Extending *DL-Lite*$_R$ TBoxes with View Definitions

Manuel Namici, Giuseppe De Giacomo, and Maurizio Lenzerini

Dipartimento di Ingegneria Informatica, Automatica e Gestionale "Antonio Ruberti"
Sapienza, Università di Roma
lastname@diag.uniroma1.it

Abstract. Views are a mechanisms for precomputing answers to query of particular significance. Views have a definition (the query itself) and an extension obtained by evaluating the query over the data sources. Views are used for controlling the access to data and keep data even when the original sources are not accessible anymore. In this paper we introduce views definitions in *DL-Lite*$_R$ ontologies as an additional form of assertions in the TBox, and we study the basic reasoning tasks involving them, including consistency, containment, disjointness, projection classification, and query answering.

1 Introduction

The notion of *view* is used in data management to create an abstraction of a query (which forms the definition of the view), and to see it as an ordinary database object. When such an object is mentioned in a data service (query or update), the data manager substitutes it with its "content", and then executes the service resulting from the substitution. In relational databases, views can be seen as new relations derived from the elementary ones (base relations). But views are also available in NoSQL systems (e.g., MongoDB), and they have been studied for graph databases as well. Note that there are two interpretations of the notion of "content" here: for dynamic views, the content is simply the definition of the associated query, whereas for static views (also called materialized views), the content is the so-called "extension" of the view, i.e., the result that was pre-computed and cached for the associated query.

There are many uses of views in data management. For example, they can provide the user with concepts that are derived from the elementary database objects, thus improving abstraction. Or, they can represent a selection of the data contained in a particular portion of the database, so as to limit the degree of exposure of the underlying data to the outer world. Indeed, users of a given class may have permission to query the view, while denied access to the rest of the database. Views can also present aggregations to the users, thus providing calculated results as part of the data temselves. Finally, views can be used in

data integration to specify which portion of a data source is mapped to which concepts in the global schema, or in data exchange to characterize the portion of the data to be transferred from the sources to the target system.

Generally speaking, the notion of view can be very well used in knowledge bases too. Surprisingly, however, there are only few works dealing with views in such area in general, and in Description Logics in particular. Here, we mention those works that are the most relevant for the present paper. In [4], the authors propose to split the TBox of a knowledge base in two parts, one for declaring the classical terminological axioms regarding the concepts and the relation of interest (forming the schema), and one for defining views, that are new concepts (unary predicates), each one with an associated definition. Axioms regarding the concepts in the schema refer only to names of schema elements, whereas concept expressions constituting the view definitions may refer to names of both schema and view elements. Note that views can be recursive, in the sense they contain (either direct or indirect) references to themselves, and fixpoint semantics are advocated for dealing with cycles in the definitions. In the seminal paper [3], the authors introduce the problem of rewriting queries using views in Description Logics: find a query expression that uses only a set of conjunctive views V defined over a Description Logic knowledge base, and is equivalent to a given conjunctive query Q. While query rewriting aims at computing a query over the views that is equivalent to the original one, view-based query answering has the goal of computing the answers to a query by relying only on the pre-computed extensions of a set of views. Obviously, rewriting is a technique for addressing view-based query answering, but the latter is a more general problem than the one of rewriting queries using views. Algorithms and complexity for view-based query answering in lighweight Description Logics have been investigated in [8, 9].

In this paper we follow an approach similar to the one adopted in [4], but with the following differences: (*i*) Each view is given as a conjunctive query, rather than a concept expression, thus allowing the use of variables, joins, projections, and selections in the expression associated to the view definition. (*ii*) We do not limit the views to unary or binary predicates, and thus each view symbol has an associated arity (greater than 0), which corresponds to the number of distinguished variables of the query constituting its definition. (*iii*) We disallow the use of recursion in view definitions.

Example 1. Consider an ontology containing the following axioms:

$$\exists \texttt{hasTaken} \sqsubseteq \texttt{Student},\ \exists \texttt{hasGiven} \sqsubseteq \texttt{Professor},\ \exists \texttt{isTheCourseOf} \sqsubseteq \texttt{Course},$$
$$\exists \texttt{teaches} \sqsubseteq \texttt{Professor},\ \exists \texttt{teaches}^- \sqsubseteq \texttt{Course}$$

where the notion of "exam" is represented in terms of a number of binary relations linking the various actors (student, professor) and components (course, grade) of the exam. If we want to provide the user with a different abstraction of the notion of exam, in particular as a specific predicate with 4 arguments, indicating who has taken the exam, what is the corresponding course, who is the professor (who must teach the course) who has given the exam, and which is the assigned grade, then we can define the following view:

$$\texttt{Exam}(x, y, w, z) \leftarrow \exists t\ \texttt{hasTaken}(x, t),\ \texttt{isTheCourseOf}(y, t),\ \texttt{HasGiven}(w, t),\ \texttt{teaches}(w, y),\ \texttt{grade}(t, z)$$

The above example shows that with the new predicate symbols corresponding to the views, the ontology is enriched with new abstractions that can be used during both ontology exploration for domain analysis, and query answering. The introduction of such new predicates may help overcoming some limitations of Ontology-based Data Access (OBDA) systems. The first limitation has to do with the nature of the ontology languages used in OBDA, which only allows unary and binary predicates. The use of views provides the possibility of using n-ary predicates in modeling the domain. The second limitation is related to the controlled expressive power in specifying the axioms of the ontology. For example, no lightweight ontology languages allows the free use of joins in concepts and role expressions. Once again, the possibility of defining conjunctive views may help overcoming this limitation.

Obviously, once we have introduced views in our ontology, we should be able to reason about them in all the tasks where the ontology plays a role. We discuss two basic classes of reasoning tasks, namely query answering, and reasoning about views. It is easy to see that, with the limitation (*iii*), computing the certain answers to (unions of) conjunctive queries whose atoms are ordinary concepts and relations, and view symbols can be still done in AC^0 in the case where the ontology is expressed in *DL-Lite*$_R$. Indeed, a simple unfolding strategy suffices for this purpose. Reasoning on views, on the other hand, is more challenging. Indeed, we aim at designing algorithms and study the complexity of checking view consistency, view containment and view disjointness. This is exactly the topic of this paper.

The paper is organized as follows. In Section 2 we illustrate some preliminary notions of queries in Description Logics. Section 3 describes our approach for extending ontologies with views. Section 4 presents algorithms and complexity analyses for reasoning about views. Finally, Section 5 concludes the paper, by highlighting possible directions for future work.

2 Preliminaries

In this section we briefly recap the basic notions of Description Logics [2] (DLs) and conjunctive queries [1]. DLs are a family of formal knowledge representation languages that are essentially fragments of first-order logic that can be used to model the domain of interest in terms of individuals, representing single individuals in the domain, concepts, representing sets of individuals, and roles, representing binary relations on sets of individuals. A DL knowledge base (KB) or ontology is composed by a set of logical axioms, which are typically separated into terminological (TBox) axioms, that describe relationships between concepts and roles, and assertional (ABox) axioms, that describe knowledge about (tuples of) individuals in the domain. In this paper we consider the Description Logic *DL-Lite*$_R$ [6] of the *DL-Lite* family of lightweight Description Logics.

DL-Lite$_R$: syntax and semantics. Given a signature $\mathcal{S} = \langle N_c, N_r, N_i \rangle$, where N_c is a set of *atomic concept* names, N_r is a set of *atomic role* names, and N_i is a set of *individual* names, then *DL-Lite$_R$* concept and roles are constructed according to the following syntax:

$$B ::= A \mid \exists R \qquad\qquad R ::= P \mid P^-$$
$$C ::= B \mid \neg B \qquad\qquad E ::= R \mid \neg R$$

where $A \in N_c$ denotes an *atomic concept*, $P \in N_r$ denotes an *atomic role*, B denotes a *basic concept*, that can either be an atomic concept or an expression of the form $\exists R$, and R denotes a *basic role*, that can be either an atomic role or the inverse of an atomic role. Finally, C and E denote, respectively, a *general concept*, that can be either a basic concept or its negation, and a *general role*, that can be either a basic role or its negation.

A *DL-Lite$_R$* TBox $\mathcal{T}$ over a signature $\mathcal{S}$ is formed by a finite set of *inclusion assertions* of the form:

$$B \sqsubseteq C \qquad\qquad R \sqsubseteq E$$

where general concept expressions (resp. general role expressions) are allowed to occurs only on the right-hand side of the inclusion assertion. Intuitively, an assertion of the form $B \sqsubseteq C$ states that all instances of the basic concept B are also instances of the general concept C, whereas an assertion of the form $R \sqsubseteq E$ states that all instances of the basic role R are also instances of the general role E.

A *DL-Lite$_R$* ABox $\mathcal{A}$ over a signature $\mathcal{S}$ is formed by a finite set of *membership assertions* on atomic concepts and roles, of the form:

$$A(a) \qquad\qquad P(a, b)$$

where $a, b \in N_i$ are constants denoting object (individuals) in the domain, $A(a)$ states that the object denoted by the constant a is an instance of the atomic concept A, and $P(a, b)$ states that the pair of objects identified by constants a and b is an instance of the atomic role P. Given a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ defined as above, a *DL-Lite$_R$* knowledge base (KB) over a signature $\mathcal{S}$ is defined as the pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$.

The semantics of a *DL-Lite$_R$* KB $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is given in terms of first-order *interpretations*. An interpretation $\mathcal{I}$ over a signature $\mathcal{S}$ is defined as a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}}$ is a nonempty *interpretation domain*, and $\cdot^{\mathcal{I}}$ is an *interpretation function*, that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, and to each individual a an element $a^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, according to the following rules:

$$a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \qquad\qquad (R^-)^{\mathcal{I}} \subseteq \left\{ \langle b, a \rangle \mid \langle a, b \rangle \in R^{\mathcal{I}} \right\}$$
$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \qquad\qquad (\neg B)^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$$
$$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \qquad\qquad (\neg R)^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$$
$$(\exists R)^{\mathcal{I}} \subseteq \left\{ a \mid \exists b. \langle a, b \rangle \in R^{\mathcal{I}} \right\}$$

Given a KB $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ over a signature $\mathcal{S}$, and an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ over the same signature, then an inclusion assertion $B \sqsubseteq C$ (resp. $R \sqsubseteq E$) in $\mathcal{T}$ is said to *hold* in $\mathcal{I}$ if and only if $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ (resp. $R^{\mathcal{I}} \subseteq E^{\mathcal{I}}$), while a membership assertion $A(a)$ (resp. $P(a, b)$) is said to *hold* in $\mathcal{I}$ if and only if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp. $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in P^{\mathcal{I}}$). An interpretation $\mathcal{I}$ is a *model* for $\mathcal{O}$, denoted as $\mathcal{I} \models \mathcal{O}$, if and only if it satisfies every inclusion assertion and every membership assertion in $\mathcal{O}$. The set of all models of a KB is indicated as $Mod(\mathcal{O}) = \{\mathcal{I} \mid \mathcal{I} \models \mathcal{O}\}$. We use the same notation $\mathcal{I} \models \mathcal{T}$ and $Mod(\mathcal{T})$ also when we refer only the TBox $\mathcal{T}$.

Conjunctive Queries over DL-Lite$_R$. A *conjunctive query* (CQ) over a DL knowledge base is an open *first-order logic* formula of the form:

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

where $\vec{x}$ are the *free* (or *distinguished*) variables, and $\vec{y}$ are the *existentially quantified* (or *non-distinguished*) variables, and *conj* is a conjunction of atoms over the signature of the knowledge base, each of the form $A(z)$ or $P(z, w)$, where A is an atomic concept, P is an atomic role, and z, w are constants appearing in $\mathcal{A}$ or variable names. A CQ that does not involve any free variable is called a *boolean CQ* corresponding to a closed formula of first-order logic. Sometimes, it is convenient to denote CQs using *datalog notation* [1]:

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

where $q(\vec{x})$, called the *head* of the query, is a symbol that is not part of the signature of the KB, and $conj(\vec{x}, \vec{y})$ is called the *body* of the query. Using this notation, variables in $\vec{x}$ appearing in the head of the query denote the free variables, while variables in $\vec{y}$, that do not appear in the head of the query, are the non-distinguished variables, and are assumed to be implicitly existentially quantified. Moreover, conjunction in datalog notation is typically represented by separating atoms with a comma. Using datalog notation, a boolean CQ can be simply written as: $q() \leftarrow conj(\vec{y})$.

The semantics of a CQ are given in terms of *answers*: Given an interpretation $\mathcal{I}$, the answers to a CQ $\exists \vec{y}.conj(\vec{x}, \vec{y})$ is given in terms of the assignments $\alpha : Vars \rightarrow \Delta^{\mathcal{I}}$ from the free variables in $\vec{x}$ to the constants in $\mathcal{I}$, such that, when substituted, make the formula true in $\mathcal{I}$.

$$conj^{\mathcal{I}} = \{\alpha(\vec{x}) \mid \mathcal{I}, \alpha \models \exists \vec{y}.conj(\vec{x}, \vec{y})\}$$

The crucial characteristic of *DL-Lite*$_R$ ontologies is that they enable the so-called Ontology-based Data Access [6, 7, 10]. Indeed, *DL-Lite*$_R$ enjoys *first-order rewritability* of query answering. That is, every (union of) CQ over a *DL-Lite*$_R$ ontology can be rewritten into a first-order query to be evaluated over the ABox only (i.e., the individual data) considered as a database. This property, on the one hand, gives us a very low worst-case computational complexity bound w.r.t. data, namely AC^0 data complexity. On the other hand, it gives us a very effective practical technique to deal with ontologies that include very large ABoxes (i.e.,

a lot of individual data): perform the rewriting; transform the first-order query into SQL, or SPARQL, depending on how data are stored; and perform the resulting query exploiting a data management engine to take advantage of all optimizations available for these standard languages.

While in this paper we do not consider directly query answering, we will exploit *DL-Lite*$_R$ nice computational features when reasoning on views.

3 Extending *DL-Lite*$_R$ TBoxes with Views Definitions

In this section we describe an extension to the notion of *DL-Lite*$_R$ KB that includes *views* defined as conjunctive queries.

If $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a *DL-Lite*$_R$ KB over a signature $\mathcal{S}$, a *view* V over $\mathcal{O}$ is defined as follows:

$$V = \langle v, \exists \vec{y}.conj(\vec{x}, \vec{y}) \rangle$$

where v is the name associated with the view V, and $conj$ is a CQ over $\mathcal{O}$ with free variables $\vec{x}$. As a notation, in what follows we will indicate with $name(V)$ the corresponding view symbol v associated with V, with $def(V)$ the corresponding conjunctive query $\exists \vec{y}.conj(\vec{x}, \vec{y})$ associated with V, and with $arity(V)$ the arity of the view V, coinciding with the number of free variables $\vec{x}$.

Now, if $N_v = \{v_1, \ldots, v_n\}$ is a finite set of view names, where each v_i is a symbol not appearing in $\mathcal{S}$, and $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ is a set of views over $\mathcal{O}$, where each $V_i \in \mathcal{V}$ is defined as follows:

$$V_i = \langle v_i, \exists \vec{y}.conj_i(\vec{x}, \vec{y}) \rangle$$

then $\mathcal{O}_v = \langle \mathcal{T}, \mathcal{A}, \mathcal{V} \rangle$ is called a *view-enriched knowledge base* over the signature $\mathcal{S}_v = \langle N_c, N_r, N_i, N_v \rangle$.

For convenience, with a little abuse of notation, for view V_i we will sometimes use the abbreviated form $v_i(\vec{x}) \leftarrow conj_i(\vec{x}, \vec{y})$.

Example 2. Let's consider the following ontology, composed by:

$$\mathcal{T} = \{\texttt{Student} \sqsubseteq \texttt{Person},\ \texttt{Professor} \sqsubseteq \texttt{Person},\ \texttt{Student} \sqsubseteq \neg\texttt{Professor},\ \exists\texttt{LivesIn} \sqsubseteq \texttt{Person},\ \exists\texttt{LivesIn}^- \sqsubseteq \texttt{City},\ \exists\texttt{LocatedIn} \sqsubseteq \texttt{City},\ \exists\texttt{LocatedIn}^- \sqsubseteq \texttt{Country}\}$$

$$\mathcal{A} = \{\texttt{Student}(john),\ \texttt{Professor}(mary),\ \texttt{LivesIn}(john, london),\ \texttt{LocatedIn}(london, uk)\}$$

Then we can introduce a view $\mathcal{V} = \{V_1\}$ as follows:

$$V_1 = \langle \texttt{PersonCountry},\ \exists y.\texttt{Person}(x_1) \wedge \texttt{LivesIn}(x_1, y) \wedge \texttt{LocatedIn}(y, x_2) \rangle$$

$$name(V_1) = \texttt{PersonCountry} \qquad arity(V_1) = 2$$
$$def(V_1) = \exists y.\texttt{Person}(x_1) \wedge \texttt{LivesIn}(x_1, y) \wedge \texttt{LocatedIn}(y, x_2)$$

The abbreviated form for the view V_1 is:

$$\texttt{PersonCountry}(x_1, x_2) \leftarrow \texttt{Person}(x_1), \texttt{LivesIn}(x_1, y), \texttt{LocatedIn}(y, x_2)$$

In order to define the semantics for $\mathcal{O}_v$, we need to provide the semantics for the views in $\mathcal{V}$. Thus, starting from the notion of interpretation as described in Section 2, we extend the interpretation function to views as follows:

$$v_i^{\mathcal{I}} = conj_i^{\mathcal{I}} = \{\alpha(\vec{x}) \mid \mathcal{I}, \alpha \models \exists \vec{y}.conj(\vec{x}, \vec{y})\}$$

that is, the interpretation of a view v_i is equivalent to the assignments that make the corresponding formula $conj_i$ true in $\mathcal{I}$. Notice that by adopting this definition, such views act as a conservative extension to the KB language.

Example 3. An interpretation $\mathcal{I}$ that satisfies the ontology in the example above is the following:

$$\begin{aligned} \Delta^{\mathcal{I}} &= \{j, k, l, m\} & \texttt{LivesIn}^{\mathcal{I}} &= \{\langle j, l\rangle\} \\ \texttt{Person}^{\mathcal{I}} &= \{j, m\} & \texttt{LocatedIn}^{\mathcal{I}} &= \{\langle l, k\rangle\} \\ \texttt{Student}^{\mathcal{I}} &= \{j\} & john^{\mathcal{I}} &= j \\ \texttt{Professor}^{\mathcal{I}} &= \{m\} & mary^{\mathcal{I}} &= m \\ \texttt{City}^{\mathcal{I}} &= \{l\} & london^{\mathcal{I}} &= l \\ \texttt{Country}^{\mathcal{I}} &= \{k\} & uk^{\mathcal{I}} &= k \end{aligned}$$

Consider that the query $def(V_1) = \exists y.\texttt{Person}(x) \wedge \texttt{LivesIn}(x, y) \wedge \texttt{LocatedIn}(y, z)$ is evaluated as $(def(V_1))^{\mathcal{I}} = \{\langle j, k\rangle\}$, and this evaluation is assigned to $name(V_1) = \texttt{PersonCountry}$, hence, $\texttt{PersonCountry}^{\mathcal{I}} = \{\langle j, k\rangle\}$.

4 Basic Reasoning Services on Views Definitions

We now proceed in defining a set of reasoning services in *DL-Lite*$_R$ that are of interest when taking into account views as defined in Section 3. In particular, we are interested in the following reasoning tasks:

- **View consistency**: Given a TBox $\mathcal{T}$ and a view V, defined as:

  $$V = \langle v, \exists \vec{y}.conj(\vec{x}, \vec{y})\rangle$$

  we are interested in checking whether V *is consistent w.r.t.* $\mathcal{T}$, denoted as $V \neq_{\mathcal{T}} \emptyset$. In other words, we want to know if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$, and $v^{\mathcal{I}} \neq \emptyset$.

  Example 4. Consider the TBox from Example 2, and the view V_1:

  $$\texttt{PersonCountry}(x_1, x_2) \leftarrow \texttt{Person}(x_1), \texttt{LivesIn}(x_1, y), \texttt{LocatedIn}(y, x_2)$$

  it is easy to see that this view is consistent with $\mathcal{T}$. Indeed the interpretation $\mathcal{I}$ in Example 3 is a model of $\mathcal{T}$ for which the interpretation of `PersonCountry` is non-empty. However, if we consider for example a view V_2 defined as:

  $$\begin{aligned} \texttt{PersonCountry}(x_1, x_2) \leftarrow\ & \texttt{Student}(x_1), \texttt{Professor}(x_1), \texttt{LivesIn}(x_1, y) \\ & \texttt{LocatedIn}(y, x_2) \end{aligned}$$

then it is immediate to see that this view must be empty in every possible model of $\mathcal{T}$ since the TBox assertion $\texttt{Student} \sqsubseteq \neg\texttt{Professor}$ implies that their intersection is empty in every model of $\mathcal{T}$, and hence $\texttt{Student}(x_1) \wedge \texttt{Professor}(x_1)$ is false for every assignment of x_1 in every model of $\mathcal{T}$.

– **View containment**: Given a TBox $\mathcal{T}$ and two views V_1, V_2, where $arity(V_1) = arity(V_2)$, defined as:

$$V_1 = \langle v_1, \exists \vec{y_1}.conj_1(\vec{x}, \vec{y_1}) \rangle \qquad V_2 = \langle v_2, \exists \vec{y_2}.conj_2(\vec{x}, \vec{y_2}) \rangle$$

we are interested in checking whether v_1 *is contained in* v_2, denoted as $V_1 \subseteq_{\mathcal{T}} V_2$. In other words, we are interested in checking whether $v_1^{\mathcal{I}} \subseteq v_2^{\mathcal{I}}$, for every interpretation $\mathcal{I}$ that is a model of $\mathcal{T}$.

Example 5. Consider again the TBox $\mathcal{T}$ from Example 2, and the views V_1 and V_2 defined respectively as follows:

$$\texttt{PersonCountry}(x_1, x_2) \leftarrow \texttt{Person}(x_1), \texttt{LivesIn}(x_1, y), \texttt{LocatedIn}(y, x_2)$$
$$\texttt{StudentCountry}(x_1, x_2) \leftarrow \texttt{Student}(x_1), \texttt{LivesIn}(x_1, y), \texttt{LocatedIn}(y, x_2)$$

it follows from the TBox axioms that V_2 must be contained in V_1 in every possible model of $\mathcal{T}$ since the TBox contains the assertion $\texttt{Student} \sqsubseteq \texttt{Person}$.

– **View disjointness**: Given a TBox $\mathcal{T}$ and two views V_1, V_2 where $arity(V_1) = arity(V_2)$, defined as:

$$V_1 = \langle v_1, \exists \vec{y_1}.conj_1(\vec{x}, \vec{y_1}) \rangle \qquad V_2 = \langle v_2, \exists \vec{y_2}.conj_2(\vec{x}, \vec{y_2}) \rangle$$

we are interested in checking whether v_1 *is disjoint from* v_2, denoted as $V_1 \cap V_2 =_{\mathcal{T}} \emptyset$. In other words, we are interested in checking whether $v_1^{\mathcal{I}} \cap v_2^{\mathcal{I}} = \emptyset$, for every interpretation $\mathcal{I}$ that is a model of $\mathcal{T}$.

Example 6. Consider once more the TBox $\mathcal{T}$ from Example 2, and the views V_1 and V_2 defined respectively as follows:

$$\texttt{StudentCountry}(x_1, x_2) \leftarrow \texttt{Student}(x_1), \texttt{LivesIn}(x_2, y), \texttt{LocatedIn}(y, x_2)$$
$$\texttt{ProfessorCountry}(x_1, x_2) \leftarrow \texttt{Professor}(x_1), \texttt{LivesIn}(x_1, y), \texttt{LocatedIn}(y, x_2)$$

we have that V_1 and V_2 must be disjoint in every possible model of $\mathcal{T}$ since $\texttt{Student}$ and $\texttt{Professor}$ are disjoint in $\mathcal{T}$.

4.1 Checking View Consistency

We show how view consistency can be reduced to knowledge base satisfiability. To do so, we introduce the notion of *canonical ABox* $\mathcal{A}_q$ of a boolean CQ q.

Definition 1. *Given a boolean conjunctive query q, defined as $\exists \vec{y}.conj(\vec{y})$, the canonical ABox $\mathcal{A}_q$ for q is formed by the set of membership assertions as follows:*

- *For each variable y in q we include a new constant c_y in the signature.*
- *One membership assertion $A(\hat{t})$ for each atom of the form $A(t) \in \exists\vec{y}.conj(\vec{y})$, where $\hat{t} = c$ if t is a constant c, and $\hat{t} = c_y$ if t is a variable y.*
- *One membership assertion $P(\hat{t_1}, \hat{t_2})$ for each atom of the form $P(t_1, t_2) \in \exists\vec{y}.conj(\vec{y})$, where $\hat{t_i} = c$ if t_i is a constant c, and $\hat{t_i} = c_y$ if t_i is a variable y.*

Using the canonical ABox of a view $V = \langle v, \exists\vec{y}.conj(\vec{x}, \vec{y})\rangle$, we can check consistency $v \neq_{\mathcal{T}} \emptyset$ by applying the boolean procedure ViewConsistency$(\mathcal{T}, V)$, defined as follows:

1. Freeze the free variables $\vec{x}$, by replacing them with fresh constants $\vec{a}$. Notice that when we replace the free variables $\vec{x}$ in $\exists\vec{y}.conj(\vec{x}, \vec{y})$, we obtain a boolean query $v_{\vec{a}}$ of the form $\exists\vec{y}.conj(\vec{a}, \vec{y})$.
2. Compute the canonical ABox $\mathcal{A}_{v_{\vec{a}}}$ of $\exists\vec{y}.conj(\vec{a}, \vec{y})$.
3. Return true iff the KB $\langle\mathcal{T}, A_{v_{\vec{a}}}\rangle$ is consistent.

Theorem 1. *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox, and V be a view, $V \neq_{\mathcal{T}} \emptyset$ iff* ViewConsistency$(\mathcal{T}, V)$ *evaluates to true.*

Proof. ($\Leftarrow$) Let $V = \langle v, \exists\vec{y}.conj(\vec{x}, \vec{y})\rangle$. By definition ViewConsistency$(\mathcal{T}, V)$ returns *true* iff $\langle\mathcal{T}, \mathcal{A}_{v_{\vec{a}}}\rangle$ is satisfiable. This means that there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \langle\mathcal{T}\rangle, \mathcal{A}_{v_{\vec{a}}}$. Then the query $\exists\vec{y}.conj(\vec{a}, \vec{y})$ evaluates to *true* in $\mathcal{I}$. Notice that $\exists\vec{y}.conj(\vec{a}, \vec{y})$ is obtained from $\exists\vec{y}.conj(\vec{x}, \vec{y})$ by freezing the free variables $\vec{x}$ with fresh constants $\vec{a}$. Thus, if we consider the assignment α such that $\alpha(\vec{x}) = \vec{a}^{\mathcal{I}}$ we have that $\mathcal{I}, \alpha \models \exists\vec{y}.conj(\vec{x}, \vec{y})$, hence, $v^{\mathcal{I}} = \{\alpha(\vec{x})\}$ and hence V is consistent.

($\Rightarrow$) If there exists an interpretation $\mathcal{I}$ for which $v^{\mathcal{I}} \neq \emptyset$, then there exists an assignment α from the free variables $\vec{x}$ such that $\mathcal{I}, \alpha \models \exists\vec{y}.conj(\vec{x}, \vec{y})$. Moreover, considering the existentially quantified variables, there exists an assignment α' such that $\mathcal{I}, \alpha' \models conj(\vec{x}, \vec{y})$. Let us introduce fresh constants $\vec{a}$ for $\vec{x}$ and $\vec{b}$ for $\vec{y}$ and extend the interpretation $\mathcal{I}$ to $\mathcal{I}$' that interprets these fresh constants as $\vec{a}^{\mathcal{I}'} = \alpha'(\vec{x})$ and $\vec{b}^{\mathcal{I}'} = \alpha'(\vec{y})$. Then we immediately get that $\mathcal{I}'$ is such that $\mathcal{I}' \models \langle\mathcal{T}, \mathcal{A}_{v_{\vec{a}}}\rangle$, hence $\langle\mathcal{T}, \mathcal{A}_{v_{\vec{a}}}\rangle$ is satisfiable. □

Considering the complexity of KB consistency in *DL-Lite*$_R$, it is immediate to assess the complexity of view consistency.

Theorem 2. *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox, and V be a view, checking whether $V \neq_{\mathcal{T}} \emptyset$ is in* PTime *in $\mathcal{T}$, and* AC^0 *in $def(V)$.*

4.2 Checking View Containment

We now show how view containment can be reduced to query answering. We will again make use of the canonical ABox of a view, as defined above. Consider two views, $V_1 = \langle v_1, \exists\vec{y_1}.conj_1(\vec{x}, \vec{y_1})\rangle$ and $V_2 = \langle v_2, \exists\vec{y_2}.conj_2(\vec{x}, \vec{y_2})\rangle$, we can check containment $V_1 \subseteq_{\mathcal{T}} V_2$, by applying the boolean procedure ViewContainment$(\mathcal{T}, V_1, V_2)$, defined as follows:

1. Freeze the free variables $\vec{x}$, by replacing them with fresh constants $\vec{a}$. Notice that we obtain two boolean queries $v_{i,\vec{a}}$ of the form $\exists \vec{y}.conj_i(\vec{a}, \vec{y})$.
2. Compute the canonical ABox $\mathcal{A}_{v_{1,\vec{a}}}$ of $v_{1,\vec{a}}$.
3. Return $\langle \mathcal{T}, \mathcal{A}_{v_{1,\vec{a}}} \rangle \models v_{2,\vec{a}}$ (note that $v_{2,\vec{a}}$ is a boolean query).

Theorem 3. *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox, and V_1, V_2 be two views of the same arity, then $V_1 \subseteq_{\mathcal{T}} V_2$ iff* ViewContainment$(\mathcal{T}, V_1, V_2)$ *evaluates to true.*

Proof. ($\Rightarrow$) If $V_1 \subseteq_{\mathcal{T}} V_2$, then for every interpretation $\mathcal{I}$ that is a model of $\mathcal{T}$, we have that $v_1^{\mathcal{I}} \subseteq v_2^{\mathcal{I}}$. Let's now consider the boolean queries $v_{1,\vec{a}}, v_{2,\vec{a}}$, obtained by freezing the variables $\vec{x}$. It is easy to see that, by construction, $\mathcal{T}, \mathcal{A}_{v_{1,\vec{a}}} \models v_{1,\vec{a}}$ is always true. Since we know that $v_1^{\mathcal{I}} \subseteq v_2^{\mathcal{I}}$, we can conclude also that $\mathcal{T}, \mathcal{A}_{v_{1,\vec{a}}} \models v_{2,\vec{a}}$.
($\Leftarrow$) If $\mathcal{T}, \mathcal{A}_{v_{1,\vec{a}}} \models v_{2,\vec{a}}$, then by applying the deduction theorem we have that $\mathcal{T} \models \wedge_{A \in \mathcal{A}_{v_{1,\vec{a}}}} A \implies v_{2,\vec{a}}$ but $\wedge_{A \in \mathcal{A}_{v_{1,\vec{a}}}} A$ is $v_{1,\vec{a}}$, hence we have $\mathcal{T} \models v_{1,\vec{a}} \implies v_{2,\vec{a}}$. That is, for every model $\mathcal{I}'$ of $\mathcal{T}$ extended with the interpretation of the fresh constants $\vec{a}$, we have that $v_{1,\vec{a}}^{\mathcal{I}'} \subseteq v_{2,\vec{a}}^{\mathcal{I}'}$. If we now replace the fresh constants with variables we get that for every model $\mathcal{I}$ of $\mathcal{T}$ and every assignment α of the free variables $\vec{x}$, we have $\mathcal{I}, \alpha \models \exists \vec{y_1}.conj_1(\vec{x}, \vec{y_1})$ implies $\mathcal{I}, \alpha \models \exists \vec{y_2}.conj_2(\vec{x}, \vec{y_1})$. Hence the claim. □

Considering the complexity of query answering in *DL-Lite$_R$*, it is immediate to assess the complexity of view containment.

Theorem 4. *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox, and V_1, V_2 be two views of the same arity, checking whether $V_1 \subseteq_{\mathcal{T}} V_2$ is in* PTime *in $\mathcal{T}$,* AC^0 *in $def(V_1)$, and* NP *in $def(V_2)$.*

4.3 Checking View Disjointness

In this section we show how view disjointness can be reduced to KB satisfiability. Consider two views, $V_1 = \langle v_1, \exists \vec{y_1}.conj_1(\vec{x}, \vec{y_1}) \rangle$ and $V_2 = \langle v_2, \exists \vec{y_2}.conj_2(\vec{x}, \vec{y_2}) \rangle$, we can check disjointness $V_1 \cap V_2 =_{\mathcal{T}} \emptyset$, by applying the boolean procedure ViewDisjointness$(\mathcal{T}, V_1, V_2)$, defined as follows:

1. Freeze the free variables $\vec{x}$, by replacing them with fresh constants $\vec{a}$. Notice that we obtain two boolean queries $v_{i,\vec{a}}$ of the form $\exists \vec{y}.conj_i(\vec{a}, \vec{y})$.
2. Compute the canonical ABoxes $\mathcal{A}_{v_{1,\vec{a}}}$ of $v_{1,\vec{a}}$ and $\mathcal{A}_{v_{2,\vec{a}}}$ of $v_{2,\vec{a}}$.
3. Return true iff $\langle \mathcal{T}, \mathcal{A}_{v_{1,\vec{a}}} \cup \mathcal{A}_{v_{2,\vec{a}}} \rangle$ is inconsistent.

Theorem 5. *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox, and V_1, V_2 be two views of the same arity, then $V_1 \cap V_2 =_{\mathcal{T}} \emptyset$ iff* ViewDisjointness$(\mathcal{T}, V_1, V_2)$ *evaluates to true.*

Proof. From the definition of view disjointness we have that, for every model $\mathcal{I}$ of $\mathcal{T}$, $V_1 \cap V_2 =_{\mathcal{T}} \emptyset$ if and only if $v_1^{\mathcal{I}} \cap v_2^{\mathcal{I}} = (\exists \vec{y_1}.conj_1(\vec{x}, \vec{y_1}))^{\mathcal{I}} \cap (\exists \vec{y_2}.conj_2(\vec{x}, \vec{y_2}))^{\mathcal{I}} = (\exists \vec{y_1}, \vec{y_2}.(conj_1(\vec{x}, \vec{y_1}) \wedge conj_2(\vec{x}, \vec{y_2})))^{\mathcal{I}} = \emptyset$, that is, the conjunction of $def(V_1)$ and $def(V_2)$ is inconsistent w.r.t. $\mathcal{T}$. Moreover, it is easy to see that the canonical ABox of the conjunction of $def(V_1)$ and $def(V_2)$, after replacing the free variables with fresh constants $\vec{a}$ is equal to $\mathcal{A}_{v_{1,\vec{a}}} \cup \mathcal{A}_{v_{2,\vec{a}}}$. Hence, we can apply the same reasoning that we adopted for view consistency, and get the thesis. □

Considering the complexity of KB consistency in *DL-Lite*$_R$, it is immediate to assess the complexity of view disjointness.

Theorem 6. *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox, and V_1, V_2 be two views of the same arity, then $V_1 \wedge V_2 =_{\mathcal{T}} \emptyset$ is in* PTIME *in $\mathcal{T}$, and* AC^0 *in $def(V_1)$ and $def(V_2)$.*

4.4 Other Reasoning Services on Views Definitions

By using the basic reasoning services introduced above, we can define more complex reasoning services on view definitions. Here we discuss a few of them.

We start with view classification, which is one of the key reasoning services available in ontology-based systems. Classification consists typically in computing the hierarchical representation of the subsumption ("is-a") relation between the concepts in the ontology as logical consequences of the TBox assertions [2]. Specifically, *concept classification* is the task of computing all subsumptions $A \sqsubseteq B$ between atomic concepts $A, B \in N_c$ of the ontology such that $\mathcal{T} \models A \sqsubseteq B$.

Classification can be extended to views. *View classification* corresponds to systematically checking whether $V_1 \subseteq_{\mathcal{T}} V_2$ for all views in $\mathcal{V}$. This is of particular interest to understand if a certain view can be used for approximating another one without actually computing the latter, for example.

It is also of interest to check how *unary views are classified w.r.t. concepts* by systematically checking whether $V \subseteq_{\mathcal{T}} V_A$ and $V_A \subseteq_{\mathcal{T}} V$ for all unary views and atomic concepts A, where V_A is the trivial unary view $V_A = \langle v_A, A(x) \rangle$. In this way we can understand how unary view relate to the concepts in the ontology. Note that a similar classification can be done with binary views and roles.

Finally, we can also check containment and disjointness w.r.t. to projections over views. One interesting use of this possibility is to actually *type each component of a view* with the most specific subsumer and subsumee atomic concept to understand which concepts an n-ary view relates to.

5 Conclusions and Further Work

We have presented a proposal for enriching Description Logic knowledge bases with views. Our paper can be seen as a first step in a path towards a complete approach to the management of views in Description Logic and in Ontology-Based Data Access systems. There are many interesting directions to explore following the path. Probably, the most relevant is the introduction of views defined using epistemic operators in the body, in the spirit of [5]. The use of epistemic logic allows more sophisticated views to be defined. For example, one could use the epistemic operator for distinguishing between semantically different concepts expressed as views, such as "exam given by a professor working for a department", and "exam given by a professor for which the corresponding department is known". The challenge in this case is to devise algorithms for reasoning about ordinary and epistemic views in the context of the same knowledge base.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995), http://webdam.inria.fr/Alice/
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
3. Beeri, C., Levy, A.Y., Rousset, M.: Rewriting queries using views in description logics. In: Mendelzon, A.O., Özsoyoglu, Z.M. (eds.) Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA. pp. 99–108. ACM Press (1997), https://doi.org/10.1145/263661.263673
4. Buchheit, M., Donini, F.M., Nutt, W., Schaerf, A.: A refined architecture for terminological systems: Terminology = schema + views. Artif. Intell. 99(2), 209–260 (1998), https://doi.org/10.1016/S0004-3702(97)00079-9
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Eql-lite: Effective first-order query processing in description logics. In: Veloso, M.M. (ed.) IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. pp. 274–279 (2007), http://ijcai.org/Proceedings/07/Papers/042.pdf
6. 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. Autom. Reasoning 39(3), 385–429 (2007), https://doi.org/10.1007/s10817-007-9078-x
7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. Artif. Intell. 195, 335–360 (2013)
8. Calvanese, D., De Giacomo, G., Lenzerini, M.: Answering queries using views over description logics knowledge bases. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA. pp. 386–391. AAAI Press / The MIT Press (2000), http://www.aaai.org/Library/AAAI/2000/aaai00-059.php
9. Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: View-based query answering in description logics: Semantics and complexity. J. Comput. Syst. Sci. 78(1), 26–46 (2012), https://doi.org/10.1016/j.jcss.2011.02.011
10. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semantics 10, 133–173 (2008)