cycle of the algorithm. □

*Example 5.9.* Consider the TBox $\mathcal{T}_u$ in Example 5.7 and the CQ

$$q(x) \leftarrow \textit{TEACHES-TO}(x, y), \textit{HAS-TUTOR}(y, \_)$$

asking for professors that teach to students that have a tutor.

Let us analyze the execution of the algorithm PerfectRef$(\{q\}, \mathcal{T}_u)$. At the first execution of step (a), the algorithm applies to the atom *HAS-TUTOR*$(y, _)$ the PI *Student* $\sqsubseteq \exists$*HAS-TUTOR* and inserts in $pr$ the new query

$$q(x) \leftarrow \textit{TEACHES-TO}(x, y), \textit{Student}(y).$$

Then, at a second execution of step (a), the algorithm applies to the atom *Student*$(y)$ the PI $\exists$*TEACHES-TO*$^-$ $\sqsubseteq$ *Student* and inserts in $pr$ the query

$$q(x) \leftarrow \textit{TEACHES-TO}(x, y), \textit{TEACHES-TO}(\_, y).$$

Since the two atoms of the second query unify, step (b) of the algorithm inserts into $pr$ the query

$$q(x) \leftarrow \textit{TEACHES-TO}(x, \_).$$

Notice that the variable $y$ is unbound in the new query, hence it has been replaced by the symbol $\_$. At a next iteration, step (a) applies *Professor* $\sqsubseteq \exists$*TEACHES-TO* to *TEACHES-TO*$(x, \_)$ and inserts into $pr$ the query

$$q(x) \leftarrow \textit{Professor}(x).$$

Then, at a further execution of step (a), it applies $\exists$*HAS-TUTOR*$^-$ $\sqsubseteq$ *Professor* to *Professor*$(x)$ and inserts into $pr$ the query

$$q(x) \leftarrow \textit{HAS-TUTOR}(\_, x).$$

The set constituted by the above five queries and the original query $q$ is then returned by the algorithm PerfectRef$(\{q\}, \mathcal{T}_u)$. ∎

*Example 5.10.* As a further example, consider now the TBox $\mathcal{T}'_u$ obtained from $\mathcal{T}_u$ in Example 5.7 by adding the role inclusion assertion *HAS-TUTOR*$^-$ $\sqsubseteq$ *TEACHES-TO*, expressing that a tutor also teaches the student s/he is tutoring. Notice that $\mathcal{T}'_u$ is a valid *DL-Lite*$_\mathcal{A}$ TBox. Consider the CQ

$$q'(x) \leftarrow \textit{Student}(x).$$

Then, the result of PerfectRef$(\{q'\}, \mathcal{T}'_u)$ is the UCQ

$$\begin{aligned} q'(x) &\leftarrow \textit{Student}(x) \\ q'(x) &\leftarrow \textit{TEACHES-TO}(\_, x) \\ q'(x) &\leftarrow \textit{HAS-TUTOR}(x, \_) \end{aligned}$$

Notice that the insertion of the last CQ in the result of the execution of the algorithm is due to an application of the role inclusion assertion. ∎

We note that the UCQ produced by PerfectRef is not necessarily minimal, i.e., it may contain pairs of CQs that are one contained into the other. Though this does not affect the worst-case computational complexity, for practical purposes this set of queries can be simplified, using well-known minimization techniques for relational queries. For example, it is possible to check ordinary containment between each pair of CQs in the produced UCQs, and remove from the result UCQ those CQs that are contained in some other CQ in the set. It is easy to see that such an optimization step will not affect completeness of the algorithm.

### 5.3 Query Evaluation

In order to compute the certain answers to a UCQ $q$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we need to evaluate the set $pr$ of CQs produced by $\mathsf{PerfectRef}(q, \mathcal{O})$ over the ABox $\mathcal{A}$ considered as a relational database.

In Figure 14, we define the algorithm Answer that, given a ontology $\mathcal{O}$ and a UCQ $q$, computes $cert(q, \mathcal{O})$. The following theorem shows that the algorithm Answer terminates, when applied to a UCQ and a *DL-Lite*$_\mathcal{A}$ TBox.

**Theorem 5.11.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology and $q$ a UCQ over $\mathcal{O}$. Then, the algorithm* $\mathsf{Answer}(q, \mathcal{O})$ *terminates.*

*Proof.* Termination of $\mathsf{Answer}(q, \mathcal{O})$ follows straightforwardly from Lemma 4.13 and Lemma 5.8, which respectively establish termination of the algorithms $\mathsf{Satisfiable}(\mathcal{O})$ and $\mathsf{PerfectRef}(q, \mathcal{T})$. □

*Example 5.12.* Let us consider again the query of Example 5.9

$$q(x) \leftarrow \mathit{TEACHES\text{-}TO}(x, y), \mathit{HAS\text{-}TUTOR}(y, \_)$$

expressed over the ontology $\mathcal{O}_u = \langle \mathcal{T}_u, \mathcal{A}_u \rangle$, where $\mathcal{T}_u$ is the TBox defined in Example 5.7, and $\mathcal{A}_u$ consists of the membership assertions

$$\mathit{Student}(\mathit{john}), \qquad \mathit{HAS\text{-}TUTOR}(\mathit{john}, \mathit{mary}), \qquad \mathit{TEACHES\text{-}TO}(\mathit{mary}, \mathit{bill}).$$

By executing $\mathsf{Answer}(\{q\}, \mathcal{O}_u)$, since $\mathcal{O}_u$ is satisfiable (see Section 4), it executes $\mathsf{PerfectRef}(\{q\}, \mathcal{T}_u)$, which returns the UCQ described in Example 5.9. Let $q_1$ be such a query, then it is easy to see that $q_1^{DB(\mathcal{A}_u)}$ is the set $\{\mathit{mary}\}$.

Let us now consider again the query

$$q'(x) \leftarrow \mathit{Student}(x)$$

expressed over the ontology $\mathcal{O}'_u = \langle \mathcal{T}'_u, \mathcal{A}'_u \rangle$, where $\mathcal{T}'_u$ is as in Example 5.10, and $\mathcal{A}'_u$ consists of the membership assertions

$$\mathit{HAS\text{-}TUTOR}(\mathit{john}, \mathit{mary}), \qquad \mathit{TEACHES\text{-}TO}(\mathit{mary}, \mathit{bill}).$$

Obviously, $\mathcal{O}'_u$ is satisfiable, and executing $\mathsf{Answer}(\{q'\}, \mathcal{O}'_u)$ results in the evaluation of the UCQs returned by $\mathsf{PerfectRef}(\{q'\}, \mathcal{T}'_u)$, and which we have described in Example 5.10, over $\mathcal{A}'_u$. This produces the answer set $\{\mathit{john}, \mathit{bill}\}$. Notice that, without considering the additional role inclusion assertion, we would have obtained only $\{\mathit{bill}\}$ as answer to the query. ■

**Algorithm** $\mathsf{Answer}(q, \mathcal{O})$
**Input:** UCQ $q$, *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$
**Output:** $cert(q, \mathcal{O})$
**if not** $\mathsf{Satisfiable}(\mathcal{O})$
**then return** $\mathit{AllTup}(q, \mathcal{O})$;
**else return** $(\mathsf{PerfectRef}(q, \mathcal{T}))^{DB(\mathcal{A})}$;

**Fig. 14.** The algorithm Answer that computes the certain answers to a UCQ over a *DL-Lite*$_\mathcal{A}$ ontology

### 5.4 Correctness

We now prove correctness of the query answering technique described above. As discussed, from Theorem 5.2 it follows that query answering can in principle be done by evaluating the query over the model $can(\mathcal{O})$. However, since $can(\mathcal{O})$ is in general infinite, we obviously need to avoid the construction of $can(\mathcal{O})$. Instead, we compile the TBox into the query, thus simulating the evaluation of the query over $can(\mathcal{O})$ by evaluating a finite reformulation of the query over the ABox considered as a database.

**Lemma 5.13.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, $q$ a UCQ over $\mathcal{T}$, and $pr$ the UCQ returned by* $\mathsf{PerfectRef}(q, \mathcal{T})$. *For every DL-Lite$_\mathcal{A}$ ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = pr^{DB(\mathcal{A})}$.*

*Proof.* We first introduce the notion of witness of a tuple of constants with respect to a CQ. For a CQ $q'(\boldsymbol{x}) = \exists \boldsymbol{y}.\, conj(\boldsymbol{x}, \boldsymbol{y})$, we denote with $conj'(\boldsymbol{x}, \boldsymbol{y})$ the set of atoms corresponding to the conjunction $conj(\boldsymbol{x}, \boldsymbol{y})$. Given a *DL-Lite$_\mathcal{A}$* knowledge base $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, a CQ $q'(\boldsymbol{x}) = \exists \boldsymbol{y}.\, conj(\boldsymbol{x}, \boldsymbol{y})$ over $\mathcal{O}$, and a tuple $\boldsymbol{t}$ of constants occurring in $\mathcal{O}$, a set of membership assertions $\mathcal{G}$ is a *witness of $\boldsymbol{t}$ w.r.t. $q'$* if there exists a substitution $\sigma$ from the variables $\boldsymbol{y}$ in $conj'(\boldsymbol{t}, \boldsymbol{y})$ to constants in $\mathcal{G}$ such that the set of atoms in $\sigma(conj'(\boldsymbol{t}, \boldsymbol{y}))$ is equal to $\mathcal{G}$. In particular, we are interested in witnesses of a tuple $\boldsymbol{t}$ w.r.t. a CQ $q'$ that are contained in $chase(\mathcal{O})$. Intuitively, each such witness corresponds to a subset of $chase(\mathcal{O})$ that is sufficient in order to have that the formula $\exists \boldsymbol{y}.\, conj(\boldsymbol{t}, \boldsymbol{y})$ evaluates to $true$ in the canonical interpretation $can(\mathcal{O})$, and therefore the tuple $\boldsymbol{t} = \boldsymbol{t}^{can(\mathcal{O})}$ belongs to $q'^{can(\mathcal{O})}$. More precisely, we have that $\boldsymbol{t} \in q'^{can(\mathcal{O})}$ iff there exists a witness $\mathcal{G}$ of $\boldsymbol{t}$ w.r.t. $q'$ such that $\mathcal{G} \subseteq chase(\mathcal{O})$. The cardinality of a witness $\mathcal{G}$, denoted by $|\mathcal{G}|$, is the number of membership assertions in $\mathcal{G}$.

Since $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, by Theorem 5.2, $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = q^{can(\mathcal{O})}$, and, by Theorem 5.5, $pr^{DB(\mathcal{A})} = \bigcup_{\overline{q} \in pr} \overline{q}^{DB(\mathcal{A})}$, where $pr$ is the UCQ returned by $\mathsf{PerfectRef}(q, \mathcal{T})$. Consequently, to prove the claim it is sufficient to show that $\bigcup_{\overline{q} \in pr} \overline{q}^{DB(\mathcal{A})} = q^{can(\mathcal{O})}$. We show both inclusions separately.

"$\subseteq$" We have to prove that $\overline{q}^{DB(\mathcal{A})} \subseteq q^{can(\mathcal{O})}$, for each CQ $\overline{q} \in pr$. We show by induction on the number of steps (a) and (b) executed by the algorithm $\mathsf{PerfectRef}$ to obtain $\overline{q}$ that $\overline{q}^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$. The claim then follows from the fact that $DB(\mathcal{A})$ is contained in $can(\mathcal{O})$ and that CQs are monotone.

Base step: trivial, since $\overline{q} \in q$.

Inductive step: Let the CQ $\overline{q} = q_{i+1}$ be obtained from $q_i$ by means of step (a) or step (b) of the algorithm $\mathsf{PerfectRef}$. We show in both cases that $q_{i+1}^{can(\mathcal{O})} \subseteq q_i^{can(\mathcal{O})}$. By the inductive hypothesis we then have that $q_i^{can(\mathcal{O})} \subseteq q^{can(\mathcal{O})}$, and the claim follows. We first consider the case in which $q_{i+1}$ is obtained from $q_i$ by applying step (a) of the algorithm. Let $\boldsymbol{t}$ be a tuple of constants occurring in $\mathcal{O}$ such that $\boldsymbol{t}^{can(\mathcal{O})} \in q_{i+1}^{can(\mathcal{O})}$. Then, it follows that there exists $\mathcal{G} \subseteq can(\mathcal{O})$ such that $\mathcal{G}$ is a witness of $\boldsymbol{t}$ w.r.t. $q_{i+1}$. Let us assume that $q_{i+1}$ is obtained from $q_i$ by applying step (a) when the positive inclusion assertion $\alpha$ of $\mathcal{T}$ is of the form $A_1 \sqsubseteq A$, i.e., $q_{i+1} = q_i[A(x)/A_1(x)]$ (the proof when $\alpha$ is of the other forms listed in Definition 5.6 is analogous). Then, either $\mathcal{G}$ is a witness of $\boldsymbol{t}$ w.r.t. $q_i$, or there exists a membership assertion in $\mathcal{G}$ to which the PI $A_1 \sqsubseteq A$ is applicable. In both cases there exists a witness of $\boldsymbol{t}$ w.r.t. $q_i$ contained in $chase(\mathcal{O})$.

Therefore, $\boldsymbol{t}^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$. We consider now the case in which $q_{i+1}$ is obtained from $q_i$ by applying step (b) of the algorithm, i.e., $q_{i+1} = \mathit{anon}(\mathit{reduce}(q_i, g_1, g_2))$, where $g_1$ and $g_2$ are two atoms belonging to $q_i$ that unify. It is easy to see that in such a case $\mathcal{G}$ is also a witness of $\boldsymbol{t}$ w.r.t. $q_i$, and therefore $\boldsymbol{t}^{can(\mathcal{O})} \in q_i^{can(\mathcal{O})}$.

"$\supseteq$" We have to show that for each tuple $\boldsymbol{t} \in q^{can(\mathcal{O})}$, there exists $\overline{q} \in pr$ such that $\boldsymbol{t} \in \overline{q}^{DB(\mathcal{A})}$. First, since $\boldsymbol{t} \in q^{can(\mathcal{O})}$, it follows that there exists a CQ $q_0$ in $q$ and a finite number $k$ such that there is a witness $\mathcal{G}_k$ of $\boldsymbol{t}$ w.r.t. $q_0$ contained in $chase_k(\mathcal{O})$. Moreover, without loss of generality, we can assume that every rule **cr1**, **cr2**, **cr3**, **cr4**, and **cr5** used in the construction of $chase(\mathcal{O})$ is necessary in order to generate such a witness $\mathcal{G}_k$: i.e., $chase_k(\mathcal{O})$ can be seen as a forest (set of trees) where: *(i)* the roots correspond to the membership assertions of $\mathcal{A}$; *(ii)* $chase_k(\mathcal{O})$ contains exactly $k$ edges, where each edge corresponds to an application of a rule; *(iii)* each leaf is either one of the roots or a membership assertion in $\mathcal{G}_k$. In the following, we say that a membership assertion $\beta$ is an *ancestor* of a membership assertion $\beta'$ in a set of membership assertions $\mathcal{S}$, if there exist $\beta_1, \ldots, \beta_n$ in $\mathcal{S}$, such that $\beta_1 = \beta$, $\beta_n = \beta'$, and each $\beta_i$ can be generated by applying a chase rule to $\beta_{i-1}$, for $i \in \{2, \ldots, n\}$. We also say that $\beta'$ is a successor of $\beta$. Furthermore, for each $i \in \{0, \ldots, k\}$, we denote with $\mathcal{G}_i$ the *pre-witness* of $\boldsymbol{t}$ w.r.t. $q$ in $chase_i(\mathcal{O})$, defined as follows:

$$\mathcal{G}_i = \bigcup_{\beta' \in \mathcal{G}_k} \{\, \beta \in chase_i(\mathcal{O}) \mid \beta \text{ is an ancestor of } \beta' \text{ in } chase_k(\mathcal{O}) \text{ and there exists no successor of } \beta \text{ in } chase_i(\mathcal{O}) \text{ that is an ancestor of } \beta' \text{ in } chase_k(\mathcal{O}) \,\}.$$

Now we prove by induction on $i$ that, starting from $\mathcal{G}_k$, we can "go back" through the rule applications and find a query $\overline{q}$ in $pr$ such that the pre-witness $\mathcal{G}_{k-i}$ of $\boldsymbol{t}$ w.r.t. $q_0$ in $chase_{k-i}(\mathcal{O})$ is also a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$. To this aim, we prove that there exists $\overline{q} \in pr$ such that $\mathcal{G}_{k-i}$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_{k-i}|$, where $|\overline{q}|$ indicates the number of atoms in the CQ $\overline{q}$. The claim then follows for $i = k$, since $chase_0(\mathcal{O}) = \mathcal{A}$.

Base step: There exists $\overline{q} \in pr$ such that $\mathcal{G}_k$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_k|$. This is an immediate consequence of the fact that $q_0 \in pr$ and that $pr$ is closed with respect to step (b) of the algorithm PerfectRef. Indeed, if $|\mathcal{G}_k| < |q_0|$ then there exist two atoms $g_1$, $g_2$ in $q_0$ and a membership assertion $\beta$ in $\mathcal{G}_k$ such that $\beta$ and $g_1$ unify and $\beta$ and $g_2$ unify, which implies that $g_1$ and $g_2$ unify. Therefore, by step (b) of the algorithm, it follows that there exists a query $q_1 \in pr$ (with $q_1 = \mathit{reduce}(q_0, g_1, g_2)$) such that $\mathcal{G}_k$ is a witness of $\boldsymbol{t}$ w.r.t. $q_1$ and $|q_1| = |q| - 1$. Now, if $|\mathcal{G}_k| < |q_1|$, we can iterate the above argument, thus we conclude that there exists $\overline{q} \in pr$ such that $\mathcal{G}_k$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_k|$.

Inductive step: suppose that there exists $\overline{q} \in pr$ such that $\mathcal{G}_{k-i+1}$ is a witness of $\boldsymbol{t}$ w.r.t. $\overline{q}$ and $|\overline{q}| = |\mathcal{G}_{k-i+1}|$. Let us assume that $chase_{k-i+1}(\mathcal{O})$ is obtained by applying **cr2** to $chase_{k-i}(\mathcal{O})$ (the proof is analogous for rules **cr1**, **cr3**, **cr4**, and **cr5**). This means that a PI of the form $A \sqsubseteq \exists P$[12], where $A$ is an atomic concept and $P$ is an atomic role, is applied in $chase_{k-i}(\mathcal{O})$ to a membership assertion of the form $A(a)$, such that there does not exists $a' \in \Gamma_C$ such that $P(a, a') \in chase_{k-i}(\mathcal{O})$. Therefore,

[12] The other execution of rule **cr2** is for the case where the PI is $A \sqsubseteq \exists P^-$, which is analogous.

$chase_{k-i+1}(\mathcal{O}) = chase_{k-i}(\mathcal{O}) \cup \{P(a, a'')\}$, where $a'' \in \Gamma_C$ follows lexicographically all constants occurring in $chase_i(\mathcal{O})$.

Since $a''$ is a new constant of $\Gamma_C$, i.e., a constant not occurring elsewhere in $\mathcal{G}_{k-i+1}$, and since $|\overline{q}| = |\mathcal{G}_{k-i+1}|$, it follows that the atom $P(x, \_)$ occurs in $\overline{q}$. Therefore, by step (a) of the algorithm, it follows that there exists a query $q_1 \in pr$ (with $q_1 = \overline{q}[P(x, \_)/A(x)]$) such that $\mathcal{G}_{k-i}$ is a witness of $\boldsymbol{t}$ w.r.t. $q_1$.

Now, there are two possible cases: either $|q_1| = |\mathcal{G}_{k-i}|$, and in this case the claim is immediate; or $|q_1| = |\mathcal{G}_{k-i}| + 1$. This last case arises if and only if the membership assertion $A(a)$ to which the rule **cr2** is applied is both in $\mathcal{G}_{k-i}$ and in $\mathcal{G}_{k-i+1}$. This implies that there exist two atoms $g_1$ and $g_2$ in $q_1$ such that $A(a)$ and $g_1$ unify and $A(a)$ and $g_2$ unify, hence $g_1$ and $g_2$ unify. Therefore, by step (b) of the algorithm (applied to $q_1$), it follows that there exists $q_2 \in pr$ (with $q_2 = \mathit{reduce}(q_1, g_1, g_2)$) such that $\mathcal{G}_{k-i}$ is a witness of $\boldsymbol{t}$ w.r.t. $q_2$ and $|q_2| = |\mathcal{G}_{k-i+1}|$, which proves the claim. □

Based on the above property, we are finally able to establish correctness of the algorithm Answer.

**Theorem 5.14.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ ontology and $q$ a UCQ. Then, $cert(q, \mathcal{O}) = \mathsf{Answer}(q, \mathcal{O})$.*

*Proof.* In the case where $\mathcal{O}$ is satisfiable, the proof follows immediately from Lemma 5.13 and Theorem 5.5. In the case where $\mathcal{O}$ is not satisfiable, it is immediate to verify that the set $\mathit{AllTup}(q, \mathcal{O})$ returned by $\mathsf{Answer}(q, \mathcal{O})$ corresponds to $cert(q, \mathcal{O})$, according to the semantics of queries given in Section 2.2. □

As an immediate corollary of the above properties, it follows that the problem of answering UCQs over satisfiable *DL-Lite*$_\mathcal{A}$ ontologies is FOL-rewritable. Moreover, it is easy to see that FOL-rewritability extends also to the case of arbitrary (both satisfiable and unsatisfiable) *DL-Lite*$_\mathcal{A}$ ontologies. Indeed, the whole query answering task can be encoded into a single UCQ, obtained by adding to the UCQ $\mathsf{PerfectRef}(q, \mathcal{T})$ a finite number of CQs encoding the fact that every tuple in $\mathit{AllTup}(q, \mathcal{O})$ is in the answer set of the query if $\mathcal{O}$ is unsatisfiable. (For details on the construction of such a query see e.g. [17], which defines a similar encoding in the context of relational database integrity constraints.) We therefore get the following theorem.

**Theorem 5.15.** *Answering UCQs in DL-Lite$_\mathcal{A}$ is FOL-rewritable.*

### 5.5 Computational Complexity

We first establish the complexity of the algorithm PerfectRef.

**Lemma 5.16.** *Let $\mathcal{T}$ be a DL-Lite$_\mathcal{A}$ TBox, and $q$ a UCQ over $\mathcal{T}$. The algorithm $\mathsf{PerfectRef}(q, \mathcal{T})$ runs in time polynomial in the size of $\mathcal{T}$.*

*Proof.* Let $n$ be the query size, and let $m$ be the number of predicate symbols (concept or role names) that occur either in the TBox or in the query. As shown in Lemma 5.8, the number of distinct CQs generated by the algorithm is less than or equal to $(m \cdot (n + 1)^2)^n$, which corresponds to the maximum number of executions of the repeat-until

cycle of the algorithm. Since $m$ is linearly bounded by the size of the TBox $\mathcal{T}$, while $n$ does not depend on the size of $\mathcal{T}$, from the above argument it follows that the algorithm $\mathsf{PerfectRef}(q, \mathcal{T})$ runs in time polynomial in the size of $\mathcal{T}$. □

Based on the above property, we are able to establish the complexity of answering UCQs in *DL-Lite*$_\mathcal{A}$.

**Theorem 5.17.** *Answering UCQs in DL-Lite*$_\mathcal{A}$ *is in* PTIME *in the size of the ontology, and in* $\mathrm{AC}^0$ *in the size of the ABox (data complexity).*

*Proof.* The proof is an immediate consequence of the correctness of the algorithm Answer, established in Theorem 5.14, and the following facts: *(i)* Lemma 5.16, which implies that the query $\mathsf{PerfectRef}(q, \mathcal{T})$ can be computed in time polynomial in the size of the TBox and constant in the size of the ABox (data complexity). *(ii)* Theorem 4.22, which states the computational complexity of checking satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies. *(iii)* The fact that the evaluation of a UCQ over a database can be computed in $\mathrm{AC}^0$ with respect to the size of the database (since UCQs are a subclass of FOL queries) [1]. □

We are also able to characterize the combined complexity (i.e., the complexity w.r.t. the size of $\mathcal{O}$ and $q$) of answering UCQs in *DL-Lite*$_\mathcal{R}$.

**Theorem 5.18.** *Answering UCQs in DL-Lite*$_\mathcal{A}$ *is NP-complete in combined complexity.*

*Proof.* To prove membership in NP, observe that a version of the algorithm PerfectRef that nondeterministically returns only one of the CQs belonging to the reformulation of the input query, runs in nondeterministic polynomial time in combined complexity, since every query returned by PerfectRef can be generated after a polynomial number of transformations of one of the input CQs (i.e., after a polynomial number of executions of steps (a) and (b) of the algorithm). This allows the corresponding nondeterministic version of the algorithm Answer to run in nondeterministic polynomial time when the input is a boolean UCQ. NP-hardness follows directly from NP-hardness of CQ evaluation over relational databases [1]. □

To summarize, the above results show a very nice computational behavior of queries in *DL-Lite*$_\mathcal{A}$: answering UCQs over ontologies expressed in such a logic is computationally no worse than standard UCQ answering (and containment) in relational databases.

### 5.6 Dealing with Identification Assertions

We address now the addition of identification assertions, and present a technique for satisfiability and query answering in *DL-Lite*$_{\mathcal{A},id}$. We start with the following result, which extends Lemma 4.11 holding for *DL-Lite*$_\mathcal{A}$ ontologies to ontologies that contain also identification assertions.

**Lemma 5.19.** *Let $\mathcal{O}$ be a DL-Lite*$_{\mathcal{A},id}$ *ontology. Then, $can(\mathcal{O})$ is a model of $\mathcal{O}$ if and only if $\mathcal{O}$ is satisfiable.*

*Proof (sketch).* "⇒" If $can(\mathcal{O})$ is a model of $\mathcal{O}$, then $\mathcal{O}$ is obviously satisfiable.

"⇐" Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable *DL-Lite*$_{\mathcal{A},id}$ ontology, and let us show that $can(\mathcal{O})$ satisfies all assertions in $\mathcal{O}$. By Lemma 4.11, it is sufficient to show that $can(\mathcal{O})$ satisfies all IdCs. Since roles occurring in IdCs cannot be specialized, it is easy to see that the following crucial property holds: for each basic role $Q$ and for each constant $a$ introduced in $chase(\mathcal{O})$ during a chase step, and hence present in $can(\mathcal{O})$, there is in $chase(\mathcal{O})$ at most one fact of the form $Q(a, a')$ and at most one fact of the form $Q(a', a)$, where $a'$ is some constant originally present in $\mathcal{A}$ or introduced during a previous chase step. From this property, it immediately follows that an IdC[13] can *not* be violated by a constant introduced during the chase. On the other hand, if an IdC was violated in $chase(\mathcal{O})$, and hence in $can(\mathcal{O})$, by some pair of constants of $\mathcal{A}$, then such an IdC would be violated in every model of $\mathcal{A}$, and hence $\mathcal{O}$ would be unsatisfiable, thus contradicting the hypothesis. Consequently, no IdC of $\mathcal{O}$ is violated in $can(\mathcal{O})$, thus $can(\mathcal{O})$ is a model of all IdCs, and hence a model of $\mathcal{O}$. □

The above lemma allows us to establish a fundamental "separation" property for IdCs, similar to the one for functionality assertions and negative inclusions stated in Theorem 4.12. However, instead to resorting to a notion of closure of a set of (identification) assertions, to check satisfiability of a *DL-Lite*$_{\mathcal{A},id}$ ontology we rely on the perfect reformulation of the query that expresses the violation of an identification assertion.

As a preliminary step, we associate to each IdC $\alpha$ a boolean CQ with an inequality $\delta(\alpha)$ that encodes the violation of $\alpha$ (similarly to what we have done for negative inclusions and functionality assertions). We make use of the following notation, where $B$ is a basic concept and $x$ a variable:

$$\gamma(B, x) = \begin{cases} A(x), & \text{if } B = A, \\ P(x, y_{new}), & \text{if } B = \exists P, \text{ where } y_{new} \text{ is a fresh variable,} \\ P(y_{new}, x), & \text{if } B = \exists P^-, \text{ where } y_{new} \text{ is a fresh variable.} \end{cases}$$

Then, given an IdC $\alpha = (\mathsf{id}\ B\ \pi_1, \ldots, \pi_n)$, we define the boolean CQ with inequality

$$\delta(\alpha) \ = \ \exists \boldsymbol{x}. \gamma(B, x) \wedge \gamma(B, x') \wedge x \neq x' \wedge \bigwedge_{1 \leq i \leq n} (\rho(\pi_i(x, x_i)) \wedge \rho(\pi_i(x', x_i))),$$

where $\boldsymbol{x}$ are all variables appearing in the atoms of $\delta(\alpha)$, and $\rho(\pi(x, y))$ is inductively defined on the structure of path $\pi$ as follows:

(1) If $\pi = B_1? \circ \cdots \circ B_h? \circ Q \circ B'_1? \circ \cdots \circ B'_k?$ (with $h \geq 0$, $k \geq 0$), then

$$\rho(\pi(x, y)) = \gamma(B_1, x) \wedge \cdots \wedge \gamma(B_h, x) \wedge Q(x, y) \wedge \gamma(B'_1, y) \wedge \cdots \wedge \gamma(B'_k, y).$$

(2) If $\pi = \pi_1 \circ \pi_2$, where $length(\pi_1) = 1$ and $length(\pi_2) \geq 1$, then

$$\rho(\pi(x, y)) = \rho(\pi_1(x, z)) \wedge \rho(\pi_2(z, y)),$$

where $z$ is a fresh variable symbol (i.e., a variable symbol not occurring elsewhere in the query).

[13] Recall that we consider only so-called local IdCs, which have at least one path of length 1.

**Algorithm** SatisfiableIdC($\mathcal{O}$)
**Input:** *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$
**Output:** $true$ if $\mathcal{O}$ is satisfiable, $false$ otherwise
**begin**
  **if not** Satisfiable($\langle \mathcal{T}, \mathcal{A} \rangle$)
  **then return** $false$;
  **else begin**
    $q_{\mathcal{T}_{id}} := \{\bot\}$;
    **for each** $\alpha \in \mathcal{T}_{id}$ **do** $q_{\mathcal{T}_{id}} := q_{\mathcal{T}_{id}} \cup \{\delta(\alpha)\}$;
    $q_{unsat(\mathcal{T}_{id})} :=$ PerfectRefIdC($q_{\mathcal{T}_{id}}, \mathcal{T}$);
    **if** $q_{unsat(\mathcal{T}_{id})}^{DB(\mathcal{A})} = \emptyset$ **then return** $true$; **else return** $false$;
  **end**
**end**

**Fig. 15.** The algorithm SatisfiableIdC that checks satisfiability of a *DL-Lite*$_{\mathcal{A},id}$ ontology

Intuitively, $\delta(\alpha)$ encodes the violation of $\alpha$ by asking for the existence of two distinct instances of $B$ identified, according to $\alpha$, by the same set of objects.

Consider now a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$, where we have denoted the TBox of such an ontology as the union $\mathcal{T} \cup \mathcal{T}_{id}$ of a set $\mathcal{T}$ of *DL-Lite*$_{\mathcal{A}}$ inclusion and functionality assertions and of a set $\mathcal{T}_{id}$ of IdCs. Let us assume that $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable. In order to check the satisfiability of $\mathcal{O}$ (i.e., assess the impact on satisfiability of the IdCs), we can consider the *perfect reformulation* of the query $q_{\mathcal{T}_{id}} = \bigcup_{\alpha \in \mathcal{T}_{id}} \{\delta(\alpha)\}$ encoding the violation of all IdCs in $\mathcal{T}_{id}$. However, we need to consider a variation of the reformulation algorithm PerfectRef shown in Figure 13 that takes into account the presence of inequalities in $q_{\mathcal{T}_{id}}$. Such an algorithm, denoted PerfectRefIdC, considers the inequality predicate as a new primitive role, and never "reduces" variables occurring in inequality atoms, i.e., such variables are never transformed by unification steps into non-join variables (cf. Section 5.2). Exploiting Lemma 5.19, we can prove the following result.

**Theorem 5.20.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable DL-Lite*$_{\mathcal{A}}$ *ontology, let $\mathcal{T}_{id}$ be a set of IdCs, and let $q_{\mathcal{T}_{id}} = \bigcup_{\alpha \in \mathcal{T}_{id}} \{\delta(\alpha)\}$. Then the DL-Lite*$_{\mathcal{A},id}$ *ontology $\mathcal{O}_{id} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$ is satisfiable if and only if* $(\text{PerfectRefIdC}(q_{\mathcal{T}_{id}}))^{DB(\mathcal{A})} = \emptyset$.

We present in Figure 15 the algorithm SatisfiableIdC that checks the satisfiability of a *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$. First, the algorithm uses the algorithm Satisfiable to check satisfiability of the ordinary *DL-Lite*$_{\mathcal{A}}$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ obtained from $\mathcal{O}$ by discarding all IdCs: if $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable, then also $\mathcal{O}$ is unsatisfiable. Otherwise, the algorithm first computes the union $q_{\mathcal{T}_{id}}$ of the queries $\delta(\alpha)$, for all IdCs $\alpha \in \mathcal{T}_{id}$, encoding the violation of all IdCs in $\mathcal{O}$. Then, it uses PerfectRefIdC to compute the query $q_{unsat(\mathcal{T}_{id})}$ corresponding to the *perfect reformulation* of $q_{\mathcal{T}_{id}}$ with respect to the TBox assertions in $\mathcal{T}$. Finally, the algorithm evaluates $q_{unsat(\mathcal{T}_{id})}$ over $DB(\mathcal{A})$, i.e., the ABox $\mathcal{A}$ considered as a relational database, (which can be done in $AC^0$ w.r.t. the size of $\mathcal{A}$) and checks whether such an evaluation returns the empty set (i.e., whether the boolean UCQ evaluates to $false$). If this is the case, then the algorithm returns $true$

(i.e., that $\mathcal{O}$ is satisfiable), since the ABox does not violate any IdC that is logically implied by $\mathcal{T} \cup \mathcal{T}_{id}$ (note that considering $\mathsf{PerfectRefIdC}(q, \mathcal{T})$ rather than simply $q$ is essential for this). Instead, if the evaluation of $q_{unsat(\mathcal{T}_{id})}$ over $DB(\mathcal{A})$ returns $true$, then the ABox violates some IdC, and the algorithm reports that $\mathcal{O}$ is unsatisfiable by returning $false$.

The following lemma establishes the correctness of SatisfiableIdC.

**Lemma 5.21.** *Let $\mathcal{O}$ be a DL-Lite$_{\mathcal{A},id}$ ontology. Then, the algorithm* $\mathsf{SatisfiableIdC}(\mathcal{O})$ *terminates, and $\mathcal{O}$ is satisfiable if and only if* $\mathsf{SatisfiableIdC}(\mathcal{O}) = true$.

*Proof.* Termination follows immediately from termination of PerfectRefIdC and of evaluation of a FOL query over a database. The correctness is an immediate consequence of Theorem 5.20.

Correctness of the algorithm, together with the fact that the perfect reformulation is independent of the ABox (see Section 5.2), and, according to Lemma 5.16 can be computed in PTIME in the size of the TBox, allows us to extend the complexity results of Theorem 4.22 for ontology satisfiability in *DL-Lite*$_{\mathcal{A}}$ also to *DL-Lite*$_{\mathcal{A},id}$ ontologies.

**Theorem 5.22.** *In DL-Lite$_{\mathcal{A},id}$, ontology satisfiability is FOL-rewritable, and hence in* $\mathrm{AC}^0$ *in the size of the ABox (data complexity), and in* PTIME *in the size of the whole ontology (combined complexity).*

We observe that also for checking the satisfiability of a *DL-Lite*$_{\mathcal{A}}$ ontology, specifically with respect to negative inclusion assertions, we could have adopted an approach similar to the one presented in this subsection based on query reformulation, rather than the one presented in Section 4.2 based on computing the closure $cln(\mathcal{T})$ of the negative inclusions. Specifically, one can check that the query $q_{unsat(\mathcal{T})}$ computed by Algorithm Satisfiable in Figure 9 starting from $cln(\mathcal{T})$, actually corresponds to

$$\mathsf{PerfectRef}\Big(\bigcup_{\alpha \in \mathcal{T}_n} \delta(\alpha), \mathcal{T}_p\Big) \cup \bigcup_{\alpha \in \mathcal{T}_f} \delta(\alpha),$$

where $\mathcal{T}_p$, $\mathcal{T}_n$, and $\mathcal{T}_f$ are respectively the sets of positive inclusions, negative inclusions, and functionality assertions in $\mathcal{T}$.

We now turn our attention to query answering in the presence of identification assertions. To this aim, we observe that Lemma 5.1 and Theorem 5.2 hold also for *DL-Lite*$_{\mathcal{A},id}$ ontologies, from which we can derive the analogue of Corollary 5.3, establishing separability for query answering in *DL-Lite*$_{\mathcal{A},id}$.

**Corollary 5.23.** *Let $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$ be a satisfiable DL-Lite$_{\mathcal{A},id}$ ontology, and let $q$ be a UCQ over $\mathcal{O}$. Then, $cert(q, \mathcal{O}) = cert(q, \langle \mathcal{T}_p, \mathcal{A} \rangle)$, where $\mathcal{T}_p$ is the set of positive inclusions in $\mathcal{T}$.*

Then, Lemma 5.13 does not depend on the presence of identification assertions, except for the fact that they may affect satisfiability of an ontology. Hence, for answering UCQs over a *DL-Lite*$_{\mathcal{A},id}$ ontology we can resort to the Algorithm AnswerIdC, shown in Figure 16, which is analogous to the Algorithm Answer shown in Figure 14, with

**Algorithm** $\mathsf{AnswerIdC}(q, \mathcal{O})$
**Input:** UCQ $q$, *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$
**Output:** $cert(q, \mathcal{O})$
**if not** $\mathsf{SatisfiableIdC}(\mathcal{O})$
**then return** $\mathit{AllTup}(q, \mathcal{O})$;
**else return** $(\mathsf{PerfectRef}(q, \mathcal{T}))^{DB(\mathcal{A})}$;

**Fig. 16.** The algorithm AnswerIdC that computes the certain answers to a UCQ over a *DL-Lite*$_{\mathcal{A},id}$ ontology

the only difference that now the satisfiability check is done by taking into account also identification assertions.

The following theorem establishes termination and correctness of the algorithm AnswerIdC, when applied to a UCQ and a *DL-Lite*$_{\mathcal{A},id}$ ontology.

**Theorem 5.24.** *Let $\mathcal{O} = \langle \mathcal{T} \cup \mathcal{T}_{id}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology and $q$ a UCQ. Then,* $\mathsf{AnswerIdC}(q, \mathcal{O})$ *terminates and* $cert(q, \mathcal{O}) = \mathsf{AnswerIdC}(q, \mathcal{O})$.

Also, we obtain for query answering over *DL-Lite*$_{\mathcal{A},id}$ ontologies exactly the same complexity bounds as for *DL-Lite*$_{\mathcal{A}}$ ontologies.

**Theorem 5.25.** *Answering UCQs in DL-Lite*$_{\mathcal{A},id}$ *is in* PTIME *in the size of the ontology, in* $\mathrm{AC}^0$ *in the size of the ABox (data complexity), and NP-complete in combined complexity.*

Finally, it can be shown that adding identification assertions to *DL-Lite*$_{\mathcal{A}}$ does not increase the (data and combined) complexity of all other reasoning services, including logical implication of identification assertions.

## 6 Beyond *DL-Lite*$_{\mathcal{A},id}$

We now analyze the impact on the computational complexity of inference of extending the *DL-Lite*$_{\mathcal{A},id}$ DL as presented in Section 2. Specifically, we will concentrate on data complexity, and note that, whenever the data complexity of an inference problem goes beyond $\mathrm{AC}^0$, then the problem is not FOL-rewritable. Hence, if we want to base inference on evaluating queries over a relational database, the lack of FOL-rewritability means that a more powerful query answering engine than those available in standard relational database technology is required. An immediate consequence of this fact is that we cannot take advantage anymore of data management tools and query optimization techniques of current DBMSs (cf. also Section 7).

There are two possible ways of extending *DL-Lite*$_{\mathcal{A},id}$. The first one corresponds to a proper language extension, i.e., adding new DL constructs to *DL-Lite*$_{\mathcal{A},id}$, while the second one consists of changing/strengthening the semantics of the formalism. We analyze both types of extensions.

### 6.1 Extending the Ontology Language

Concerning the extension of the *DL-Lite*$_{\mathcal{A},id}$ language, the results in [22], which we report below, and those in [4], show that, apart from number restrictions, it is not possible to add any of the usual DL constructs to *DL-Lite*$_{\mathcal{A},id}$ while keeping the data complexity of query answering within AC$^0$. This means that *DL-Lite*$_{\mathcal{A},id}$ is essentially the most expressive DL allowing for data integration systems where query answering is FOL-rewritable.

In addition to the constructs of *DL-Lite*$_{\mathcal{A},id}$, we consider here also the following common construct in DLs [7]:

- *concept conjunction*, denoted $C_1 \sqcap C_2$, and interpreted as $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$, for an interpretation $\mathcal{I}$;
- *concept disjunction*, denoted $C_1 \sqcup C_2$, and interpreted as $C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$;
- *universal quantification on a role*, denoted $\forall P.A$, and interpreted as:

$$(\forall P.A)^{\mathcal{I}} \;=\; \{\, o \mid \forall o'.\, (o, o') \in P^{\mathcal{I}} \rightarrow o' \in A^{\mathcal{I}} \,\}.$$

We then consider variations of *DL-Lite*$_{\mathcal{A}}$ TBoxes, consisting of:

- concept inclusion assertions of the form $Cl \sqsubseteq Cr$, where the constructs that may occur in $Cl$ and $Cr$ will vary according to the language considered;
- possibly role inclusion assertions between atomic roles, i.e., of the form $P \sqsubseteq P'$;
- possibly functionality assertions of the form (funct $P$) and/or (funct $P^-$).

We first consider the case where we use qualified existential quantification in the left-hand side of inclusion assertions. This alone is sufficient to lose FOL-rewritability of instance checking. The same effect can be achieved with universal quantification on the right-hand side of inclusion assertions, or with functionality interacting with qualified existential on the right-hand side.

**Theorem 6.1.** *Instance checking (and hence ontology satisfiability and query answering) is* NLOGSPACE*-hard in data complexity for ontologies $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{A}$ is an ABox, and $\mathcal{T}$ is a TBox of one of the following forms:*

1. $Cl \;\longrightarrow\; A \mid \exists P.A$
   $Cr \;\longrightarrow\; A$
   *Assertions in* $\mathcal{T}$*:* $Cl \sqsubseteq Cr$.
2. $Cl \;\longrightarrow\; A$
   $Cr \;\longrightarrow\; A \mid \forall P.A$
   *Assertions in* $\mathcal{T}$*:* $Cl \sqsubseteq Cr$.
3. $Cl \;\longrightarrow\; A$
   $Cr \;\longrightarrow\; A \mid \exists P.A$
   *Assertions in* $\mathcal{T}$*:* $Cl \sqsubseteq Cr$, (funct $P$).

*Proof.* For Case 1, the proof is by a LOGSPACE reduction of reachability in directed graphs, which is NLOGSPACE-complete [40], to instance checking. Let $G = \langle V, E \rangle$ be a directed graph, where $V$ is a set of vertexes and $E \subseteq V \times V$ is a set of directed edges, and let $s$, $t$ be two vertexes in $V$. *Reachability* is the problem of checking whether

there are vertexes $v_0, v_1, \ldots, v_n$ in $V$ with $v_0 = s$, $v_n = t$, and $(v_{i-1}, v_i) \in E$, for $i \in \{1, \ldots, n\}$, i.e., whether there is an oriented path formed by edges in $E$ that, starting from $s$ allows one to reach $t$.

We define an ontology $\mathcal{O} = \langle \mathcal{T}_{reach}, \mathcal{A}_G \rangle$, where the TBox $\mathcal{T}_{reach}$ is constituted by a single inclusion assertion

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

and the ABox $\mathcal{A}_G$ has as constants the nodes of $G$, and is constituted by the membership assertion $A(t)$, and by one membership assertion $P(v, v')$ for each edge $(v, v') \in E$. The TBox $\mathcal{T}_{reach}$ does not depend on $G$, and it is easy to see that $\mathcal{A}_G$ can be constructed in LOGSPACE from $G$, $s$, and $t$. We show that there is an oriented path in $G$ from $s$ to $t$ if and only if $\mathcal{O} \models A(s)$.

"$\Leftarrow$" Suppose there is no path in $G$ from $s$ to $t$. We construct a model $\mathcal{I}$ of $\mathcal{O}$ such that $s^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = V$, $v^{\mathcal{I}} = v$ for each $v \in V$, $P^{\mathcal{I}} = E$, and $A^{\mathcal{I}} = \{ v \mid$ there is a path in $G$ from $v$ to $t \}$. We show that $\mathcal{I}$ is a model of $\mathcal{O}$. By construction, $\mathcal{I}$ satisfies all membership assertions $P(v, v')$ and the membership assertion $A(t)$. Consider an object $v \in (\exists P.A)^{\mathcal{I}}$. Then there is an object $v' \in A^{\mathcal{I}}$ such that $(v, v') \in P^{\mathcal{I}}$. Then, by definition of $\mathcal{I}$, there is a path in $G$ from $v'$ to $t$, and $(v, v') \in E$. Hence, there is also a path in $G$ from $v$ to $t$ and, by definition of $\mathcal{I}$, we have that $v \in A^{\mathcal{I}}$. It follows that also the inclusion assertion $\exists P.A \sqsubseteq A$ is satisfied in $\mathcal{I}$.

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

For Case 2, the proof follows from Case 1 and the observation that an assertion $\exists P.A_1 \sqsubseteq A_2$ is logically equivalent to the assertion $A_1 \sqsubseteq \forall P^-.A_2$, and that we can get rid of inverse roles by inverting the edges of the graph represented in the ABox.

For Case 3, the proof is again by a LOGSPACE reduction of reachability in directed graphs, and is based on the idea that an assertion $\exists P.A_1 \sqsubseteq A_2$ can be simulated by the assertions $A_1 \sqsubseteq \exists P^-.A_2$ and (funct $P^-$). Moreover, the graph can be encoded using only functional roles (see proof of Theorem 6.5), and we can again get rid of inverse roles by inverting edges. □

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

Notably, Case 3 of Theorem 6.1 tells us that instance checking (and therefore query answering), in the DL obtained from *DL-Lite*$_{\mathcal{A}}$ by removing the restriction on the interaction between functionality assertions and role inclusions (cf. Definition 2.1) is not in AC$^0$, and hence not FOL-rewritable. This can be seen easily by considering the encoding of inclusion assertions involving qualified existential restriction on the right-hand side in terms of inclusion assertions between roles, illustrated at the beginning of Section 4. Indeed, once we apply such an encoding, the ontology used in the reduction to prove Case 3 of Theorem 6.1 contains functional roles that are specialized. In fact,

as shown in [4] with a more involved proof, TBox reasoning in such ontologies is EXPTIME-complete (hence as hard as TBox reasoning in much more expressive DLs [7]), and instance checking and (U)CQ query answering are PTIME-complete in data complexity.

We now analyze the cases obtained from those considered in Theorem 6.1 by allowing for conjunction of concepts in the left-hand side of inclusion assertions[14].

**Theorem 6.2.** *Instance checking (and hence ontology satisfiability and query answering) is* PTIME*-hard in data complexity for ontologies $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{A}$ is an ABox, and $\mathcal{T}$ is a TBox of one of the following forms:*

1. $Cl \longrightarrow A \mid \exists P.A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A$
   *Assertions in $\mathcal{T}$:* $Cl \sqsubseteq Cr$.
2. $Cl \longrightarrow A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A \mid \forall P.A$
   *Assertions in $\mathcal{T}$:* $Cl \sqsubseteq Cr$.
3. $Cl \longrightarrow A \mid A_1 \sqcap A_2$
   $Cr \longrightarrow A \mid \exists P.A$
   *Assertions in $\mathcal{T}$:* $Cl \sqsubseteq Cr$, $(\mathsf{funct}\ P)$.

*Proof.* For Case 1, the proof is by a LOGSPACE reduction of Path System Accessibility, which is PTIME-complete [40]. An instance of *Path System Accessibility* is defined as $PS = (V, E, S, t)$, where $V$ is a set of vertexes, $E \subseteq V \times V \times V$ is an accessibility relation (we call its elements edges), $S \subseteq V$ is a set of source vertexes, and $t \in V$ is a terminal vertex. $PS$ consists in verifying whether $t$ is *accessible*, where accessibility is defined inductively as follows:

- each vertex $v \in S$ is accessible;
- if vertexes $v_1$ and $v_2$ are accessible and $(v, v_1, v_2) \in E$, then $v$ is accessible;
- nothing else is accessible.

Given $PS$, we define the ontology $\mathcal{O} = \langle \mathcal{T}_{psa}, \mathcal{A}_{PS} \rangle$, where the TBox $\mathcal{T}_{psa}$ is constituted by the inclusion assertions

$$\exists P_1.A \sqsubseteq A_1 \qquad \exists P_2.A \sqsubseteq A_2 \qquad A_1 \sqcap A_2 \sqsubseteq A \qquad \exists P_3.A \sqsubseteq A$$

and the ABox $\mathcal{A}_{PS}$ makes use of the vertexes in $V$ and the edges in $E$ as constants, as described below. Consider a vertex $v \in V$, and let $e_1, \ldots, e_k$ be all edges in $E$ that have $v$ as their first component, taken in some arbitrarily chosen order. Then the ABox $\mathcal{A}$ contains the following membership assertions:

- $P_3(v, e_1)$, and $P_3(e_i, e_{i+1})$ for $i \in \{1, \ldots, k-1\}$,
- $P_1(e_i, v_1)$ and $P_2(e_i, v_2)$, where $e_i = (v, v_1, v_2)$, for $i \in \{1, \ldots, k-1\}$.

[14] Note that allowing for conjunction of concepts in the right-hand side of inclusion assertions does not have any impact on expressivity or complexity, since an assertion $B \sqsubseteq C_1 \sqcap C_2$ is equivalent to the pair of assertions $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$.

Additionally, $\mathcal{A}_{PS}$ contains one membership assertion $A(v)$ for each vertex $v \in S$. Again, $\mathcal{T}_{psa}$ does not depend on $PS$, and it is easy to see that $\mathcal{A}_{PS}$ can be constructed in LOGSPACE from $PS$. We show that $t$ is accessible in $PS$ if and only if $\mathcal{O} \models A(t)$.

"$\Leftarrow$" Suppose that $t$ is not accessible in $PS$. We construct a model $\mathcal{I}$ of $\mathcal{O}$ such that $t^{\mathcal{I}} \notin A^{\mathcal{I}}$. Consider the interpretation $\mathcal{I}$ with $\Delta^{\mathcal{I}} = V \cup E$, and in which each constant of the ABox is interpreted as itself, $P_1^{\mathcal{I}}$, $P_2^{\mathcal{I}}$, and $P_3^{\mathcal{I}}$ consist of all pairs of nodes directly required by the ABox assertions, $A_1^{\mathcal{I}}$ consists of all edges $(v', v_1, v_2)$ such that $v_1$ is accessible in $PS$, $A_2^{\mathcal{I}}$ consists of all edges $(v', v_1, v_2)$ such that $v_2$ is accessible in $PS$, and $A^{\mathcal{I}}$ consists of all vertexes $v$ that are accessible in $PS$ union all edges $(v', v_1, v_2)$ such that both $v_1$ and $v_2$ are accessible in $PS$. It is easy to see that $\mathcal{I}$ is a model of $\mathcal{O}$, and since $t$ is not accessible in $PS$, we have that $t \notin A^{\mathcal{I}}$.

"$\Rightarrow$" Suppose that $t$ is accessible in $PS$. We prove by induction on the structure of the derivation of accessibility that if a vertex $v$ is accessible, then $\mathcal{O} \models A(v)$. Base case (direct derivation): $v \in S$, hence, by definition, $\mathcal{A}$ contains the assertion $A(v)$ and $\mathcal{O} \models A(v)$. Inductive case (indirect derivation): there exists an edge $(v, v_1, v_2) \in E$ and both $v_1$ and $v_2$ are accessible. By the inductive hypothesis, we have that $\mathcal{O} \models A(v_1)$ and $\mathcal{O} \models A(v_2)$. Let $e_1, \ldots, e_h$ be the edges in $E$ that have $v$ as their first component, up to $e_h = (v, v_1, v_2)$ and in the same order used in the construction of the ABox. Then, by $P_1(e_h, v_1)$ in the ABox and the assertions $\exists P_1.A \sqsubseteq A_1$ we have that $\mathcal{O} \models A_1(e_h)$. Similarly, we get $\mathcal{O} \models A_2(e_h)$, and hence $\mathcal{O} \models A(e_h)$. By exploiting assertions $P_3(e_i, e_{i+i})$ in the ABox, and the TBox assertion $\exists P_3.A \sqsubseteq A$, we obtain by induction on $h$ that $\mathcal{O} \models A(e_1)$. Finally, by $P_3(v, e_1)$, we obtain that $\mathcal{O} \models A(v)$.

For Cases 2 and 3, the proof follows from Case 1 and observations analogous to the ones for Theorem 6.1. □

We also state, without a proof the following result, which shows that qualified existential restrictions on the left-hand side of inclusion assertions together with inverse roles are sufficient to obtain PTIME-hardness.

**Theorem 6.3.** *Instance checking (and hence ontology satisfiability and query answering) is* PTIME*-hard in data complexity for ontologies $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{A}$ is an ABox, and $\mathcal{T}$ is a TBox of the form:*

$$\begin{array}{lcl} Cl & \longrightarrow & A \mid \exists P.A \mid \exists P^- A \\ Cr & \longrightarrow & A \mid \exists P \\ \multicolumn{3}{l}{\textit{Assertions in } \mathcal{T}: \quad Cl \sqsubseteq Cr.} \end{array}$$

We now show three cases where the TBox language becomes so expressive that the data complexity of answering CQs becomes coNP-hard, i.e., as hard as for very expressive DLs [71].

**Theorem 6.4.** *Answering CQs is coNP-hard in data complexity for ontologies $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{A}$ is an ABox, and $\mathcal{T}$ is a TBox of one of the following forms:*

1. $$\begin{array}{lcl} Cl & \rightarrow & A \\ Cr & \rightarrow & A \mid A_1 \sqcup A_2 \\ R & \rightarrow & P \\ \multicolumn{3}{l}{\textit{Assertions in } \mathcal{T}: \quad Cl \sqsubseteq Cr.} \end{array}$$

2. $Cl \rightarrow A \mid \neg A$
   $Cr \rightarrow A$
   *Assertions in* $\mathcal{T}$*:* $Cl \sqsubseteq Cr$.
3. $Cl \rightarrow A \mid \forall P.A$
   $Cr \rightarrow A$
   *Assertions in* $\mathcal{T}$*:* $Cl \sqsubseteq Cr$.

*Proof.* In all three cases, the proof is an adaptation of the proof of coNP-hardness of instance checking for the DL $\mathcal{ALE}$ presented in [39]. The proof is based on a reduction of $2+2$-CNF unsatisfiability, shown to be coNP-complete in [39], to CQ answering. A $2+2$-CNF formula on an alphabet $\mathcal{P} = \{\ell_1, \ldots, \ell_m\}$ is a CNF formula $F = C_1 \wedge \cdots \wedge C_n$ in which each clause $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$ has exactly four literals, two positive ones, $L^i_{1+}$ and $L^i_{2+}$, and two negative ones, $\neg L^i_{1-}$ and $\neg L^i_{2-}$, where the propositional letters $L^i_{1+}$, $L^i_{2+}$, $L^i_{1-}$, and $L^i_{2-}$ are elements of $\mathcal{P} \cup \{true, false\}$.

We first consider Case 1. Given a $2+2$-CNF formula $F$ as above, we associate with it an ontology $\mathcal{O}_F = \langle \mathcal{T}, \mathcal{A}_F \rangle$ and a boolean CQ $q$ as follows. $\mathcal{O}_F$ has one constant $\ell$ for each letter $\ell \in \mathcal{P}$, one constant $c_i$ for each clause $C_i$, plus two constants $true$ and $false$ for the corresponding propositional constants. The atomic concepts of $\mathcal{O}_F$ are $O$, $A_t$, and $A_f$, and the atomic roles are $P_1$, $P_2$, $N_1$, $N_2$. Then, we set

$$
\begin{aligned}
\mathcal{T} &= \{\ O \ \sqsubseteq A_t \sqcup A_f\ \},\\
\mathcal{A}_F &= \{\ A_t(true),\ A_f(false),\ O(\ell_1),\ \ldots\ O(\ell_m),\\
&\qquad P_1(c_1, \ell^1_{1+}),\ P_2(c_1, \ell^1_{2+}),\ N_1(c_1, \ell^1_{1-}),\ N_2(c_1, \ell^1_{2-}),\\
&\qquad \cdots\\
&\qquad P_1(c_n, \ell^n_{1+}),\ P_2(c_n, \ell^n_{2+}),\ N_1(c_n, \ell^n_{1-}),\ N_2(c_n, \ell^n_{2-})\ \},\ \text{and}\\
q() &= P_1(c, f_1), A_f(f_1),\ P_2(c, f_2), A_f(f_2),\ N_1(c, t_1), A_t(t_1),\ N_2(c, t_2), A_t(t_2).
\end{aligned}
$$

Notice that only the ABox $\mathcal{A}_F$ depends on the formula $F$, and that the TBox contains a single inclusion assertion involving a concept disjunction.

Intuitively, the membership to the extension of $A_f$ or $A_t$ corresponds to the truth values $true$ and $false$ respectively and checking whether $() \in cert(q, \mathcal{O}_F)$ (i.e., the query evaluates to true in $\mathcal{O}_F$) corresponds to checking whether in every truth assignment for the formula $F$ there exists a clause whose positive literals are interpreted as $false$, and whose negative literals are interpreted as $true$, i.e., a clause that is not satisfied. Note that the ABox $\mathcal{A}_F$ contains the assertions $A_t(true)$ and $A_f(false)$ in order to guarantee that in each model $\mathcal{I}$ of $\mathcal{O}_F$ the constants $true$ and $false$ are respectively in $A^{\mathcal{I}}_t$ and $A^{\mathcal{I}}_f$ (and possibly in both).

Now, it remains to prove that the formula $F$ is unsatisfiable if and only if $() \in cert(q, \mathcal{O}_F)$.

"$\Rightarrow$" Suppose that $F$ is unsatisfiable. Consider a model $\mathcal{I}$ of $\mathcal{O}_F$ (which always exists since $\mathcal{O}_F$ is always satisfiable), and let $\delta_{\mathcal{I}}$ be the truth assignment for $F$ such that $\delta_{\mathcal{I}}(\ell) = true$ iff $\ell^{\mathcal{I}} \in A^{\mathcal{I}}_t$, for every letter $\ell \in \mathcal{P}$ (and corresponding constant in $\mathcal{O}_F$). Since $F$ is unsatisfiable, there exists a clause $C_i$ that is not satisfied by $\delta_{\mathcal{I}}$, and therefore $\delta_{\mathcal{I}}(L^i_{1+}) = false$, $\delta_{\mathcal{I}}(L^i_{2+}) = false$, $\delta_{\mathcal{I}}(L^i_{1-}) = true$ and $\delta_{\mathcal{I}}(L^i_{2-}) = true$. It follows that in $\mathcal{I}$ the interpretation of the constants related in $\mathcal{A}_F$ to $c_i$ through the roles $P_1$ and $P_2$ is not in $A^{\mathcal{I}}_t$ and, since $\mathcal{I}$ satisfies $O \sqsubseteq A_t \sqcup A_f$, it is in $A^{\mathcal{I}}_f$. Similarly, the

interpretation of the constants related to $c_i$ through the roles $N_1$ and $N_2$ is in $A_t^{\mathcal{I}}$. Thus, there exists a substitution $\sigma$ that assigns the variables in $q$ to elements of $\Delta^{\mathcal{I}}$ in such a way that $\sigma(q)$ evaluates to true in $\mathcal{I}$ (notice that this holds even if the propositional constants $true$ or $false$ occur in $F$). Therefore, since this argument holds for each model $\mathcal{I}$ of $\mathcal{O}_F$, we can conclude that $() \in cert(q, \mathcal{O}_F)$.

"$\Leftarrow$" Suppose that $F$ is satisfiable, and let $\delta$ be a truth assignment satisfying $F$. Let $\mathcal{I}_\delta$ be the interpretation for $\mathcal{O}_F$ defined as follows:

$$
\begin{aligned}
O^{\mathcal{I}_\delta} &= \{\ \ell^{\mathcal{I}_\delta} \mid \ell \text{ occurs in } F\ \},\\
A_t^{\mathcal{I}_\delta} &= \{\ \ell^{\mathcal{I}_\delta} \mid \delta(\ell) = true\ \} \cup \{true\},\\
A_f^{\mathcal{I}_\delta} &= \{\ \ell^{\mathcal{I}_\delta} \mid \delta(\ell) = false\ \} \cup \{false\},\\
P^{\mathcal{I}_\delta} &= \{\ (a_1^{\mathcal{I}_\delta}, a_2^{\mathcal{I}_\delta}) \mid P(a_1, a_2) \in \mathcal{A}_F\}, \text{ for } P \in \{P_1, P_2, N_1, N_2\}.
\end{aligned}
$$

It is easy to see that $\mathcal{I}_\delta$ is a model of $\mathcal{O}_F$. On the other hand, since $\delta$ satisfies $F$, for every clause $c_i$ in $F$ there exists a positive literal $\ell_+^i$ such that $\delta(\ell_+^i) = true$, or a negative literal $\ell_-^i$ such that $\delta(\ell_-^i) = false$. It follows that for every constant $c_i$, there exists either a role ($P_1$ or $P_2$) that relates $c_i$ to a constant whose interpretation is in $A_t^{\mathcal{I}_\delta}$ or there exists a role ($N_1$ or $N_2$) that relates $c_i$ to a constant whose interpretation is in $A_f^{\mathcal{I}_\delta}$. Since the query $q$ evaluates to true in $\mathcal{I}_\delta$ only if there exists a constant $c_i$ in $\mathcal{O}_F$ such that the interpretations of the constants related to $c_i$ by roles $P_1$ and $P_2$ are both in $A_f^{\mathcal{I}_\delta}$ and the interpretations of the constants related to $c_i$ by roles $N_1$ and $N_2$ are both in $A_t^{\mathcal{I}_\delta}$, it follows that the query $q$ evaluates to $false$ in $\mathcal{I}_\delta$ and therefore $() \notin cert(q, \mathcal{O}_F)$.

The proofs for Case 2 and Case 3 are obtained by reductions of $2+2$-CNF unsatisfiability to CQ answering analogous to the one for Case 1. More precisely, for Case 2 the ontology $\mathcal{O}_F = \langle \mathcal{T}, \mathcal{A}_F \rangle$ has the same constants and the same atomic roles as for Case 1, and has only the atomic concepts $A_t$ and $A_f$. Then, $\mathcal{T}_F = \{\neg A_t \sqsubseteq A_f\}$ and $\mathcal{A}_F$ is as for Case 1 but without the assertions involving the concept $O$. The query $q$ is as for Case 1.

For Case 3, $\mathcal{O}_F$ has the same constants as for Cases 1 and 2, the same atomic roles as for Cases 1 and 2 plus an atomic role $P_t$, and two atomic concepts $A$ and $A_f$. Then, $\mathcal{T} = \{\forall P_t.A \sqsubseteq A_f\}$ and $\mathcal{A}_F$ is as for Case 2 but without the assertion $A_t(true)$, which is substituted by the assertion $P_t(true, a)$, where $a$ is a new constant not occurring elsewhere in $\mathcal{O}_F$. The query is

$$
\begin{aligned}
q() = P_1(c, f_1), A_f(f_1), \quad & P_2(c, f_2), A_f(f_2),\\
N_1(c, t_1), P_t(t_1, x_1), \quad & N_2(c, t_2), P_t(t_2, x_2).
\end{aligned}
$$

The correctness of the above reductions can be proved as done for Case 1. We finally point out that the intuition behind the above results is that in all three cases it is possible to require a reasoning by case analysis, caused by set covering assertions. Indeed, in Case 2 we have explicitly asserted $O \sqsubseteq A_t \sqcup A_f$, while in Case 1 and Case 3, $A_t$ and $A_f$, and $\forall P_t.A$ and $\exists P_t$ cover the entire domain, respectively. □

The results proved in Theorems 6.1, 6.2, 6.3, and 6.4 are summarized in Table 1. Notice that, while the NLOGSPACE-hardness and PTIME-hardness results in the table hold already for instance checking (i.e., answering atomic queries), the coNP-hardness results

**Table 1.** Data Complexity of query answering for various extensions of *DL-Lite*$_{\mathcal{A},id}$

| $Cl$ | $Cr$ | $\mathcal{F}$ | $\mathcal{R}$ | Data complexity of query answering | Proved in |
|---|---|---|---|---|---|
| *DL-Lite*$_{\mathcal{A},id}$ | | $\surd$ | $\surd^*$ | in AC$^0$ | Theorems 5.17, 5.25 |
| $A \mid \exists P.A$ | $A$ | – | – | NLOGSPACE-hard | Theorem 6.1, Case 1 |
| $A$ | $A \mid \forall P.A$ | – | – | NLOGSPACE-hard | Theorem 6.1, Case 2 |
| $A$ | $A \mid \exists P.A$ | $\surd$ | – | NLOGSPACE-hard | Theorem 6.1, Case 3 |
| $A \mid \exists P.A \mid A_1 \sqcap A_2$ | $A$ | – | – | PTIME-hard | Theorem 6.2, Case 1 |
| $A \mid A_1 \sqcap A_2$ | $A \mid \forall P.A$ | – | – | PTIME-hard | Theorem 6.2, Case 2 |
| $A \mid A_1 \sqcap A_2$ | $A \mid \exists P.A$ | $\surd$ | – | PTIME-hard | Theorem 6.2, Case 3 |
| $A \mid \exists P.A \mid \exists P^-.A$ | $A \mid \exists P$ | – | – | PTIME-hard | Theorem 6.3 |
| $A$ | $A \mid A_1 \sqcup A_2$ | – | – | coNP-hard | Theorem 6.4, Case 1 |
| $A \mid \neg A$ | $A$ | – | – | coNP-hard | Theorem 6.4, Case 2 |
| $A \mid \forall P.A$ | $A$ | – | – | coNP-hard | Theorem 6.4, Case 3 |

**Legenda:** $A$ (possibly with subscript) = atomic concept, $P$ = atomic role,
$Cl/Cr$ = left/right-hand side of inclusion assertions, $\mathcal{F}$ = functionality assertions allowed,
$\mathcal{R}$ = role/relationship inclusions allowed, where $^*$ denotes restricted interaction between functionality and role inclusion, according to Definition 2.1.
The NLOGSPACE and PTIME hardness results hold already for instance checking.

proved in Theorem 6.4 hold for CQ answering, but do *not* hold for instance checking. Indeed, as shown in [4], instance checking (and hence ontology satisfiability) stays in AC$^0$ in data complexity for *DL-Lite*$_{\mathcal{A}}$ extended with arbitrary boolean combinations (i.e., negation, and disjunction) of concepts, both in the left-hand side and in the right-hand side of inclusion assertions. [4] shows also that *DL-Lite*$_{\mathcal{A}}$ can be extended with *number restrictions* (cf. also Section 2.2) and with the additional role constraints present in OWL 2 QL that are not already expressible in *DL-Lite*$_{\mathcal{A}}$, such as reflexivity, irreflexivity, and asymmetry, without losing FOL-rewritability of satisfiability and UCQ query answering.

Finally, the following result from [25] motivates the locality restriction in identification assertions, i.e., that at least one of the paths in an identification assertion must have length 1. Indeed, if such a restriction is removed, we lose again FOL-rewritability of reasoning.

**Theorem 6.5.** *Ontology satisfiability (and hence instance checking and query answering) in DL-Lite*$_{\mathcal{A}}$ *extended with single-path IdCs that are non-local is* NLOGSPACE*-hard in data complexity.*

*Proof.* The proof is based again on a reduction of reachability in directed graphs (see proof of Theorem 6.1) to ontology satisfiability. Let $G = \langle V, E \rangle$ be a directed graph, where $V$ is a set of vertexes and $E$ a set of directed edges, and let $s$ and $t$ be two vertexes of $G$. We consider the graph represented through functional relations $F$ (to connect a vertex to the first element of the chain of its children), $N$ (to connect an element of the chain to the next), and $S$ (to connect the elements forming the chain to the actual

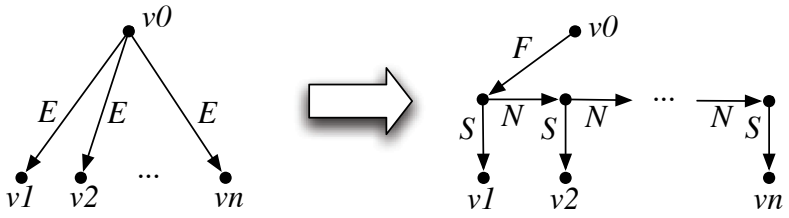



**Fig. 17.** Representation of a graph through the functional relations $F$, $N$, $S$

child vertexes of the graph), and denote with $V^+$ the set of vertexes augmented by the vertexes used in such a representation (cf. Figure 17).

From $G$ and the two vertexes $s$ and $t$, we define the ontology $\mathcal{O}_{idcs} = \langle \mathcal{T}_{idcs}, \mathcal{A}_G \rangle$ as follows:

- The alphabet of $\mathcal{T}_{idcs}$ consists of an atomic concept $A$, that intuitively denotes the vertexes of two copies of $G$, of an atomic concept $A_t$, and of atomic roles $P_F$, $P_N$, $P_S$, and $P_0$. Then

  $$\mathcal{T}_{idcs} \;=\; \{A \sqsubseteq \exists P_0,\ (\mathsf{id}\ A_t\ P_0)\} \cup \{(\mathsf{id}\ \exists P_0^-\ P_0^- \circ P_{\mathcal{R}}^- \circ P_0) \mid \mathcal{R} \in \{F, N, S\}\}.$$

  Notice that $\mathcal{T}_{idcs}$ does not depend on $G$.
- The ABox $\mathcal{A}_G$ is defined from the graph $G$ and the two vertexes $s$ and $t$ as follows:

  $$\begin{aligned}\mathcal{A}_G \;=\; &\{P_{\mathcal{R}}(a_1, a_2),\ P_{\mathcal{R}}(a_1', a_2') \mid (a_1, a_2) \in \mathcal{R},\ \text{for } \mathcal{R} \in \{F, N, S\}\} \cup \\ &\{A(a),\ A(a') \mid a \in V^+\} \cup \{P_0(s, a_{init}),\ P_0(s', a_{init}),\ A_t(t),\ A_t(t')\}.\end{aligned}$$

  In other words, we introduce for each node $a$ of the graph $G$ two constants $a$ and $a'$ in $\mathcal{O}$, and we encode in $\mathcal{A}_G$ two copies of (the representation of) $G$. In addition, we include in $\mathcal{A}_G$ the assertions $P_0(s, a_{init})$ and $P_0(s', a_{init})$ connecting the two copies of the start vertex $s$ to an additional constant $a_{init}$ that does not correspond to any vertex of (the representation of) $G$. We also include the assertions $A_t(t)$ and $A_t(t')$, which are exploited to encode the reachability test (cf. Figure 18).

It can be shown by induction on the length of paths from $s$, that $t$ is reachable from $s$ in $G$ iff $\mathcal{O}_{idcs}$ is unsatisfiable. Intuitively, the TBox enforces that each individual contributing to the encoding of the two copies of $G$ has an outgoing $P_0$ edge. Moreover, the path-identification assertions enforce that each object that is in the range of such a $P_0$ edge is identified by a suitable path. Hence, starting from $a_{init}$, corresponding pairs of objects (in the range of $P_0$) in the two copies of the graph that are reachable from $s$ and $s'$, respectively, will be unified with each other. If $t$ is reachable from $s$, also the two objects connected to $t$ and $t'$ via $P_0$ will be unified. Hence by the identification assertion $(\mathsf{id}\ A_t\ P_0)$, we have that $t$ and $t'$ are forced to be equal, which makes the ontology unsatisfiable (due to the unique name assumption). Notice that, for the reduction to work, we needed to make sure that each vertex has at most one outgoing edge, hence we have preliminarily encoded the edge relation $E$ using the functional relations $F$, $N$, and $S$. □

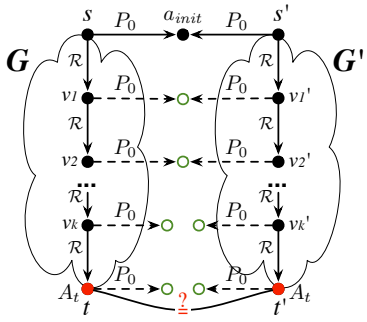



**Fig. 18.** Structure of a potential model of the ontology $\mathcal{O}_{idcs}$ used in the proof of Theorem 6.5

### 6.2 Changing the *DL-Lite* Semantics

Concerning the possibility of strengthening the semantics, we analyze the consequences of removing the *unique name assumption* (UNA), i.e., the assumption that, in every interpretation of an ontology, two distinct constants denote two different domain elements. Unfortunately, this leads instance checking (and satisfiability) out of $AC^0$, and therefore instance checking and query answering are not FOL-rewritable anymore.

**Theorem 6.6.** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology interpreted without the unique name assumption. Then instance checking with respect to $\mathcal{O}$ is* NLOGSPACE*-hard in the size of $\mathcal{A}$.*

*Proof.* The proof is based again on a LOGSPACE reduction of reachability in directed graphs to instance checking. Let $G = \langle V, E \rangle$ be a directed graph and $s$ and $t$ two vertexes of $G$. As in the proof of Theorem 6.5, we consider $G$ represented through first-child and next-sibling functional relations $F$, $N$, $S$ (cf. Figure 17).

From $G$ and the two vertexes $s$ and $t$, we define an ontology $\mathcal{O}_{una} = \langle \mathcal{T}_{una}, \mathcal{A}_G \rangle$ as follows:

- The alphabet of $\mathcal{T}_{una}$ consists of an atomic concept $A$ and of atomic roles $P_F$, $P_N$, $P_S$, and $P_0$. The TBox itself imposes only that all roles are functional, i.e.,

$$\mathcal{T}_{una} \;=\; \{(\mathsf{funct}\ P_0)\} \cup \{(\mathsf{funct}\ P_{\mathcal{R}}) \mid \mathcal{R} \in \{F, N, S\}\}.$$

  Notice that $\mathcal{T}_{una}$ does not depend on $G$.
- The ABox $\mathcal{A}_G$ is defined from the graph $G$ and the two vertexes $s$ and $t$ as follows:

$$\begin{aligned}\mathcal{A}_G \;=\; &\{P_{\mathcal{R}}(a_1, a_2), P_{\mathcal{R}}(a_1', a_2') \mid (a_1, a_2) \in \mathcal{R}, \text{ for } \mathcal{R} \in \{F, N, S\}\} \cup \\ &\{A(t),\ P_0(a_{init}, s),\ P_0(a_{init}, s')\}\end{aligned}$$

  In other words, we introduce for each node $a$ of the graph $G$ two constants $a$ and $a'$ in $\mathcal{O}$, and we encode in $\mathcal{A}_G$ two copies of (the representation of) $G$. In addition, we include in $\mathcal{A}_G$ the facts $P_0(a_{init}, s)$, $P_0(a_{init}, s')$, and $A(t)$, where $a_{init}$ is an additional constant that does not correspond to any vertex of (the representation of) $G$ (cf. Figure 19).

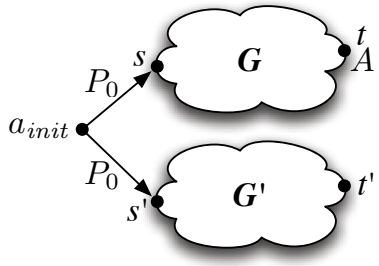



**Fig. 19.** Structure of the ABox $\mathcal{A}_G$ used in the proof of Theorem 6.6

It is now possible to prove that $t$ is reachable from $s$ in $G$ if and only if $\mathcal{O}_{una} \models A(t')$. Indeed, it is easy to verify that the latter holds if and only if for every model $\mathcal{I}$ of $\mathcal{O}_{una}$, the constants $t$ and $t'$ are interpreted as the same object, i.e., $t^{\mathcal{I}} = t'^{\mathcal{I}}$. This is the case if and only if $t^{\mathcal{I}}$ and $t'^{\mathcal{I}}$ are forced to be equal by the functionality of the roles $P_0$, $P_F$, $P_N$, and $P_S$. By exploiting the structure of the ABox $\mathcal{A}_G$, one can prove by induction on the length of paths from $s$, that such an equality is enforced if and only if $t$ is reachable from $s$ in $G$. □

## 7 Accessing Data through *DL-Lite*$_{\mathcal{A},id}$ Ontologies

The discussion presented in the previous sections on *DL-Lite*$_{\mathcal{A},id}$ ontologies assumed a relational representation for the ABox assertions. This is a reasonable assumption only in those cases where the ontology is managed by an ad hoc system, and is built from scratch for the specific application.

We argue that this is not a typical scenario in current applications (e.g., in Enterprise Application Integration). Indeed, we believe that one of the most interesting real-world usages of ontologies is what we have called *ontology-based data access* (OBDA). OBDA is the problem of accessing a set of existing data sources by means of a conceptual representation expressed in terms of an ontology. In such a scenario, the TBox of the ontology provides a shared, uniform, abstract view of the intensional level of the application domain, whereas the information about the extensional level (the instances of the ontology) resides in the data sources, which are developed independently of the conceptual layer, and are managed by traditional technologies (such as relational database technology). In other words, the ABox of the ontology does not exist as an independent syntactic object. Rather, the instances of concepts and roles in the ontology are simply an abstract and virtual representation of some real data stored in existing data sources. Therefore, the problem arises of establishing sound mechanisms for linking existing data to the instances of the concepts and the roles in the ontology.

In this section, we present a solution that has been proposed recently for this problem [75], based on a mapping mechanism that enables a designer to link existing data sources to an ontology expressed in *DL-Lite*$_{\mathcal{A},id}$, and by illustrating a formal framework capturing the notion of *DL-Lite*$_{\mathcal{A},id}$ *ontology with mappings*. In the following, we assume that the data sources are expressed in terms of the relational data model. In other

words, all the technical development presented in the rest of this section assumes that the set of sources to be linked to the ontology constitutes a single relational database. Note that this is a realistic assumption, since many data federation tools are now available that are able to wrap a set of heterogeneous sources and present them as a single relational database.

Before delving into the details of the method, a preliminary discussion on the notorious *impedance mismatch problem* between values (data) and objects is in order [66]. When mapping relational data sources to ontologies, one should take into account that sources store values, whereas instances of concepts are objects, where each object should be denoted by an ad hoc identifier (e.g., a constant in logic), not to be confused with any data item. For example, if a data source stores data about persons, it is likely that values for social security numbers, names, etc. will appear in the sources. However, at the conceptual level, the ontology will represent persons in terms of a concept, and instances of such concepts will be denoted by object constants.

One could argue that data sources might, in some cases, store directly object identifiers. However, in order to use such object identifiers at the conceptual level, one should make sure that such identifiers have been chosen on the basis of an "agreement" among the sources on the form used to represent objects. This is something occurring very rarely in practice. For all the above reasons, in *DL-Lite*$_{\mathcal{A},id}$, we take a radical approach. To face the impedance mismatch problem, and to tackle the possible lack of an a-priori agreement on identification mechanisms at the sources, we keep data values appearing in the sources separate from object identifiers at the conceptual level. In particular, we consider object identifiers formed by (logic) terms built out of data values stored at the sources. The way by which these terms will be defined starting from the data at the sources will be specified through suitable mapping assertions, to be described below. Note that this idea traces back to the work done in deductive object-oriented databases [54].

### 7.1 Linking Relational Data to Ontologies

To realize the above described idea from a technical point of view, we specialize the alphabets of object constants in a particular way, which we now describe in detail.

We remind the reader that $\Gamma_V$ is the alphabet of value constants in *DL-Lite*$_{\mathcal{A},id}$. We assume that data appearing at the sources are denoted by constants in $\Gamma_V$[15], and we introduce a new alphabet $\Lambda$ of *function symbols*, where each function symbol has an associated *arity*, specifying the number of arguments it accepts. On the basis of $\Gamma_V$ and $\Lambda$, we inductively define the set $\tau(\Lambda, \Gamma_V)$ of all *object terms* (or simply, *terms*) of the form $\mathbf{f}(d_1, \ldots, d_n)$ such that

- $\mathbf{f} \in \Lambda$,
- the arity of $\mathbf{f}$ is $n > 0$, and
- $d_1, \ldots, d_n \in \Gamma_V$.

[15] We could also introduce suitable conversion functions in order to translate values stored at the sources into value constants in $\Gamma_V$, but, for the sake of simplicity, we do not deal with this aspect here.

We finally sanction that the set $\Gamma_O$ of symbols used in *DL-Lite*$_{\mathcal{A},id}$ for denoting objects actually coincides with $\tau(\Lambda, \Gamma_V)$. In other words, we use the terms built from $\Gamma_V$ using the function symbols in $\Lambda$ for denoting the instances of concepts in ontologies.

All the notions defined for our logics remain unchanged. In particular, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ still assigns a different element of $\Delta^{\mathcal{I}}$ to every element of $\Gamma$, and, given that $\Gamma_O$ coincides with $\tau(\Lambda, \Gamma_V)$, this implies that different terms in $\tau(\Lambda, \Gamma_V)$ are interpreted as different objects in $\Delta^{\mathcal{I}}_O$, i.e., we enforce the unique name assumption on terms. Formally, this means that $\mathcal{I}$ is such that

- for each $a \in \Gamma_V$, $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}_V$,
- for each $a \in \Gamma_O$, i.e., for each $a \in \tau(\Lambda, \Gamma_V)$, $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}_O$,
- for each $a_1, a_2 \in \Gamma$, $a_1 \neq a_2$ implies $a_1^{\mathcal{I}} \neq a_2^{\mathcal{I}}$.

The syntax and the semantics of a *DL-Lite*$_{\mathcal{A}}$ TBox, ABox, and UCQ, introduced in Section 2, do not need to be modified. In particular, from the point of view of the semantics of queries, the notion of certain answers is exactly the same as the one presented in Section 2.4.

We can now turn our attention to the problem of specifying mapping assertions linking the data at the sources to the objects in the ontology. As mentioned, we assume that the data sources are wrapped into a relational database $\mathcal{D}$ (constituted by the relational schema, and the extensions of the relations), so that we can query such data by using SQL, and that all value constants stored in $\mathcal{D}$ belong to $\Gamma_V$ Also, the database $\mathcal{D}$ is independent from the ontology; in other words, our aim is to link to the ontology a collection of data that exist autonomously, and have not been necessarily structured with the purpose of storing the ontology instances.

In the following, we denote with $ans(\varphi, \mathcal{D})$ the set of tuples (of the arity of $\varphi$) of value constants returned as the result of the evaluation of the SQL query $\varphi$ over the database $\mathcal{D}$.

With these assumptions in place, to actually realize the link between the data and the ontology, we adapt principles and techniques from the literature on data integration [63]. In particular, we resort to *mappings* as described in the following definition. We make use of the notion of *variable term*, which is a term of the same form as the object terms introduced above, with the difference that variables may appear as arguments of the function. In other words, a variable term has the form $\mathbf{f}(\boldsymbol{z})$, where $\mathbf{f}$ is a function symbol in $\Lambda$ of arity $m$, and $\boldsymbol{z}$ denotes an $m$-tuple of variables or value constants.

**Definition 7.1.** *A DL-Lite*$_{\mathcal{A},id}$ ontology with mappings *is a triple* $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$*, where:*

- $\mathcal{T}$ *is a DL-Lite*$_{\mathcal{A},id}$ *TBox;*
- $\mathcal{D}$ *is a relational database;*
- $\mathcal{M}$ *is a set of* mapping assertions*, partitioned into two sets,* $\mathcal{M}^t$ *and* $\mathcal{M}^a$*, where:*
  - $\mathcal{M}^t$ *is a set of so-called* typing mapping assertions*, each one of the form*

$$\Phi \rightsquigarrow T_i,$$

*where* $\Phi$ *is a query of arity 1 over* $\mathcal{D}$*, denoting the projection of one relation over one of its columns, and* $T_i$ *is one of the DL-Lite*$_{\mathcal{A},id}$ *data types;*

- $\mathcal{M}^a$ *is a set of* data-to-object mapping assertions *(or simply mapping assertions), each one of the form*

$$\Phi(\boldsymbol{x}) \rightsquigarrow \Psi(\boldsymbol{y}, \boldsymbol{t}),$$

*where*
  - $\boldsymbol{x}$ *is a non-empty set of variables,*
  - $\boldsymbol{y} \subseteq \boldsymbol{x}$*,*
  - $\boldsymbol{t}$ *is a set of variable terms of the form* $\boldsymbol{f}(\boldsymbol{z})$*, with* $\boldsymbol{f} \in \Lambda$ *and* $\boldsymbol{z} \subseteq \boldsymbol{x}$*,*
  - $\Phi(\boldsymbol{x})$ *is an arbitrary SQL query over* $\mathcal{D}$*, with* $\boldsymbol{x}$ *as output variables, and*
  - $\Psi(\boldsymbol{y}, \boldsymbol{t})$ *is a CQ over* $\mathcal{T}$ *of arity* $n > 0$ *without non-distinguished variables, whose atoms are over the variables* $\boldsymbol{y}$ *and the variable terms* $\boldsymbol{t}$*.*

We briefly comment on the assertions in $\mathcal{M}$ as defined above. Typing mapping assertions are used to assign appropriate types to constants in the relations of $\mathcal{D}$. Basically, these assertions are used for interpreting the values stored in the database in terms of the types used in the ontology, and their usefulness is evident in all cases where the types in the data sources do not directly correspond to the types used in the ontology. Data-to-object mapping assertions, on the other hand, are used to map data in the database to instances of concepts, roles, and attributes in the ontology.

We next give an example of *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings.

*Example 7.2.* Let $\mathcal{D}_{pr}$ be the database constituted by a set of relations with the following signature:

$$\begin{array}{l} D_1[\texttt{SSN:}\textbf{STRING}\texttt{, PROJ:}\textbf{STRING}\texttt{, D:}\textbf{DATE}], \\ D_2[\texttt{SSN:}\textbf{STRING}\texttt{, NAME:}\textbf{STRING}], \\ D_3[\texttt{CODE:}\textbf{STRING}\texttt{, NAME:}\textbf{STRING}], \\ D_4[\texttt{CODE:}\textbf{STRING}\texttt{, SSN:}\textbf{STRING}] \end{array}$$

We assume that, from the analysis of the above data sources, the following meaning of the above relations has been derived.

- Relation $D_1$ stores tuples $(s, p, d)$, where $s$ and $p$ are strings and $d$ is a date, such that $s$ is the social security number of a temporary employee, $p$ is the name of the project she works for (different projects have different names), and $d$ is the ending date of the employment.
- Relation $D_2$ stores tuples $(s, n)$ of strings consisting of the social security number $s$ of an employee and her name $n$.
- Relation $D_3$ stores tuples $(c, n)$ of strings consisting of the code $c$ of a manager and her name $n$.
- Finally, relation $D_4$ relates managers' code with their social security number.

A possible extension for the above relations is given by the following sets of tuples:

$$\begin{array}{l} D_1 = \{(\texttt{20903}, \texttt{"Tones"}, \texttt{25/03/09})\} \\ D_2 = \{(\texttt{20903}, \texttt{"Rossi"}), (\texttt{55577}, \texttt{"White"})\} \\ D_3 = \{(\texttt{"X11"}, \texttt{"White"}), (\texttt{"X12"}, \texttt{"Black"})\} \\ D_4 = \{(\texttt{"X11"}, \texttt{29767})\} \end{array}$$

| | |
|---|---|
| $\mathit{Manager} \sqsubseteq \mathit{Employee}$ | $\mathit{Project} \sqsubseteq \delta(\mathbf{projName})$ |
| $\mathit{TempEmp} \sqsubseteq \mathit{Employee}$ | $\rho(\mathbf{projName}) \sqsubseteq \texttt{xsd:string}$ |
| $\mathit{Employee} \sqsubseteq \mathit{Person}$ | $(\mathsf{funct}\ \mathbf{projName})$ |
| $\mathit{Employee} \sqsubseteq \exists \mathit{WORKS\text{-}FOR}$ | $\mathit{TempEmp} \sqsubseteq \delta(\mathbf{until})$ |
| $\exists \mathit{WORKS\text{-}FOR}^{-} \sqsubseteq \mathit{Project}$ | $\delta(\mathbf{until}) \sqsubseteq \exists \mathit{WORKS\text{-}FOR}$ |
| $\mathit{Person} \sqsubseteq \delta(\mathbf{persName})$ | $\rho(\mathbf{until}) \sqsubseteq \texttt{xsd:date}$ |
| $\rho(\mathbf{persName}) \sqsubseteq \texttt{xsd:string}$ | $(\mathsf{funct}\ \mathbf{until})$ |
| $(\mathsf{funct}\ \mathbf{persName})$ | $\mathit{Manager} \sqsubseteq \neg\delta(\mathbf{until})$ |

**Fig. 20.** The *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}_{pr}$ for the projects example

| | | | |
|---|---|---|---|
| $m_1^t$ : | SELECT SSN FROM $D_1$ | $\leadsto$ | xsd:string |
| $m_2^t$ : | SELECT SSN FROM $D_2$ | $\leadsto$ | xsd:string |
| $m_3^t$ : | SELECT CODE FROM $D_3$ | $\leadsto$ | xsd:string |
| $m_4^t$ : | SELECT CODE FROM $D_4$ | $\leadsto$ | xsd:string |
| $m_5^t$ : | SELECT PROJ FROM $D_1$ | $\leadsto$ | xsd:string |
| $m_6^t$ : | SELECT NAME FROM $D_2$ | $\leadsto$ | xsd:string |
| $m_7^t$ : | SELECT NAME FROM $D_3$ | $\leadsto$ | xsd:string |
| $m_8^t$ : | SELECT SSN FROM $D_4$ | $\leadsto$ | xsd:string |
| $m_9^t$ : | SELECT D FROM $D_1$ | $\leadsto$ | xsd:date |

**Fig. 21.** The typing mapping assertions $\mathcal{M}_{pr}^t$ for the projects example

Consider now the TBox $\mathcal{T}_{pr}$ shown in Figure 20, which models information about employees and projects they work for. Specifically, the assertions in $\mathcal{T}_{pr}$ state the following. Managers and temporary employees are two kinds of employees, and employees are persons. Each employee works for at least one project, whereas each person and each project has a unique name. Both person names and project names are strings, whereas the attribute **until** associates objects with a unique date. In particular, any temporary employee has an associated date (which indicates the expiration date of her contract), and everyone having a value for the attribute **until** participates in the role *WORKS-FOR*. Finally, $\mathcal{T}_{pr}$ specifies that a manager does not have any value for the attribute **until**, meaning that a manager has a permanent position. Note that this implies that no employee is simultaneously a temporary employee and a manager.

Now, let $\Lambda = \{\mathbf{pers}, \mathbf{proj}, \mathbf{mgr}\}$ be a set of function symbols, all of arity 1. Consider the *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings $\mathcal{OM}_{pr} = \langle \mathcal{T}_{pr}, \mathcal{M}_{pr}, \mathcal{D}_{pr} \rangle$, where $\mathcal{M}_{pr} = \mathcal{M}_{pr}^t \cup \mathcal{M}_{pr}^a$, with $\mathcal{M}_{pr}^t$ shown in Figure 21, and $\mathcal{M}_{pr}^a$ shown in Figure 22. We briefly comment on the data-to-ontology mapping assertions in $\mathcal{M}_{pr}^a$:

- $m_1^a$ maps every tuple $(s, p, d)$ in $D_1$ to a temporary employee $\mathbf{pers}(s)$, working until $d$ for project $\mathbf{proj}(p)$ with name $p$.
- $m_2^a$ maps every tuple $(s, n)$ in $D_2$ to an employee $\mathbf{pers}(s)$ with name $n$.
- $m_3^a$ and $m_4^a$ tell us how to map data in $D_3$ and $D_4$ to managers and their name in the ontology. Note that, if $D_4$ provides the social security number $s$ of a manager whose

code is in $D_3$, then we use the social security number to form the corresponding object term, i.e., the object term has the form **pers**$(s)$. Instead, if $D_4$ does not provide this information, then we use an object term of the form **mgr**$(c)$, where $c$ is a code, to denote the corresponding instance of the concept *Manager*. ■

### 7.2 Semantics of Ontologies with Mappings

In order to define the semantics of a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings, we need to define when an interpretation *satisfies an assertion in* $\mathcal{M}$ *w.r.t. a database* $\mathcal{D}$. To this end, we make use of the notion of ground instance of a formula. Let $\Psi(\boldsymbol{x})$ be a formula over a *DL-Lite*$_{\mathcal{A},id}$ TBox with $n$ distinguished variables $\boldsymbol{x}$, and let $\boldsymbol{v}$ be a tuple of value constants of arity $n$. Then the ground instance $\Psi[\boldsymbol{x}/\boldsymbol{v}]$ of $\Psi(\boldsymbol{x})$ is the formula obtained from $\Psi(\boldsymbol{x})$ by substituting every occurrence of $x_i$ with $v_i$, for $i \in \{1, \ldots, n\}$. We are now ready to define when an interpretation satisfies a mapping assertion.

In the following, we denote with $ans(\varphi, \mathcal{D})$ the set of tuples (of the arity of $\varphi$) of value constants returned as the result of the evaluation of the SQL query $\varphi$ over the database $\mathcal{D}$.

**Definition 7.3.** *Let* $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$*, with* $\mathcal{M} = \mathcal{M}^t \cup \mathcal{M}^a$*, be a DL-Lite*$_{\mathcal{A},id}$ *ontology with mappings and* $\mathcal{I}$ *an interpretation of* $\mathcal{OM}$*.*

- *Let* $m^t$ *be an assertion in* $\mathcal{M}^t$ *of the form* $\Phi \rightsquigarrow T_i$*. We say that* $\mathcal{I}$ satisfies $m^t$ w.r.t. $\mathcal{D}$*, if for every* $v \in ans(\Phi, \mathcal{D})$*, we have that* $v \in val(T_i)$*.*
- *Let* $m^a$ *be an assertion in* $\mathcal{M}^a$ *of the form* $\Phi(\boldsymbol{x}) \rightsquigarrow \Psi(\boldsymbol{y}, \boldsymbol{t})$*, where* $\boldsymbol{x}$*,* $\boldsymbol{y}$*, and* $\boldsymbol{t}$ *are as in Definition 7.1. We say that* $\mathcal{I}$ satisfies $m^a$ w.r.t. $\mathcal{D}$*, if for every tuple of values* $\boldsymbol{v}$ *such that* $\boldsymbol{v} \in ans(\Phi, \mathcal{D})$*, and for every ground atom* $X$ *in* $\Psi[\boldsymbol{x}/\boldsymbol{v}]$*, we have that:*
  - *if* $X$ *has the form* $A(s)$*, then* $s^{\mathcal{I}} \in A^{\mathcal{I}}$*;*
  - *if* $X$ *has the form* $F(s)$*, then* $s^{\mathcal{I}} \in F^{\mathcal{I}}$*;*
  - *if* $X$ *has the form* $P(s_1, s_2)$*, then* $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$*;*
  - *if* $X$ *has the form* $U(s_1, s_2)$*, then* $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in U^{\mathcal{I}}$*.*

$m_1^a$ : `SELECT SSN, PROJ, D FROM` $D_1$ $\rightsquigarrow$ *TempEmp*(**pers**(SSN)), *WORKS-FOR*(**pers**(SSN), **proj**(PROJ)), **projName**(**proj**(PROJ), PROJ), **until**(**pers**(SSN), D)

$m_2^a$ : `SELECT SSN, NAME FROM` $D_2$ $\rightsquigarrow$ *Employee*(**pers**(SSN)), **persName**(**pers**(SSN), NAME)

$m_3^a$ : `SELECT SSN, NAME FROM` $D_3$`,` $D_4$ `WHERE` $D_3$`.CODE =` $D_4$`.CODE` $\rightsquigarrow$ *Manager*(**pers**(SSN)), **persName**(**pers**(SSN), NAME)

$m_4^a$ : `SELECT CODE, NAME FROM` $D_3$ `WHERE CODE NOT IN (SELECT CODE FROM` $D_4$`)` $\rightsquigarrow$ *Manager*(**mgr**(CODE)), **persName**(**mgr**(CODE), NAME)

**Fig. 22.** The object-to-data mapping assertions $\mathcal{M}_{pr}^a$ for the projects example

*We say that* $\mathcal{I}$ satisfies $\mathcal{M}$ w.r.t. $\mathcal{D}$*, if it satisfies every assertion in* $\mathcal{M}$ *w.r.t.* $\mathcal{D}$*. We say that* $\mathcal{I}$ *is a* model *of* $\mathcal{OM}$ *if* $\mathcal{I}$ *is a model of* $\mathcal{T}$ *and satisfies* $\mathcal{M}$ *w.r.t.* $\mathcal{D}$*. Finally, we denote with* $Mod(\mathcal{OM})$ *the set of models of* $\mathcal{OM}$*, and we say that* $\mathcal{OM}$ *is* satisfiable *if* $Mod(\mathcal{OM}) \neq \emptyset$*.*

*Example 7.4.* One can easily verify that the ontology with mappings $\mathcal{OM}_{pr}$ of Example 7.2 is satisfiable. ∎

Note that the mapping mechanism described above nicely deals with the fact that the database $\mathcal{D}$ and the ontology $\mathcal{OM}$ are based on different semantic assumptions. Indeed, the semantics of $\mathcal{D}$ follows the so-called "closed world assumption" [79], which intuitively sanctions that every fact that is not explicitly stored in the database is false. On the contrary, the semantics of $\mathcal{OM}$ is open, in the sense that nothing is assumed about the facts that do not appear explicitly in the ABox. In a mapping assertion of the form $\Phi \rightsquigarrow \Psi$, the closed semantics of $\mathcal{D}$ is taken into account by the fact that $\Phi$ is evaluated as a standard relational query over the database $\mathcal{D}$, while the open semantics of $\mathcal{OM}$ is reflected by the fact that mappings assertions are interpreted as "material implication" in logic. It is well known that a material implication of the form $\Phi \rightsquigarrow \Psi$ imposes that every tuple of $\Phi$ contributes to the answers to $\Psi$, leaving open the possibility of additional tuples satisfying $\Psi$.

Let $q$ denote a UCQ expressed over the TBox $\mathcal{T}$ of $\mathcal{OM}$. We call *certain answers to* $q$ *over* $\mathcal{OM}$, denoted $cert(q, \mathcal{OM})$, the set of $n$-tuples of terms in $\Gamma$, defined as

$$cert(q, \mathcal{OM}) = \{\boldsymbol{t} \mid \boldsymbol{t}^{\mathcal{I}} \in q^{\mathcal{I}}, \text{ for all } \mathcal{I} \in Mod(\mathcal{OM})\}.$$

Given an ontology with mappings and a query $q$ over its TBox, *query answering* is the problem of computing the certain answers to $q$.

### 7.3 Satisfiability and Query Answering for Ontologies with Mappings

Our goal is to illustrate a method for checking satisfiability and for query answering for *DL-Lite*$_{\mathcal{A},id}$ ontologies with mappings. We will give here just an overview of the method, concentrating on query answering, and refer to [75] for more details.

The simplest way to tackle reasoning over a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings is to use the mappings to produce an actual ABox, and then reason on the ontology constituted by the ABox and the original TBox by applying the techniques described in Sections 4 and 5. We call such an approach "bottom-up". However, the bottom-up approach requires to actually build the ABox starting from the data at the sources, thus somehow duplicating the information already present in the data sources. To avoid this redundancy, we propose an alternative approach, called "top-down", which essentially keeps the ABox virtual.

We sketch the main ideas of both approaches below. As said, we refer in particular to query answering, but similar considerations hold for satisfiability checking too. Before delving into the discussion, we define the notions of split version of an ontology and of virtual ABox, which will be useful in the sequel.

We first show how to compute the *split version* of an ontology with mappings $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, which has a particularly "friendly form". Specifically, we denote with $split(\mathcal{M})$ a new set of mapping assertions, obtained from $\mathcal{M}$ as follows:

$m^a_{11}$ : SELECT SSN, PROJ, D FROM $D_1$ $\rightsquigarrow$ *TempEmp*(**pers**(SSN))

$m^a_{12}$ : SELECT SSN, PROJ, D FROM $D_1$ $\rightsquigarrow$ *WORKS-FOR*(**pers**(SSN), **proj**(PROJ))

$m^a_{13}$ : SELECT SSN, PROJ, D FROM $D_1$ $\rightsquigarrow$ **projName**(**proj**(PROJ), PROJ)

$m^a_{14}$ : SELECT SSN, PROJ, D FROM $D_1$ $\rightsquigarrow$ **until**(**pers**(SSN), D)

$m^a_{21}$ : SELECT SSN, NAME FROM $D_2$ $\rightsquigarrow$ *Employee*(**pers**(SSN))

$m^a_{22}$ : SELECT SSN, NAME FROM $D_2$ $\rightsquigarrow$ **persName**(**pers**(SSN), NAME)

$m^a_{31}$ : SELECT SSN, NAME FROM $D_3$, $D_4$ WHERE $D_3$.CODE = $D_4$.CODE $\rightsquigarrow$ *Manager*(**pers**(SSN))

$m^a_{32}$ : SELECT SSN, NAME FROM $D_3$, $D_4$ WHERE $D_3$.CODE = $D_4$.CODE $\rightsquigarrow$ **persName**(**pers**(SSN), NAME)

$m^a_{41}$ : SELECT CODE, NAME FROM $D_3$ WHERE CODE NOT IN (SELECT CODE FROM $D_4$) $\rightsquigarrow$ *Manager*(**mgr**(CODE))

$m^a_{42}$ : SELECT CODE, NAME FROM $D_3$ WHERE CODE NOT IN (SELECT CODE FROM $D_4$) $\rightsquigarrow$ **persName**(**mgr**(CODE), NAME)

**Fig. 23.** The split version of the object-to-data mapping assertions $\mathcal{M}^a_{pr}$ for the projects example

*(1)* $split(\mathcal{M})$ contains all typing assertions in $\mathcal{M}$.
*(2)* $split(\mathcal{M})$ contains one mapping assertion $\Phi' \rightsquigarrow X$, for each mapping assertion $\Phi \rightsquigarrow \Psi \in \mathcal{M}$ and for each atom $X \in \Psi$, where $\Phi'$ is the projection of $\Phi$ over the variables occurring in $X$.

We denote with $split(\mathcal{OM})$ the ontology $\langle \mathcal{T}, split(\mathcal{M}), \mathcal{D} \rangle$.

*Example 7.5.* Consider the ontology with mappings $\mathcal{OM}_{pr} = \langle \mathcal{T}_{pr}, \mathcal{M}_{pr}, \mathcal{D}_{pr} \rangle$ of Example 7.2. By splitting the mappings as described above, we obtain the ontology $split(\mathcal{OM}_{pr}) = \langle \mathcal{T}_{pr}, split(\mathcal{M}_{pr}), \mathcal{D}_{pr} \rangle$, where $split(\mathcal{M}_{pr})$ contains all typing assertions in $\mathcal{M}_{pr}$ and the split mapping assertions shown in Figure 23. ∎

The relationship between an ontology with mappings and its split version is characterized by the following theorem.

**Proposition 7.6.** *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology with mappings. Then, we have that*

$$Mod(split(\mathcal{OM})) = Mod(\mathcal{OM}).$$

*Proof.* The result follows straightforwardly from the syntax and the semantics of the mappings. □

This result essentially tells us that every ontology with mappings is logically equivalent to the corresponding split version. Therefore, given an arbitrary *DL-Lite$_{\mathcal{A},id}$* ontology

with mappings, we can always reduce it to its split version. Moreover, such a reduction can be computed in LOGSPACE in the size of the mappings and does not depend on the size of the data. Therefore, in the following, we will to deal only with split versions of *DL-Lite*$_{\mathcal{A},id}$ ontologies with mappings.

In order to express the semantics of ontologies with mappings in terms of the semantics of conventional ontologies, we introduce now the notion of virtual ABox. Intuitively, given a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, the virtual ABox corresponding to $\mathcal{OM}$ is the ABox whose assertions are computed by "applying" the mapping assertions in $\mathcal{M}$ starting from the data in $\mathcal{D}$. Note that in our method this ABox is "virtual", in the sense that it is not explicitly built.

**Definition 7.7.** *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology with mappings, and let $m = \Phi(\boldsymbol{x}) \rightsquigarrow X(\boldsymbol{y}, \boldsymbol{t})$ be a (split) mapping assertion in $\mathcal{M}$. The* virtual ABox generated by $m$ from $\mathcal{D}$ *is the set of membership assertions*

$$\mathcal{A}(m, \mathcal{D}) = \{X[\boldsymbol{x}/\boldsymbol{v}] \mid \boldsymbol{v} \in ans(\Phi, \mathcal{D})\}.$$

*Moreover, the* virtual ABox for $\mathcal{OM}$*, denoted $\mathcal{A}(\mathcal{M}, \mathcal{D})$, is the set of membership assertions*

$$\mathcal{A}(\mathcal{M}, \mathcal{D}) = \bigcup_{m \in \mathcal{M}} \mathcal{A}(m, \mathcal{D}).$$

Notice that, in the above definition, $\boldsymbol{v}$ is an $n$-tuple of constants of $\Gamma_V$, where $n$ is the arity of $\Phi$, and $X[\boldsymbol{x}/\boldsymbol{v}]$ denotes the ground atom obtained from $X(\boldsymbol{x})$ by substituting the $n$-tuple of variables $\boldsymbol{x}$ with $\boldsymbol{v}$. Also, $\mathcal{A}(\mathcal{M}, \mathcal{D})$ is an ABox over the constants $\Gamma = \Gamma_V \cup \tau(\Lambda, \Gamma)$.

*Example 7.8.* Let $split(\mathcal{OM}_{pr})$ be the *DL-Lite*$_{\mathcal{A},id}$ ontology with split mappings of Example 7.5. Consider in particular the mappings $m^a_{21}$ and $m^a_{22}$ and suppose we have $D_2 = \{(\texttt{20903}, \texttt{"Rossi"}), (\texttt{55577}, \texttt{"White"})\}$ in the database $\mathcal{D}$. Then, the sets of assertions $\mathcal{A}(m^a_{21}, \mathcal{D})$ and $\mathcal{A}(m^a_{22}, \mathcal{D})$ are as follows:

$$\begin{aligned}
\mathcal{A}(m^a_{21}, \mathcal{D}) &= \{\ \textit{Employee}(\mathbf{pers}(\texttt{20903})),\ \textit{Employee}(\mathbf{pers}(\texttt{55577}))\ \} \\
\mathcal{A}(m^a_{22}, \mathcal{D}) &= \{\ \mathbf{persName}(\mathbf{pers}(\texttt{20903}), \texttt{"Rossi"}), \\
&\qquad\quad \mathbf{persName}(\mathbf{pers}(\texttt{55577}), \texttt{"White"})\ \}
\end{aligned}$$

By proceeding in the same way for each mapping assertion in $split(\mathcal{M}_{pr})$, we easily obtain the whole virtual ABox $\mathcal{A}(\mathcal{M}_{pr}, \mathcal{D}_{pr})$ for $split(\mathcal{OM}_{pr})$, and hence for $\mathcal{OM}_{pr}$. ∎

The following result, which follows easily from the definitions, establishes the relationship between the semantics of *DL-Lite*$_{\mathcal{A},id}$ ontologies with mappings and the semantics of *DL-Lite*$_{\mathcal{A},id}$ ontologies by resorting to virtual ABoxes:

**Proposition 7.9.** *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ ontology with mappings. Then we have that*

$$Mod(\mathcal{OM}) = Mod(\langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle).$$

Notice that, for convenience, we have defined $\mathcal{A}(\mathcal{M}, \mathcal{D})$ for the case where the mappings in $\mathcal{M}$ are split. However, from Proposition 7.6, we also obtain that, for an ontology $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ with non-split mapping assertions, we have that

$$Mod(\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle) = Mod(\langle \mathcal{T}, split(\mathcal{M}), \mathcal{D} \rangle) = Mod(\langle \mathcal{T}, \mathcal{A}(split(\mathcal{M}), \mathcal{D}) \rangle).$$

### 7.4 Approaches for Query Answering over Ontologies with Mappings

We discuss now in more detail both the bottom-up and the top-down approach for query answering. Proposition 7.9 above suggests an obvious, and "naive", *bottom-up algorithm* to answer queries over a satisfiable *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ with mappings:

1. Materialize the virtual ABox for $\mathcal{OM}$, i.e., compute $\mathcal{A}(\mathcal{M}, \mathcal{D})$.
2. Apply to the *DL-Lite*$_{\mathcal{A},id}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}(\mathcal{M}, \mathcal{D}) \rangle$, the query answering algorithm described in Section 5.

Unfortunately, this approach has the drawback that the resulting algorithm is not anymore $AC^0$ (or even LOGSPACE) in the size of the database, since it requires the generation and storage of the whole virtual ABox, which in general is polynomial in the size of the database. Moreover, since the database is independent of the ontology, it may happen that, during the lifetime of the ontology with mappings, the data it contains are modified. This would clearly require to set up a mechanism for keeping the virtual ABox up-to-date with respect to the database evolution, similarly to what happens in data warehousing. This is the reason why such a bottom-up approach is only of theoretical interest, but not efficiently realizable in practice.

Hence, we propose a different approach, called "top-down", which uses an algorithm that avoids materializing the virtual ABox, but, rather, takes into account the mapping specification *on-the-fly*, during reasoning. In this way, we can both keep the computational complexity of the algorithm low, which turns out to be as the one of the query answering algorithm for ontologies without mappings (i.e., in $AC^0$), and avoid any further procedure for data refreshment. We present an overview of our top-down approach to query answering.

Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ be a *DL-Lite*$_{\mathcal{A},id}$ ontology with split mappings, and let $q$ be a UCQ [16] over $\mathcal{OM}$. According to the top-down approach, the certain answers to $q$ over $\mathcal{OM}$ are computed by performing the following steps:

1. **Reformulation.** In this step, we compute the perfect reformulation $q_1 = \mathsf{PerfectRefIdC}(q, \mathcal{T})$ of $q$, according to the technique presented in Section 5. The query $q_1$ is a UCQ satisfying the following property: the certain answers to $q$ with

[16] Notice that, although we do not consider query answering for UCQs with inequalities, in general we would need to consider also the case where $q$ may contain inequalities. Indeed, such inequalities may result from the queries that encode the violation of a functionality or an identification assertion, and whose evaluation is required to check the satisfiability of $\mathcal{OM}$, cf. Section 5.6. For simplicity we do not consider inequalities here, but they can be dealt with by replacing them with a suitable predicate, which in the end gets translated into an SQL inequality check, see [75] for more details.

respect to $\mathcal{OM}$ coincide with the set of tuples computed by evaluating $q_1$ over $DB(\mathcal{A}(\mathcal{M}, \mathcal{D}))$[17], i.e., the database representing $\mathcal{A}(\mathcal{M}, \mathcal{D})$.

2. **Filtering.** In this step we take care of a particular problem that the CQs in $q_1$ might have. Specifically, such a CQ is called *ill-typed* if it has at least one join variable $x$ appearing in two incompatible positions in the query, i.e., such that the TBox $\mathcal{T}$ of the ontology logically implies that $x$ is both of type $T_i$, and of type $T_j$, with $i \neq j$ (we remind that in *DL-Lite*$_{\mathcal{A},id}$, data types are pairwise disjoint). The purpose of the filtering step is to remove from the query $q_1$ all the ill-typed CQs. Intuitively, such a step is needed because the query $q_1$ has to be unfolded and then evaluated over the source database $\mathcal{D}$ (cf. the next two steps of the algorithm). These last two steps, performed for an ill-typed CQ might produce incorrect results. Let $q_2$ be the UCQ produced as result of this step.

3. **Unfolding.** Instead of materializing $\mathcal{A}(\mathcal{M}, \mathcal{D})$ and evaluating $q_2$ over $DB(\mathcal{A}(\mathcal{M}, \mathcal{D}))$ (as in the bottom-up approach), we "unfold" $q_2$ according to $\mathcal{M}$, i.e., we compute a new query $q_3$, which is an SQL query over the source relations. As shown in detail in [75], and illustrated briefly below, this computation is done by using logic programming techniques. It allows us to get rid of $\mathcal{M}$, in the sense that the set of tuples computed by evaluating the SQL query $q_3$ over the database $\mathcal{D}$ coincides with the set of tuples computed by evaluating $q_2$ over $DB(\mathcal{A}(\mathcal{M}, \mathcal{D}))$.

4. **Evaluation.** The evaluation step consists simply in delegating the evaluation of the SQL query $q_3$, produced by the unfolding step, over the database $\mathcal{D}$ to the DBMS managing such a database. Formally, such an evaluation returns $ans(q_3, \mathcal{D})$, i.e., the set of tuples obtained from the evaluation of $q_3$ over $\mathcal{D}$.

The unfolding step for $q_2$ can be carried out as follows:

(3a) We introduce for each non-split mapping assertion $m_i = \Phi_i(\boldsymbol{x}) \rightsquigarrow \Psi_i(\boldsymbol{y}, \boldsymbol{t})$ in $\mathcal{M}$ an auxiliary predicate $Aux_i$ of the same arity as $\Phi_i$. Intuitively, $Aux_i$ denotes the result of the evaluation over $\mathcal{D}$ of the SQL query $\Phi_i$ in the left-hand side of the mapping.

(3b) We introduce for each atom $X(\boldsymbol{y}, \boldsymbol{t})$ in $\Psi_i(\boldsymbol{y}, \boldsymbol{t})$, a logic programming clause

$$X(\boldsymbol{y}, \boldsymbol{t}) \leftarrow Aux_i(\boldsymbol{x}).$$

Notice that, in general, the atom $X(\boldsymbol{y}, \boldsymbol{t})$ in the mapping will contain not only variables but also variable terms, and hence such a clause will contain function symbols in its head.

(3c) From each CQ $q'$ in $q_2$, we obtain a set of CQs expressed over the $Aux_i$ predicates by *(i)* finding, in all possible ways, the most general unifier $\vartheta$ between all atoms in $q'$ and the heads $X(\boldsymbol{y}, \boldsymbol{t})$ of the clauses introduced in the previous step, *(ii)* replacing in $q'$ each head of a clause with the corresponding body, and *(iii)* applying to the resulting CQ the most general unifier $\vartheta$.

[17] The function $DB(\cdot)$ is defined in Section 2.6.

(3d) From the resulting UCQ over the $Aux_i$ predicates, we obtain an SQL query that is a union of select-project-join queries, by substituting each $Aux_i$ predicate with the corresponding SQL query $\Phi_i$.

We refer to [75] for more details, and illustrate the steps above by means of an example.

*Example 7.10.* Consider the ontology $\mathcal{OM}_{pr}$ of Example 7.2, and assume it is satisfiable. The mapping assertions in $\mathcal{M}_{pr}$ of $\mathcal{OM}_{pr}$ can be encoded in the following portion of a logic program, where for each mapping assertion $m_i^a$ in Figure 22, we have introduced an auxiliary predicate $Aux_i$:

$$\begin{aligned}
\textit{TempEmp}(\mathbf{pers}(s)) &\leftarrow Aux_1(s,p,d)\\
\textit{WORKS-FOR}(\mathbf{pers}(s),\mathbf{proj}(p)) &\leftarrow Aux_1(s,p,d)\\
\mathbf{projName}(\mathbf{proj}(p),p) &\leftarrow Aux_1(s,p,d)\\
\mathbf{until}(\mathbf{pers}(s),d) &\leftarrow Aux_1(s,p,d)\\
\textit{Employee}(\mathbf{pers}(s)) &\leftarrow Aux_2(s,n)\\
\mathbf{persName}(\mathbf{pers}(s),n) &\leftarrow Aux_2(s,n)\\
\textit{Manager}(\mathbf{pers}(s)) &\leftarrow Aux_3(s,n)\\
\mathbf{persName}(\mathbf{pers}(s),n) &\leftarrow Aux_3(s,n)\\
\textit{Manager}(\mathbf{mgr}(c)) &\leftarrow Aux_4(c,n)\\
\mathbf{persName}(\mathbf{mgr}(c),n) &\leftarrow Aux_4(c,n)
\end{aligned}$$

Now, consider the query over $\mathcal{OM}$

$$q(x,n) \leftarrow \textit{WORKS-FOR}(x,y),\ \mathbf{persName}(x,n).$$

Its reformulation $q_1 = \mathsf{PerfectRef}(q,\mathcal{T})$, computed according to the technique presented in Section 5.2, is the UCQ

$$\begin{aligned}
q_1(x,n) &\leftarrow \textit{WORKS-FOR}(x,y),\ \mathbf{persName}(x,n)\\
q_1(x,n) &\leftarrow \mathbf{until}(x,y),\ \mathbf{persName}(x,n)\\
q_1(x,n) &\leftarrow \textit{TempEmp}(x),\ \mathbf{persName}(x,n)\\
q_1(x,n) &\leftarrow \textit{Employee}(x),\ \mathbf{persName}(x,n)\\
q_1(x,n) &\leftarrow \textit{Manager}(x),\ \mathbf{persName}(x,n)
\end{aligned}$$

One can verify that in this case none of the CQs of $q_1$ will be removed by the filtering step, hence $q_2 = q_1$. In order to compute the unfolding of $q_2$, we unify each of its atoms in all possible ways with the left-hand side of the mapping assertions in $\mathit{split}(\mathcal{M}_{pr})$, i.e., with the heads of the clauses introduced in Step (3b) above, and we obtain the following UCQ $q_2'$:

$$\begin{aligned}
q_2'(\mathbf{pers}(s),n) &\leftarrow Aux_1(s,p,d),\ Aux_2(s,n)\\
q_2'(\mathbf{pers}(s),n) &\leftarrow Aux_1(s,p,d),\ Aux_3(s,n)\\
q_2'(\mathbf{pers}(s),n) &\leftarrow Aux_2(s,n),\ Aux_2(s,n)\\
q_2'(\mathbf{pers}(s),n) &\leftarrow Aux_2(s,n),\ Aux_3(s,n)\\
q_2'(\mathbf{pers}(s),n) &\leftarrow Aux_3(s,n),\ Aux_2(s,n)\\
q_2'(\mathbf{pers}(s),n) &\leftarrow Aux_3(s,n),\ Aux_3(s,n)\\
q_2'(\mathbf{mgr}(c),n) &\leftarrow Aux_4(c,n),\ Aux_4(c,n)
\end{aligned}$$

```
SELECT CONCAT(CONCAT('pers (',D1.SSN),')'), D2.NAME
FROM D1, D2
WHERE D1.SSN = D2.SSN
  UNION
SELECT CONCAT(CONCAT('pers (',D1.SSN),')'), D3.NAME
FROM D1, D3, D4
WHERE D1.SSN = D4.SSN AND D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('pers (',D2.SSN),')'), D2.NAME
FROM D2
  UNION
SELECT CONCAT(CONCAT('pers (',D2.SSN),')'), D3.NAME
FROM D2, D3, D4
WHERE D2.SSN = D4.SSN AND D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('pers (',D2.SSN),')'), D3.NAME
FROM D2, D3, D4
WHERE D2.SSN = D4.SSN AND D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('pers (',D3.SSN),')'), D3.NAME
FROM D3, D4
WHERE D3.CODE = D4.CODE
  UNION
SELECT CONCAT(CONCAT('mgr (',D3.CODE),')'), D3.NAME
FROM D3
WHERE D3.CODE NOT IN (SELECT D4.CODE FROM D4)
```

**Fig. 24.** The SQL query for the projects example produced by the query answering algorithm

Notice that each of the clauses in $q'_2$ is actually generated in many different ways from $q_1$ and the clauses above.

From $q'_2$, it is now possible to derive the SQL query $q_3$ shown in Figure 24, where in the derivation we have assumed that duplicate atoms in a clause are eliminated. The SQL query $q_3$ can be directly issued over the database $\mathcal{D}$ to produce the requested certain answers. □

It is also possible to show that the above procedure can also be used to check satisfiability of an ontology with mappings, by computing the answer to the boolean query that encodes the violation of the constraints in the TBox, and checking whether such an answer is empty.

Let the algorithms for satisfiability and query answering over a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings resulting from the above described method be called $\mathsf{SatisfiableDB}(\mathcal{OM})$ and $\mathsf{AnswerDB}(q, \mathcal{OM})$, respectively. A complexity analysis of the various steps of these algorithms, allows us to establish the following result, for whose proof we refer to [75].

**Theorem 7.11.** *Given a DL-Lite*$_{\mathcal{A},id}$ *ontology with mappings* $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$*, and a UCQ* $q$ *over* $\mathcal{OM}$*, both* $\mathsf{SatisfiableDB}(\mathcal{OM})$ *and* $\mathsf{AnswerDB}(q, \mathcal{OM})$ *run in* $\mathrm{AC}^0$ *in the size of* $\mathcal{D}$ *(data complexity), in polynomial time in the size of* $\mathcal{M}$*, and in polynomial time in the size of* $\mathcal{T}$*. Moreover,* $\mathsf{AnswerDB}(q, \mathcal{OM})$ *runs in exponential time in the size of* $Q$*.*

### 7.5 Extending the Mapping Formalism

We investigate now the impact of extending the language used to express the mapping on the computational complexity of query answering. In particular, we consider so-called GLAV mappings [63], i.e., assertions that relate CQs over the database to CQs over the ontology. Such assertions are therefore an extension of both the GAV mappings considered above, and of LAV mappings typical of the data integration setting. Unfortunately, even with LAV mappings only, i.e., mappings where the query over the database simply returns the instances of a single relation, instance checking and query answering are no more in $\text{AC}^0$ with respect to data complexity [20].

**Theorem 7.12.** *Let $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ be an ontology with mappings, where the mapping $\mathcal{M}$ is constituted by a set of LAV mapping assertions. Instance checking (and hence CQ and UCQ query answering) over $\mathcal{OM}$ is* NLOGSPACE*-hard in the size of $\mathcal{D}$.*

*Proof.* The proof is again by a LOGSPACE reduction from reachability in directed graphs. Let $G = \langle V, E \rangle$ be a directed graph, where $V$ is a set of vertexes and $E$ a set of directed edges, and let $s$ and $t$ be two vertexes of $G$. As in the proof of Theorem 6.6, we consider the graph represented through first-child and next-sibling functional relations $F$, $N$, $S$ (cf. Figure 17).

We define the ontology with LAV mappings $\mathcal{OM}_{lav} = \langle \mathcal{T}_{lav}, \mathcal{M}_{lav}, \mathcal{D}_{lav} \rangle$ as follows:

- The alphabet of $\mathcal{T}_{lav}$ consists of the atomic concepts $A$ and $A'$ and of the atomic roles $P_F$, $P_N$, $P_S$, $P_0$, and $P_{copy}$. The TBox itself imposes only that all roles are functional, i.e.,

$$\mathcal{T}_{lav} \;=\; \{(\mathsf{funct}\ P_0), (\mathsf{funct}\ P_{copy})\} \cup \{(\mathsf{funct}\ P_{\mathcal{R}}) \mid \mathcal{R} \in \{F, N, S\}\}.$$

- The schema of $\mathcal{D}$ contains a unary relational table $A_d$ and three binary relational tables $F_d$, $N_d$, and $S_d$.
- The LAV mapping $\mathcal{M}_{lav}$ is defined as follows:[18]

$$\begin{array}{rcl} A_d(x) & \rightsquigarrow & q_A(x) \leftarrow A(x), P_{copy}(x, x'), P_0(z, x), P_0(z, x') \\ \mathcal{R}_d(x, y) & \rightsquigarrow & q_{\mathcal{R}}(x, y) \leftarrow P_{\mathcal{R}}(x, y), P_{copy}(x, x'), \\ & & \qquad P_{\mathcal{R}}(x', y'), P_{copy}(y, y'), A'(y'), \quad \text{for } \mathcal{R} \in \{F, N, S\}. \end{array}$$

Figure 25 provides a graphical representation of the kinds of interpretations generated by the LAV mapping above. Notice that the TBox $\mathcal{T}_{lav}$ and the mapping $\mathcal{M}_{lav}$ do not depend on the graph $G$.

Then, from the graph $G$ and the two vertexes $s$, $t$, we define the instance $D_G$ of the database $\mathcal{D}_{lav}$ as follows:

$$D_G \;=\; \{\mathcal{R}_d(a, b) \mid (a, b) \in \mathcal{R}, \text{ for } \mathcal{R} \in \{F, N, S\}\} \cup \{A_d(s)\}$$

[18] For simplicity, we do not include function symbols in the mapping since they would play no role in the reduction. Also, instead of using SQL code, we denote the query over the database $\mathcal{D}$ simply by the relation that is returned.

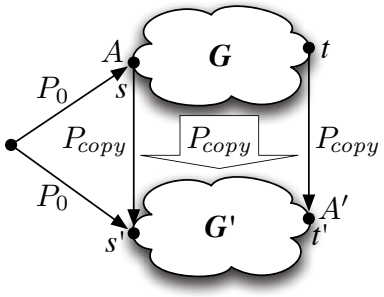



**Fig. 25.** Interpretation generated by the LAV mapping used in the proof of Theorem 7.12

Intuitively, $D_G$ is simply constituted by the binary relations $F_d$, $N_d$, and $S_d$, used to represent the graph $G$, and a unary relation $A_d$ containing only $s$.

Now consider the concept $A'$. It is possible to show by induction on the length of a path from $s$ to $t$ in $G$ that $t$ is reachable from $s$ in $G$ if and only if $\mathcal{OM}_{lav} \models A'(t)$, i.e., $t$ is an instance of $A'$ in every model of $\mathcal{OM}_{lav}$. Intuitively, this is due to the fact that the functionality of the roles of $\mathcal{T}_{lav}$ forces the objects corresponding to the nodes of $G$ and retrieved through the mapping to be unified with their "copies" generated by the existentially quantified variables in the mapping. Hence, the node $t$ will be forced to become an instance of $A'$ if it is connected to $s$, but not otherwise. □

The above result shows that, if we allowed more general forms of mappings than the ones considered here for ontologies with mappings, such as LAV mappings, we would lose FOL-rewritability of inference.

Notice that for the above proof to go through, the presence of functionality assertions is crucial. Indeed, it is possible to show that, without functionality assertions (and without identification assertions), query answering even in the presence of GLAV mappings can be done in $AC^0$ in data complexity, essentially by transforming the GLAV mapping into GAV mappings and introducing additional constraints (and relations of arbitrary arity) in the TBox [16].

## 8 Ontology-Based Data Access Software Tools

In this section we introduce two tools specifically designed for OBDA as described in the previous sections, namely DIG-QUONTO and the OBDA Plugin for Protégé 3.3.1. The first, presented in Section 8.1, is a server for the QUONTO reasoner [2,76] that exposes it's reasoning services and OBDA functionality through an extended version of the DIG Interface [10], a standard communication protocol for DL reasoners. The second, presented in Section 8.2, is a plugin for the ontology editor Protégé[19] that provides facilities to model *ontologies with mappings* (see Section 7), to *synchronize* these models with an OBDA enabled reasoner through an extended DIG protocol, and to access CQ services offered by DIG 1.2 compatible reasoners [76].

[19] `http://protege.stanford.edu/`

Both tools can be used together in order to design, deploy and use a fully functional OBDA layer on top of existing relational databases.

### 8.1 DIG-QUONTO, the OBDA-DIG Server for QUONTO

DIG-QUONTO [76] is a module for the QUONTO system that exposes the functionality of the QUONTO reasoner and its *RDBMS-ontology mapping* module through an extended version of the DIG 1.1 Interface [10], the HTTP/XML based communication protocol for DL reasoners. Using DIG-QUONTO, it is possible to extend an existing relational database with an OBDA layer, which can be used to cope with incomplete information or function as a data integration layer.

The QUONTO reasoner is a DL reasoner that implements the reasoning and query answering algorithms for *DL-Lite*$_{\mathcal{A},id}$ presented in Sections 4 and 5. Built on top of the QUONTO reasoner is its *RDBMS-ontology mapping* module, a module that implements the mapping techniques described in Section 7. DIG-QUONTO wraps both components, thus combining their functionalities in a common interface accessible to clients and providing several features that we will now describe.

**Ontology Representation.** QUONTO and DIG-QUONTO work with *ontologies with mappings* of the form $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ as presented in Section 7, where $\mathcal{T}$ is a *DL-Lite*$_{\mathcal{A},id}$ TBox, $\mathcal{D}$ is a relational database (constituted by the relational schema, and the extensions of the relations), and $\mathcal{M}$ is a set of mapping assertions. As mentioned, $\mathcal{D}$ and $\mathcal{M}$ together define an ABox $\mathcal{A}(\mathcal{M}, \mathcal{D})$, which however is never materialized by QUONTO. Instead, using the *RDBMS-ontology mapping* module, QUONTO is able to rewrite an UCQ $q$ over $\mathcal{T}$ into an SQL query that is executed by the RDBMS managing $\mathcal{D}$ and that retrieves the answers to $q$. Hence, QUONTO is able to exploit many optimizations available in modern RDBMS engines in all operations related to the extensional level of the ontology, and as a consequence, it is able to handle large amounts of data. A further consequence is that data has never to be imported into the ontology, as is done, e.g., with OWL ontologies. Instead, in DIG-QUONTO it is possible to let the data reside in the original relational source. Note that this has the consequence that if the data source is not a traditional database, but in fact a virtual schema created by a data federation tool, DIG-QUONTO would act as a highly efficient data integration system.

For a detailed description of the reasoning process used in DIG-QUONTO see Sections 4, 5 and 7.

**Reasoning Services.** QUONTO provides the following reasoning services over a *DL-Lite*$_{\mathcal{A},id}$ ontology with mappings $\mathcal{OM} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$:

*Answering UCQs.* Given $\mathcal{OM}$ and a UCQ $q$, compute the certain answers for $q$ over $\mathcal{OM}$. This is the central reasoning service of the QUONTO system, to which all other reasoning services are reduced. This service is unique in QUONTO in that, at the moment of writing, QUONTO is the only DL reasoner that offers UCQ query answering under the *standard certain answer* semantics (as opposed to weaker semantics, such as the *grounded semantics*, implemented in other reasoners). This is especially important in settings of incomplete information (e.g., the data integration

setting), as it allows QUONTO to bind variables in the body of queries to *unknown* individuals which are only deduced to exist due to $\mathcal{T}$ but which are not produced by the data in $\mathcal{D}$ (considering the mappings $\mathcal{M}$).

The implementation of this service is based on the algorithms described in Sections 4, 5, and 7.

*Checking ontology satisfiability.* Given $\mathcal{OM}$, check if $\mathcal{OM}$ has a model. This is a classical reasoning service available in most DL reasoners, which mixes intensional and extensional reasoning. In QUONTO, this service is reduced to query answering. Specifically, to check satisfiability of $\mathcal{OM}$, QUONTO checks whether the answer to the query that checks for a violation of one of the negative inclusion, functionality, or identification assertions in the TBox is empty, as illustrated in Section 5.6. Hence, it can be done efficiently in the size of the data, which is maintained in the database and managed through a DBMS.

*Checking atomic satisfiability.* Given $\mathcal{OM}$ and an atomic entity $X$ (i.e., a concept, a role, or an attribute) in the alphabet of $\mathcal{OM}$, check whether there exists a model $\mathcal{I}$ of $\mathcal{OM}$ such that $X^{\mathcal{I}} \neq \emptyset$. This is a purely intensional reasoning service, i.e., it actually depends only on the TBox $\mathcal{T}$ of $\mathcal{OM}$. In QUONTO, this service is reduced to checking ontology satisfiability, as illustrated in Section 4.4, and hence ultimately to query answering. In fact, since satisfiability of the entity $X$ depends only on the TBox of $\mathcal{OM}$ (and not on the database and mappings), QUONTO constructs for the purpose an ad-hoc database and mappings, that are distinct from the ones of $\mathcal{OM}$, and answers the appropriate queries over such a database.

*Checking subsumption.* Given $\mathcal{OM}$ and two atomic entities $X_1$ and $X_2$ in the alphabet of $\mathcal{OM}$, check whether $X_1^{\mathcal{I}} \subseteq X_2^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{OM}$. In QUONTO, also subsumption is reduced to ontology satisfiability, as illustrated in Section 4.4.

*Queries regarding the concept/role hierarchy.* Given $\mathcal{OM}$, QUONTO provides the following intensional reasoning services regarding the structure of the concept and role hierarchy in $\mathcal{T}$:

- Ancestors. Given an atomic entity $X$ in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities $X'$ such that $\mathcal{T} \models X \sqsubseteq X'$.
- Parents. Given an atomic entity $X$ in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities $X'$ such that $\mathcal{T} \models X \sqsubseteq X'$ and there is no $X''$ such that $\mathcal{T} \models X \sqsubseteq X''$ and $\mathcal{T} \models X'' \sqsubseteq X'$. This corresponds to the immediate subsumption relation.
- Descendants. Given an atomic entity $X$ in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities $X'$ such that $\mathcal{T} \models X' \sqsubseteq X$.
- Children. Given an atomic entity $X$ in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities $X'$ such that $\mathcal{T} \models X' \sqsubseteq X$ and there is no $X''$ such that $\mathcal{T} \models X' \sqsubseteq X''$ and $\mathcal{T} \models X'' \sqsubseteq X$.
- Equivalents. Given an atomic entity $X$ in the alphabet of $\mathcal{OM}$, retrieve the set of atomic entities $X$ such that $\mathcal{T} \models X \sqsubseteq X'$ and $\mathcal{T} \models X' \sqsubseteq X$.

In QUONTO, such services are also ultimately reduced to query answering, although for efficiency reasons this is done in an ad-hoc manner.

*Checking implication of assertions.* Given $\mathcal{OM}$ and a functionality or identification assertion $\alpha$, check whether $\mathcal{T} \models \alpha$. We point out that implication of identification assertions is unique to QUONTO, given the fact that these kinds of constraints are not available in most DL reasoners.

**The OBDA-DIG Communication Layer.** All functionalities of DIG-QUONTO are accessible through an extended version of the DIG 1.1 Interface [10], which is an effort carried out by the DL Implementation Group (DIG) to standardize interaction with DL reasoners in a networked environment. The original specification defines the communication mechanism to which DL reasoners (i.e., DIG servers) and clients comply. The interface has been widely accepted and most well known ontology design tools and DL reasoners implement it. XML messages, divided in `tells` and `asks`, allow the client to: *(i)* query for the server's reasoning capabilities, *(ii)* transfer the assertions of an ontology to the server, *(iii)* perform certain ontology manipulations, and *(iv)* ask standard DL queries about the given ontology, e.g., concept subsumption, satisfiability, equivalence, etc. The concept language used to describe DIG ontologies is based on the $\mathcal{SHOIQ}(D)$ description logic [53].

As mentioned above, DIG-QUONTO offers ontology constructs and services not available in traditional DL reasoners and therefore not considered in DIG 1.1. Hence, it has been necessary to implement not only the original DIG 1.1 interface, but also extensions that enable the use of the functionality available in DIG-QUONTO.

Being QUONTO's main reasoning task answering UCQs, the DIG-QUONTO server implements the so called DIG 1.2 specification [77], an extension to the original DIG 1.1 interface that provides an ABox query language for DIG clients and server. Concretely, it provides the ability to pose UCQs. It doesn't restrict the semantics for the expressed queries, hence in DIG-QUONTO we use *traditional certain answer semantics*, as described in Section 2.4. At the moment of writing, DIG 1.2 is implemented in the DIG modules for the RacerPro[20] and QUONTO reasoners.

The core component of the protocol implemented in the DIG-QUONTO server are the OBDA extensions to DIG 1.1 [80,81]. These extensions have as main objective to augment DIG 1.1 with the concepts of *Data Source* and *Mapping*, which are at the core of the OBDA setting and fundamental to the functionality offered by the *rdbms-ontology mapping* module for QUONTO. Moreover, the extension aims at the standardization of the interaction with reasoners offering OBDA functionality, not only with QUONTO. For more information about the OBDA extension to DIG 1.1 we refer to the extension's website[21].

DIG-QUONTO can be executed as a user service or as a system wide service. Once initiated, DIG-QUONTO listens for DIG-OBDA requests issued by clients. A number of parameters are available at initialization time. These allow the user to indicate whether DIG-QUONTO should perform automatic consistency checking at query time, and whether it should use *view based* unfolding procedures, among other things.

[20] `http://www.racer-systems.com/`

[21] `http://obda.inf.unibz.it/dig-11-obda/`

The status and operation of DIG-QUONTO can be monitored with its web interface, which is available at:

```
http://[QUONTOHOST]:[QUONTOPORT]/index.jsp
```

Through the interface, users can obtain information such as the ontologies currently loaded into the system, review system logs, or visualize system parameters.

Regarding implementation details, we note that DIG-QUONTO is written in Java and requires Sun's Java Runtime Environment (JRE)[22]. Moreover, DIG-QUONTO uses Java JDBC connectors to establish communication with a DBMS. The following DBMSs are supported by DIG-QUONTO at the moment of writing: MySQL 5.0.45[23], PostgreSQL 8.3, Oracle 10g and 11g , DB2 8.2 and 9.1, SQLite 3.5.9, H2 1.0.74, and Derby 10.3.2.1. We refer to QUONTO's main website[24] for detailed information about the software and for download links.

### 8.2 The OBDA Plugin for Protégé

The OBDA Plugin [81,82] is an open source add-on for the ontology editor Protégé 3.3.1 (see Footnote 19) whose main objectives are, on the one hand, to extend Protégé so as to allow its use in the design of *ontologies with mappings* and, on the other hand, to extend the way in which Protégé interacts with DL reasoners so as to support the interaction with reasoners that are designed for the OBDA architecture and that can make use of the OBDA information introduced using the plugin's facilities. These objectives are accomplished by extending the GUI, back-end mechanisms, and communication protocols of Protégé. In the following paragraphs, we will briefly describe these extensions, referring to the plugin's website[25] for a comprehensive overview of all the features of the OBDA Plugin.

**Database and Mapping Definition.** The main GUI component of the OBDA Plugin is the *Datasource Manager* tab (see Figure 26). Using this tab, users are able to associate JDBC data sources to the currently open ontology. Moreover, for each data source, users are able to define a set of mappings that relate the data returned by an arbitrary SQL query over the source to the entities (i.e., classes and properties) of the ontology.

The plugin offers several features to facilitate these tasks. Among them we can find simple ones, such as syntax coloring for the mapping editor or database connection validation, and more complex ones, such as SQL query validation, direct DBMS query facilities (see Figure 27), and database schema inspection facilities (see Figure 28).

All data introduced with the plugin's GUI components is made persistent in so called `.obda` files. These are XML files that can be read and edited easily by users or applications and which are independent from Protégé's `.owl` and `.ppjr` files. This feature enables users to easily extend existing OWL ontologies with OBDA features without compromising the original model.

[22] `http://java.sun.com/javase/`

[23] The use of MySQL is highly discouraged due to performance limitations of this engine.

[24] `http://www.dis.uniroma1.it/~quonto/`

[25] `http://obda.inf.unibz.it/protege-plugin/`

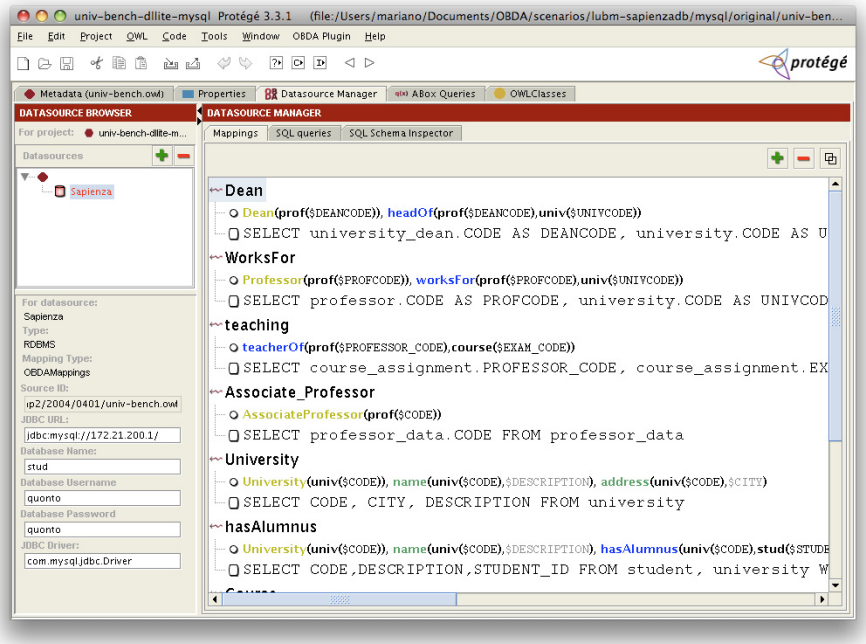



**Fig. 26.** OBDA Plugin's Datasource Manager tab

**OBDA-DIG Synchronization for DL Reasoners.** The mechanism used in Protégé 3.3.1 for interaction with DL reasoners is the DIG 1.1 Interface [10]. In order to allow for the interaction with reasoners designed for the OBDA setting, the OBDA Plugin extends Protégé's DIG implementation with the *OBDA Extensions for DIG* [80,81]. Using these extensions, the plugin is able to transmit the OBDA model created by the user to any reasoner implementing both specifications (see Section 8.1 for a short overview of these protocols).

Moreover, the plugin offers several possibilities to tweak the way in which synchronization takes place. For example, to interact with a traditional DL reasoner while the OBDA Plugin is installed, it is possible to configure the OBDA Plugin so as to use the original DIG 1.1 synchronization mechanism instead of the extended one.

**UCQs with the OBDA Plugin.** Another key feature of the OBDA Plugin is its ability to interact with reasoners that offer the service of answering (U)CQs. Using the *ABox Queries* tab of the OBDA Plugin (see Figure 29), users are able to express UCQs written in a SPARQL-like syntax. To provide the answers to the query, the OBDA Plugin translates the UCQ to a DIG 1.2 [77] request, which is sent to the reasoner. Clearly, the target reasoner must provide a UCQ answering service through a DIG 1.2 interface. At the moment of writing there exist two such systems, namely the DIG-QuOnto server and the Racer++ system.

In addition to the basic query handling facilities, the *ABox Queries* tab offers extra functionality such as persistent support for query *collections*, batch mode result retrieval and result set export facilities.

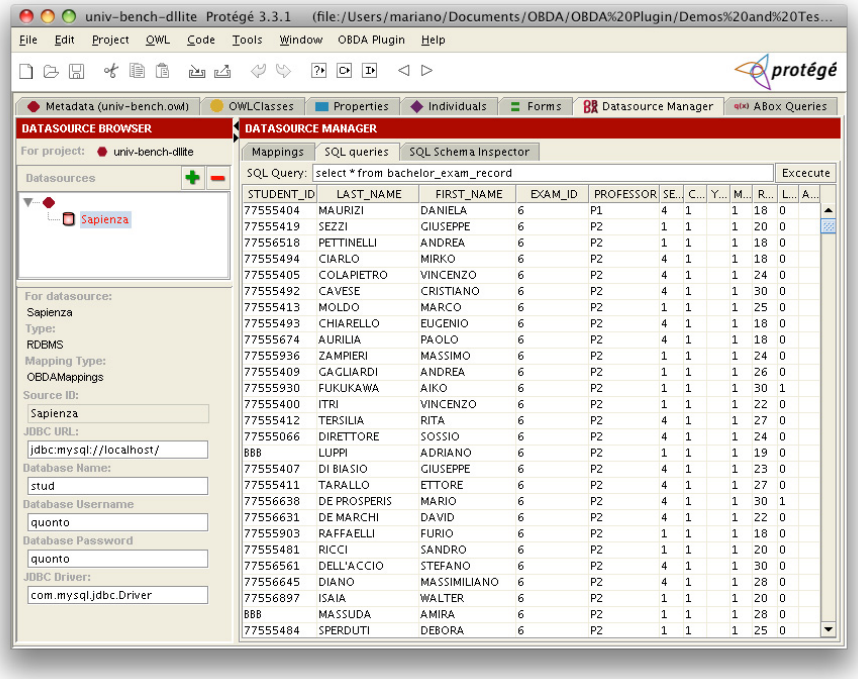



**Fig. 27.** OBDA Plugin's direct SQL Query tab

**DIG-QUONTO Specific Features.** Since the OBDA Plugin is being developed in parallel with DIG-QUONTO, the plugin incorporates several DIG-QUONTO specific facilities. These include the ability to enable and disable reasoning during query answering, to retrieve the expansion/rewriting and unfolding of a UCQ, to visualize the QUONTO TBoxes, and to request ontology satisfiability checking.

**Extensible OBDA API.** An important feature of the OBDA Plugin is its modular architecture. The OBDA Plugin for Protégé 3.3.1 has been built on top of a Java API for OBDA. The API is independent from the final GUI (e.g., Protégé) and, more importantly, independent from particular characterizations of the OBDA architecture. This enables the API to accommodate for different mapping techniques or data source types, having as only requirement that the mapping technique regards a mappings as a pair composed by a query $\Phi$ over a data source and a query $\Psi$ over the ontology. Note that here, the term *query* is used in the most general sense of the word, as a query can stand for any arbitrary computation.

We observe that the GUI independence aspect of the core API has been used to provide a port of the OBDA Plugin for the NeOn Toolkit platform[26] and to build the initial prototypes of the Protégé 4 and Protégé 3.4 versions of the plugin. The core API of the OBDA Plugin will soon be released with an open source license.

[26] `http://obda.inf.unibz.it/neon-plugin/`

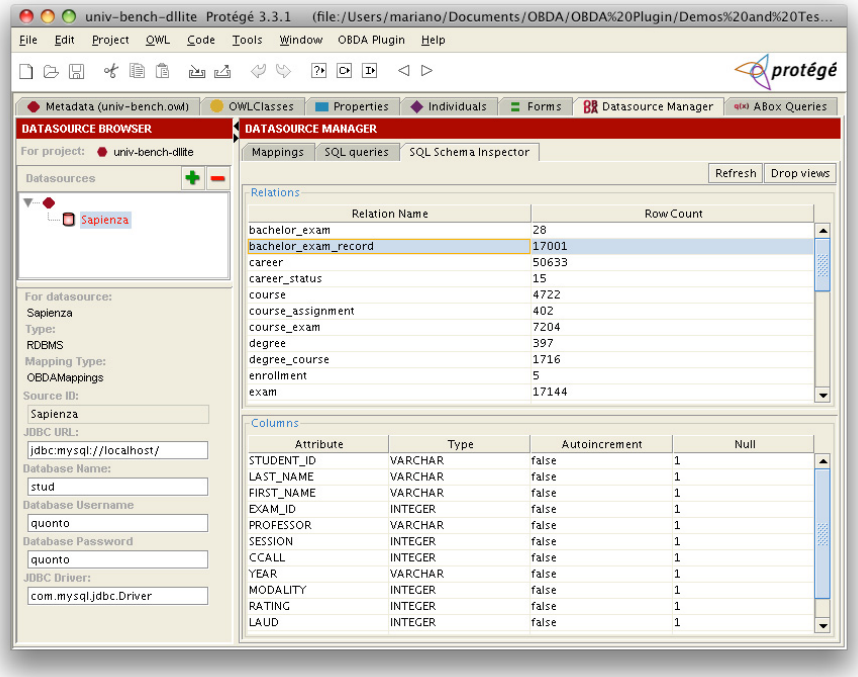



**Fig. 28.** OBDA Plugin's RDBMS Schema Inspector

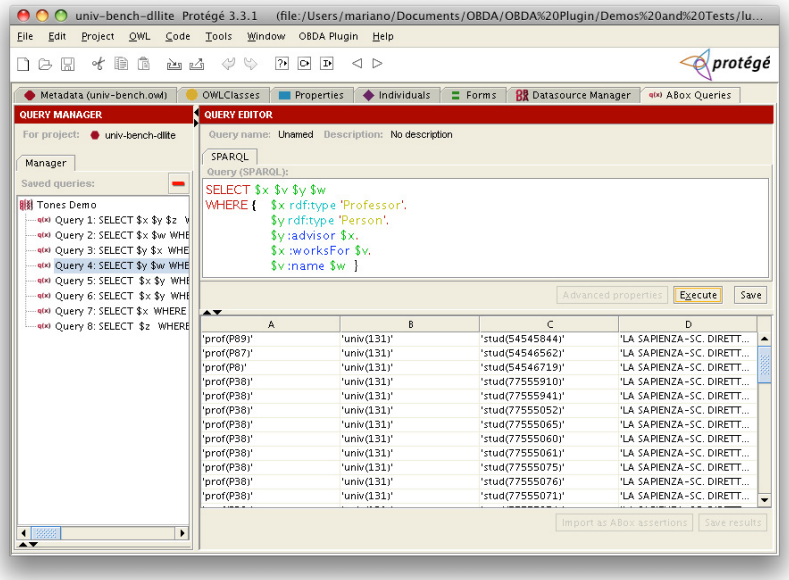



**Fig. 29.** OBDA Plugin's ABox Queries Tab

## 9 Conclusions

In this article, we have summarized the main technical results concerning the *DL-Lite* family of description logics, which has been developed, studied, and implemented in recent years as a solution to the problem of ontology-based data access (OBDA). The *DL-Lite* family, and in particular *DL-Lite*$_{\mathcal{A},id}$, an expressive member of this family that we have used as the basis for our technical development, provides a solution to the trade-off between expressiveness of the language and complexity of inference that is optimized towards the requirements arising in OBDA, namely *(i)* the ability to capture the main modeling constructs of conceptual modeling languages, such as UML class diagrams and the Entity-Relationship model, and *(ii)* efficient reasoning over large amounts of data and the ability to compute the certain answers to conjunctive queries and unions thereof w.r.t. an ontology by rewriting a query into a new query and directly evaluating such a rewritten query over the available data using a relational DBMS.

The results we have presented here are based mainly on work that has been carried out in the last years and published in the following articles: [22,24,25,20,75]. However, the *DL-Lite* family has spurred a lot of interest in the research community, and recently various follow-up activities have emerged and research is still very active and ongoing. We summarize here the major research efforts:

- Extensions of the ontology language with further constructs, such as number restrictions (which are a generalization of functionality), full booleans, and additional role constructs present in OWL 2, and a systematic analysis of the computational complexity of inference for all meaningful combinations of constructs and under various assumptions [3,4].
- Extension of the query language to support full first-order queries (or equivalently, SQL queries) under a weakened epistemic semantics, overcoming the undecidability of full first-order inference over ontologies [23].
- Updating *DL-Lite* ontologies [35,36].
- Fuzzy extensions to the DLs of the *DL-Lite* family [87,72].
- Extensions of *DL-Lite* with temporal operators, which has applications to temporal conceptual data modeling [5].
- Computation and extraction of modules from an ontology [60,59].

**Acknowledgements.** This research has been partially supported by the FET project TONES (Thinking ONtologiES), funded within the EU 6th Framework Programme under contract FP6-7603, and by the large-scale integrating project (IP) OntoRule (ONTOlogies meet Business RULEs), funded by the EC under ICT Call 3 FP7-ICT-2008-3, contract number FP7-231875.

## References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading (1995)
2. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: Querying ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 1670–1671 (2005)

3. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: DL-Lite in the light of first-order logic. In: Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007), pp. 361–366 (2007)
4. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. Technical Report BBKCS-09-03, School of Computer Science and Information Systems, Birbeck College, London (2009), http://www.dcs.bbk.ac.uk/research/techreps/2009/bbkcs-09-03.pdf
5. Artale, A., Kontchakov, R., Lutz, C., Wolter, F., Zakharyaschev, M.: Temporalising tractable description logics. In: Proc. of the 14th Int. Symp. on Temporal Representation and Reasoning (TIME 2007), pp. 11–22 (2007)
6. Baader, F.: Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence, IJCAI 1991 (1991)
7. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
8. Baader, F., Hladik, J., Lutz, C., Wolter, F.: From tableaux to automata for description logics. Fundamenta Informaticae 57, 1–33 (2003)
9. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69(1), 5–40 (2001)
10. Bechhofer, S., Möller, R., Crowther, P.: The DIG description logic interface. In: Proc. of the 2003 Description Logic Workshop (DL 2003). CEUR Electronic Workshop Proceedings, vol. 81, pp. 196–203 (2003), http://ceur-ws.org/
11. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1–2), 70–118 (2005)
12. Brachman, R.J., Levesque, H.J.: The tractability of subsumption in frame-based description languages. In: Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI 1984), pp. 34–37 (1984)
13. Brachman, R.J., Levesque, H.J. (eds.): Readings in Knowledge Representation. Morgan Kaufmann, San Francisco (1985)
14. Brachman, R.J., Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. Cognitive Science 9(2), 171–216 (1985)
15. Buchheit, M., Donini, F.M., Schaerf, A.: Decidable reasoning in terminological knowledge representation systems. J. of Artificial Intelligence Research 1, 109–138 (1993)
16. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: On the expressive power of data integration systems. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 338–350. Springer, Heidelberg (2002)
17. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), pp. 16–21 (2003)
18. Calvanese, D., De Giacomo, G.: Expressive description logics. In: Baader, et al. (eds.) [7], ch. 5, pp. 178–218 (2003)
19. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Linking data to ontologies: The description logic *DL-Lite*$_A$. In: Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED 2006). CEUR Electronic Workshop Proceedings, vol. 216 (2006), http://ceur-ws.org/
20. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R., Ruzzi, M.: Data integration through *DL-lite*$_{\mathcal{A}}$ ontologies. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 26–47. Springer, Heidelberg (2008)

21. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 602–607 (2005)
22. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 260–270 (2006)
23. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: EQL-Lite: Effective first-order query processing in description logics. In: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007), pp. 274–279 (2007)
24. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
25. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Path-based identification constraints in description logics. In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008), pp. 231–241 (2008)
26. Calvanese, D., De Giacomo, G., Lenzerini, M.: Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI 1999), pp. 84–89 (1999)
27. Calvanese, D., De Giacomo, G., Lenzerini, M.: Answering queries using views over description logics knowledge bases. In: Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000), pp. 386–391 (2000)
28. Calvanese, D., De Giacomo, G., Lenzerini, M.: 2ATAs make DLs easy. In: Proc. of the 2002 Description Logic Workshop (DL 2002). CEUR Electronic Workshop Proceedings, vol. 53, pp. 107–118 (2002), `http://ceur-ws.org/`
29. Calvanese, D., De Giacomo, G., Lenzerini, M.: Description logics for information integration. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS, vol. 2408, pp. 41–60. Springer, Heidelberg (2002)
30. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D.: Reasoning in expressive description logics. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 23, vol. II, pp. 1581–1634. Elsevier Science Publishers, Amsterdam (2001)
31. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. J. of Artificial Intelligence Research 11, 199–240 (1999)
32. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of the 9th ACM Symp. on Theory of Computing (STOC 1977), pp. 77–90 (1977)
33. Chen, C., Haarslev, V., Wang, J.: LAS: Extending Racer by a Large ABox Store. In: Proc. of the 2005 Description Logic Workshop (DL 2005). CEUR Electronic Workshop Proceedings, vol. 147 (2005), `http://ceur-ws.org/`
34. Cosmadakis, S.S., Kanellakis, P.C., Vardi, M.: Polynomial-time implication problems for unary inclusion dependencies. J. of the ACM 37(1), 15–46 (1990)
35. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the update of description logic ontologies at the instance level. In: Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006), pp. 1271–1276 (2006)
36. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the approximation of instance level update and erasure in description logics. In: Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007), pp. 403–408 (2007)
37. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: Ontology based access to distributed and semi-structured information. In: Meersman, R., Tari, Z., Stevens, S. (eds.) Database Semantic: Semantic Issues in Multimedia Systems, ch. 20, pp. 351–370. Kluwer Academic Publishers, Dordrecht (1999)

38. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: The complexity of concept languages. Information and Computation 134, 1–58 (1997)
39. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Deduction in concept languages: From subsumption to instance checking. J. of Logic and Computation 4(4), 423–452 (1994)
40. Garey, M.R., Johnson, D.S.: Computers and Intractability — A guide to NP-completeness. W. H. Freeman and Company, San Francisco (1979)
41. Goasdoue, F., Lattes, V., Rousset, M.-C.: The use of CARIN language and algorithms for information integration: The Picsel system. Int. J. of Cooperative Information Systems 9(4), 383–401 (2000)
42. Gruber, T.: Towards principles for the design of ontologies used for knowledge sharing. Int. J. of Human and Computer Studies 43(5/6), 907–928 (1995)
43. Gruber, T.R.: A translation approach to portable ontology specification. Knowledge Acquisition 5(2), 199–220 (1993)
44. Guarino, N.: Formal ontology in information systems. In: Proc. of the Int. Conf. on Formal Ontology in Information Systems (FOIS 1998). Frontiers in Artificial Intelligence, pp. 3–15. IOS Press, Amsterdam (1998)
45. Haarslev, V., Möller, R.: RACER system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 701–705. Springer, Heidelberg (2001)
46. Halevy, A.Y.: Answering queries using views: A survey. J. of Very Large Database 10(4), 270–294 (2001)
47. Heflin, J., Hendler, J.: A portrait of the Semantic Web in action. IEEE Intelligent Systems 16(2), 54–59 (2001)
48. Heymans, S., Ma, L., Anicic, D., Ma, Z., Steinmetz, N., Pan, Y., Mei, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Feier, C., Hench, G., Wetzstein, B., Keller, U.: Ontology reasoning with large data repositories. In: Hepp, M., De Leenheer, P., de Moor, A., Sure, Y. (eds.) Ontology Management, Semantic Web, Semantic Web Services, and Business Applications. Semantic Web And Beyond Computing for Human Experience, vol. 7, pp. 89–128. Springer, Heidelberg (2008)
49. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 1998), pp. 636–647 (1998)
50. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: DL reasoning with large numbers of individuals. In: Proc. of the 2004 Description Logic Workshop (DL 2004). CEUR Electronic Workshop Proceedings, vol. 104 (2004), `http://ceur-ws.org/`
51. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics 1(1), 7–26 (2003)
52. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. of Logic and Computation 9(3), 385–410 (1999)
53. Horrocks, I., Sattler, U.: A tableau decision procedure for $\mathcal{SHOIQ}$. J. of Automated Reasoning 39(3), 249–276 (2007)
54. Hull, R.: A survey of theoretical research on typed complex database objects. In: Paredaens, J. (ed.) Databases, pp. 193–256. Academic Press, London (1988)
55. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), pp. 466–471 (2005)
56. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. of Computer and System Sciences 28(1), 167–189 (1984)
57. Kolaitis, P.G.: Schema mappings, data exchange, and metadata management. In: Proc. of the 24rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2005), pp. 61–75 (2005)

58. Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 1998), pp. 205–213 (1998)
59. Kontchakov, R., Pulina, L., Sattler, U., Schneider, T., Selmer, P., Wolter, F., Zakharyaschev, M.: Minimal module extraction from DL-Lite ontologies using QBF solvers. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence, IJCAI 2009 (2009)
60. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Can you tell the difference between DL-Lite ontologies? In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008), pp. 285–295 (2008)
61. Kozen, D.: Theory of Computation. Springer, Heidelberg (2006)
62. Krisnadhi, A., Lutz, C.: Data complexity in the $\mathcal{EL}$ family of description logics. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 333–347. Springer, Heidelberg (2007)
63. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), pp. 233–246 (2002)
64. Libkin, L.: Data exchange and incomplete information. In: Proc. of the 25th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2006), pp. 60–69 (2006)
65. Maedche, A.: Ontology learning for the Semantic Web. Kluwer Academic Publishers, Dordrecht (2003)
66. Meseguer, J., Qian, X.: A logical semantics for object-oriented databases. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 89–98 (1993)
67. Minsky, M.: A framework for representing knowledge. In: Haugeland, J. (ed.) Mind Design. The MIT Press, Cambridge (1981); A longer version appeared in The Psychology of Computer Vision (1975), Republished in [13]
68. Möller, R., Haarslev, V.: Description logic systems. In: Baader, et al. (eds.) [7], ch. 8, pp. 282–305
69. Nebel, B.: Computational complexity of terminological reasoning in BACK. Artificial Intelligence 34(3), 371–383 (1988)
70. Noy, N.F.: Semantic integration: A survey of ontology-based approaches. SIGMOD Record 33(4), 65–70 (2004)
71. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of query answering in expressive description logics via tableaux. J. of Automated Reasoning 41(1), 61–98 (2008)
72. Pan, J.Z., Stamou, G.B., Stoilos, G., Thomas, E.: Expressive querying over fuzzy DL-Lite ontologies. In: Proc. of the 2007 Description Logic Workshop (DL 2007). CEUR Electronic Workshop Proceedings, vol. 250 (2007), `http://ceur-ws.org/`
73. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Publ. Co., Reading (1994)
74. Patel-Schneider, P.F., McGuinness, D.L., Brachman, R.J., Resnick, L.A., Borgida, A.: The CLASSIC knowledge representation system: Guiding principles and implementation rational. SIGART Bull. 2(3), 108–113 (1991)
75. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)
76. Poggi, A., Rodriguez, M., Ruzzi, M.: Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. In: Clark, K., Patel-Schneider, P.F. (eds.) Proc. of the 4th Int. Workshop on OWL: Experiences and Directions, OWLED 2008 DC (2008)
77. Racer Systems GmbH & Co. KG. Release notes for RacerPro 1.9.2 beta, `http://www.sts.tu-harburg.de/~r.f.moeller/racer/Racer-1-9-2-beta-Release-Notes/release-notes-1-9-2se8.html` (last access, July 2008)

78. Reingold, O.: Undirected connectivity in log-space. J. of the ACM 55(4) (2008)
79. Reiter, R.: On closed world data bases. In: Gallaire, H., Minker, J. (eds.) Logic and Databases, pp. 119–140. Plenum Publ. Co., New York (1978)
80. Rodríguez-Muro, M., Calvanese, D.: An OBDA extension to the DIG 1.1 Interface (July 2008), http://www.inf.unibz.it/~rodriguez/OBDA/dig-11-obda/
81. Rodriguez-Muro, M., Calvanese, D.: Towards an open framework for ontology based data access with Protégé and DIG 1.1. In: Proc. of the 5th Int. Workshop on OWL: Experiences and Directions, OWLED 2008 (2008)
82. Rodriguez-Muro, M., Lubyte, L., Calvanese, D.: Realizing ontology based data access: A plug-in for Protégé. In: Proc. of the ICDE Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008), pp. 286–289. IEEE Computer Society Press, Los Alamitos (2008)
83. Schild, K.: A correspondence theory for terminological logics: Preliminary report. In: Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI 1991), pp. 466–471 (1991)
84. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)
85. Sirin, E., Parsia, B.: Pellet system description. In: Proc. of the 2006 Description Logic Workshop (DL 2006). CEUR Electronic Workshop Proceedings, vol. 189 (2006), http://ceur-ws.org/
86. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. Technical report, University of Maryland Institute for Advanced Computer Studies, UMIACS (2005)
87. Straccia, U.: Towards top-k query answering in description logics: The case of DL-lite. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS, vol. 4160, pp. 439–451. Springer, Heidelberg (2006)
88. Uschold, M., Grüninger, M.: Ontologies and semantics for seamless connectivity. SIGMOD Record 33(4), 58–64 (2004)
89. van der Meyden, R.: Logical approaches to incomplete information. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, pp. 307–356. Kluwer Academic Publishers, Dordrecht (1998)
90. Vardi, M.Y.: The complexity of relational query languages. In: Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC 1982), pp. 137–146 (1982)
91. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
92. Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. J. of Computer and System Sciences 32, 183–221 (1986)
93. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, Heidelberg (1999)