Improving Query Answering over *DL-Lite* Ontologies

Riccardo Rosati, Alessandro Almatelli
Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma, Italy

Abstract

The *DL-Lite* family of Description Logics has been designed with the specific goal of allowing for answering complex queries (in particular, conjunctive queries) over ontologies with very large instance sets (ABoxes). So far, in *DL-Lite* systems, this goal has been actually achieved only for relatively simple (short) conjunctive queries. In this paper we present Presto, a new query answering technique for *DL-Lite* ontologies, and an experimental comparison of Presto with the main previous approaches to query answering in *DL-Lite*. In practice, our experiments show that, in real ontologies, current techniques are only able to answer conjunctive queries of less than 7-10 atoms (depending on the complexity of the TBox), while Presto is actually able to handle conjunctive queries of up to 30 atoms. Furthermore, in the cases that are already successfully handled by previous approaches, Presto is significantly more efficient.

Introduction

The *DL-Lite* family of Description Logics (Calvanese et al. 2007) has been designed with the specific goal of allowing for answering complex queries (in particular, conjunctive queries) over ontologies with very large instance sets (ABoxes). The ideas underlying *DL-Lite* are currently very popular both in the theoretical KR community (see, e.g., (Artale et al. 2009)) and in the more practical world of standard ontology languages for the Semantic Web (*DL-Lite* provides the logical underpinnings of the OWL2 QL language).

The strategy used by most of the existing systems and algorithms for *DL-Lite*, (e.g., Quonto (Acciarri et al. 2005), Owlgres (Stocker and Smith 2008), and Requiem (Pérez-Urbina, Motik, and Horrocks 2009)) is *query answering by query rewriting*. More precisely, query answering is performed by first computing a rewriting of the query with respect to the intensional part of the ontology (TBox), thus obtaining a so-called *perfect reformulation* of the initial query. Such a perfect reformulation is then evaluated over the extensional part of the ontology (ABox) only. A distinguishing feature of *DL-Lite* with respect to the other DLs is that the perfect reformulation of conjunctive queries can be expressed by first-order queries. This property, also called *first-order rewritability* of conjunctive queries, is extremely important, because it allows to delegate the management of the ABox to a relational database system (RDBMS) and to solve query answering by shipping the perfect reformulation of the initial query (expressed in the SQL language) to the RDBMS. This implementation strategy actually allows to handle ABoxes of very large size (comparable to the size of a database).

However, the bottleneck of the above algorithms and systems is constituted by the fact that the perfect reformulation computed increases exponentially with the number of atoms of the conjunctive query. Empirical studies have shown that, while this is not a serious problem for conjunctions of up to 5-7 atoms (depending on the complexity of the TBox), for larger queries the above algorithms typically either fail in computing the perfect reformulation in a reasonable amount of time, or produce a query that is too large (e.g., a union of thousands of conjunctive queries) to be handled by current RDBMSs. This constitutes a serious practical limitation, since experiments have shown that some natural and interesting conjunctive queries over real ontologies fall into the class that cannot be handled by current query answering techniques for *DL-Lite*.

In this paper we try to overcome the above limitation of current query answering techniques over *DL-Lite* ontologies. In particular, we present a new algorithm, called Presto, for the perfect reformulation of unions of conjunctive queries over *DL-Lite* ontologies. Presto is based on the following innovative ideas: (i) differently from previous approaches, Presto does not generate a union of conjunctive queries, but a nonrecursive datalog program. In fact, the use of a disjunctive normal form is one of the reasons for the exponential blow-up of previous techniques, which can thus be avoided by Presto; (ii) the query expansion rules (based on resolution) used by previous techniques are deeply optimized in Presto. In particular, Presto applies expansion rules driven by the goal of *eliminating existential joins* from the query based on the computation of *most general subsumees* of concept and role expressions, which turns out to be a much smarter strategy than previous approaches. As a consequence of the above innovations, the query produced by Presto is not exponential anymore with respect to the number of atoms of the initial conjunctive query, but is only exponential with respect to the number of *eliminable exis-*

tential join variables of the query: such variables are a subset of the join variables of the query, and are typically much less than the number of atoms of the query.

Then, we present a set of experimental results which show that: (i) both the time for computing the rewriting and the size of the query generated by our algorithm are smaller than the corresponding time and size of all previous approaches; (ii) the evaluation of our nonrecursive datalog query (after a translation in SQL) by the RDBMS is computationally much easier than the evaluation of the queries produced by previous techniques. In particular, the experiments show that Presto scales much better than all previous approaches with respect to the size of the query: in practice, in all ontologies we have used in our experiments, Presto allows for effectively computing the perfect reformulation of conjunctive queries of even 30 atoms (and more), while previous techniques are able to deal with conjunctions of at most 7-10 atoms (depending on the complexity of the TBox).

The above results prove that Presto constitutes a real advancement with respect to the current techniques for query answering over *DL-Lite* ontologies, since it overcomes the main computational limitation of the *DL-Lite* approach to query answering over ontologies, allowing for effectively answering even very complex (unions of) conjunctive queries over *DL-Lite* ontologies.

The structure of the paper is the following. In the next section, we briefly recall the description logics and query languages used in the paper. Then, we describe the Presto query rewriting algorithm, and show formal properties of the algorithm (correctness and computational complexity). Finally, we report on the experiments conducted with Presto.

Preliminaries

In this section we briefly recall the description logic *DL-Lite*$_R$, unions of conjunctive queries, and nonrecursive datalog queries. We focus on *DL-Lite*$_R$ for ease of presentation, since our technique is actually able to handle other *DL-Lite* logics, in particular *DL-Lite*$_F$, *DL-Lite*$_A$, and (under a slight extension) the OWL2 profile OWL2 QL.

***DL-Lite* ontologies** We start from three mutually disjoint alphabets: an alphabet of concept names, an alphabet of role names, and an alphabet of constant (or individual) names. We call *basic concept* an expression of the form $B ::= A \mid \exists P \mid \exists P^-$, where A is a concept name and P is a role name, and we call *basic role* an expression of the form $R ::= P \mid P^-$, where P is a role name.

With a slight abuse of notation, we will also make use of expressions of the form $\exists R$ and $\exists R^-$, where R represents either a role name P or the basic role P^-. In the latter case, $\exists R^-$ stands for the basic concept $\exists P$.

A *DL-Lite*$_R$ TBox assertion is an expression of one of the following forms (where B_1, B_2 are basic concepts and R_1, R_2 are basic roles): (i) $B_1 \sqsubseteq B_2$ (concept inclusion); (ii) $R_1 \sqsubseteq R_2$ (role inclusion); (iii) $B_1 \sqsubseteq \neg B_2$ (concept disjointness); (iv) $R_1 \sqsubseteq \neg R_2$ (role disjointness).

A *DL-Lite*$_R$ TBox is a set of *DL-Lite*$_R$ TBox assertions.

A *membership assertion* is a ground atom, i.e., an expression of the form $A(a)$, $P(a, b)$ where A is a concept name, P is a role name, and a, b are constant names.

An ABox is a set of membership assertions.

A *DL-Lite*$_R$ *ontology* is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a *DL-Lite*$_R$ TBox and $\mathcal{A}$ is an ABox.

Queries An *atom* is an expression of the form $p(X)$, where p is a predicate of arity n and X is a n-tuple of variables or constants. If no variable symbol occurs in X, then $p(X)$ is called a *ground atom* (or *fact*).

A *Datalog rule* r is an expression of the form α :– $\beta_1, \ldots, \beta_m$, where α is an atom, each β_i is an atom, and every variable occurring in α must appear in at least one of the atoms $\beta_1, \ldots, \beta_m$. The atom α is called the *head* of r, while the expression $\beta_1, \ldots, \beta_m$ is called the *body* of r. The predicate of α is called the *head predicate* of r. The *arity* of r is the number of arguments of the head predicate α. The variables occurring in α are called the *head variables* of r, while the variables only occurring in the body of r are called the *existential variables* of r. An existential variable that occurs only once in r is called an *unbound variable*, otherwise it is called an *existential join variable* (*ej-var* for short). Head variables, ej-vars, and constants occurring in r are the *bound terms* of r, while unbound variables are the *unbound terms* of r.

A *Datalog program* over an ontology $\mathcal{O}$ is a set of Datalog rules such that, for every rule r of the program, the head predicate of r is not a predicate (concept or role) used in the ontology.

A *nonrecursive datalog program* is a Datalog program such that there exists an ordering $r_1, \ldots, r_n$ of its rules such that the head predicate of r_i does not occur in the body of rule r_j for every i, j such that $1 \leq i \leq j \leq n$. W.l.o.g., we assume that all occurrences of the same predicate in a program have the same arity.

A *nonrecursive Datalog (nr-datalog) query* over an ontology $\mathcal{O}$ is a pair (q, Q) such that q is not a predicate of $\mathcal{O}$ and Q is a nonrecursive Datalog program over $\mathcal{O}$. The *arity* of a nr-datalog query (q, Q) is the arity of the predicate q in Q. We recall that nr-datalog queries correspond to first-order positive existential first-order queries (i.e., to positive relational algebra queries) (Abiteboul, Hull, and Vianu 1995).

A *union of conjunctive queries (UCQ)* over an ontology $\mathcal{O}$ is a nr-datalog query (q, Q) over $\mathcal{O}$ such that: (i) all rules in Q have q as their head predicate; (ii) for every rule $r \in Q$, all the predicates occurring in the body of r are predicates of $\mathcal{O}$. We recall that every nr-datalog query can be *unfolded* into a finite UCQ. A *conjunctive query (CQ)* over an ontology $\mathcal{O}$ is a UCQ over $\mathcal{O}$ whose program consists of a single rule. Finally, a *Boolean* nr-datalog query is a nr-datalog query of arity 0.

From now on, to keep notation to a minimum, we will not explicitly mention the query predicate of nr-datalog queries, assuming that the query predicate is always q (and of course, we also assume that q is not used as a predicate by any ontology): thus, we will denote the query (q, Q) simply by Q.

Semantics The semantics of a DL is given in terms of interpretations, where an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$. In particular, we have:

$$\begin{array}{rcl} A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\ P^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\ (P^{-})^{\mathcal{I}} & = & \{(o_2, o_1) \mid (o_1, o_2) \in P^{\mathcal{I}}\} \\ (\exists R)^{\mathcal{I}} & = & \{o \mid \exists o'. (o, o') \in R^{\mathcal{I}}\} \\ (\neg B)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}} \\ (\neg R)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}} \end{array}$$

An interpretation $\mathcal{I}$ is a *model* of $B \sqsubseteq C$, where C is either a basic concept or the negation of a basic concept, if $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. Similarly, $\mathcal{I}$ is a *model* of $R \sqsubseteq E$, where R is a basic role and E is either a basic role or the negation of a basic role, if $R^{\mathcal{I}} \subseteq E^{\mathcal{I}}$.

To specify the semantics of membership assertions, we extend the interpretation function to constants, by assigning to each constant a an object $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model of a membership assertion $A(a)$, (resp., $P(a, b)$) if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$).

Given an (inclusion, or membership) assertion ϕ, and an interpretation $\mathcal{I}$, we denote by $\mathcal{I} \models \phi$ the fact that $\mathcal{I}$ is a model of ϕ. Given a (finite) set of assertions Φ, we denote by $\mathcal{I} \models \Phi$ the fact that $\mathcal{I}$ is a model of every assertion in Φ. A *model of an ontology* $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$. An ontology is *satisfiable* if it has at least one model.

A Boolean nr-datalog query Q is *satisfied* in an interpretation $\mathcal{I}$ if $\mathcal{I} \models FO(Q)$, where $FO(Q)$ is the positive existential first-order sentence corresponding to Q which is obtained by "unfolding" the defined predicates of Q (Abiteboul, Hull, and Vianu 1995). For instance, if Q is the query

$$\begin{array}{l} q() :- C(x), p(x, y), R(y, z) \\ q() :- D(x), S(x, a) \\ p(x, y) :- R(y, x), D(x), S(y, z) \\ p(x, y) :- T(x, y) \end{array}$$

then $FO(Q)$ is the first-order sentence

$$\begin{array}{l} (\,\exists x, y, z.C(x) \wedge \\ \quad ((\exists z'.R(y, x) \wedge D(x) \wedge S(y, z')) \vee T(x, y)) \;\wedge R(y, z)\,) \vee \\ (\,\exists x.D(x) \wedge S(x, a)\,) \end{array}$$

For the sake of simplicity, the reasoning task we will formally address in this paper is query entailment, i.e., query answering restricted to Boolean queries. An ontology $\mathcal{O}$ *entails* a Boolean nr-datalog query q, denoted by $\mathcal{O} \models q$, if q is satisfied in all models of $\mathcal{O}$. Since we focus on query entailment only, from now on when we speak about nr-datalog queries we always mean *Boolean* nr-datalog queries. We remark, however, that, as in the case of UCQs (see e.g. (Glimm et al. 2007)), answering arbitrary (i.e., non-Boolean) nr-datalog queries can be easily reduced to nr-datalog query entailment.

Given an ABox $\mathcal{A}$, the *canonical model of* $\mathcal{A}$ (denoted by $can(\mathcal{A})$) is the interpretation isomorphic to $\mathcal{A}$, i.e., an interpretation $\mathcal{I}$ whose domain is the set of individuals occurring in $\mathcal{A}$ and such that: (i) for every individual a occurring in $\mathcal{A}$, $a^{\mathcal{I}} = a$; (ii) for every concept name C, $C^{\mathcal{I}} = \{a \mid C(a) \in \mathcal{A}\}$; (iii) for every role name R, $R^{\mathcal{I}} = \{\langle a, b\rangle \mid R(a, b) \in \mathcal{A}\}$.

Given a TBox $\mathcal{T}$ and a query Q, a *perfect reformulation* of Q with respect to $\mathcal{T}$ is a query Q' such that, for every ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, $\langle \mathcal{T}, \mathcal{A} \rangle \models Q$ iff Q' is satisfied in $can(\mathcal{A})$. Informally, a perfect reformulation of Q is able to "encode" the TBox $\mathcal{T}$, and thus allows to answer Q by only looking at the ABox (and considering it as a relational database, i.e., a single model).

Most general subsumees Given a *DL-Lite*$_R$ TBox $\mathcal{T}$ and a non-empty set of basic concepts $\mathcal{B} = \{B_1, \ldots, B_n\}$, a *most general subsumee (MGS)* of $\mathcal{B}$ in $\mathcal{T}$ is a basic concept B such that: (1) $\mathcal{T} \not\models B \sqsubseteq \bot$ where $\bot$ denotes the empty concept; (2) for each $B_i \in \mathcal{B}$, $\mathcal{T} \models B \sqsubseteq B_i$; (3) for each basic concept B' satisfying the above conditions 1 and 2, either $\mathcal{T} \not\models B \sqsubseteq B'$ or $\mathcal{T} \models B' \sqsubseteq B$.

The above notion of MGS of a set of concept expressions is the usual one: however, for our purposes we actually need to extend the above notion of MGS to a set of both concept and role expressions.

Given a *DL-Lite*$_R$ TBox $\mathcal{T}$ and a set of basic concepts and basic roles $\mathcal{P} = \{B_1, \ldots, B_n, R_1, \ldots, R_m\}$ with $n \geq 1$, $m > 1$, a *most general subsumee* (MGS) of $\mathcal{P}$ in $\mathcal{T}$ is a basic concept $\exists R$ such that:

1. $\mathcal{T} \not\models \exists R \sqsubseteq \bot$;
2. for each basic concept $B_i \in \mathcal{P}$, $\mathcal{T} \models \exists R \sqsubseteq B_i$;
3. for each basic role $R_i \in \mathcal{P}$, $\mathcal{T} \models R \sqsubseteq R_i$;
4. for each basic concept $\exists S$ satisfying the above conditions 1–4, either $\mathcal{T} \not\models \exists R^{-} \sqsubseteq \exists S^{-}$ or $\mathcal{T} \models \exists S^{-} \sqsubseteq \exists R^{-}$.

We denote by $\mathcal{MGS}(\mathcal{P}, \mathcal{T})$ the set of most general subsumees of $\mathcal{P}$ in $\mathcal{T}$. It can be easily shown that checking whether a basic concept is an MGS of $\mathcal{P}$ in a *DL-Lite*$_R$ TBox $\mathcal{T}$ can be done in time polynomial in the size of $\mathcal{P} \cup \mathcal{T}$.

The Presto algorithm

We now present Presto, an algorithm that computes the perfect reformulation of a UCQ with respect to a *DL-Lite*$_R$ TBox. The algorithm is displayed in Figure 1. Before delving into its details, let us provide an informal explanation of the intuitions behind the algorithm. Presto is based on three main ideas:

(1) Split every rule into its existential join connected components. As explained below in the description of the function $Split$, the body of every rule is systematically divided into subsets of atoms connected by existential join variables, and every such subset of the query body constitutes the definition of a new auxiliary predicate. This step is fundamental, since (as explained at the end of this section) the size of the query computed by Presto is exponential with respect to the maximum number of (eliminable) ej-vars that appear in a single rule of the query, and applying the above splitting step may decrease such a number.

(2) Eliminate ej-vars through the use of most general subsumees. This step (formalized by the $EliminateEJVar$ function) corresponds to a sequence of resolution steps in the

previous query rewriting algorithms for *DL-Lite*, in particular PerfectRef (Calvanese et al. 2007) and Requiem (Pérez-Urbina, Motik, and Horrocks 2009), and is based on the use of the above defined most general subsumees of concept and role expressions with respect to a TBox. In practice, this step realizes a crucial optimization of the rewriting rules of previous methods (in particular, the reduce-rule of the PerfectRef algorithm): in Presto, only unifications among terms that bring to the generation of a "significant" new rule (i.e., a rule not subsumed by an already generated rule of the query) are considered. Instead, in the algorithm PerfectRef of (Calvanese et al. 2007), unifications are derived in a "blind" way from every unifiable pair of atoms, regardless of the consequences of this unification: indeed (as already observed by (Pérez-Urbina, Motik, and Horrocks 2009)), in many cases the vast majority of the unification steps performed by PerfectRef is superfluous. The idea of aggregating a sequence of unification and resolution steps, avoiding useless unifications, is actually the central idea of the Presto algorithm, and constitutes a dramatic improvement in terms of computational cost with respect to the above mentioned previous methods, as will be shown by the experimental results reported in the next section.

(3) Define predicates (views) corresponding to the TBox expansion of basic concepts and roles. First, the query is transformed by introducing (through the $Rename$ and $DeleteUnboundVars$ functions) concept and role expressions, besides concept and role names. Then, for every concept expression B (respectively, role name R) occurring in the query, a new predicate is introduced (called OA-predicate in the following): such a predicate is defined in the query as the union of the concepts (respectively, roles) that are a specialization of B (respectively, R) in the TBox (this is realized by the function $DefineAtomView$). In fact, all such concepts constitute all the possible rewritings of the rule atom corresponding to the concept B. (In this step we also consider the TBox expansion of Boolean propositions corresponding to an extensional non-emptiness check of atomic concepts and roles, i.e., the property whether a concept (or a role) is populated in every model of the ontology. Such Boolean propositions are denoted by B^0 and R^0 in the following.) The advantage of introducing concept and role views is that the exponential blowup due to the Cartesian product of the rewritings of the single rule atoms is avoided. For instance, suppose we want to compute the perfect reformulation of a CQ having 10 atoms in its body and such that each atom has a rewriting of 10 atoms. If we are forced to produce a UCQ, then we should produce a query containing at least 10^{10} CQs, while if we can define an intermediate predicate for (the rewriting of) every query atom, the nr-datalog program thus computed has $10^2 + 1$ rules.

We now define the auxiliary functions used by Presto. In the following, given a rule r we use the symbols $b, b_1, b_2, \ldots$ to denote bound terms of r (i.e., head variables, ej-vars, and constants), we use the symbols $u, u_1, u_2, \ldots$ to denote unbound variables, we use the symbols $x, x_1, x_2, \ldots$ to denote generic variables (which may be either bound or unbound), we use the symbols $a, a_1, a_2, \ldots$ to denote constants, and we use the symbols $t, t_1, t_2, \ldots$ to denote generic terms (either bound or unbound).

An *ontology-annotated predicate* (OA-predicate for short) is an expression of the form p_e^k such that either e is a role name and $k \in \{0, 2\}$, or e is a basic concept expression and $k \in \{0, 1\}$. Given a *DL-Lite*$_R$ TBox $\mathcal{T}$, the OA-predicates of $\mathcal{T}$ are all the OA-predicates built using the role and concept names occurring in $\mathcal{T}$.

The function $Rename$**.** Let r be a rule over $\mathcal{T}$. Then, $Rename(r)$ is the rule obtained from r as follows: (i) every atom of the form $R(t, t_1)$, where R is a role name, is replaced by $p_R^2(t, t_1)$; (ii) every atom of the form $A(t)$, where A is a concept name, is replaced by $p_A^1(t)$. Then, given a query Q, we define $Rename(Q) = \bigcup_{r \in Q} Rename(r)$. Thus, the function $Rename$ replaces concept and role names with OA-predicates in the rule. Notice that, in every OA-predicate, the superscript represents the arity of the predicate.

The function $DeleteUnboundVars$**.** Let r be a rule over $\mathcal{T}$. Then, $DeleteUnboundVars(r)$ is the rule obtained from r as follows:

- every atom of the form $p_R^2(b, u)$ is replaced by $p_{\exists R}^1(b)$;
- every atom of the form $p_R^2(u, b)$ is replaced by $p_{\exists R^-}^1(b)$;
- every atom of the form $p_R^2(u_1, u_2)$ is replaced by p_R^0;
- every atom of the form $p_A^1(u)$ is replaced by p_A^0.

Then, given a query Q, we define $DeleteUnboundVars(Q) = \bigcup_{r \in Q} DeleteUnboundVars(r)$. Thus, the function $DeleteUnboundVars$ eliminates unbound variables in a systematic way, through the use of new OA-predicates.

The function $DeleteRedundantAtoms$**.** Given a *DL-Lite*$_R$ TBox $\mathcal{T}$ and a rule r, $DeleteRedundantAtoms(r, \mathcal{T})$ eliminates the redundant atoms in the body of r with respect to $\mathcal{T}$, i.e., the atoms that are implied (under $\mathcal{T}$) by other atoms in the body of r. More precisely:

- if $p_R^2(t_1, t_2)$ and $p_S^2(t_1, t_2)$ occur in r and $\mathcal{T} \models R \sqsubseteq S$, then eliminate $p_S^2(t_1, t_2)$ from r;
- if $p_R^2(t_1, t_2)$ and $p_S^2(t_2, t_1)$ occur in r and $\mathcal{T} \models R \sqsubseteq S^-$, then eliminate $p_S^2(t_2, t_1)$ from r;
- if $p_B^1(t)$ and $p_C^1(t)$ occur in r and $\mathcal{T} \models B \sqsubseteq C$, then eliminate $p_C^1(t)$ from r;
- if $p_R^2(t_1, t_2)$ and $p_C^1(t_1)$ occur in r and $\mathcal{T} \models \exists R \sqsubseteq C$, then eliminate $p_C^1(t_1)$ from r;
- if $p_R^2(t_1, t_2)$ and $p_C^1(t_2)$ occur in r and $\mathcal{T} \models \exists R^- \sqsubseteq C$, then eliminate $p_C^1(t_2)$ from r;
- if p_α^0 and p_β^0 occur in r and $\mathcal{T} \models \alpha^0 \sqsubseteq \beta^0$, then eliminate p_β^0 from r;
- if $p_B^1(t)$ and p_α^0 occur in r and $\mathcal{T} \models B^0 \sqsubseteq \alpha^0$, then eliminate p_α^0 from r;
- if $p_R^2(t_1, t_2)$ and p_α^0 occur in r and $\mathcal{T} \models R^0 \sqsubseteq \alpha^0$, then eliminate p_α^0 from r.

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

Figure 1: The Presto algorithm.

Then, given a query Q, we define $DeleteRedundantAtoms(Q, \mathcal{T}) = \bigcup_{r \in Q} DeleteRedundantAtoms(r, \mathcal{T})$. In the above definition, we have indicated implications of Boolean propositions using the notation $\alpha^0 \sqsubseteq \beta^0$, with α, β basic concepts or role names. E.g., $R^0 \sqsubseteq A^0$ stands for the sentence $(\exists x, y.R(x, y)) \rightarrow (\exists x.A(x))$. Checking the entailment of such formulas (which is a non-standard form of TBox reasoning in DLs) can easily be done in *DL-Lite*$_R$ in polynomial time with respect to the size of the TBox, by slightly extending the procedure for checking entailment of ordinary concept and role inclusion assertions (we omit details on this aspect due to lack of space). Notice that, after the above elimination of atoms, some ej-vars may have turned into unbound variables: we assume that such variables are eliminated by executing the function $DeleteUnboundVars$ on the query.

The function $Split$**.** Given a rule r, $Split(r)$ is the program that is obtained by splitting r into the connected components of the *ej-graph* of r (and using a new auxiliary predicate for each connected component). The ej-graph of r is an undirected graph whose nodes are the atoms in the body of r and such that there is an edge between two nodes if the corresponding atoms share an ej-var.

Example. Given the following rule r:

$$\begin{aligned} q(x,y) :- \; & p^2_R(x,z),\; p^2_S(z,y),\; p^1_{\exists T}(z),\; p^2_T(y,w), \\ & p^2_R(w,v),\; p^1_E(v),\; p^2_S(w,x),\; p^2_T(x,t),\; p^2_R(t,a) \end{aligned}$$

where x, y, z, v, w, t are variables and a is a constant, the ej-connected components of r are:

$$\begin{array}{l} p^2_R(x,z),\; p^2_S(z,y),\; p^1_{\exists T}(z) \\ p^2_T(y,w),\; p^2_R(w,v),\; p^1_E(v),\; p^2_S(w,x) \\ p^2_T(x,t),\; p^2_R(t,a) \end{array}$$

and $Split(r)$ is the following program (using the new auxiliary predicates q_1, q_2, q_3):

$$\begin{array}{l} q(x,y) :- q_1(x,y),\; q_2(x,y),\; q_3(x) \\ q_1(x,y) :- p^2_R(x,z),\; p^2_S(z,y),\; p^1_{\exists T}(z) \\ q_2(x,y) :- p^2_T(y,w),\; p^2_R(w,v),\; p^1_E(v),\; p^2_S(w,x) \\ q_3(x) :- p^2_T(x,t),\; p^2_R(t,a) \end{array}$$

Given a program Q, we define $Split(Q) = \bigcup_{r \in Q} Split(r)$.

The functions $Eliminable$ **and** $EliminateEJVar$**.** Let x be an ej-var in r, let $p^1_{B_1}(x), \ldots, p^1_{B_n}(x)$ be the unary atoms in which x occurs, and let

$$\alpha = \{p^2_{R_1}(x,t_1), \ldots, p^2_{R_m}(x,t_m), p^2_{S_1}(t'_1,x), \ldots, p^2_{S_\ell}(t'_\ell,x)\}$$

be the set of binary atoms of r in which x occurs. Furthermore, let $\mathcal{P} = \{B_1, \ldots, B_n, R_1, \ldots, R_m, S^-_1, \ldots, S^-_\ell\}$, and let $\mathcal{T}$ be a *DL-Lite*$_R$ TBox.

First, we say that the ej-var x is *$\mathcal{T}$-eliminable in* r if:

1. x does not occur twice in the same binary atom;
2. at most one constant symbol is contained in the set of terms $\{t_1, \ldots, t_m, t'_1, \ldots, t'_\ell\}$;
3. $\mathcal{MGS}(\mathcal{P}, \mathcal{T})$ is non-empty.

Then, $Eliminable(x, r, \mathcal{T})$ is defined as a Boolean function that returns $true$ iff x is $\mathcal{T}$-eliminable in r.

Finally, we define the function $EliminateEJVar$ as follows:

- if $m = 0$ and $\ell = 0$, then $\mathit{EliminateEJVar}(r, x, \mathcal{T})$ is the set of rules $\{r[\alpha/p^0_B] \mid B \in \mathcal{MGS}(\mathcal{P}, \mathcal{T})\}$, where $r[\alpha/p^0_B]$ denotes the rule obtained from r by replacing the set of atoms α in the body of r with the atom p^0_B;
- if $m \geq 1$ or $\ell \geq 1$, then $\mathit{EliminateEJVar}(r, x, \mathcal{T})$ is the set of rules $\{\sigma(r[\alpha/p^1_{\exists R^-}(\tau)]) \mid \exists R \in \mathcal{MGS}(\mathcal{P}, \mathcal{T})\}$, where: (i) $\tau = t_1$ if $m \geq 1$, otherwise $\tau = t'_1$; (ii) $r[\alpha/p^1_{\exists R^-}(\tau)]$ denotes the rule obtained from r by replacing the set of atoms α in the body of r with the atom $p^1_{\exists R^-}(\tau)$; (iii) σ is the variable substitution obtained from the equalities $t_1 = t_2 = \ldots = t_m = t'_1 = t'_2 = \ldots = t'_\ell$ (i.e., $\sigma(r)$ denotes the application of the above substitution to rule r).

Example. Let r be the rule

$$q \text{:-} p^2_{R_3}(x, y),\ p^2_{R_4}(z, x),\ p^1_A(x),\ p^1_B(z),\ p^1_C(y)$$

and let $\mathcal{T}$ be such that $\mathcal{MGS}(\{A, R_3, R_4^-\}, \mathcal{T}) = \{\exists R_1, \exists R_2^-\}$. Then, $\mathit{Eliminable}(x, r, \mathcal{T}) = \mathit{true}$, and $\mathit{EliminateEJVar}(q, x, \mathcal{T})$ is the set of rules

$$\{\ q \text{:-} p^1_{\exists R_1^-}(y),\ p^1_B(y),\ p^1_C(y) \quad q \text{:-} p^1_{\exists R_2}(y),\ p^1_B(y),\ p^1_C(y)\ \}$$

(notice that in this case $\sigma = \{z \rightarrow y\}$).

The function $\mathit{DefineAtomView}$**.** Finally, given an OA-predicate V and a *DL-Lite*$_R$ TBox $\mathcal{T}$, the function $\mathit{DefineAtomView}(V, \mathcal{T})$ is defined as follows:

(i) if $V = p^2_R$ with R a role name, then $\mathit{DefineAtomView}(V, \mathcal{T})$ is the set of rules

$$\{p^2_R(x, y) \text{:-} P(x, y) \mid P \text{ is a role name and } \mathcal{T} \models P \sqsubseteq R\} \cup$$
$$\{p^2_R(x, y) \text{:-} P(y, x) \mid P \text{ is a role name and } \mathcal{T} \models P^- \sqsubseteq R\}$$

(ii) if $V = p^1_B$ with B a basic concept, then $\mathit{DefineAtomView}(V, T)$ is the set of rules

$$\{p^1_B(x) \text{:-} A(x) \mid A \text{ is a concept name and } \mathcal{T} \models A \sqsubseteq B\} \cup$$
$$\{p^1_B(x) \text{:-} R(x, \_) \mid R \text{ is a role name and } \mathcal{T} \models \exists R \sqsubseteq B\} \cup$$
$$\{p^1_B(x) \text{:-} R(\_, x) \mid R \text{ is a role name and } \mathcal{T} \models \exists R^- \sqsubseteq B\}$$

(iii) if $V = p^0_N$ with N concept or role name, then $\mathit{DefineAtomView}(V, \mathcal{T})$ is the set of rules

$$\{p^0_N \text{:-} A(\_) \mid A \text{ is a concept name and } \mathcal{T} \models A^0 \sqsubseteq N^0\} \cup$$
$$\{p^0_N \text{:-} R(\_, \_) \mid R \text{ is a role name and } \mathcal{T} \models R^0 \sqsubseteq N^0\}$$

Let us now go back to the main algorithm. As shown by Figure 1, the structure of the algorithm Presto is rather simple: the input query Q is initially transformed by the Rename, $\mathit{DeleteUnboundVars}$, $\mathit{DeleteRedundantAtoms}$ and Split functions. Then, a cycle is executed whose purpose is to close the query (set of rules) with respect to the elimination of ej-vars. At every iteration, the query is augmented with new rules obtained from the elimination of an ej-var from a rule already present in the query.

Example 1 Consider the *DL-Lite*$_R$ TBox (over the concepts $A, A_1, A_2, A_3, B, B_1, B_2, B_3, C, C_1, C_2, C_3$ and the roles U, T, W, V, Q, P, S, R) displayed in Figure 2(a), and consider the conjunctive query q displayed in Figure 2(b). The nonrecursive datalog program returned by Presto is shown in Figure 2(c). First, the query obtained after applying the functions Rename and $\mathit{DeleteUnboundVars}$ is

$$q(x, y) \text{:-} p^1_{\exists S}(z), p^2_R(x, z), p^1_{\exists T^-}(w), p^1_{A_1}(z), p^2_R(y, w)$$

Then, the application of the function $\mathit{DeleteRedundantAtoms}$ deletes the atom $p^1_{A_1}(z)$ (because the TBox entails the inclusion $\exists R^- \sqsubseteq A_1$). Next, the function Split is applied, which splits the query into the two connected components $p^1_{\exists S}(z), p^2_R(x, z)$ and $p^1_{\exists T^-}(w), p^2_R(y, w)$, thus producing the set of rules Q' corresponding to the rules (R0), (R1) and (R2) which use the auxiliary predicates q_1 and q_2. Then, the algorithm executes a first iteration of the repeat–until cycle, and picks rule (R1). The function $\mathit{Eliminable}$ identifies z as an eliminable ej-var in (R1), since it is easy to verify that $\mathcal{MGS}(\{\exists S, R^-\}, \mathcal{T}) = \{\exists T, \exists U^-\}$: therefore, the function $\mathit{EliminateEJVar}(z, \text{(R1)}, \mathcal{T})$ returns the rules (R3) and (R4). At its second iteration, the algorithm picks rule (R2), and the function $\mathit{Eliminable}$ identifies w as an eliminable ej-var in (R2), since it is easy to verify that $\mathcal{MGS}(\{\exists T^-, R^-\}, \mathcal{T}) = \{\exists T, \exists U^-\}$: therefore, the function $\mathit{EliminateEJVar}(w, \text{(R2)}, \mathcal{T})$ returns the rules (R5) and (R6). Finally, the repeat–until loop ends, because there are no more eliminable ej-vars in the rules generated so far, and the function $\mathit{DefineAtomView}$ adds to the program the other rules shown in Figure 2(c).[1] □

It is immediate to verify that the algorithm always terminates, and that the set of rules Q' returned by the algorithm is always a nr-datalog query over $\mathcal{T}$ (i.e., Q' is always nonrecursive). Correctness of the algorithm is established by the following theorem.

Theorem 2 *Let $\mathcal{T}$ be a DL-Lite$_R$ TBox, let Q be a UCQ and let Q' be the nr-datalog query returned by Presto$(Q, \mathcal{T})$. Then, for every ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is a satisfiable DL-Lite$_R$ knowledge base, $\langle \mathcal{T}, \mathcal{A} \rangle \models Q$ iff Q' is satisfied in $can(\mathcal{A})$.*

Proof (sketch). The proof heavily relies on the correctness of the algorithm PerfectRef presented in (Calvanese et al. 2007) (also called "CGLLR algorithm" in (Pérez-Urbina, Motik, and Horrocks 2009)). The algorithm PerfectRef iteratively construct a set of CQs (i.e., a UCQ) by making use of two rules that produce a new CQ from a previously generated CQ: the atom-rewrite-rule (step (a) of the algorithm) and the reduce-rule (step (b)). We refer to (Calvanese et al. 2007) for a detailed description of PerfectRef.

Let Q'_c be the query returned by the algorithm PerfectRef executed on input Q and $\mathcal{T}$. We prove that Q'_c is equivalent to the query Q'_p returned by Presto, i.e., in every interpretation $\mathcal{I}$, the evaluation of Q'_c and Q'_p in $\mathcal{I}$ coincide.

First, the proof that the query Q'_p is contained in the query Q'_c is quite easy. We first consider the unfolding of Q'_p, i.e., the UCQ Q''_p obtained by unfolding all defined predicates

[1] We remark that, for this query and TBox, the algorithm PerfectRef produces a perfect reformulation which consists of the union of 650 conjunctive queries.

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl@{\qquad}rcl@{\qquad}rcl}
A & \sqsubseteq & A_1 & C_1 & \sqsubseteq & B_3 & \exists R & \sqsubseteq & \exists U & \exists T & \sqsubseteq & \exists S & & & \\
A & \sqsubseteq & A_3 & C_2 & \sqsubseteq & C & \exists R & \sqsubseteq & A & \exists T & \sqsubseteq & B_3 & T & \sqsubseteq & P \\
A_1 & \sqsubseteq & B & C_2 & \sqsubseteq & C_1 & \exists R & \sqsubseteq & \exists S & \exists T & \sqsubseteq & B_2 & T & \sqsubseteq & P^- \\
B_2 & \sqsubseteq & A_3 & C_3 & \sqsubseteq & B & \exists R & \sqsubseteq & C & \exists T^- & \sqsubseteq & \exists P & T & \sqsubseteq & R \\
B_3 & \sqsubseteq & A & C_3 & \sqsubseteq & B_1 & \exists R & \sqsubseteq & A_2 & \exists T^- & \sqsubseteq & A_1 & T & \sqsubseteq & R^- \\
B_3 & \sqsubseteq & A_2 & \exists P & \sqsubseteq & B & \exists R^- & \sqsubseteq & A_3 & \exists T^- & \sqsubseteq & C & T^- & \sqsubseteq & S \\
C & \sqsubseteq & A & \exists P & \sqsubseteq & \exists U & \exists R^- & \sqsubseteq & A & \exists U & \sqsubseteq & C & U & \sqsubseteq & T^- \\
C & \sqsubseteq & B & \exists P & \sqsubseteq & B_2 & \exists R^- & \sqsubseteq & \exists P^- & \exists U & \sqsubseteq & C_2 & U & \sqsubseteq & S^- \\
C_1 & \sqsubseteq & B & \exists R & \sqsubseteq & C_1 & \exists R^- & \sqsubseteq & C_3 & \exists U^- & \sqsubseteq & \exists P & & &
\end{array}
$$

(a) TBox

$$q(x,y) \coloneq S(z,j), R(x,z), T(k,w), A_1(z), R(y,w).$$

(b) Query

$$
\begin{array}{ll}
\text{(R0)} & q(x,y) \coloneq q_1(x), q_2(y) \\
\text{(R1)} & q_1(x) \coloneq p^2_R(x,z), p^1_{\exists S}(z) \\
\text{(R2)} & q_2(y) \coloneq p^1_{\exists T^-}(w), p^2_R(y,w) \\
\text{(R3)} & q_1(x) \coloneq p^1_{\exists U}(x) \\
\text{(R4)} & q_1(x) \coloneq p^1_{\exists T^-}(x) \\
\text{(R5)} & q_2(y) \coloneq p^1_{\exists T^-}(y) \\
\text{(R6)} & q_2(y) \coloneq p^1_{\exists U}(y)
\end{array}
$$

$$
\begin{array}{l@{\qquad}l@{\qquad}l@{\qquad}l}
p^1_{\exists S}(x) \coloneq T(\_,x) & p^1_{\exists T^-}(x) \coloneq U(x,\_) & p^1_{\exists U}(x) \coloneq T(x,\_) & p^2_R(x,y) \coloneq U(x,y) \\
p^1_{\exists S}(x) \coloneq U(\_,x) & p^1_{\exists T^-}(x) \coloneq P(x,\_) & p^1_{\exists U}(x) \coloneq U(x,\_) & p^2_R(x,y) \coloneq T(x,y) \\
p^1_{\exists S}(x) \coloneq R(x,\_) & p^1_{\exists T^-}(x) \coloneq U(\_,x) & p^1_{\exists U}(x) \coloneq P(x,\_) & p^2_R(x,y) \coloneq R(x,y) \\
p^1_{\exists S}(x) \coloneq S(x,\_) & p^1_{\exists T^-}(x) \coloneq R(x,\_) & p^1_{\exists U}(x) \coloneq R(x,\_) & p^2_R(x,y) \coloneq U(y,x) \\
p^1_{\exists S}(x) \coloneq T(x,\_) & p^1_{\exists T^-}(x) \coloneq T(x,\_) & p^1_{\exists U}(x) \coloneq U(\_,x) & p^2_R(x,y) \coloneq T(y,x) \\
p^1_{\exists S}(x) \coloneq U(x,\_) & p^1_{\exists T^-}(x) \coloneq T(\_,x) & p^1_{\exists U}(x) \coloneq T(\_,x) & \\
p^1_{\exists S}(x) \coloneq P(x,\_) & & &
\end{array}
$$

(c) Nonrecursive datalog query produced by Presto

Figure 2: TBox, query, and rewriting of Example 1.

in Q'_p. Obviously, we have that Q'_p and Q''_p are equivalent. Then, we prove that for every CQ r_p in Q''_p there exists a CQ r_c in Q'_c such that r_p is equal to r_c up to variable names, which easily follows from the definition of Presto and the CGLLR algorithm. This implies that r_p is contained in r_c, which in turn proves that Q'_p is contained in Q'_c.

Then, we show that Q'_c is contained in Q''_p. This part of the proof is less trivial, since there are CQs in Q'_c that are not equal (up to variable renaming) to any CQ in Q''_p. For every such CQ r_c, we prove that there exists a CQ r_p in Q''_p such that r_c is contained in r_p. More precisely, we prove that, for every $r_c \in Q'_c$, there exists a CQ $r_p \in Q''_p$ such that r_c is *b-contained* in r_p, i.e., there exists a homomorphism from r_p to r_c that maps bound variables of r_p to bound terms of r_c. The proof is by induction on the structure of Q'_c. We consider a bottom-up inductive definition of Q'_c in which $(Q'_c)_i$ is the UCQ computed by CGLLR after i iterations. The base case is immediate. As for the inductive case, let r_c be the CQ added by CGLLR at its $i+1$-th iteration. then, there are two possible cases:
(1) r_c has been generated by the atom-rewrite-rule from a CQ $r'_c \in (Q'_c)_i$ using a TBox inclusion I. In this case, by the inductive hypothesis it follows that there exists a CQ $r'_p \in Q''_p$ such that the atom-rewrite-rule can be applied to r'_p using the same TBox inclusion I: let r_p be the CQ thus obtained. Now, it is immediate to verify that the usage of OA-predicates in Q'_p guarantees that Q''_p is closed under application of the atom-rewrite-rule: hence, r_p belongs to Q''_p. Moreover, by definition of the atom-rewrite-rule (which does not eliminate ej-vars, thus it cannot introduce new unbound variables), it follows that r_c is b-contained in r_p;
(2) r_c has been generated by the reduce-rule from a CQ $r'_c \in (Q'_c)_i$. In this case, there are two possible cases: (2.a) the reduce-rule does not eliminate an ej-var, i.e., it does not introduce a new unbound variable. In this case, by the inductive hypothesis, there exists $r'_p \in Q''_p$ such that r'_c is b-contained in r'_p. Now, it is immediate to see that r'_c is also b-contained in r_p, hence the thesis follows; (2.b) the reduce-rule eliminates an ej-var, i.e., it introduces a new unbound variable. In this case, it can be proved that there exists a corresponding rule r_p generated by the $EliminateEJVar$ step such that r_c is b-contained in r_p. The proof of this property is again by induction on the structure of Q'_c, and a crucial role in this proof is played by the notion of MGS.

We have thus proved that Q'_c is equivalent to Q'_p. Then, since Q'_c is a perfect reformulation of Q' ((Calvanese et al. 2007, Lemma 39)) and in turn Q' is equivalent to Q, it follows that Q'_p is a perfect reformulation of Q. □

We denote with Q'_s is the nr-datalog query computed by Presto after the first $Split$ step (i.e., right before entering the repeat-until cycle); moreover, *#elim*(r) denotes the number of eliminable ej-vars in rule r. The following property follows from the fact that every auxiliary function can be computed in polynomial time, due to the computational complexity of TBox reasoning in *DL-Lite*$_R$.

Theorem 3 *Presto*$(Q, \mathcal{T})$ *runs in polynomial time with respect to the size of the TBox, and in exponential time with respect to* $\max_{r \in Q'_s}\{$*#elim*$(r)\}$. *The same upper bounds hold for the size of the query returned by Presto*$(Q, \mathcal{T})$.

Experimental results

We now report on a set of experiments on both query rewriting and query answering using Presto. The goal of these experiments is to compare Presto with the previous techniques for query answering and rewriting in *DL-Lite*.

Our experiments have been executed on the ontologies and queries used in the experimental evaluations of previous approaches: in particular, we have considered: (A) the experimental setting used in (Pérez-Urbina, Motik, and Horrocks 2009); (B) the experimental setting used in (Kontchakov et al. 2009); (C) the well-known LUBM ontology benchmark (`swat.cse.lehigh.edu/projects/lubm/`); (D) a newly created ontology. Due to space limits, here we provide detailed results only on tests (A) and (D). In some cases, we have additionally tested more complex queries than the ones already available, to verify the effectiveness of Presto on such queries. We have conducted our experiments on a Pentium dual core 2.00GHz CPU, with 2GB RAM and Windows Vista OS.

Query rewriting: comparison with Quonto and Requiem

To test the pure query rewriting abilities of the Presto approach (i.e., ignoring the cost of evaluating queries over the ABox), we have used the experimental setting of (Pérez-Urbina, Motik, and Horrocks 2009), which presents a detailed comparison between (different versions of) Requiem and an implementation of the PerfectRef query rewriting algorithm of (Calvanese et al. 2007). We have considered the three versions of the Requiem algorithm which are currently available (see `www.comlab.ox.ac.uk/projects/requiem`), our implementation of Presto, and the query rewriting module currently used in the Quonto system (called QPerfRef in the following), which is an optimized version of the PerfectRef algorithm of (Calvanese et al. 2007). (At this stage of our experiments, we have not considered Owlgres (Stocker and Smith 2008), since its rewriting technique is similar to PerfectRef.) Moreover, we have considered the same ontologies and queries used in (Pérez-Urbina, Motik, and Horrocks 2009), adding some more complex queries.

Here we present only an excerpt of the results of this analysis. In particular, Figure 3 displays the results obtained about six different ontologies (V, S, P1, P5X, A, U) of different complexity: for each ontology, several different conjunctive queries (Q1–Q7) of increasing size have been considered. In the figure, the empty cells correspond to the cases in which the rewriting was not returned after 30 minutes (times are expressed in milliseconds).

The table clearly shows the effectiveness of Presto. Except from few (simple) queries, Presto is almost always better than all the other techniques, in terms of both the time needed for computing the rewriting and the size of the rewriting generated. Moreover, the results show that Presto scales much better than all other techniques.

Ontology	Query ID	Number of rules/CQs of the generated query					Time (msec) to generate the query				
		QPerfRef	Requiem	RequiemF	RequiemG	PRESTO	QPerfRef	Requiem	RequiemF	RequiemG	PRESTO
V	Q1	15	15	15	15	16	1	1	1	15	1
V	Q2	10	10	10	10	11	31	15	15	15	15
V	Q3	72	576	72	72	28	47	328	655	468	15
V	Q4	185	185	185	185	43	94	78	125	125	15
V	Q5	30	30	30	30	14	140	16	31	15	16
V	Q6	1850	1850	1850	1850	53	920	2341	6428	8909	15
V	Q7	7200	7200	7200	7200	83	3323	36596	114599	327463	15
S	Q1	6	6	6	6	7	1	1	1	15	1
S	Q2	2	160	2	2	3	1	109	140	47	1
S	Q3	4	480	4	4	5	46	1248	1779	171	1
S	Q4	4	960	4	4	5	16	2341	3463	47	15
S	Q5	8	2880	8	8	7	47	51481	75349	296	16
S	Q6	8			8	9	32			60153	16
S	Q7	16			16	12	78			164611	16
P1	Q1	2	2	2	2	3	15	1	1	1	1
P1	Q2	2	2	2	2	3	16	1	1	1	1
P1	Q3	2	2	2	2	3	31	1	1	1	1
P1	Q4	2	2	2	2	3	32	15	16	16	1
P1	Q5	2	2	2	2	3	47	15	16	31	15
P1	Q6	2	32	2	32	3	94	7098	7286	7238	15
P1	Q7	2	64	2	64	3	171	168379	172549	172218	16
P5X	Q1	10	14	14	14	11	15	16	16	15	1
P5X	Q2	50	77	25	25	16	46	46	63	63	1
P5X	Q3	250	390	58	58	16	125	297	499	639	15
P5X	Q4	1254	1953	179	179	16	749	6476	12247	16880	15
P5X	Q5	6330	9766	718	718	16	7239	223955	427426	567713	15
P5X	Q6	32338				16	114233				16
P5X	Q7					16					16
A	Q1	558	114	27	27	69	171	62	94	78	1
A	Q2	1739	74	50	50	52	592	47	63	94	1
A	Q3	4741	104	104	104	55	2200	94	140	374	15
A	Q4	6589	285	224	224	93	2340	156	234	374	15
A	Q5	66068	624	624	624	71	35365	672	1248	2247	16
A	Q6		2496	2496	2496	91		9221	18799	36443	15
A	Q7					131					31
U	Q1	5	2	2	2	6	1	1	1	1	1
U	Q2	1	148	1	1	1	1	78	93	47	1
U	Q3	12	224	4	4	8	31	156	234	15	16
U	Q4	5	1628	2	2	6	1	1998	4430	78	16
U	Q5	25	2960	10	10	11	31	9953	18157	297	16
U	Q6	40	2368	16	16	14	47	7322	14238	725	16
U	Q7	560	33152		224	28	296	1734331		121888	16

Figure 3: Results for query rewriting.

	query	Quonto	Quonto-Presto
0	q(x,y) :- A1(j), R(k,y), R(x,y), S(x,z), T(l,m)	33	3
1	q(x,y) :- S(z,k), R(x,z), T(y,z), R(x,x), P(x,m)	227	47
2	q(x,y) :- S(z,j), R(x,z), T(k,w), A1(z), R(y,w)	14478	64
3	q(x,y) :- A1(j), S(z,j), R(y,k), R(k,z), R('c7',x)	13369	453
4	q(x,y) :- R(z,j), T(j,w), R(x,k), P(w,y), S(z,'a4'), A1(k)	13791	7039
5	q(x,y) :- S(z,k), A1(i), T(k,w), T(x,z), P(w,j), R(y,j), T('c7',i)	7269	65
6	q(x,y) :- Q('a4',w), S(j,i), S(z,k), R(x,z), T(k,m), R(j,w), U(i,y), A1(m)	14341	127
7	q(x,y) :- S(j,i), S(z,k), R(x,z), T(k,m), R(j,w), Q(n,w), U(i,y), A1(m), W(y,n)	11956	47
8	q(x,y) :- S(j,i), T(n,u), S(z,k), S(u,x), W(k,n), T(k,m), U(x,z), R(j,w), Q(n,w), U(i,y), A1(m)	191226	16008
9	q(x,y) :- S(j,i), B2(t), S(z,k), R(x,z), W('a4',n), T(k,m), W(u,t), W(n,u), R(j,w), Q(n,w), U(i,y), A1(m)		64
10	q(x,y) :- R(n,u), T(k,l), W('a4',n), T(k,m), R(j,w), U(i,y), P(u,t), S(j,i), S(z,k), B2(t), R(x,z), C3(l), Q(n,w), A1(m)		548
11	q(x,y) :- S(l,r), W('a4',n), T(k,m), R(j,w), U(i,y), S(r,'c7'), P(n,u), R(u,t), S(j,i), B2(t), S(z,k), R(x,z), Q(z,l), Q(n,w), A1(m)		422
12	q(x,y) :- S(l,r), W('a4',n), T(k,m), R(j,w), U(i,y), S(r,'c7'), P(n,u), P(s,'a4'), R(u,t), S(j,i), B2(t), S(z,k), C3(s), R(x,z), Q(z,l), Q(n,w), A1(m)		846
13	q(x,y) :- S(l,r), W('a4',n), T(k,m), R(e,i), R(j,w), U(i,y), S(r,'c7'), P(n,u), T(y,e), P(s,'a4'), R(u,t), S(j,i), B2(t), S(z,k), C3(s), R(x,z), Q(z,l), Q(n,w), A1(m)		1998
14	q(x,y) :- S(l,r), W('a4',n), T(k,m), P(x,f), R(e,i), R(j,w), A(f), U(i,y), S(r,'c7'), P(n,u), T(y,e), P(s,'a4'), R(u,t), S(j,i), B2(t), S(z,k), C3(s), R(x,z), W(f,w), Q(z,l), Q(n,w), A1(m)		3875
15	q(x,y) :- S(l,r), W('a4',n), T(k,m), R(e,i), R(j,w), T(v,g), U(i,y), S(r,'c7'), P(n,u), T(y,e), P(s,'a4'), R(u,t), S(j,i), B2(t), R(i,v), S(z,k), C3(s), S(g,x), R(x,z), Q(z,l), Q(n,w), A1(m)		20860
16	q(x,y) :- R(x,p), S(l,r), W('a4',n), T(k,m), R(e,i), R(j,w), T(v,g), U(i,y), S(r,'c7'), P(n,u), T(y,e), A1(p), P(s,'a4'), R(u,t), S(j,i), B2(t), R(i,v), S(z,k), C3(s), S(g,x), R(x,z), Q(z,l), Q(n,w), A1(m)		20421
17	q(x,y) :- R(x,p), T(k,m), U(i,y), S(r,'c7'), P(n,u), T(y,e), A1(p), P(s,'a4'), S(j,i), S(z,k), R(i,v), Q(z,l), Q(n,w), S(l,r), W('a4',n), R(j,w), R(e,i), T(v,g), P(y,d), R(u,t), B2(t), S(g,x), C3(s), R(x,z), A1(m), U(d,z)		21174
18	q(x,y) :- R(x,p), T(k,m), U(i,y), P(f,'a1'), S(r,'c7'), P(n,u), T(y,e), A1(p), P(s,'a4'), S(j,i), S(z,k), R(i,v), Q(z,l), Q(n,w), S(z,f), S(l,r), W('a4',n), R(j,w), R(e,i), T(v,g), P(y,d), R(u,t), B2(t), C3(s), S(g,x), R(x,z), A1(m), U(d,z)		22724
19	q(x,y) :- R(x,p), T(k,m), U(i,y), P(f,'a1'), S(r,'c7'), P(n,u), T(y,e), A1(p), P(s,'a4'), S(j,i), S(z,k), R(i,v), T(j,h), P(h,x), Q(z,l), Q(n,w), S(z,f), S(l,r), W('a4',n), R(j,w), R(e,i), T(v,g), P(y,d), R(u,t), B2(t), C3(s), S(g,x), R(x,z), A1(m), U(d,z)		127862
20	q(x,y) :- R(x,p), T(k,m), U(i,y), P(f,'a1'), S(r,'c7'), P(n,u), T(y,e), A1(p), P(s,'a4'), S(j,i), S(z,k), R(i,v), T(j,h), P(h,x), R(o,h), Q(z,l), Q(n,w), S(z,f), S(l,r), W('a4',n), R(j,w), R(e,i), T(v,g), P(y,d), R(u,t), A(o), B2(t), C3(s), S(g,x), R(x,z), A1(m), U(d,z)		141279

Figure 4: Results for query answering (query rewriting and evaluation).

We remark that some of these ontologies are actually expressed in a language that is slightly more expressive than *DL-Lite*$_R$, due to the presence of qualified existential concepts on the right-hand side of concept inclusion assertions. Differently from both Requiem and QPerfRef, the Presto algorithm is not optimized to handle qualified existential concepts, therefore, to deal with such expressions in Presto, an encoding of qualified existential concepts using auxiliary role names and role inclusions is needed, which causes an increase in the size of the TBox and of the ontology alphabet. Nevertheless, the results show that Presto is able to compete with the methods that explicitly handle qualified existential concepts.

Notice also that, in some cases, even if the size of the rewriting computed by both Requiem and QPerfRef is small, these algorithms take a considerable amount of time to compute the rewriting. Roughly, this is due to the fact that, differently from Presto, even when the number of actual solutions (conjunctive queries) is not high, the search space scanned by such algorithms may be very large, i.e., a huge number of candidate solutions may be generated.

Query answering: comparison with Quonto

Finally, in order to test the overall effectiveness of our approach for query answering, we have experimented the evaluation of the queries generated by Presto (after a translation in SQL) on some of the most popular current RDBMSs (IBM DB2, PostgreSQL, and MySQL). To this aim, we have produced a modified version of the Quonto system based on the Presto query rewriting algorithm: we call Quonto-Presto such a system. We have compared the query answering performance of Quonto-Presto with the original Quonto system on the various versions of the LUBM ontology (which provides large ABoxes) and on an ontology explicitly created for this purpose (the TBox of such an ontology is the one reported in Figure 2(a)). Figure 4 displays the results obtained on the latter ontology. For each of the queries displayed in the first column of the table, each row reports the total query evaluation time (query rewriting plus evaluation of the SQL translation of the query on the DBMS). Empty cells represent the cases in which Quonto did not produce any answer before the timeout of 30 minutes. These results are relative to the tests performed using MySQL Server 5.1 (however, analogous results have been otained with the other DBMSs mentioned above).

The results clearly show that not only Presto allows for a more efficient query rewriting than QPerfRef, but also the evaluation of (the SQL translation of) the queries generated by Presto is more efficient than the evaluation of the UCQs generated by QPerfRef. In other words, our results show that the increased complexity in the structure of the query (from UCQ to nr-datalog query) does not actually compromise the gain obtained by Presto in terms of size of the rewritten query. These results have been confirmed by the query evaluation tests that we have performed on the various versions of the LUBM ontology: with the exception of few cases (corresponding to very short queries and/or queries with no join variables, where Quonto is already very efficient), Quonto-Presto outperforms Quonto.

Conclusions

In this paper we have presented a new query rewriting method for *DL-Lite*. We believe that our technique is extremely significant, since it overcomes serious limitations of the previous query answering techniques for *DL-Lite*. Presto allows for providing a new upper bound on the size of the perfect reformulation of a UCQ in *DL-Lite*: this result has not only a theoretical significance, but also a strong practical impact, since it allows for effectively solving the problem of answering "real" complex queries over *DL-Lite* ontologies, as witnessed by our experiments.

The present work can be extended in several directions. First of all, the present version of Presto can be certainly optimized in many ways. For instance, the algorithm can be improved in the case when there are no role inclusion assertions in the TBox (e.g., for *DL-Lite*$_F$ TBoxes): in fact, the problem of computing perfect reformulations in this setting is significantly simplified by this assumption. An approach in this direction has been pursued in (Kontchakov et al. 2009), although under a strategy that mixes query answering by query rewriting with ABox preprocessing.

From the theoretical viewpoint, it would be interesting to see whether the ideas underlying Presto can be applied to DLs more expressive than *DL-Lite*: although it is well-known that CQs are not first-order rewritable in such DLs, it could be possible in principle to generalize Presto to improve query rewriting and query answering in such logics.

Finally, the current implementation of query answering based on Presto, which we used to run our experiments, is at a very early stage and needs several optimizations (e.g., in the translation scheme from nr-datalog to SQL queries).

References

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

Acciarri, A.; Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Palmieri, M.; and Rosati, R. 2005. QuOnto: QUerying ONTOlogies. In *Proc. of AAAI 2005*, 1670–1671.

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

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

Glimm, B.; Horrocks, I.; Lutz, C.; and Sattler, U. 2007. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of IJCAI 2007*, 399–404.

Kontchakov, R.; Lutz, C.; Toman, D.; Wolter, F.; and Zakharyaschev, M. 2009. Combined FO rewritability for conjunctive query answering in DL-Lite. In *Proc. of DL 2009*.

Pérez-Urbina, H.; Motik, B.; and Horrocks, I. 2009. A comparison of query rewriting techniques for DL-lite. In *Proc. of DL 2009*.

Stocker, M., and Smith, M. 2008. Owlgres: A scalable owl reasoner. In *Proc. of OWLED 2008*.