Efficient Query Answering for OWL 2

Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik

Oxford University Computing Laboratory, Oxford, UK
{hector.perez-urbina,ian.horrocks,boris.motik}@comlab.ox.ac.uk

Abstract. The QL profile of OWL 2 has been designed so that it is possible to use database technology for query answering via query rewriting. We present a comparison of our resolution based rewriting algorithm with the standard algorithm proposed by Calvanese et al., implementing both and conducting an empirical evaluation using ontologies and queries derived from realistic applications. The results indicate that our algorithm produces significantly smaller rewritings in most cases, which could be important for practicality in realistic applications.

1 Introduction

Ontologies can be used as conceptual schemas to provide an intuitive and unified view over one or more data repositories, allowing queries to be independent of the structure and location of the data. The use of ontologies as conceptual schemas for data repositories has been extensively studied in a variety of contexts, such as information integration [4]. The use of data repositories to store instance data is becoming increasingly important, for instance in the semantic Web, due to the scalability requirements of many applications and the widespread use of ontologies.

In OWL 2—a new version of the OWL ontology language that is currently a W3C candidate recommendation—scalability requirements are addressed by *profiles*—subsets of the language that enjoy desirable computational properties. The OWL 2 QL profile was designed to allow query answering via *query rewriting*: a query over an OWL 2 QL ontology and a set of instance data stored in a data repository can be answered by *rewriting* the query w.r.t. the ontology and then answering the rewritten query in the data repository. In this paper we focus on the case where the data is stored in a relational database and accessed using SQL queries, but the same technique could be applied to data stored in a triple store and accessed via SPARQL queries.

OWL 2 QL is based on DL-Lite$_R$—one of a family of description logics developed by Calvanese et al. [3]. The DL-Lite$_R$ rewriting algorithm of Calvanese et al., which we will refer to as CGLLR, transforms a conjunctive query Q and a DL-Lite$_R$ ontology $\mathcal{O}$ into a *union of conjunctive queries* $Q_\mathcal{O}$ such that the answers to Q and any set of instance data $\mathcal{A}$ can be obtained by evaluating $Q_\mathcal{O}$ over $\mathcal{A}$ only. This technique has been implemented in reasoners such as QuOnto[1] and

[1] http://www.dis.uniroma1.it/~quonto/

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 489–504, 2009.

Owlgres[2]. Unfortunately, as shown by Calvanese et al., the size of $Q_{\mathcal{O}}$ is worst-case exponential w.r.t. the size of Q and $\mathcal{O}$ [3], and as we show in Section 4, realistic ontologies and queries can result in $Q_{\mathcal{O}}$ being extremely large (e.g., containing tens of thousands of conjunctive queries). Thus, on the one hand, $Q_{\mathcal{O}}$ may be costly to compute, and, on the other hand, evaluation by RDBMSs may be costly or even unfeasible. Trying to produce small rewritings is therefore of critical importance to the practical application of query rewriting in general, and of OWL 2 QL in particular.

Motivated by the prospect of applying deductive database techniques to improve the scalability of reasoners, in our previous work [9] we considered the problem of query rewriting for various logics of the DL-Lite and $\mathcal{EL}$ families, the latter being the basis for the OWL 2 EL profile. Our algorithm takes as input a conjunctive query Q and an ontology $\mathcal{O}$, and uses a resolution-based technique to produce a rewritten query $Q_{\mathcal{O}}$. Although $Q_{\mathcal{O}}$ will, in general, be a (possibly recursive) *datalog query*, and thus necessitate the use of a deductive database system, the algorithm exhibits "pay-as-you-go" behavior for various logics. In particular, if $\mathcal{O}$ is a DL-Lite$_R$ ontology, then $Q_{\mathcal{O}}$ is a union of conjunctive queries. Our algorithm can therefore be seen as a generalization and extension of CGLLR.

In this paper we describe a simplified version of our algorithm that we will refer to as RQR (Resolution-based Query Rewriting). Like CGLLR, RQR rewrites a query w.r.t. a DL-Lite$_R$ ontology to produce a union of conjunctive queries. RQR differs from CGLLR mainly in the way it handles existential restrictions in an ontology. First, RQR uses functional terms to keep track of successors whose existence is implied by such restrictions, while CGLLR relies on a so-called reduction step. Second, RQR directly handles qualified existential restrictions (i.e., those where the restriction class is not owl:Thing), whereas CGLLR requires the elimination of such restrictions using an encoding that introduces new "auxiliary" properties. We describe both algorithms in Section 3 and further discuss their differences in Section 3.3.

Both the reduction step and the introduction of auxiliary properties can increase the size of the rewriting. Therefore, we conjectured that RQR will often produce smaller rewritings than CGLLR. In order to test the practicality of query rewriting and the efficiency of the different rewriting techniques, we have implemented RQR in a query rewriting system that we call REQUIEM[3] (REsolution-based QUery rewrIting for Expressive Models), and compared its behavior with that of our implementation of CGLLR that we refer to as C. The comparison uses a benchmark suite containing realistic DL-Lite$_R$ ontologies and test queries as well as some artificial ontologies and queries designed to highlight the differences between the two algorithms. The benchmark suite also included versions of the ontologies in which qualified existential restrictions have been explicitly encoded using auxiliary properties, as this allowed us to compare the two

[2] http://pellet.owldl.com/owlgres/

[3] http://www.comlab.ox.ac.uk/projects/requiem/

implementations in cases where RQR's native handling of qualified existential restrictions is not advantageous.

Both algorithms are amenable to optimizations that can reduce the size of the rewritings. One obvious optimization would be to use query subsumption checks to eliminate redundant conjunctive queries from the rewriting; we discuss this and other optimization techniques in Section 3.4. In order to compare optimized versions of the two algorithms we additionally implemented REQUIEM-SC and C-SC, both of which first compute the rewriting as in the original version, and then apply the above-mentioned query subsumption optimization to the result.

Our evaluation showed that, even for ontologies in which qualified existential restrictions were already encoded, REQUIEM produced significantly smaller rewritings than C in most cases. In one case, for instance, C exceeded the maximum allowed run time (600 seconds) after producing more than 42,000 conjunctive queries; in contrast, REQUIEM completed the rewriting in less than half a second having produced only 624 queries. Moreover, the rewritings produced by REQUIEM were often similar or identical to those produced by REQUIEM-SC, something that was less often the case for C and C-SC; using RQR should, therefore, reduce the need for (potentially costly) query subsumption checking.

2 Ontology-Based Data Access via Query Rewriting

We next describe how to answer queries over an OWL 2 QL ontology and a database via query rewriting, illustrating the process by means of an example.

Suppose we have a relational database DB_0 containing a table `Professor` with attributes `name`, `department`, and `telephone`; and a table `Student` with attributes `name`, `major`, `address`, and `tutor`. We can use a suitable ontology as a conceptual schema that describes the structure of the data. For example, we might use the following ontology $\mathcal{O}_0$ to describe DB_0:[4]

$$\mathrm{Teacher} \sqsubseteq \exists \mathrm{teaches} \tag{1}$$

$$\mathrm{Professor} \sqsubseteq \mathrm{Teacher} \tag{2}$$

$$\exists \mathrm{hasTutor}^{-} \sqsubseteq \mathrm{Professor} \tag{3}$$

Axiom (1) states that teachers teach at least someone, axiom (2) states that professors are teachers, and axiom (3) states that the range of the property hasTutor is Professor.

Given suitable mappings from the classes and properties in the ontology to data in the database, queries posed in terms of the ontology can be answered using the database. This has several advantages: on the one hand, queries can be posed in terms of the conceptual structure of the data rather than its arrangement in the database, and on the other hand, the database provides data persistence and scalability.

Mappings from the ontology to the database are typically defined using expressions of the form $D \mapsto Q_D$, where D is a class or property occurring in

[4] We use the description logic syntax for the sake of compactness.

the ontology and Q_D is an SQL query over the database; Q_D could, however, equally well be a SPARQL query that accesses data in an RDF triple store. In our example, the mapping $\mathcal{M}_0$ between $\mathcal{O}_0$ and DB_0 is defined as follows:

$$\begin{aligned} \text{Professor} &\mapsto \texttt{SELECT Name FROM Professor} \\ \text{hasTutor} &\mapsto \texttt{SELECT Name, Tutor FROM Student} \end{aligned}$$

Queries posed over the ontology are answered in two steps: first, the ontology is used to rewrite the query into a union of conjunctive queries; and second, the mappings are used to transform the rewritten query into an SQL query and evaluate it using an RDBMS. For example, consider the query

$$Q_0(x) \leftarrow \text{teaches}(x, y) \tag{4}$$

posed over $\mathcal{O}_0$. The rewriting $Q_{\mathcal{O}_0}$ of query (4) w.r.t. $\mathcal{O}_0$ contains (4) and the following queries:

$$Q_0(x) \leftarrow \text{Teacher}(x) \tag{5}$$
$$Q_0(x) \leftarrow \text{Professor}(x) \tag{6}$$
$$Q_0(x) \leftarrow \text{hasTutor}(y, x) \tag{7}$$

Transforming $Q_{\mathcal{O}_0}$ into an SQL query $\mathsf{sql}(Q_{\mathcal{O}_0})$ basically amounts to using $\mathcal{M}_0$ to replace each class or property D occurring in a query contained in $Q_{\mathcal{O}_0}$ with the corresponding SQL query Q_D, and forming the union of the resulting queries. Note that $\mathcal{M}_0$ does not contain a mapping for every class and property of $\mathcal{O}_0$. The answer to any query containing an atom for which there is no mapping will necessarily be empty, and we can therefore discard such queries. Consequently, queries (4) and (5) can be discarded in our example. As a result, we obtain the following rewritten SQL query:

$$\begin{aligned} \mathsf{sql}(Q_{\mathcal{O}_0}) = \texttt{SELECT Name FROM Professor UNION} \\ \texttt{SELECT Tutor FROM Student} \end{aligned}$$

This query can now be directly evaluated in the RDBMS to compute the answers to the original query $Q_0(x)$.

In the rest of the paper we focus on the problem of computing the rewriting $Q_{\mathcal{O}}$ of a given query Q w.r.t. an OWL 2 QL ontology $\mathcal{O}$.

3 Query Rewriting Algorithms

In this section we describe the RQR and CGLLR query rewriting algorithms, discuss the differences between them, and present various optimizations that can help us reduce the size of the rewritings.

Before presenting the algorithms, we introduce some notation and definitions. We use the well-known notions of constants, variables, function symbols, terms, and atoms of first-order logic. A Horn clause C is an expression

of the form $D_0 \leftarrow D_1 \wedge ... \wedge D_n$, where each D_i is an atom. The atom D_0 is called the *head*, and the set $\{D_1, ..., D_n\}$ is called the *body*. We require that all the variables occurring in the head of C occur at least in one of its body atoms. For instance, the expression $\text{teaches}(x, f(x)) \leftarrow \text{Professor}(x)$ is a Horn clause. The *depth* of a term t is defined as $\mathsf{depth}(t) = 0$ if t is a constant or a variable, and $\mathsf{depth}(f(s)) = 1 + \mathsf{depth}(s)$ if t is a functional term $f(s)$. The notion of depth is extended to an atom $R(t_1, \ldots, t_n)$ in the natural way: $\mathsf{depth}(R(t_1, \ldots, t_n)) = \max(\mathsf{depth}(t_i))$ for $1 \leq i \leq n$. An atom D occurring in a Horn clause C is said to be the *deepest* in C if $\mathsf{depth}(D) \geq \mathsf{depth}(D_i)$ for every atom D_i of C. For instance, the atom $\text{teaches}(x, f(x))$ is the deepest in the previously mentioned example clause.

A *conjunctive query* (CQ) Q posed over an ontology $\mathcal{O}$ is a Horn clause whose head predicate does not occur in $\mathcal{O}$, and whose body predicates are class and property names occurring in $\mathcal{O}$. For instance, (4) is a CQ over the ontology $\mathcal{O}_0$. A *union of conjunctive queries* (UCQ) over $\mathcal{O}$ is a set of conjunctive queries over $\mathcal{O}$ with the same head up to variable renaming [1]. For instance, the query $Q_{\mathcal{O}_0}$ composed of queries (4)–(7) is a UCQ over the ontology $\mathcal{O}_0$. A tuple of constants $\vec{a}$ is a *certain answer* to a UCQ Q over $\mathcal{O}$ and a set of instance data $\mathcal{A}$ iff $\mathcal{O} \cup \mathcal{A} \cup Q \models Q_P(\vec{a})$, where Q_P is the head predicate of Q, and Q is considered to be a set of universally quantified implications with the usual first-order semantics. The set of all answers to Q over $\mathcal{O}$ and $\mathcal{A}$ is denoted by $\mathsf{ans}(Q, \mathcal{O} \cup \mathcal{A})$. Given a conjunctive query Q and an ontology $\mathcal{O}$, a query $Q_{\mathcal{O}}$ is said to be a *rewriting* of Q w.r.t. $\mathcal{O}$ if $\mathsf{ans}(Q, \mathcal{O} \cup \mathcal{A}) = \mathsf{ans}(Q_{\mathcal{O}}, \mathcal{A})$ for every $\mathcal{A}$.

Both algorithms compute the rewriting $Q_{\mathcal{O}}$ of a given query Q w.r.t. a DL-Lite$_R$ ontology $\mathcal{O}$. DL-Lite$_R$ is the basis for OWL 2 QL. Extending the algorithms to handle the additional features of OWL 2 QL (e.g., datatypes, negative inclusions) is straightforward; we omit the details for the sake of simplicity.

3.1 CGLLR

The algorithm computes $Q_{\mathcal{O}}$ by using the axioms of $\mathcal{O}$ as rewrite rules and applying them to the body atoms of Q. The algorithm is shown in Algorithm 1. The partial function ref takes as input an axiom α and an atom D, and returns an atom $\mathsf{ref}(D, \alpha)$ as follows.

- If $D = A(x)$, then we have that (i) if $\alpha = B \sqsubseteq A$, then $\mathsf{ref}(D, \alpha) = B(x)$; (ii) if $\alpha = \exists P \sqsubseteq A$, then $\mathsf{ref}(D, \alpha) = P(x, \_)$; and (iii) if $\alpha = \exists P^- \sqsubseteq A$, then $\mathsf{ref}(D, \alpha) = P(\_, x)$.
- If $D = P(x, \_)$, then we have that (i) if $\alpha = A \sqsubseteq \exists P$, then $\mathsf{ref}(D, \alpha) = A(x)$; (ii) if $\alpha = \exists S \sqsubseteq \exists P$, then $\mathsf{ref}(D, \alpha) = S(x, \_)$; and (iii) if $\alpha = \exists S^- \sqsubseteq \exists P$, then $\mathsf{ref}(D, \alpha) = S(\_, x)$.
- If $D = P(\_, x)$, then we have that (i) if $\alpha = A \sqsubseteq \exists P^-$, then $\mathsf{ref}(D, \alpha) = A(x)$; (ii) if $\alpha = \exists S \sqsubseteq \exists P^-$, then $\mathsf{ref}(D, \alpha) = S(x, \_)$; and (iii) if $\alpha = \exists S^- \sqsubseteq \exists P^-$, then $\mathsf{ref}(D, \alpha) = S(\_, x)$.
- If $D = P(x, y)$, then we have that (i) if either $\alpha = S \sqsubseteq P$ or $\alpha = S^- \sqsubseteq P^-$, then $\mathsf{ref}(D, \alpha) = S(x, y)$; and (ii) if either $\alpha = S \sqsubseteq P^-$ or $\alpha = S^- \sqsubseteq P$, then $\mathsf{ref}(D, \alpha) = S(y, x)$.

Input: Conjunctive query Q, DL-Lite$_R$ ontology $\mathcal{O}$
$Q_\mathcal{O} = \{Q\}$;
repeat
  foreach *query* $Q' \in Q_\mathcal{O}$ **do**
    (reformulation) **foreach** *atom* D *in* Q' **do**
      foreach *axiom* $\alpha \in \mathcal{O}$ **do**
        if α *is applicable to* D **then**
          $Q_\mathcal{O} = Q_\mathcal{O} \cup \{Q'[D/\mathsf{ref}(D, \alpha)]\}$;
        end
      end
    end
    (reduction) **forall** *atoms* D_1, D_2 *in* Q' **do**
      if D_1 *and* D_2 *unify* **then**
        $\sigma = \mathsf{MGU}(D_1, D_2)$;
        $Q_\mathcal{O} = Q_\mathcal{O} \cup \{\lambda(Q'\sigma))\}$;
      end
    end
  end
until *no query unique up to variable renaming can be added to* $Q_\mathcal{O}$;
return $Q_\mathcal{O}$;

Algorithm 1. The CGLLR algorithm

Input: Conjunctive query Q, DL-Lite$_R$ ontology $\mathcal{O}$
$R = \Xi(\mathcal{O}) \cup \{Q\}$;
repeat
  (saturation) **forall** *clauses* C_1, C_2 *in* R **do**
    $R = R \cup \mathsf{resolve}(C_1, C_2)$;
  end
until *no query unique up to variable renaming can be added to* R ;
$Q_\mathcal{O} = \{C \mid C \in \mathsf{unfold}(\mathsf{ff}(R))$, and C has the same head predicate as $Q\}$;
return $Q_\mathcal{O}$;

Algorithm 2. Our resolution-based algorithm

If $\mathsf{ref}(D, \alpha)$ is defined for α and D, we say that α is *applicable* to D. The expression $Q[D/D']$ denotes the CQ obtained from Q by replacing the body atom D with a new atom D'. The function MGU takes as input two atoms and returns their most general unifier [1]. The function λ takes as input a CQ Q and returns a new CQ obtained by replacing each variable that occurs only once in Q with the symbol "_".

Starting with the original query Q, CGLLR continues to produce queries until no new queries can be produced. In the *reformulation* step the algorithm rewrites the body atoms of a given query Q' by using applicable ontology axioms as rewriting rules, generating a new query for every atom reformulation. Then, in the *reduction* step the algorithm produces a new query $\lambda(Q'\sigma)$ for each pair of body atoms of Q' that unify.

3.2 RQR

The algorithm first transforms Q and $\mathcal{O}$ into clauses and then computes $Q_{\mathcal{O}}$ by using a resolution-based calculus to derive new clauses from the initial set. The procedure is presented in Algorithm 2, where we show only those parts of the original algorithm that are relevant to DL-Lite$_R$. The expression $\Xi(\mathcal{O})$ denotes the set of clauses obtained from $\mathcal{O}$ according to Table 1. The function resolve takes two clauses C_1 and C_2, and it returns a set containing every clause C_R that can be obtained by combining the atoms of C_1 and C_2 according to the *inference templates* shown in Table 2. A template of the form $\frac{P_1 \quad P_2}{R}$ denotes that, if C_1 is a clause of the form of P_1 and C_2 is a clause of the form of P_2, then $\mathsf{resolve}(C_1, C_2)$ contains all clauses of the form of R that can be constructed from C_1 and C_2; otherwise, $\mathsf{resolve}(C_1, C_2) = \emptyset$. The function ff takes a set of clauses N and returns the subset of the function-free clauses in N. The function unfold takes a set of clauses N, and returns the set obtained by unfolding every clause in N; for example, if we have that $N = \{Q_P(x) \leftarrow A(x), A(x) \leftarrow B(x)\}$, then $\mathsf{unfold}(N) = N \cup \{Q_P(x) \leftarrow B(x)\}$, where $Q_P(x) \leftarrow B(x)$ is the result of unfolding $A(x) \leftarrow B(x)$ into $Q_P(x) \leftarrow A(x)$. A formal description of the unfolding step can be found in [9].

RQR computes $Q_{\mathcal{O}}$ in three steps: first, in the *clausification* step, the algorithm transforms Q and $\mathcal{O}$ into a set of clauses $\Xi(\mathcal{O}) \cup \{Q\}$; second, in the *saturation* step, the algorithm continues to produce clauses until no new clauses can be produced; third, in the *unfolding* and *pruning* step, clauses that are not function free are discarded, the remaining clauses are unfolded, and then clauses that do not have the same head predicate as Q are also discarded.

3.3 Differences

The algorithms mainly differ in the way they handle existential restrictions. This difference is twofold: first, while RQR deals with axioms containing existential quantifiers on the right-hand side by introducing functional terms, CGLLR does so by restricting the applicability of such axioms and relying on the reduction step; second, unlike RQR, CGLLR does not handle qualified existential restrictions natively—that is, there is no rewriting rule for axioms of the form $A \sqsubseteq \exists R.B$; instead, the algorithm requires a preliminary step in which each such axiom occurring in $\mathcal{O}$ is replaced with a set of axioms $\{A \sqsubseteq \exists P_1, \exists P_1^- \sqsubseteq B, P_1 \sqsubseteq R\}$, where P_1 is a new atomic property not occurring in $\mathcal{O}$. We explore these differences and their impact on the size of the rewritings by means of an example.

Consider an OWL 2 QL ontology $\mathcal{O}_1$ that consists of the following axiom

$$\text{Professor} \sqsubseteq \exists \text{teaches.Student}, \tag{8}$$

which states that a professor teaches at least some student, and the query

$$Q_1(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y). \tag{9}$$

Table 1. Translating $\mathcal{O}$ into a set of clauses $\Xi(\mathcal{O})$

DL-Lite$_R$ clause	DL-Lite$_R$ axiom
$B(x) \leftarrow A(x)$	$A \sqsubseteq B$
$P(x, f(x)) \leftarrow A(x)$	$A \sqsubseteq \exists P$
$P(x, f(x)) \leftarrow A(x)$	$A \sqsubseteq \exists P.B$
$B(f(x)) \leftarrow A(x)$	
$P(f(x), x) \leftarrow A(x)$	$A \sqsubseteq \exists P^-$
$P(f(x), x) \leftarrow A(x)$	$A \sqsubseteq \exists P^-.B$
$B(f(x)) \leftarrow A(x)$	
$A(x) \leftarrow P(x, y)$	$\exists P \sqsubseteq A$
$A(x) \leftarrow P(y, x)$	$\exists P^- \sqsubseteq A$
$S(x, y) \leftarrow P(x, y)$	$P \sqsubseteq S, P^- \sqsubseteq S^-$
$S(x, y) \leftarrow P(y, x)$	$P \sqsubseteq S^-, P^- \sqsubseteq S$

Note 1. Each axiom of the form $A \sqsubseteq \exists R.B$ is uniquely associated with a function symbol f.

Table 2. Inference templates for the function resolve

$$\frac{C(x) \leftarrow B(x) \quad B(f(x)) \leftarrow A(x)}{C(f(x)) \leftarrow A(x)}$$

$$\frac{B(x) \leftarrow P(x, y) \quad P(x, f(x)) \leftarrow A(x)}{B(x) \leftarrow A(x)} \qquad \frac{B(x) \leftarrow P(x, y) \quad P(f(x), x) \leftarrow A(x)}{B(f(x)) \leftarrow A(x)}$$

$$\frac{B(x) \leftarrow P(y, x) \quad P(x, f(x)) \leftarrow A(x)}{B(f(x)) \leftarrow A(x)} \qquad \frac{B(x) \leftarrow P(y, x) \quad P(f(x), x) \leftarrow A(x)}{B(x) \leftarrow A(x)}$$

$$\frac{S(x, y) \leftarrow P(x, y) \quad P(x, f(x)) \leftarrow A(x)}{S(x, f(x)) \leftarrow A(x)} \qquad \frac{S(x, y) \leftarrow P(x, y) \quad P(f(x), x) \leftarrow A(x)}{S(f(x), x) \leftarrow A(x)}$$

$$\frac{S(x, y) \leftarrow P(y, x) \quad P(x, f(x)) \leftarrow A(x)}{S(f(x), x) \leftarrow A(x)} \qquad \frac{S(x, y) \leftarrow P(y, x) \quad P(f(x), x) \leftarrow A(x)}{S(x, f(x)) \leftarrow A(x)}$$

$$\frac{Q_P(\vec{u}) \leftarrow B(t) \wedge \bigwedge D_i(\vec{t_i}) \quad B(f(x)) \leftarrow A(x)}{Q_P(\vec{u})\sigma \leftarrow A(x)\sigma \wedge \bigwedge D_i(\vec{t_i})\sigma}$$

where $\sigma = \mathsf{MGU}(B(t), B(f(x)))$, and $B(t)$ is deepest in its clause.

$$\frac{Q_P(\vec{u}) \leftarrow P(s, t) \wedge \bigwedge D_i(\vec{t_i}) \quad P(x, f(x)) \leftarrow A(x)}{Q_P(\vec{u})\sigma \leftarrow A(x)\sigma \wedge \bigwedge D_i(\vec{t_i})\sigma}$$

where $\sigma = \mathsf{MGU}(P(s, t), P(x, f(x)))$, and $P(s, t)$ is deepest in its clause.

$$\frac{Q_P(\vec{u}) \leftarrow P(s, t) \wedge \bigwedge D_i(\vec{t_i}) \quad P(f(x), x) \leftarrow A(x)}{Q_P(\vec{u})\sigma \leftarrow A(x)\sigma \wedge \bigwedge D_i(\vec{t_i})\sigma}$$

where $\sigma = \mathsf{MGU}(P(s, t), P(f(x), x))$, and $P(s, t)$ is deepest in its clause.

We first analyze the execution of CGLLR. Note that CGLLR cannot handle axiom (8) natively, and it must first be replaced with the following axioms:

$$\text{Professor} \sqsubseteq \exists R_{aux} \tag{10}$$

$$\exists R_{aux}^{-} \sqsubseteq \text{Student} \tag{11}$$

$$R_{aux} \sqsubseteq \text{teaches} \tag{12}$$

In the first iteration, axiom (12) is applicable to the atom $\text{teaches}(x, y)$ in (9). Similarly, axiom (11) is applicable to $\text{Student}(y)$ in (9). Therefore, we obtain the following queries in the reformulation step:

$$Q_1(x) \leftarrow R_{aux}(x, y) \wedge \text{Student}(y) \tag{13}$$

$$Q_1(x) \leftarrow \text{teaches}(x, y) \wedge R_{aux}(\_, y) \tag{14}$$

In this iteration no query can be obtained in the reduction step. In the next iteration, axiom (10) is not applicable to the atom $R_{aux}(x, y)$ in (13) because y occurs in (13) in more than one place. Axiom (10) cannot be applied to (13) because CGLLR does not keep track of information about role successors; furthermore, if we naively allowed existential quantification axioms to be applied, the resulting calculus would become unsound. To illustrate this point, suppose that (10) were applicable to $R_{aux}(x, y)$ in (13), and $\mathsf{ref}(R_{aux}(x, y), (10)) = \text{Professor}(x)$; we would then obtain the query

$$Q_1(x) \leftarrow \text{Professor}(x) \wedge \text{Student}(y). \tag{15}$$

Note that the relation between x and y is lost—that is, the fact that the individual represented by y must be a teaches-successor of the individual represented by x is not captured by query (15).

Although the applicability of (10) is restricted, axiom (11) is applicable to $\text{Student}(y)$ in (13). Similarly, axiom (12) is applicable to $\text{teaches}(x, y)$ in (14). Both reformulations produce the query

$$Q_1(x) \leftarrow R_{aux}(x, y) \wedge R_{aux}(\_, y). \tag{16}$$

In the next iteration, no axiom is applicable to any body atom of (16), so no query is added in the reformulation step. In the reduction step, however, the algorithm produces

$$Q_1(x) \leftarrow R_{aux}(x, \_) \tag{17}$$

by unifying the body atoms of (16). In the following iteration, axiom (10) is applicable to the only body atom of (17), producing

$$Q_1(x) \leftarrow \text{Professor}(x). \tag{18}$$

Note that without the reduction step, the algorithm would not have produced query (18). It can be easily verified that no more new queries can be produced; thus, CGLLR returns $\{(9), (13), (14), (16), (17), (18)\}$.

We now analyze the execution of RQR. According to Table 1, axiom (8) is translated into the following clauses:

$$\text{teaches}(x, f(x)) \leftarrow \text{Professor}(x) \tag{19}$$
$$\text{Student}(f(x)) \leftarrow \text{Professor}(x) \tag{20}$$

In the saturation step the algorithm produces

$$\mathsf{resolve}((9),(19)) = Q_1(x) \leftarrow \text{Professor}(x) \wedge \text{Student}(f(x)) \tag{21}$$
$$\mathsf{resolve}((9),(20)) = Q_1(x) \leftarrow \text{teaches}(x, f(x)) \wedge \text{Professor}(x) \tag{22}$$
$$\mathsf{resolve}((19),(22)) = Q_1(x) \leftarrow \text{Professor}(x) \tag{23}$$

Note the difference between queries (15) and (21). Since the function symbol f is uniquely associated with clause (19), unlike query (15), query (21) captures the fact that the individual represented by $f(x)$ must be a teaches-successor of the individual represented by x. It can easily be verified that no other clause is produced in the first step. Clearly, $\mathsf{ff}(R) = \{(9),(23)\}$. In this case, there is no unfolding to be done, so RQR returns $\{(9),(23)\}$.

As shown in the above example, the introduction of auxiliary properties can lead to an increase in the size of the rewritings. The reduction step alone, however, can also lead to larger rewritings. This situation arises especially in the case where part of the data of the database describes a graph. As a simple example, consider an OWL 2 QL ontology $\mathcal{O}_2$ that consists of the axiom

$$\text{Student} \sqsubseteq \exists \text{hasTutor}, \tag{24}$$

which states that a student has at least one tutor, and the query

$$Q_2(x) \leftarrow \text{hasTutor}(x, y) \wedge \text{hasTutor}(z, y) \wedge \text{hasTutor}(z, w) \wedge \text{hasTutor}(x, w). \tag{25}$$

When using CGLLR, axiom (24) is not applicable to query (25), so no query is produced in the reformulation step. However, every pair of body atoms in query (25) unify, and it is easy to see that for each query of this form with m body atoms, CGLLR produces $\binom{m}{2}$ new queries in the reduction step. Eventually, the reduction step produces the query

$$Q_2(x) \leftarrow \text{hasTutor}(x, \_). \tag{26}$$

Axiom (24) is now applicable to query (26), and the following query is produced in the reformulation step:

$$Q_2(x) \leftarrow \text{Student}(x) \tag{27}$$

Note, however, that several queries needed to be produced in the reduction step in order to produce query (27) in the reformulation step.

An important remark is in order. For every query Q' produced from a query Q in the reduction step, there is a substitution σ such that $Q\sigma \subseteq Q'$, in which case

we say that Q *subsumes* Q'. It is well known that every query that is subsumed by another can be discarded after the rewriting has been computed without affecting completeness [5]; however, identifying such queries is not straightforward since CGLLR does not keep track of which queries were produced in the reduction step. In our example, query (25) subsumes query (26) by the substitution $\sigma = \{z \mapsto x, w \mapsto y\}$; therefore, query (26) can be safely discarded from the final rewriting. In this case, however, note that query (26) subsumes query (25) as well; therefore, it is sensible to eliminate query (25) instead since it is larger. Since both queries subsume each other, we say that they are *equivalent*. Moreover, since query (26) is the minimal equivalent subquery of query (25), we say that query (26) is a *condensation* of query (25) [2]. The potential generation of condensations by the reduction step plays an important role in the optimization of the rewritings. We discuss this aspect further in the following section.

The use of functional terms makes RQR more goal-oriented, in the sense that it does not need to derive the "irrelevant" queries produced by the reduction step of CGLLR in order to be complete. Moreover, RQR handles qualified existential restrictions natively, whereas CGLLR needs to encode them away by introducing new properties and axioms.

3.4 Optimizations

As discussed in the introduction, both algorithms are amenable to optimization. One obvious optimization technique is to check subsumption between pairs of conjunctive queries and eliminate any query that is subsumed by another. Such a procedure can be simply (albeit not necessarily optimally) applied a posteriori to the rewritings produced by RQR and CGLLR.

It is important to note that using the query subsumption optimization with RQR and CGLLR does not necessarily result in exactly the same rewritings. This is due to the fact that the CGLLR reduction step may produce *condensations*. We illustrate this point with an example. Consider an OWL 2 QL ontology $\mathcal{O}_3$ that consists of the following axiom

$$\exists \text{teaches}^- \sqsubseteq \text{Student}, \tag{28}$$

which states that someone that is taught is a student, and the query

$$Q_3(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y). \tag{29}$$

CGLLR produces a set containing (29) and the following queries:

$$Q_3(x) \leftarrow \text{teaches}(x, y) \wedge \text{teaches}(\_, y) \tag{30}$$

$$Q_3(x) \leftarrow \text{teaches}(x, \_) \tag{31}$$

Note that query (31) was produced in the reduction step from (30) and it is a condensation of (30). In the query subsumption check we have that query (31) subsumes query (29), so the latter is discarded. Note, however, that query (31) subsumes query (30) and *vice versa*. Therefore, since it is sensible to discard the

larger query, the condensation (31) is kept and query (30) is discarded instead. It is easy to see that in the end we obtain {(31)}.

When using RQR, axiom (28) is translated into the following clause:

$$\text{Student}(x) \leftarrow \text{teaches}(y, x) \tag{32}$$

The algorithm then produces a set containing (29) and the following clause:

$$Q_3(x) \leftarrow \text{teaches}(x, y) \wedge \text{teaches}(z, y) \tag{33}$$

Since (33) subsumes (29), it is easy to see that after the query subsumption check we obtain {(33)}. As can be seen, (31) is slightly smaller than (33); it is also a condensation of (33). In our empirical evaluation (see Section 4), the optimized versions of RQR and CGLLR produced the same rewritings modulo the condensations produced by CGLLR. Modifying RQR to replace queries with their condensations before the query subsumption check would be straightforward.

Finally, we briefly describe two other well-known optimizations: forward and backward subsumption [2]. Both optimizations compare each new clause C produced in the saturation step with the set of previously generated clauses. In forward subsumption, C is discarded if the set of clauses already contains a clause C' that subsumes C; in backward subsumption, C' is removed from the set of clauses if it is subsumed by C.

Since RQR is based on a resolution calculus, both forward and backward subsumption can be straightforwardly applied without affecting completeness [2]. In the case of CGLLR, however, forward subsumption cannot be (straightforwardly) applied: every query produced in the reduction step is subsumed by another previously produced query; forward subsumption would thus effectively eliminate the reduction step, and so compromise completeness. For example, forward subsumption would remove query (26) in the above example, preventing the generation of query (27). It is not clear whether backward subsumption can be applied to CGLLR without affecting completeness.

4 Evaluation

In this section we present an empirical evaluation of our implementations of the RQR and CGLLR algorithms. RQR is implemented in a rewriting system that we call REQUIEM, while our CGLLR implementation is called C. We also implemented optimized versions of the two algorithms that try to reduce the size of the rewriting using an a posteriori query subsumption check—these are called REQUIEM-SC and C-SC, respectively. Note that C-SC eliminates queries that contain auxiliary properties (introduced by the encoding of qualified existential restrictions) *before* performing the query subsumption check. Both REQUIEM and C are available at REQUIEM's Web site.

The main goal of the evaluation is to compare the algorithms w.r.t. the *size of the rewritings* they produce. Simply counting the number of conjunctive queries

in each rewriting might not provide a fair comparison as the queries themselves could differ in size; we therefore additionally measured the total number of symbols needed to represent the complete rewriting in the standard datalog notation. We also measured the time taken for each rewriting procedure. In view of our relatively naïve implementations, however, this may not provide a very meaningful measure of the likely cost of the rewriting process. We therefore also measured the number of *inferences* performed by each algorithm, where by an inference we mean the derivation of a query. Note that the number of inferences is not necessarily the same as the number of queries in the final rewriting since an algorithm may derive the same query more than once.

Tests were performed on a PC running Windows XP with a 2.59 GHz Intel processor and 1.87 GB of RAM. We used Java 1.6.0 Update 7 with a maximum heap size of 256 MB. Tests were halted if execution time exceeded 600 seconds.

4.1 Test Ontologies and Queries

The test set mainly consists of DL-Lite$_R$ ontologies that were developed in the context of real applications, along with test queries that are based on canonical examples of queries used in the corresponding application.

`V` is an ontology capturing information about European history, and developed in the EU-funded VICODI project.[5] `S` is an ontology capturing information about European Union financial institutions, and developed for ontology-based data access [10]. `U` is a DL-Lite$_R$ version of LUBM[6]—a benchmark ontology developed at Lehigh University for testing the performance of ontology management and reasoning systems—that describes the organizational structure of universities. `A` is an ontology capturing information about abilities, disabilities, and devices, and developed to allow ontology-based data access for the South African National Accessibility Portal [7].

We additionally included two synthetic ontologies in our tests in order to provide a controlled scenario to help us understand the impact of the reduction step. `P1` and `P5` model information about graphs: nodes are represented by individuals, and vertices are assertions of the form edge(a, b). The ontology `P5` contains classes representing paths of length 1–5, while `P1` contains a class representing paths of length 1 only.

Finally, for every ontology containing qualified existential restrictions, we created an ontology where the relevant axioms have been replaced by applying the encoding required in CGLLR. We included these ontologies in order to measure the impact of the encoding and the saturation step separately. These ontologies are identified with the name of the original ontology and the suffix `X`. In our discussion, we refer to these ontologies as the *AUX ontologies* and we refer to the others as the *original ontologies*. All the ontologies and queries are available at REQUIEM's Web site.

[5] `http://www.vicodi.org/`

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

		Queries				Symbols				Inferences		Time			
O	Q	R	C	RSC	CSC	R	C	RSC	CSC	R/RSC	C/CSC	R	C	RSC	CSC
V	1	15	15	15	15	454	454	454	454	14	14	16	1	31	1
	2	10	11	10	10	762	812	762	762	9	10	16	1	47	15
	3	72	72	72	72	6,525	6,525	6,525	6,525	117	117	31	31	62	47
	4	185	185	185	185	11,911	11,911	11,911	11,911	328	328	62	47	141	110
	5	30	150	30	30	3,761	16,255	3,761	3,761	59	475	31	78	63	110
S	1	6	6	6	6	158	158	158	158	48	9	15	1	16	1
	2	160	204	2	2	11,422	13,680	154	66	689	876	78	78	109	141
	3	480	1,194	4	4	56,536	121,674	488	232	3,130	7,523	438	922	1,062	2,875
	4	960	1,632	4	4	111,092	173,088	468	224	5,841	10,270	828	1,109	2,171	4,156
	5	2,880	11,487	8	8	466,896	1,602,203	1,320	648	24,332	87,324	9,829	80,031	34,681	233,958
U	1	2	5	2	2	118	286	118	118	26	4	16	1	31	1
	2	148	287	1	1	10,378	18,496	68	30	307	589	47	62	93	125
	3	224	1,260	4	4	29,376	151,848	516	372	570	4,228	94	531	203	640
	4	1,628	5,364	2	2	113,270	348,782	124	56	5,028	18,541	797	4,610	4,093	8,390
	5	2,960	9,245	10	10	279,266	822,279	932	506	13,085	43,306	3,234	16,219	15,262	29,204
A	1	402	783	27	27	21,933	39,593	901	901	934	1,958	94	157	265	350
	2	103	1,812	50	50	7,122	116,137	3,783	3,783	308	4,986	47	687	78	719
	3	104	4,763	104	104	10,108	413,760	10,108	10,108	461	14,454	78	4,187	93	4,266
	4	492	7,251	224	224	33,454	461,549	16,069	16,069	1,405	21,377	156	8,000	422	8,250
	5	624	-	624	-	70,320	-	70,320	-	2,618	-	328	-	1,031	-
P1	1	2	2	2	2	42	42	42	42	1	1	1	1	1	1
	2	2	3	2	2	70	92	70	70	4	2	1	16	16	30
	3	2	7	2	2	98	262	98	98	9	9	1	16	16	30
	4	2	16	2	2	126	734	126	126	16	38	1	16	16	30
	5	2	33	2	2	154	1,824	154	154	25	129	15	31	16	47
P5	1	6	14	6	6	122	298	122	122	9	13	1	1	1	1
	2	10	86	10	10	286	2,814	286	286	75	141	15	15	15	31
	3	13	538	13	13	486	23,740	486	486	428	1,348	94	141	95	142
	4	15	3,620	15	15	708	200,696	708	708	2,238	12,471	922	3,546	1,045	3,607
	5	16	25,256	16	16	938	1,690,902	938	938	11,350	113,277	24,172	265,875	30,000	270,520
UX	1	5	5	5	5	286	286	286	286	39	4	16	1	31	1
	2	240	286	1	1	16,208	18,496	68	30	510	589	78	78	187	156
	3	1,008	1,248	12	12	125,304	151,848	1,452	1,060	3,240	4,228	641	532	2,093	2,484
	4	5,000	5,358	5	5	332,000	348,782	283	131	17,188	18,541	5,969	4,719	36,247	38,189
	5	8,000	9,220	25	25	737,000	822,279	2,240	1,220	37,635	43,306	19,141	15,844	106,055	108,034
AX	1	782	783	41	41	39,527	39,549	1,399	1,385	1,785	1,958	218	156	766	765
	2	1,781	1,812	1,431	1,431	114,537	116,137	91,576	91,145	5,073	4,986	1,016	688	5,547	5,282
	3	4,752	4,763	4,466	4,466	413,030	413,760	388,457	388,303	14,629	14,454	7,375	4,125	49,452	45,876
	4	7,100	7,251	3,159	3,159	454,807	461,549	199,604	197,955	21,137	21,377	13,032	8,047	72,781	79,377
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P5X	1	14	14	14	14	298	298	298	298	25	13	15	16	16	1
	2	77	86	25	25	2,616	2,814	784	728	182	141	31	15	63	47
	3	390	530	58	58	18,574	23,452	2,480	2,326	1,241	1,348	156	141	406	609
	4	1,953	3,476	179	179	120,232	193,608	10,010	9,520	7,832	12,471	2,375	3,469	9,610	26,438
	5	9,766	23,744	718	-	737,890	1,597,158	50,120	-	47,100	-	69,454	249,177	280,381	-

Fig. 1. Results

The number of classes, properties, and axioms are as follows:

	V	S	U	A	P1	P5	UX	AX	P5X
classes	194	18	34	74	2	6	35	74	6
properties	10	12	26	5	1	1	31	31	5
axioms	222	51	127	137	2	10	137	189	18

4.2 Results

Figure 1 shows the results of the empirical evaluation. For each ontology and query, the column "Queries" shows the number of conjunctive queries in the rewriting, the column "Symbols" shows the number of symbols needed to represent the rewriting in datalog notation, the column "Inferences" shows the number of inferences that were performed by each implementation to compute the

rewritings (note that the number of inferences for REQUIEM and REQUIEM-SC, and for C and C-SC, is always the same), and the column "Time" shows the number of milliseconds taken to compute the rewritings.

Comparing REQUIEM to C w.r.t. the original ontologies, it can be seen that REQUIEM produced smaller rewritings than C in 25 out of 30 cases and equal sized rewritings in the remaining 5 cases. Moreover, REQUIEM was often faster and performed fewer inference steps, particularly in non-trivial cases (i.e., where both implementations took more than 1,000ms). The differences in the size of the rewritings are often significant: in the fifth queries over `U`, `S`, and `P5`, for example, the rewritings produced by C contain between two and four times as many queries (and contain correspondingly larger numbers of symbols). In the fifth query over `A`, C had already produced more than 42,000 queries when it exceeded the 600 second time limit; in contrast, REQUIEM completed the rewriting in less than 350ms and produced only 624 queries.

When we compare REQUIEM to REQUIEM-SC, we can see that they produced the same rewritings in 19 out of 30 cases. In some cases, however, the rewriting produced by REQUIEM was much larger: in the fifth query over `S`, for example, REQUIEM's rewriting contained 2,880 queries compared to only 8 for REQUIEM-SC. As we might expect given the larger rewritings produced by C, it produced the same rewritings as C-SC in only 6 out of 30 cases. The differences in size were also generally larger: in the fifth query over `S`, for example, C's rewriting contained 11,487 queries compared to only 8 for C-SC. The large size of the rewritings produced by C also mean that performing query subsumption tests over these rewritings can be costly. In the fifth query over `S`, for example, C-SC takes nearly three times as long as C.

Comparing REQUIEM-SC to C-SC, we can see that REQUIEM-SC produced the same rewritings as C-SC in 21 out of 30 cases, larger rewritings in 8 cases, and a smaller rewriting in the remaining case (due to the fact that C-SC exceeded the maximum time of 600 seconds). The differences in size are minimal, and the number of queries in the rewritings is always the same (with the exception of the case where C-SC exceeded the time limit). Note that the smaller rewritings produced by C-SC are due to its generation of condensations (see Section 3).

If we turn our attention to the AUX ontologies, we can see that REQUIEM still produced smaller rewritings than C in 12 out of 15 cases, although the differences were less marked. Moreover, REQUIEM still performed less inferences than C in 9 out of 15 cases. In contrast to the results with the original ontologies, REQUIEM was slower than C in 10 out of 15 cases; the differences, however, were generally small.

Our analysis suggests that REQUIEM will produce significantly smaller rewritings than C and will be significantly faster, particularly in cases where the queries are relatively complex and/or the ontologies contain a relatively large number of qualified existential restrictions. The size of the rewritings produced in some cases also means that a query subsumption check may be prohibitively costly in practice with CGLLR, even when queries containing auxiliary properties are removed before performing the check. Moreover, the results for the

AUX ontologies suggest that the reduction step alone has a negative impact on the size of the rewritings—that is, the introduction of auxiliary properties does contribute to producing large rewritings, but it is not the only cause.

5 Future Work

We plan to implement an ontology-based data access system using REQUIEM enhanced with various optimizations (i.e., forward/backward subsumption, query subsumption, and query condensation); we expect such a system to perform well both w.r.t. the size of the rewritings and the time needed to compute them. The practicality of such a system is, however, still open, as our results suggest that there are cases where the rewritings may be too large to evaluate. In such cases, we believe that a further optimization that uses the mappings to prune irrelevant queries (as described in Section 2) might produce rewritings of manageable proportions. We plan to test our system with actual data in order to discover if this is indeed the case. Finally, we plan to extend the system to support all of OWL 2 QL, which mainly involves adding support for datatypes.

References

1. Baader, F., Snyder, W.: Unification Theory. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 8, vol. I, pp. 445–532. Elsevier Science, Amsterdam (2001)
2. Bachmair, L., Ganzinger, H.: Resolution Theorem Proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 2, vol. 1, pp. 19–100. North Holland, Amsterdam (2001)
3. 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 (2007)
4. Calvanese, D., Giacomo, G.D., Lenzerini, M., Nardi, D., Rosati, R.: Description Logic Framework for Information Integration. Principles of Knowledge Representation and Reasoning, 2–13 (1998)
5. Chang, C.-L., Lee, R.C.-T.: Symbolic Logic and Mechanical Theorem Proving. Academic Press, Inc., Orlando (1997)
6. Kazakov, Y.: Saturation-Based Decision Procedures for Extensions of the Guarded Fragment. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany (2006)
7. Keet, C.M., Alberts, R., Gerber, A., Chimamiwa, G.: Enhancing web portals with ontology-based data access: The case study of south africa's accessibility portal for people with disabilities. In: OWLED (2008)
8. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, Germany (2006)
9. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable Query Answering and Rewriting under Description Logic Constraints. J. of Applied Logic (to appear, 2009), `http://web.comlab.ox.ac.uk/people/publications/date/Hector.Perez-Urbina.html`
10. Rodriguez-Muro, M., Lubyte, L., Calvanese, D.: Realizing ontology based data access: A plug-in for protégé. In: Proc. of the Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008), pp. 286–289. IEEE Computer Society Press, Los Alamitos (2008)