A Comparison of Query Rewriting Techniques for DL-Lite

Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks

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

1 Introduction

An incomplete database is defined by a set of constraints and a partial database instance [1]. Answering conjunctive queries over incomplete databases is an important computational task that lies at the core of many problems, such as information integration [12], data exchange [9], and data warehousing [17]. Given a query and an incomplete database, the task is to compute the so-called *certain answers*—the tuples that satisfy the query in every database instance that conforms to the partial instance and satisfies the constraints [16]. It is well known that answering queries under general constraints is undecidable; therefore, the expressivity of the constraint languages considered is typically restricted in order to achieve decidability.

Description Logics (DLs) [3] can be viewed as very expressive but decidable first-order fragments, which makes them natural candidates for constraint languages. DLs are a family of knowledge representation formalisms that represent a given domain in terms of concepts (unary predicates), roles (binary predicates), and individuals (constants). A DL Knowledge Base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of a *terminological* component $\mathcal{T}$ called the TBox, and an *assertional* component $\mathcal{A}$ called the ABox. In analogy to incomplete databases, the TBox can be seen as a conceptual schema containing the set of constraints, and the ABox as some partial instance of the schema. The use of DLs as constraint languages has already proven to be useful in a variety of scenarios such as ontology-based information integration [7] and the Semantic Web [10]. Various DLs exist for which it is possible to transform a conjunctive query Q and a TBox $\mathcal{T}$ into a datalog query Q'—a so-called *rewriting* of Q w.r.t. $\mathcal{T}$—such that for every ABox $\mathcal{A}$, the answers to Q over $\mathcal{T}$ and $\mathcal{A}$, and the answers to Q' over $\mathcal{A}$ coincide. Thus, the problem of answering Q over $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ for a specific $\mathcal{A}$ can be reduced to evaluating the datalog query Q' over $\mathcal{A}$. A motivation for answering queries via query rewriting is the prospect of using existing deductive database systems to evaluate the computed rewritings, thus taking advantage of the query optimization strategies provided by such systems.

The problem of conjunctive query rewriting for DLs has been considered by Rosati [15] for the $\mathcal{EL}$ family of languages [2], and by Calvanese et al. for the DL-Lite family of languages [6]. The rewriting algorithm proposed by Calvanese

et al., which we refer to as the CGLLR algorithm, takes as input a conjunctive query Q and a DL-Lite$_R$ TBox $\mathcal{T}$ and computes a union of conjunctive queries that is a rewriting of Q w.r.t. $\mathcal{T}$. The CGLLR algorithm lies at the core of reasoners such as QuOnto[1] and Owlgres[2]. The aforementioned approaches are closely related; however, they have been developed for two different DL families. Motivated by the prospect of developing a unified algorithm, in our previous work [13] we described a novel rewriting algorithm for the DL $\mathcal{ELHI}$ [2]. The algorithm takes as input a conjunctive query Q and an $\mathcal{ELHI}$ TBox $\mathcal{T}$, and produces a rewriting Q' of Q w.r.t. $\mathcal{T}$. A distinguishing feature of our algorithm is that it exhibits "pay-as-you-go" behavior: if $\mathcal{T}$ is in $\mathcal{ELHI}$, $\mathcal{ELH}$ or $\mathcal{EL}$, then Q' is a datalog query, as in [15]; if $\mathcal{T}$ is in DL-Lite$^+$ [13], then Q' consists of a union of conjunctive queries and a linear datalog query; and, if $\mathcal{T}$ is in DL-Lite$_R$ or DL-Lite$_{core}$, then Q' is a union of conjunctive queries, as in [6]. Hence, our approach not only handles all the aforementioned sublanguages of $\mathcal{ELHI}$, but it is worst-case optimal w.r.t. data complexity for all of them, which makes it a generalization and an extension of the approaches of [15] and [6].

Our algorithm and the CGLLR algorithm mainly differ in the way existential quantification is handled. In our algorithm, functional terms are used to keep track of role successors. In contrast, the CGLLR algorithm deals with existential quantification by relying on a so-called reduction step. Moreover, unlike our algorithm, the CGLLR algorithm does not handle qualified existential quantification axioms natively; instead, a preliminary step is required where every such an axiom is replaced with a suitable encoding that introduces a fresh role. Our algorithm does not derive the queries produced by the reduction step and it handles qualified existential quantification natively; therefore, we conjecture that *our algorithm will often produce smaller rewritings than the CGLLR algorithm.*

In order to gather empirical evidence to support our conjecture, we implemented both algorithms in their *original* form—that is, we did not implement any optimization to reduce the size of the rewritings. We then compared the two implementations using a benchmark suite containing different DL-Lite$_R$ ontologies with TBoxes of varying complexity, and a set of test queries derived from applications that use these ontologies. Our results clearly indicate that the reduction step of the CGLLR algorithm can lead to a significant increase in the size of the rewritings. The implementation of our algorithm—called REQUIEM (REsolution-based QUery rewrIting for Expressive Models)—not only produced significantly smaller rewritings, but was also significantly faster in most cases. Finally, in order to determine the level of redundancy in the rewritings produced by our algorithm, we implemented an optimized version of REQUIEM, called REQUIEM-Full, that uses forward subsumption [5] and performs an a posteriori query subsumption check on the rewritings. REQUIEM produced the same rewritings as REQUIEM-full for 63% of the queries. Therefore, our empirical evaluation suggests that REQUIEM alone can be effectively used in practice in various scenarios.

[1] `http://www.dis.uniroma1.it/~quonto/`
[2] `http://pellet.owldl.com/owlgres/`

2 Preliminaries

For P an *atomic role*, a *basic role* has the form P or P^-. For A an *atomic concept* and R a basic role, an *antecedent concept* has the form A or $\exists R$; and a *consequent concept* has the form A, $\exists R$, or $\exists R.A$. A DL-Lite$_R$ *TBox* is a set of *inclusion assertions* or inclusion axioms of the form $A \sqsubseteq C$ or $R_1 \sqsubseteq R_2$, where A is an antecedent concept, C is a consequent concept, and R_1 and R_2 are basic roles. The former inclusion assertion is called a *concept inclusion*, and the latter, a *role inclusion*. An *ABox* is a set of *membership assertions* of the form $A(a)$ or $P(a, b)$, where A is an atomic concept, P is an atomic role, and a and b are constants. A DL-Lite$_R$ *Knowledge Base* (KB) $\mathcal{K}$ is a tuple $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a DL-Lite$_R$ TBox, and $\mathcal{A}$ is an ABox. The semantics of $\mathcal{K}$ is defined as usual [3].

We use the well-known definitions of constants, variables, function symbols, terms, and atoms of first-order logic [8]. 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*. Variables that occur in the body at most once and do not occur in the head are called *unbound* variables and may be denoted with the symbol _ ; all other variables are called *bound* variables. A Horn clause is *safe* if every variable occurring in the head also occurs in the body. The *depth* of a term t is defined as $\mathsf{depth}(t) = 0$ if t is a constant or a variable, and $\mathsf{depth}(f(s_1, ..., s_n)) = 1 + \max(\mathsf{depth}(s_i))$ for $1 \leq i \leq n$ if t is a functional term $f(s_1, ..., s_n)$. The notion of depth for atoms and Horn clauses is extended in the natural way. An atom D_i occurring in a Horn clause C is said to be *deepest* in C if $\mathsf{depth}(D_i) \geq \mathsf{depth}(D_j)$ for every body atom D_j of C. A *conjunctive query* over a DL KB $\mathcal{K}$ is a safe Horn clause whose head predicate does not occur in $\mathcal{K}$, and whose body is a set of atoms whose predicates are concept and role names occurring in $\mathcal{K}$. A *union of conjunctive queries* over $\mathcal{K}$ is a set of conjunctive queries over $\mathcal{K}$ with the same head up to variable renaming [4]. A tuple of constants $\vec{a}$ is a *certain answer* to a union of conjunctive queries Q over $\mathcal{K}$ if and only if $\mathcal{K} \cup Q \models Q_P(\vec{a})$, 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 [8]. The set of all answers to Q over $\mathcal{K}$ is denoted by $\mathsf{ans}(Q, \mathcal{K})$. Given a conjunctive query Q and a TBox $\mathcal{T}$, a query Q' is said to be a rewriting of Q w.r.t. $\mathcal{T}$ if $\mathsf{ans}(Q, \langle \mathcal{T}, \mathcal{A} \rangle) = \mathsf{ans}(Q', \mathcal{A})$ for every ABox $\mathcal{A}$.

3 Query Rewriting Algorithms

In this section we briefly present the two rewriting algorithms we consider in our empirical evaluation: the CGLLR algorithm and our resolution-based algorithm. Both algorithms transform a conjunctive query Q and a DL-Lite$_R$ TBox $\mathcal{T}$ into a union of conjunctive queries Q' that is a rewriting of Q w.r.t. $\mathcal{T}$.

The CGLLR Algorithm This algorithm uses the axioms of $\mathcal{T}$ as rewriting rules and applies them to the body atoms of Q. The rewriting rules are based on a partial function $\mathsf{ref}(D, \alpha)$ that takes as input an axiom $\alpha \in \mathcal{T}$ and a body atom D of Q. We define $\mathsf{ref}(D, \alpha)$ as follows:

Input: Conjunctive query Q, DL-Lite$_R$ TBox $\mathcal{T}$
$R = \{Q\}$;
repeat
  foreach *query* $Q' \in R$ **do**
    (reformulation) **foreach** *atom* D *in* Q' **do**
      foreach *axiom* $\alpha \in \mathcal{T}$ **do**
        if α *is applicable to* D **then**
          $R = R \cup \{Q'[D/\mathsf{ref}(D, \alpha)]\}$;
        end
      end
    end
    (reduction) **forall** *atoms* D_1, D_2 *in* Q' **do**
      if D_1 *and* D_2 *unify* **then**
        $\sigma = \mathsf{MGU}(D_1, D_2)$;
        $R = R \cup \{\lambda(Q'\sigma))\}$;
      end
    end
  end
until *no query unique up to variable renaming can be added to* R ;
return R;

Algorithm 1: The CGLLR algorithm

- 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 $\mathsf{ref}(D, \alpha) = S(y, x)$.

If $\mathsf{ref}(D, \alpha)$ is defined for α and D, we say that α is *applicable* to D.

The CGLLR algorithm is shown in Algorithm 1. Starting with the original query Q, the algorithm keeps producing queries in two steps until no other query unique up to variable renaming can be produced. In the *reformulation step* the algorithm rewrites the body atoms of a given query Q' by using applicable TBox axioms as rewriting rules, generating a new query for every atom reformulation. The expression $Q[D/D']$ denotes the conjunctive query obtained from Q by replacing the body atom D with a new atom D'. Then, in the *reduction step* the algorithm produces a new query $\lambda(Q'\sigma)$ for each pair of body atoms of Q' that unify. The function MGU takes as input two atoms and returns their most general unifier [4]. The function λ takes as input a conjunctive query Q and

Input: Conjunctive query Q, DL-Lite$_R$ TBox $\mathcal{T}$
$R = \Xi(\mathcal{T}) \cup \{Q\}$;
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 ;
return $\{C \mid C \in \mathsf{unfold}(\mathsf{ff}(R)),$ *and* C *has the same head predicate as* $Q\}$;

Algorithm 2: Our resolution-based algorithm

returns a new conjunctive query obtained by replacing each occurrence of an unbound variable in Q with the symbol _ . The reduction step is necessary to achieve completeness since an axiom that is not applicable to any body atom of Q' may be applicable to some body atom of $\lambda(Q'\sigma)$.

As a final remark, we point out that the CGLLR algorithm does not handle qualified existential quantification natively—that is, there is no rewriting rule for axioms of the form $A \sqsubseteq \exists R.B$. Instead, w.l.o.g. every axiom of such a form occurring in $\mathcal{T}$ is replaced with the set of axioms $\{A \sqsubseteq \exists P_1, \exists P_1^- \sqsubseteq B, P_1 \sqsubseteq R\}$, where P_1 is a new atomic role not occurring in $\mathcal{T}$.

Resolution-based Algorithm Our algorithm takes as input a conjunctive query Q and an $\mathcal{ELHI}$ TBox $\mathcal{T}$, and produces a datalog query that is a rewriting of Q w.r.t. $\mathcal{T}$. We have shown that if $\mathcal{T}$ is in DL-Lite$_R$, then the rewriting is a union of conjunctive queries. The algorithm first transforms Q and $\mathcal{T}$ into clauses, and then computes the rewriting by using a resolution-based calculus to derive new clauses from the initial set.

Our algorithm is presented in Algorithm 2, where we show the parts of the original algorithm that are relevant to DL-Lite$_R$ only. The rewriting is computed in four steps: clausification, saturation, unfolding, and pruning. The algorithm starts by transforming Q and $\mathcal{T}$ into a set of clauses $\Xi(\mathcal{T}) \cup \{Q\}$. The expression $\Xi(\mathcal{T})$ denotes the set of clauses obtained from $\mathcal{T}$ according to Table 1. Then, the algorithm keeps producing clauses in the *saturation step* until no other clause unique up to variable renaming can be produced. 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, $\mathsf{resolve}(C_1, C_2) = \emptyset$. After the saturation step, the resulting function-free clauses are unfolded. The function ff takes a set of clauses N and returns the set of function-free clauses of 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 $\mathsf{unfold}(N) = N \cup \{Q_P(x) \leftarrow B(x)\}$, which results by unfolding $A(x) \leftarrow B(x)$ into $Q_P(x) \leftarrow A(x)$. A formal description of the unfolding step can be found in [13]. In the last step, every clause that does not have the same head predicate as Q is dropped.

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

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

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

The Main Difference The presented techniques mainly differ in their handling of *existential quantification*: while our algorithm deals with axioms containing existential quantifiers on the right-hand side by introducing functional terms, the CGLLR algorithm does so by restricting the applicability of such axioms and relying on the reduction step. We explore this difference by means of an example (taken from [6]). Consider a DL-Lite$_R$ TBox $\mathcal{T}$ that consists of the following axioms:

$$\text{Professor} \sqsubseteq \exists \text{teaches}, \tag{1}$$

$$\exists \text{teaches}^- \sqsubseteq \text{Student}. \tag{2}$$

The TBox $\mathcal{T}$ states that a professor teaches at least someone, and that someone that is taught is a student. Consider the query

$$Q(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y). \tag{3}$$

We first analyze the execution of the CGLLR algorithm. In the first iteration, the axiom (1) is not applicable to teaches(x, y) because teaches(x, y) has more than one bound variable. The reason why the applicability of (1) has to be restricted in this case is that the CGLLR algorithm does not keep track of information about role successors. In fact, naively allowing axioms of the form of (1) to be applicable in such a case would result in the loss of soundness. To illustrate this point, suppose that (1) were applicable to teaches(x, y) and $\mathsf{ref}(\text{teaches}(x, y), (1)) = \text{Professor}(x)$; the algorithm would then obtain

$$Q(x) \leftarrow \text{Professor}(x) \wedge \text{Student}(y). \tag{4}$$

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 (4). In particular, if $\mathcal{T}$ were to contain axiom (1) only, then query (4) would clearly produce wrong answers.

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.

Although the applicability of (1) is restricted, the axiom (2) is applicable to Student(y) in (3), producing

$$Q(x) \leftarrow \text{teaches}(x,y) \wedge \text{teaches}(\_\,, y). \tag{5}$$

In the next iteration, neither (1) nor (2) are applicable to any body atom of (5), so no query is added in the reformulation step. In the reduction step, however, the algorithm produces

$$Q(x) \leftarrow \text{teaches}(x, \_\,), \tag{6}$$

by unifying the body atoms of (5). In the following iteration, the axiom (1) is applicable to the only body atom of (6), producing

$$Q(x) \leftarrow \text{Professor}(x). \tag{7}$$

Note that without the reduction step, the algorithm would not have produced query (7). It can easily be verified that no more queries unique up to variable renaming can be produced; thus, the algorithm returns $\{(3), (5), (6), (7)\}$.

We now analyze the execution of our algorithm. The axioms (1) and (2) are translated into the following clauses:

$$\text{teaches}(x, f(x)) \leftarrow \text{Professor}(x), \tag{8}$$
$$\text{Student}(x) \leftarrow \text{teaches}(y, x). \tag{9}$$

In the saturation step the algorithm produces

$$\text{Student}(f(x)) \leftarrow \text{Professor}(x), \quad (\mathsf{resolve}((8),(9))) \tag{10}$$
$$Q(x) \leftarrow \text{Professor}(x) \wedge \text{Student}(f(x)), \quad (\mathsf{resolve}((3),(8))) \tag{11}$$
$$Q(x) \leftarrow \text{teaches}(x, f(x)) \wedge \text{Professor}(x), \quad (\mathsf{resolve}((3),(10))) \tag{12}$$
$$Q(x) \leftarrow \text{Professor}(x). \quad (\mathsf{resolve}((8),(12))) \tag{13}$$

Note the difference between queries (4) and (11). Since the function symbol f is uniquely associated with clause (8), unlike query (4), query (11) 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, $\mathsf{ff}(R) = \{(3), (9), (13)\}$; therefore, we have that $\mathsf{unfold}(\mathsf{ff}(R))$ consists of $\mathsf{ff}(R)$ and the query

$$Q(x) \leftarrow \text{teaches}(x, y) \wedge \text{teaches}(z, y), \tag{14}$$

which results from unfolding (9) into (3). Finally, since the clause (9) does not have the same head predicate as query (3), our algorithm returns $\{(3), (13), (14)\}$.

The use of functional terms is what mainly distinguishes our algorithm from the CGLLR algorithm. This distinctive feature makes our approach more goal-oriented, in the sense that it does not need to derive the queries produced by the reduction step of the CGLLR algorithm in order to be complete. Moreover, we have shown that every clause containing functional terms produced in the saturation step of our algorithm can be safely dropped. Furthermore, our algorithm handles qualified existential quantification natively, so it does not need to introduce auxiliary roles. These are the main reasons why we conjecture that our algorithm will often produce smaller rewritings than the CGLLR algorithm.

4 Evaluation

The main goal of the evaluation is to compare the algorithms described in Section 3 w.r.t. the size of the rewritings they produce. Since a rewriting containing fewer queries than another is not necessarily less complex, we consider the size of a rewriting as being the number of symbols needed to represent it in the standard datalog notation. In order to give an indication of likely performance, we also present the running time of both implementations; we point out, however, that no special care was taken to obtain time efficient implementations and that the times reported correspond to the computation of the rewritings only (we did not evaluate the computed rewritings).

The implementation of our algorithm is called REQUIEM and we refer to our implementation of the CGLLR algorithm as C. Both implementations are available at REQUIEM's Web site[3]. All tests were performed on a desktop computer with a 2.59 GHz Intel processor, 1.87 GB of RAM, running Windows XP. We used Java 1.6.0 Update 7 with a maximum heap size of 256 MB.

Test Ontologies and Queries We selected test DL-Lite$_R$ ontologies with varying numbers of axioms containing existential quantification. We considered ontologies that were developed in the context of real projects. The queries we used were selected based on canonical examples used in the corresponding project.

VICODI is an ontology developed in the EU-funded project of the same name,[4] that captures information about European history contextualized w.r.t. location, time and subject. STOCKEXCHANGE is an ontology developed for ontology-based data access [14], that represents the domain of financial institutions of the European Union. UNIVERSITY is a DL-Lite$_R$ version of LUBM[5]—a benchmark developed at Lehigh University for testing the performance of ontology management and reasoning systems; the ontology describes the organizational structure of universities. ADOLENA (Abilities and Disabilities OntoLogy for ENhancing Accessibility) is an ontology developed for ontology-based data access for the South African National Accessibility Portal [11]; the ontology describes abilities, disabilities, and devices. We decided to include two synthetic ontologies in our tests in order to have a controlled scenario to help us understand the impact of the reduction step of the CGLLR algorithm. PATH1 and PATH5 model information about graphs: nodes are represented by individuals, and vertices are ABox assertions of the form edge(a, b). The ontology PATH5 contains concepts representing paths of length 1–5, while PATH1 contains a concept representing paths of length 1 only. All the ontologies and queries are available at REQUIEM's Web site.

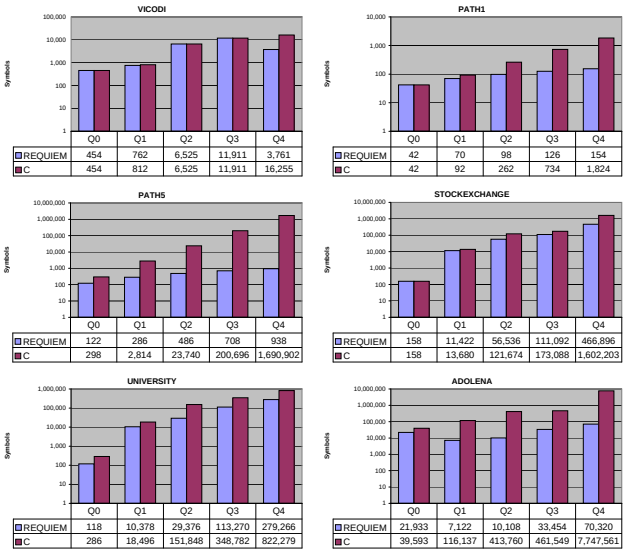
Results Figure 1 shows the size of the rewritings produced by REQUIEM and C for the different considered ontologies and queries. Figure 2 shows the time it took the implementations to compute the rewritings.

As can be seen in Figure 1, the rewritings produced by REQUIEM were never larger than those produced by C and were often significantly smaller. The most extreme case was the fifth query over ADOLENA, in which REQUIEM produced 624 queries, as opposed to the more than 78,500 produced by C. We compared the actual rewritings in order to verify whether for every query Q_R produced by REQUIEM, there was an equivalent query up to variable renaming Q_P produced by C. It turned out that this was the case for all of our tests.

A closer inspection of the rewritings produced by C revealed that most of them contained a considerable number of queries produced in the reduction step of the algorithm. The impact of the reduction step can be clearly seen in the tests involving PATH1 and PATH5. The queries we used with these two ontologies are

[3] http://www.comlab.ox.ac.uk/projects/requiem/
[4] http://www.vicodi.org/
[5] http://swat.cse.lehigh.edu/projects/lubm/

Fig. 1. Size of the rewritings

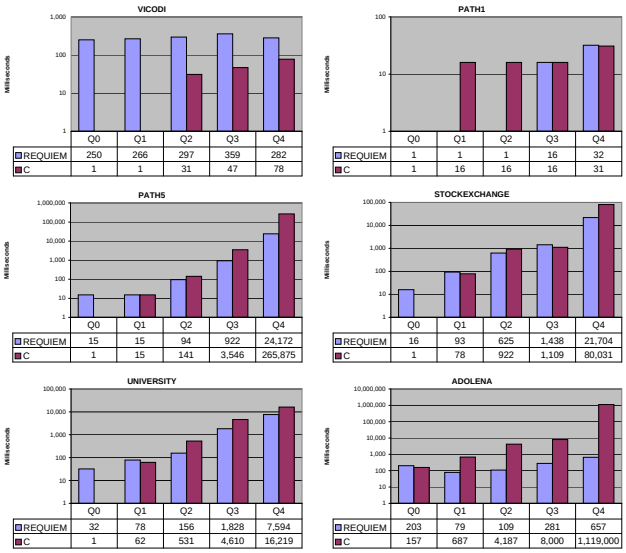


of the form $Q_n(x_0) \leftarrow \text{edge}(x_0, x_1) \wedge ... \wedge \text{edge}(x_n, x_{n+1})$, for $0 \leq n \leq 4$. Clearly, every pair of body atoms in queries of this form unify; therefore, for every Q' with m body atoms, C will produce $\binom{m}{2}$ new queries in the reduction step. The most extreme case for these ontologies was the fifth query over PATH5, in which REQUIEM produced 16 queries, as opposed to the more than 25,000 produced by C. Note that for every query Q'' produced from a query Q' in the reduction step, there is a substitution σ such that $Q'\sigma = Q''$, in which case we say that Q' subsumes Q''. It is well known that every such query Q'' can be dropped from the rewriting without sacrificing completeness [8]. Identifying such queries, however, is not straightforward since the CGLLR algorithm does not keep track of which queries were produced in the reduction step. Notably, several of the rewritings produced by C contained a considerable number of queries containing the auxiliary roles introduced to simulate axioms of the form $A \sqsubseteq \exists R.B$ (see Section 3). We believe that the CGLLR algorithm could benefit if such axioms were handled natively. To the best of our knowledge, however, there does not exist a version of the CGLLR algorithm that does so.

The results shown in Figure 2 suggest that REQUIEM will be significantly faster than C in case the queries are relatively complex and/or the ontologies contain a relatively large number of existential quantification axioms. The most extreme case was again the fifth query over ADOLENA, in which REQUIEM took a little over half a second, as opposed to the almost 20 minutes taken by C.

Finally, in order to determine the level of redundancy in the rewritings produced by our algorithm, we implemented REQUIEM-Full—an optimized version of REQUIEM that uses forward subsumption [5] and performs an a posteriori

Fig. 2. Running time



query subsumption check on the rewritings. For every clause C produced in the saturation step, the forward subsumption optimization verifies whether there is a previously produced clause C' such that C' subsumes C. If this is the case, then C is not added to the set of produced clauses. The query subsumption check takes the rewriting after the pruning step and gets rid of every clause C for which there is another clause C' such that C' subsumes C. Note that in the case of the CGLLR algorithm, there always is a previously produced query that subsumes every query produced in the reduction step; therefore, forward subsumption would get rid of every query produced in the reduction step on the fly, which would compromise completeness. Moreover, given the size of the rewritings, a straightforward optimization based on an a posteriori query subsumption check may be impractical. REQUIEM produced the same rewritings as REQUIEM-Full for 63% of the queries. This suggests that REQUIEM alone can be used effectively in practice in various scenarios.

5 Future Work

We plan to develop optimization techniques for REQUIEM in order to reduce or eliminate recursion when possible in the computed rewritings. Additionally, we plan to use REQUIEM in an ontology-based data access system using existing database technology in order to evaluate the practicality of our approach for query answering. Finally, we plan to extend REQUIEM to be a reasoner for OWL 2 QL.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the EL Envelope. In *International Joint Conferences on Artificial Intelligence (IJCAI-05)*, 2005.
3. F. Baader and W. Nutt. *Basic Description Logics*, chapter 2, pages 47–100. Cambridge University Press, 2003.
4. F. Baader and W. Snyder. Unification Theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.
5. L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–100. North Holland, 2001.
6. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *J. of Automated Reasoning*, 2007.
7. D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description Logic Framework for Information Integration. In *Principles of Knowledge Representation and Reasoning*, pages 2–13, 1998.
8. C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., Orlando, FL, USA, 1997.
9. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, pages 207–224, 2003.
10. J. Heflin and J. Hendler. A portrait of the semantic web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.
11. C. M. Keet, R. Alberts, A. Gerber, and G. Chimamiwa. Enhancing web portals with ontology-based data access: The case study of south africa's accessibility portal for people with disabilities. In *OWLED*, 2008.
12. M. Lenzerini. Data Integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.
13. H. Pérez-Urbina, B. Motik, and I. Horrocks. Rewriting Conjunctive Queries under Description Logic Constraints. In *Proceedings of the International Workshop on Logics in Databases*, May 2008.
14. M. Rodriguez-Muro, L. Lubyte, and D. Calvanese. Realizing ontology based data access: A plug-in for protégé. In *Proc. of the Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008)*, pages 286–289. IEEE Computer Society Press, 2008.
15. R. Rosati. On conjunctive query answering in EL. In *Proceedings of the 2007 International Workshop on Description Logics (DL2007)*, CEUR-WS, 2007.
16. R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.
17. J. Widom. Research Problems in Data Warehousing. In *4th International Conference on Information and Knowledge Management*, pages 25–30, Baltimore, Maryland, 1995.