Query Rewriting under Extensional Constraints in DL-Lite

Riccardo Rosati

DIAG, Sapienza Università di Roma, Italy
lastname@dis.uniroma1.it

1 Introduction

The *DL-Lite* family of description logics [4, 2] is currently one of the most studied ontology specification languages. *DL-Lite* constitutes the basis of the OWL2 QL language [1], which is part of the standard W3C OWL2 ontology specification language. The distinguishing feature of *DL-Lite* is to identify ontology languages in which expressive queries, in particular, unions of conjunctive queries (UCQs), over the ontology can be efficiently answered. Therefore, query answering is the most studied reasoning task in *DL-Lite* (see, e.g., [16, 11, 8, 18, 7, 6, 5]).

The most common approach to query answering in *DL-Lite* is through query rewriting. This approach consists of computing a so-called *perfect rewriting* of the query with respect to a TBox: the perfect rewriting of a query q for a TBox $\mathcal{T}$ is a query q' that can be evaluated on the ABox only and produces the same results as if q were evaluated on both the TBox and the ABox. This approach is particularly interesting in *DL-Lite*, because, for every UCQ q, query q' can be expressed in first-order logic (i.e., SQL), therefore query answering can be delegated to a relational DBMS, since it can be reduced to the evaluation of an SQL query on the database storing the ABox.

The shortcoming of the query rewriting approach is that the size of the rewritten query may be exponential with respect to the size of the original query. In particular, this is true when the rewritten query is in disjunctive normal form, i.e., is an UCQ. On the other hand, [6] shows the existence of polynomial perfect rewritings of the query in nonrecursive datalog.

However, it turns out that the disjunctive normal form is necessary for practical applications of the query rewriting technique, since queries of more complex forms, once translated in SQL, produce queries with nested subexpressions that, in general, cannot be evaluated efficiently by current DBMSs. So, while in some cases resorting to more compact and structurally more complex perfect rewritings may be convenient, in general this strategy does not solve the problem of arriving at an SQL expression that can be effectively evaluated on the database.

In this scenario, a very interesting way to limit the size of the rewritten UCQ has been presented in [13]. This approach proposes the use of the so-called *ABox dependencies* to optimize query rewriting in *DL-Lite*$_{\mathcal{A}}$. ABox dependencies are inclusions between concepts and roles which are interpreted as integrity constraints over the ABox: in other words, the ABox is guaranteed to satisfy such constraints. For this reason, in this paper we also call ABox dependencies *extensional constraints*. In the presence of such constraints, the query answering process can be optimized, since this additional knowledge about the extensions of concepts and roles in the ABox can be exploited for optimizing query answering. Intuitively, the presence of ABox dependencies acts in a complementary way with respect to TBox assertions: while the latter complicate query rewriting, the former simplify it, since they state that some of the TBox assertions are already satisfied by the ABox.

As explained in [13], ABox dependencies have a real practical interest, since they naturally arise in many applications of ontologies, and in particular in ontology-based data access (OBDA) applications, in which a DL ontology acts as a virtual global schema for accessing data stored in external sources, and such sources are connected through declarative mappings to the global ontology. It turns out that, in practical cases, many ABox dependencies may be (automatically) derived from the mappings between the ontology and the data sources.

In this paper, we present an approach that follows the ideas of [13]. More specifically, we present Prexto, an algorithm for computing a perfect rewriting of a UCQ in the description logic *DL-Lite*$_\mathcal{A}$. Prexto is based on the query rewriting algorithm Presto [16]: with respect to the previous technique, Prexto has been designed to fully exploit the presence of extensional constraints to optimize the size of the rewriting; moreover, differently from Presto, it also uses concept and role disjointness assertions, as well as role functionality assertions, to reduce the size of the rewritten query. As already observed in [13], the way extensional constraints interact with reasoning, and in particular query answering, is not trivial at all: e.g., [13] defines a complex condition for the deletion of a concept (or role) inclusion from the TBox due to the presence of extensional constraints. In our approach, we use extensional constraints in a very different way from [13], which uses such constraints to "deactivate" corresponding TBox assertions in the TBox: conversely, we are able to define significant query minimizations even for extensional constraints for which there exists no corresponding TBox assertions. Based on these ideas, we define the Prexto algorithm: in particular, we restructure and extend the Presto query rewriting algorithm to fully exploit the presence of extensional constraints. Finally, we show that the above optimizations allow Prexto to outperform the existing query rewriting techniques for *DL-Lite* in practical cases. In particular, we compare Prexto with Presto and with the optimization presented in [13].

This paper is an extended abstract of [15].

2 Preliminaries

We assume the reader is familiar with the basics of DLs as well as with *DL-Lite*$_\mathcal{A}$ [12].

Given an ABox $\mathcal{A}$, we denote by $\mathcal{I}_\mathcal{A}$ the *DL-Lite*$_\mathcal{A}$ interpretation such that, for every concept instance assertion $C(a)$, $a^\mathcal{I} \in C^\mathcal{I}$ iff $C(a) \in \mathcal{A}$, for every role instance assertion $R(a,b)$, $\langle a^\mathcal{I}, b^\mathcal{I}\rangle^\mathcal{I} \in R^\mathcal{I}$ iff $R(a,b) \in \mathcal{A}$, and for every attribute instance assertion $U(a,b)$, $\langle a^\mathcal{I}, b^\mathcal{I}\rangle^\mathcal{I} \in U^\mathcal{I}$ iff $U(a,b) \in \mathcal{A}$.

A conjunctive query (CQ) q is an expression of the form $q(\boldsymbol{x}) \leftarrow \alpha_1, \ldots, \alpha_n$, where $\boldsymbol{x}$ is a tuple of variables, and every α_i is an atom whose predicate is a concept name or a role name or an attribute name, and whose arguments are either variables or constants, such that every variable occurring in $\boldsymbol{x}$ also occurs in at least one α_i. The variables occurring in $\boldsymbol{x}$ are called the *distinguished variables* of q, while the variables occurring in some α_i but not in $\boldsymbol{x}$ are called the *existential variables* of q. The predicate q is called the *predicate* of the query, and the number of elements of $\boldsymbol{x}$ is called the *arity* of q. A CQ is a *Boolean* CQ if it has no distinguished variables.

A union of conjunctive queries (UCQ) Q is a set of conjunctive queries of the same arity and having the same query predicate. A UCQ Q is a Boolean UCQ if every CQ belonging to Q is Boolean.

Given a CQ q of arity n, we denote by $q(\boldsymbol{c})$ the Boolean CQ obtained from q by replacing the distinguished variables in $\boldsymbol{x}$ with the constants in the n-tuple of constants $\boldsymbol{c}$. Given a CQ q of arity n, the evaluation of q in $\mathcal{I}$, denoted by $\mathit{eval}(q, \mathcal{I})$, is the set of n tuples of constants $\boldsymbol{c}$ such that $\mathcal{I}$ satisfies the first-order sentence $q(\boldsymbol{c})$. The evaluation of a UCQ Q in $\mathcal{I}$, denoted by $\mathit{eval}(Q, \mathcal{I})$, is the set $\bigcup_{q \in Q} \mathit{eval}(q, \mathcal{I})$. The set of *certain answers* to a UCQ Q over a *DL-Lite*$_{\mathcal{A}}$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, denoted by $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle)$, is the set of tuples $\bigcap_{\mathcal{I} \in \mathit{Mod}(\langle \mathcal{T}, \mathcal{A} \rangle)} \mathit{eval}(Q, \mathcal{I})$.

Given a UCQ Q and a TBox $\mathcal{T}$, a UCQ Q' is a *perfect rewriting* of Q with respect to $\mathcal{T}$ if, for every ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent, $cert(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = \mathit{eval}(Q, \mathcal{I}_{\mathcal{A}})$. The above notion of perfect rewriting immediately extends to any query language for which the evaluation *eval* of queries on a first-order interpretation is defined. We remark that many algorithms are available to compute perfect rewritings in *DL-Lite* logics (e.g., [4, 12, 16, 11, 7, 6]).

In the following, for ease of exposition, we will not consider attributes in *DL-Lite*$_{\mathcal{A}}$ ontologies. However, all the algorithms and results that we present in this paper can be immediately extended to handle attributes (since attributes can essentially be treated in a way analogous to roles).

3 Extensional Constraints

We now define the notion of EBox, which constitutes a set of extensional constraints, i.e., constraints over the ABox. The idea of EBox has been originally introduced in [13], under the name of *ABox dependencies*.

The following definitions are valid for every DL, under the assumption that the assertions are divided into extensional assertions and intensional assertions, and extensional assertions correspond to atomic instance assertions.

Given a set of intensional assertions $\mathcal{N}$ and an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ *satisfies* $\mathcal{N}$ if $\mathcal{I}$ satisfies every assertion in $\mathcal{N}$.

An *extensional constraint box*, or simply *EBox*, is a set of intensional assertions. Notice that, from the syntactic viewpoint, an EBox is identical to a TBox. Therefore, entailment of an assertion ϕ with respect to an EBox $\mathcal{E}$ (denoted by $\mathcal{E} \models \phi$) is defined exactly in the same way as in the case of TBoxes.

Given an ABox $\mathcal{A}$ and an EBox $\mathcal{E}$, we say that $\mathcal{A}$ *is valid for* $\mathcal{E}$ if $\mathcal{I}_{\mathcal{A}}$ satisfies $\mathcal{E}$.

Definition 1. ***(Admissible ABox)*** *Given a TBox $\mathcal{T}$ and an EBox $\mathcal{E}$, an ABox $\mathcal{A}$ is an* admissible ABox for $\mathcal{T}$ and $\mathcal{E}$ *if $\mathcal{A}$ is consistent with $\mathcal{T}$ and $\mathcal{A}$ is valid for $\mathcal{E}$. We denote with ADM$(\mathcal{T}, \mathcal{E})$ the set of ABoxes $\mathcal{A}$ that are admissible for $\mathcal{T}$ and $\mathcal{E}$.*

Informally, an EBox acts as a set of *integrity constraints* over the ABox. Differently from other recent approaches that have proposed various forms of integrity constraints for DL ontologies (e.g., [9, 17]), an EBox constrains the ABox while totally discarding the TBox, since the notion of validity with respect to an EBox only considers the ABox.

We are now ready to define the notion of perfect rewriting in the presence of both a TBox and an EBox.

Definition 2. ***(Perfect rewriting in the presence of an EBox)*** *Given a TBox $\mathcal{T}$, an EBox $\mathcal{E}$, and a UCQ Q, a FOL query ϕ is a* perfect rewriting of Q with respect to $\langle \mathcal{T}, \mathcal{E} \rangle$ *if, for every ABox $\mathcal{A} \in ADM(\mathcal{T}, \mathcal{E})$, $\langle \mathcal{T}, \mathcal{A} \rangle \models Q$ iff $\mathcal{I}_{\mathcal{A}} \models \phi$.*

The above definition establishes a natural notion of perfect rewriting in the presence of an EBox $\mathcal{E}$. Since $\mathcal{E}$ constrains the admissible ABoxes, the more selective is $\mathcal{E}$ (for the same TBox $\mathcal{T}$), the more restricted the set $ADM(\mathcal{T}, \mathcal{E})$ is. If for instance, $\mathcal{E}, \mathcal{E}'$ are two EBoxes such that $\mathcal{E} \subset \mathcal{E}'$, we immediately get from the above definitions that $ADM(\mathcal{T}, \mathcal{E}) \supseteq ADM(\mathcal{T}, \mathcal{E}')$. Now, let Q be a UCQ, let ϕ be a perfect rewriting of Q with respect to $\langle \mathcal{T}, \mathcal{E} \rangle$ and let ϕ' be a perfect rewriting of Q with respect to $\langle \mathcal{T}, \mathcal{E}' \rangle$: ϕ will have to satisfy the condition $\langle \mathcal{T}, \mathcal{A} \rangle \models Q$ iff $\mathcal{I}_{\mathcal{A}} \models \phi$ for more ABoxes $\mathcal{A}$ than query ϕ'. Consequently, ϕ will have to be a more complex query than ϕ'. Therefore, larger EBoxes in principle allow for obtaining simpler perfect rewritings.

As already explained, the goal of this paper is to use extensional constraints to optimize query rewriting in *DL-Lite*$_{\mathcal{A}}$. An intuitive explanation of how extensional constraints allow for simplifying query rewriting can be given by the following very simple example. Suppose we are given a TBox $\{\mathit{Student} \sqsubseteq \mathit{Person}\}$, an empty EBox $\mathcal{E}_0$, and an EBox $\mathcal{E}_1 = \{\mathit{Student} \sqsubseteq \mathit{Person}\}$. Now, given a query $q(x) \leftarrow \mathit{Person}(x)$, a perfect rewriting of this query with respect to $\langle \mathcal{T}, \mathcal{E}_0 \rangle$ is the UCQ $\{q(x) \leftarrow \mathit{Person}(x) \quad q(x) \leftarrow \mathit{Student}(x)\}$, while a perfect rewriting of query q with respect to $\langle \mathcal{T}, \mathcal{E}_1 \rangle$ is the query q itself. Namely, under the EBox $\mathcal{E}_1$ we can ignore the TBox concept inclusion $\mathit{Student} \sqsubseteq \mathit{Person}$, since it is already satisfied by the ABox.

However, as already explained in [13], we can not always ignore TBox assertions that also appear in the EBox (and are thus already satisfied by the ABox). For instance, let q be the query $q \leftarrow C(x)$. If the TBox $\mathcal{T}$ contains the assertions $\exists R \sqsubseteq C$ and $D \sqsubseteq \exists R^-$ and the EBox $\mathcal{E}$ contains the assertion $\exists R \sqsubseteq C$, we cannot ignore this last inclusion when computing a perfect rewriting of q (or when answering query q). In fact, suppose the ABox is $\{D(a)\}$: then $\mathcal{A} \in ADM(\mathcal{T}, \mathcal{E})$ and query q is entailed by $\langle \mathcal{T}, \mathcal{A} \rangle$. But actually q is not entailed by $\langle \mathcal{T}', \mathcal{A} \rangle$ where $\mathcal{T}' = \mathcal{T} - \mathcal{E}$. From the query rewriting viewpoint, a perfect rewriting of q with respect to $\mathcal{T}$ is the UCQ $\{q \leftarrow C(x) \quad q \leftarrow R(x, y) \quad q \leftarrow D(y)\}$, while a perfect rewriting of q with respect to $\mathcal{T}'$ is the query q itself. And of course, the ABox $\mathcal{A}$ shows that this last query is not a perfect rewriting of q with respect to $\langle \mathcal{T}, \mathcal{E} \rangle$. Therefore, also when computing a perfect rewriting, we cannot simply ignore the inclusions of the TBox that are already satisfied by the ABox (i.e., that belong to the EBox).

The example above shows that we need to understand under which conditions we are allowed to use extensional constraints to optimize query rewriting.

4 Prexto

In this section we present the algorithm Prexto (Perfect Rewriting under EXTensional cOnstraints). Prexto makes use of the algorithm Presto, originally defined in [16], which computes a nonrecursive datalog program constituting a perfect rewriting of a

Algorithm $\mathsf{Presto}(Q, \mathcal{T})$
Input: UCQ Q, *DL-Lite*$_R$ TBox $\mathcal{T}$
Output: nr-datalog query Q'
begin
  $Q' = \mathsf{Rename}(Q)$;
  $Q' = \mathsf{DeleteUnboundVars}(Q')$;
  $Q' = \mathsf{DeleteRedundantAtoms}(Q', \mathcal{T})$;
  $Q' = \mathsf{Split}(Q')$;
  repeat
    if there exist $r \in Q'$ and ej-var x in r
      such that $\mathsf{Eliminable}(x, r, \mathcal{T}) = true$ and x has not already been eliminated from r
    then begin
      $Q'' = \mathsf{EliminateEJVar}(r, x, \mathcal{T})$;
      $Q'' = \mathsf{DeleteUnboundVars}(Q'')$;
      $Q'' = \mathsf{DeleteRedundantAtoms}(Q'', \mathcal{T})$;
      $Q' = Q' \cup \mathsf{Split}(Q'')$
    end
  until Q' has reached a fixpoint;
  for each OA-predicate p_α^n occurring in Q'
  do $Q' = Q' \cup \mathsf{DefineAtomView}(p_\alpha^n, \mathcal{T})$
end

Fig. 1. The original Presto algorithm [16].

UCQ Q with respect to a *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$. The algorithm Presto is reported in Figure 1. We refer the reader to [16] for a detailed explanation of the algorithm. For our purposes, it suffices to remind that the program returned by Presto uses auxiliary datalog predicates, called ontology-annotated (OA) predicates, to represent every basic concept and basic role that is involved in the query rewriting. E.g., the basic concept B is represented by the OA-predicate p_B^1, while the basic role R is represented by the OA-predicate p_R^2, where the superscript represents the arity of the predicate (actually, to handle Boolean subqueries, also 0-ary OA-predicates, i.e., predicates with no arguments, are defined: we refer the reader to [16] for more details).

In the following, we modify the algorithm Presto. In particular, we make the following changes. First, the final **for each** cycle of the algorithm (cf. Figure 1) is not executed: i.e., the rules defining the OA-predicates are not added to the returned program. Second, the algorithm DeleteRedundantAtoms is modified to take into account the presence of disjointness assertions and role functionality assertions in the TBox. More precisely, the following simplification rules are added to algorithm $\mathsf{DeleteRedundantAtoms}(Q', \mathcal{T})$ (in which we denote basic concepts by B, C, role names by R, S, and datalog rules by the symbol r):

1. if $p_R^2(t_1, t_2)$ and $p_S^2(t_1, t_2)$ occur in r and $\mathcal{T} \models R \sqsubseteq \neg S$, then delete r from Q';
2. if $p_B^1(t)$ and $p_C^1(t)$ occur in r and $\mathcal{T} \models B \sqsubseteq \neg C$, then delete r from Q';
3. if $p_R^2(t_1, t_2)$ and $p_C^1(t_1)$ occur in r and $\mathcal{T} \models \exists R \sqsubseteq \neg C$, then delete r from Q';
4. if p_α^0 and p_β^0 occur in r and $\mathcal{T} \models \alpha^0 \sqsubseteq \neg\beta^0$, then delete r from Q';
5. if $p_B^1(t)$ and p_α^0 occur in r and $\mathcal{T} \models B^0 \sqsubseteq \neg\alpha^0$, then delete r from Q';
6. if $p_R^2(t_1, t_2)$ and p_α^0 occur in r and $\mathcal{T} \models R^0 \sqsubseteq \neg\alpha^0$, then delete r from Q';

Algorithm $\mathsf{Prexto}(Q, \mathcal{T}, \mathcal{E})$
Input: UCQ Q, *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$, *DL-Lite*$_\mathcal{A}$ EBox $\mathcal{E}$
Output: UCQ Q'
begin
  $P = \mathsf{Presto}(Q, \mathcal{T})$;
  $P' = \emptyset$;
  for each OA-predicate P_R^2 occurring in P **do**
    $\Phi = \mathsf{MinimizeViews}(R, \mathcal{E}, \mathcal{T})$;
    $P' = P' \cup \{p_B^2(x,y) \leftarrow S(x,y) \mid S$ is a role name and $S \in \Phi\}$
        $\cup \{p_B^2(x,y) \leftarrow S(y,x) \mid S$ is a role name and $S^- \in \Phi\}$;
  for each OA-predicate P_B^1 occurring in P **do**
    $\Phi = \mathsf{MinimizeViews}(B, \mathcal{E}, \mathcal{T})$;
    $P' = P' \cup \{p_B^1(x) \leftarrow C(x) \mid C$ is a concept name and $C \in \Phi\}$
        $\cup \{p_B^1(x) \leftarrow R(x,y) \mid \exists R \in \Phi\} \cup \{p_B^1(x) \leftarrow R(y,x) \mid \exists R^- \in \Phi\}$;
  for each OA-predicate P_N^0 occurring in P **do**
    $\Phi = \mathsf{MinimizeViews}(N^0, \mathcal{E}, \mathcal{T})$;
    $P' = P' \cup \{p_N^0 \leftarrow C(x) \mid C$ is a concept name and $C^0 \in \Phi\}$
        $\cup \{p_N^0 \leftarrow R(x,y) \mid R$ is a role name and $R^0 \in \Phi\}$;
  $P'' = P \cup P'$;
  $Q' = \mathsf{Unfold}(P'')$;
  $Q' = \mathsf{DeleteRedundantAtoms}(Q', \mathcal{E})$;
  return Q'
end

Fig. 2. The Prexto algorithm.

7. if $p_R^2(t_1, t_2)$ and $p_R^2(t_1, t_2')$ (with $t_2 \neq t_2'$) occur in r and $(\mathsf{funct}\ R) \in \mathcal{T}$, then, if t_2 and t_2' are two different constants, then delete r from Q'; otherwise, replace r with the rule $\sigma(r)$, where σ is the substitution which poses t_2 equal to t_2'.

Analogous simplification rules (which can be immediately derived) hold when R, S are inverse roles in cases 1, 3 and 7.

Example 1. Let us show the effect of the new transformations added to DeleteRedundantAtoms through two examples. First, suppose $\mathcal{T} = \{B \sqsubseteq \neg B', (\mathsf{funct}\ R)\}$ and suppose r is the rule $q(x) \leftarrow p_B^1(y), p_R^2(x,y), p_R^2(x,z), p_{B'}^1(z)$. First, the above rule 7 of algorithm DeleteRedundantAtoms can be applied, which transforms r into the rule $q(x) \leftarrow p_B^1(y), p_R^2(x,y), p_{B'}^1(y)$. Then, the above rule 2 of algorithm DeleteRedundantAtoms can be applied, hence this rule is deleted from the program. Intuitively, this is due to the fact that this rule looks for elements belonging both to concept B and to concept B', which is impossible because the disjointness assertion $B \sqsubseteq \neg B'$ is entailed by the TBox $\mathcal{T}$. Therefore, it is correct to delete the rule from the program. □

From now on, when we speak about Presto we refer to the above modified version of the algorithm, and when we speak about DeleteRedundantAtoms we refer to the above modified version which takes into account disjointness and functionality assertions.

The Prexto algorithm is defined in Figure 2. The algorithm is constituted of the following four steps:

Algorithm MinimizeViews$(B, \mathcal{E}, \mathcal{T})$
Input: basic concept (or basic role, or 0-ary predicate) B,
  DL-Lite$_\mathcal{A}$ EBox $\mathcal{E}$, *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$
Output: set of basic concepts (or basic roles, or 0-ary predicates) Φ''
begin
  $\Phi = \{B' \mid \mathcal{T} \models B' \sqsubseteq B\}$;
  $\Phi' = \emptyset$;
  for each $B' \in \Phi$ **do**
    if there exists $B'' \in \Phi$ such that $\mathcal{E} \models B' \sqsubseteq B''$ and $\mathcal{E} \not\models B'' \sqsubseteq B'$
    then $\Phi' = \Phi' \cup \{B'\}$;
  $\Phi'' = \Phi - \Phi'$;
  while there exist $B, B' \in \Phi'$ such that $B \neq B'$ and $\mathcal{E} \models B \sqsubseteq B'$ and $\mathcal{E} \models B' \sqsubseteq B$
  do $\Phi'' = \Phi'' - \{B'\}$;
  return Φ''
end

Fig. 3. The MinimizeViews algorithm.

1. the nonrecursive datalog program P is computed by executing the Presto algorithm. This program P is not a perfect rewriting of Q yet, since the definition of the intermediate OA-predicates is missing;
2. the program P' is then constructed (by the three **for each** cycles of the program). This program contain rules defining the intermediate OA-predicates, i.e., the concept and role assertions used in the program P. To compute such rules, the algorithm makes use of the procedure MinimizeViews, reported in Figure 3. This procedure takes as input a basic concept (respectively, a basic role) B and computes a minimal subset Φ'' of the set Φ of the subsumed basic concepts (respectively, subsumed basic roles) of B which *extensionally cover* the set Φ, as explained below.
3. then, the overall nonrecursive datalog program $P \cup P'$ is unfolded, i.e., turned into a UCQ Q'. This is realized by the algorithm Unfold which corresponds to the usual unfolding of a nonrecursive program;
4. finally, the UCQ Q' is simplified by executing the algorithm DeleteRedundantAtoms which takes as input the UCQ Q' and the EBox $\mathcal{E}$ (notice that, conversely, the first execution of DeleteRedundantAtoms within the Presto algorithm uses the TBox $\mathcal{T}$ as input).

Notice that the bottleneck of the whole process is the above step 3, since the number of conjunctive queries generated by the unfolding may be exponential with respect to the length of the initial query Q (in particular, it may be exponential with respect to the maximum number of atoms in a conjunctive query of Q). As shown by the following example, the usage of extensional constraints done at step 2 through the MinimizeViews algorithm is crucial to handle the combinatorial explosion of the unfolding.

Example 2. Let $\mathcal{T}$ be the following *DL-Lite*$_\mathcal{A}$ TBox:

$\mathit{Company} \sqsubseteq \exists \mathit{givesHighSalaryTo}^-$	$\mathit{FulltimeStudent} \sqsubseteq \mathit{Unemployed}$
$\exists \mathit{givesHighSalaryTo}^- \sqsubseteq \mathit{Manager}$	$\mathit{FulltimeStudent} \sqsubseteq \mathit{Student}$
$\mathit{Manager} \sqsubseteq \mathit{Employee}$	$\mathit{isBestFriendOf} \sqsubseteq \mathit{knows}$
$\mathit{Employee} \sqsubseteq \mathit{HasJob}$	$(\mathsf{funct}\ \mathit{isBestFriendOf})$
$\exists \mathit{receivesGrantFrom} \sqsubseteq \mathit{StudentWithGrant}$	$(\mathsf{funct}\ \mathit{isBestFriendOf}^-)$
$\mathit{StudentWithGrant} \sqsubseteq \mathit{FulltimeStudent}$	$\mathit{HasJob} \sqsubseteq \neg \mathit{Unemployed}$

Moreover, let $E_1, \ldots, E_4$ be the following concept inclusions:

$E_1 = \mathit{FulltimeStudent} \sqsubseteq \mathit{StudentWithGrant}$ $\quad$ $E_3 = \mathit{HasJob} \sqsubseteq \mathit{Employee}$
$E_2 = \exists \mathit{receivesGrantFrom} \sqsubseteq \mathit{StudentWithGrant}$ $\quad$ $E_4 = \mathit{Manager} \sqsubseteq \mathit{Employee}$

and let $\mathcal{E}_1 = \{E_1\}, \mathcal{E}_2 = \{E_1, E_2\}, \mathcal{E}_3 = \{E_1, E_2, E_3\}, \mathcal{E}_4 = \{E_1, E_2, E_3, E_4\}$. Finally, let q_0, q_1, q_2, q_3 be the following simple queries:

$$
\begin{aligned}
q_0(x) &\leftarrow \mathit{Student}(x)\\
q_1(x) &\leftarrow \mathit{Student}(x), \mathit{knows}(x, y), \mathit{HasJob}(y)\\
q_2(x) &\leftarrow \mathit{Student}(x), \mathit{knows}(x, y), \mathit{HasJob}(y), \mathit{knows}(x, z), \mathit{Unemployed}(z)\\
q_3(x) &\leftarrow \mathit{Student}(x), \mathit{knows}(x, y), \mathit{HasJob}(y), \mathit{knows}(x, z), \mathit{Unemployed}(z),\\
&\qquad \mathit{knows}(x, w), \mathit{Student}(w)
\end{aligned}
$$

Let us focus on query q_1 and let us consider the empty EBox. In this case, during the execution of Prexto$(q_1, \mathcal{T}, \emptyset)$ the algorithm MinimizeViews simply computes the subsumed sets of *Student*, *knows*, *HasJob*, which are, respectively:

$$
\begin{aligned}
&\mathsf{MinimizeViews}(\mathit{Student}, \emptyset, \mathcal{T}) =\\
&\qquad \{\mathit{Student},\ \mathit{FulltimeStudent},\ \mathit{StudentWithGrant},\ \exists \mathit{receivesGrantFrom}\}\\
&\mathsf{MinimizeViews}(\mathit{knows}, \emptyset, \mathcal{T}) = \{\mathit{knows},\ \mathit{isBestFriendOf}\}\\
&\mathsf{MinimizeViews}(\mathit{HasJob}, \emptyset, \mathcal{T}) = \{\mathit{HasJob},\ \mathit{Employee},\ \mathit{Manager},\ \exists \mathit{givesHighSalaryTo}^-\}
\end{aligned}
$$

Since two sets are constituted of four predicates and one is constituted of two predicates, the UCQ returned by the unfolding step in Prexto$(q_1, \mathcal{T}, \mathcal{E})$ contains 32 CQs. This is also the size of the final UCQ, since in this case no optimizations are computed by the algorithm DeleteRedundantAtoms, because both the disjointness assertion and the role functionality assertions of $\mathcal{T}$ have no impact on the rewriting of query q_1.

Conversely, let us consider the EBox $\mathcal{E}_4$: during the execution of Prexto$(q_1, \mathcal{T}, \mathcal{E})$, we obtain the following sets from the execution of the algorithm MinimizeViews::

$$
\begin{aligned}
&\mathsf{MinimizeViews}(\mathit{Student}, \mathcal{E}_4, \mathcal{T}) = \{\mathit{Student},\ \mathit{StudentWithGrant}\}\\
&\mathsf{MinimizeViews}(\mathit{knows}, \mathcal{E}_4, \mathcal{T}) = \{\mathit{knows},\ \mathit{isBestFriendOf}\}\\
&\mathsf{MinimizeViews}(\mathit{HasJob}, \mathcal{E}_4, \mathcal{T}) = \{\mathit{Employee},\ \exists \mathit{givesHighSalaryTo}^-\}
\end{aligned}
$$

Thus, the algorithm MinimizeViews returns only two predicates for *Student* and only two predicates for *HasJob*. Therefore, the final unfolded UCQ is constituted of 8 CQs (since, as above explained, the final call to DeleteRedundantAtoms does not produce any optimization). □

It is possible to prove correctness of Prexto [15], which in turn implies the following property, which states that the computational cost of Prexto is no worse than all known query rewriting techniques for *DL-Lite*$_{\mathcal{A}}$ which compute UCQs.

Theorem 1. *Prexto$(Q, \mathcal{T}, \mathcal{E})$ runs in polynomial time with respect to the size of $\mathcal{T} \cup \mathcal{E}$, and in exponential time with respect to the maximum number of atoms in a conjunctive query in the UCQ Q.*

query	algorithm	$\mathcal{E} = \emptyset$	$\mathcal{E} = \mathcal{E}_1$	$\mathcal{E} = \mathcal{E}_2$	$\mathcal{E} = \mathcal{E}_3$	$\mathcal{E} = \mathcal{E}_4$
q_0	Presto+unfolding	4	4	4	4	4
q_0	TBox-min	4	4	4	4	4
q_0	Prexto-noEBox	4	4	4	4	4
q_0	Prexto-noDisj	4	3	2	2	2
q_0	Prexto-full	4	3	2	2	2
q_1	Presto+unfolding	32	32	32	32	32
q_1	TBox-min	32	32	32	32	32
q_1	Prexto-noEBox	32	32	32	32	32
q_1	Prexto-noDisj	32	24	16	12	8
q_1	Prexto-full	32	24	16	12	8
q_2	Presto+unfolding	256	256	256	256	256
q_2	TBox-min	256	256	256	256	256
q_2	Prexto-noEBox	224	224	224	224	224
q_2	Prexto-noDisj	256	144	64	48	32
q_2	Prexto-full	224	126	106	42	28
q_3	Presto+unfolding	2048	2048	2048	2048	2048
q_3	TBox-min	2048	2048	2048	2048	2048
q_3	Prexto-noEBox	1584	1584	1584	1584	1584
q_3	Prexto-noDisj	2048	864	256	192	128
q_3	Prexto-full	1584	708	188	141	94

Fig. 4. Comparison of query rewriting techniques on $\mathcal{T}$, $\mathcal{E}$ and queries q_0, q_1, q_2, q_3.

5 Comparison

We now compare the optimizations introduced by Prexto with some of the current techniques for query rewriting in *DL-Lite*. In particular, we consider the simple *DL-Lite*$_{\mathcal{A}}$ ontology of Example 2 and compare the size of the UCQ rewritings generated by the original Presto algorithm, the rewriting based on the TBox minimization technique TBox-min shown in [13], and the Prexto algorithm. To single out the impact of the different optimizations introduced by Prexto, we present three different execution modalities for Prexto: without considering the EBox (we call this modality Prexto-noEBox); (ii) without considering disjointness axioms and role functionality axioms in the TBox (we call this modality Prexto-noDisj); (iii) and considering all axioms both in the TBox and in the EBox (we call this modality Prexto-full). Moreover, we will consider different EBoxes of increasing size, to better illustrate the impact of the EBox on the size of the rewriting.

The table reported in Figure 4 shows the impact on rewriting queries q_0, q_1, q_2 and q_3 of: (i) the disjointness axiom and the functional role axioms in $\mathcal{T}$; (ii) the EBoxes $\mathcal{E}_1, \ldots, \mathcal{E}_4$. In the table, we denote by Presto+unfolding the UCQ obtained by unfolding the nonrecursive datalog program returned by the Presto algorithm, and denote by TBox-min the execution of Presto+unfolding which takes as input the TBox minimized by the technique presented in [13] using the extensional inclusions in the EBox. These two rows can be considered as representative of the state of the art in query rewriting in *DL-Lite* (with and without extensional constraints): indeed, due to the simple structure of the TBox and the queries, every existing UCQ query rewriting technique for plain *DL-Lite* ontologies (i.e., ontologies without EBoxes) would generate UCQs of size analogous to Presto+unfolding (of course, we are not considering the approaches where the ABox is preprocessed, in which of course much more compact query rewritings can be defined [8, 13]). The third column of the table displays the results when the empty EBox was considered, while the fourth, fifth, sixth, and seventh column respectively report the results when the EBox E_1, E_2, E_3, E_4, was considered. The numbers in these columns represent the size of the UCQ generated when rewriting the query with respect to the TBox $\mathcal{T}$ and the EBox $\mathcal{E}$: more precisely, this number is the number of CQs which constitute the generated UCQ. The results obtained in the case of query q_1 have been explained in Example 2.

The results of Figure 4 clearly show that even a very small number of EBox axioms may have a dramatic impact on the size of the rewritten UCQ, and that this is already the case for relatively short queries (like query q_2): this behavior is even more apparent for longer queries like q_3. In particular, notice that, even when only two extensional inclusions are considered (case $\mathcal{E} = \mathcal{E}_2$), the minimization of the UCQ is already very significant. Moreover, for the queries under examination, extensional inclusions are more effective than disjointness axioms and role functionalily axioms on the minimization of the rewriting size.

The results also show that the technique presented in [13] for exploiting extensional inclusions does not produce any effect in this case. This is due to the fact that the extensional inclusions considered in our experiment do not produce any minimization of the TBox according to the condition expressed in [13]. Conversely, the technique for exploiting extensional constraints of Prexto is indeed effective. For instance, notice that this technique is able to use extensional constraints (like E_2 and E_3) which have no counterpart in the TBox, in the sense that such concept inclusions are not entailed by the TBox $\mathcal{T}$.

Finally, we remark that the above simple example shows a situation which is actually not favourable for the algorithm, since there are very few extensional constraints and short (or even very short) queries: nevertheless, the experimental results show that, even in this setting, our algorithm is able to produce very significant optimizations. Indeed, the ideas which led to the Prexto algorithm came out of a large OBDA project that our research group is currently developing with an Italian Ministry. In this project, several relevant user queries could not be executed by our ontology reasoner (Quonto [3]) due to the very large size of the rewritings produced. For such queries, the minimization of the rewriting produced by the usage of the Prexto optimizations is even more dramatic than the examples reported in the paper, because the queries are more complex (at least ten atoms) and the number of extensional constraints is larger than in the example.

6 Conclusions

In this paper we have presented a query rewriting technique for fully exploiting the presence of extensional constraints in a *DL-Lite*$_{\mathcal{A}}$ ontology. Our technique clearly proves that extensional constraints may produce a dramatic improvement of query rewriting, and consequently of query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies.

We believe that the present approach can be extended in several directions. First, it would be extremely interesting to generalize the Prexto technique to ontology-based data access (OBDA), where the ABox is only virtually specified through declarative mappings over external data sources: as already mentioned in the introduction, in this scenario extensional constraints would be a very natural notion, since they could be automatically derived from the mapping specification. Then, it would be very interesting to extend the usage of extensional constraints beyond *DL-Lite*$_\mathcal{A}$ ontologies: in this respect, a central question is whether existing query rewriting techniques for other description logics (e.g., [11, 14]) can be extended with optimizations analogous to the ones of Prexto. Finally, we plan to fully implement our algorithm within the Quonto/Mastro system [3] for *DL-Lite*$_\mathcal{A}$ ontology management, and to further compare Prexto with other query rewriting techniques for *DL-Lite* (e.g., [11, 5, 10]).

Acknowledgments This research has been partially supported by the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), funded by the EU under FP7 ICT Call 5, 2009.1.2, grant agreement n. FP7-257593.

References

1. OWL 2 web ontology language profiles (2009), http://www.w3.org/TR/owl-profiles/
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The *DL-Lite* family and relations. J. of Artificial Intelligence Research 36, 1–69 (2009)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The Mastro system for ontology-based data access. Semantic Web J. 2(1), 43–53 (2011)
4. 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)
5. Chortaras, A., Trivela, D., Stamou, G.B.: Optimized query rewriting for OWL 2 QL. In: CADE. pp. 192–206 (2011)
6. Gottlob, G., Schwentick, T.: Rewriting ontological queries into small nonrecursive datalog programs. In: Proc. of DL 2011 (2011)
7. Kikot, S., Kontchakov, R., Zakharyaschev, M.: On (in)tractability of OBDA with OWL2QL. In: Proc. of DL 2011 (2011)
8. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in *DL-Lite*. In: Proc. of KR 2010. pp. 247–257 (2010)
9. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. J. of Web Semantics 7(2), 74–89 (2009)
10. Orsi, G., Pieris, A.: Optimizing query answering under ontological constraints. PVLDB 4(11), 1004–1015 (2011)
11. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. of Applied Logic 8(2), 186–209 (2010)
12. 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)
13. Rodriguez-Muro, M., Calvanese, D.: Dependencies: Making ontology based data access work in practice. In: Proc. of AMW 2011 (2011)
14. Rosati, R.: On conjunctive query answering in $\mathcal{EL}$. In: Proc. of DL 2007. CEUR, `ceur-ws.org`, vol. 250, pp. 451–458 (2007)
15. Rosati, R.: Prexto: Query rewriting under extensional constraints in *DL-Lite*. In: Proc. of ESWC 2012 (2012), to appear
16. Rosati, R., Almatelli, A.: Improving query answering over *DL-Lite* ontologies. In: Proc. of KR 2010. pp. 290–300 (2010)
17. Tao, J., Sirin, E., Bao, J., McGuinness, D.L.: Integrity constraints in OWL. In: Proc. of AAAI 2010 (2010)
18. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: Tractable OWL 2 reasoning infrastructure. In: Proc. of ESWC 2010. pp. 431–435 (2010)