Practical Aspects of Query Rewriting for OWL 2

Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik

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

Abstract. Query answering for the QL profile of OWL 2 and a substantial fragment of the EL profile can be implemented via query rewriting. In our previous work, we presented RQR—a rewriting algorithm for OWL QL that can also deal with most of the EL profile. In order to test the likely practicality of RQR, we have implemented it in a query rewriting system that we call REQUIEM. A recent empirical evaluation of REQUIEM, in which we considered OWL 2 QL ontologies, indicates that it produces significantly smaller rewritings than existing approaches in most cases. However, our results suggest that typical queries over realistic ontologies can still lead to very large rewritings (e.g., containing many thousands of queries). In this paper, we describe query rewriting, briefly present the results of our empirical evaluation, and discuss various optimization techniques aimed at reducing the size of the rewritings. We additionally discuss the consequences of rewriting queries w.r.t. OWL 2 EL ontologies and present results from a preliminary empirical evaluation of REQUIEM in which we consider realistic OWL 2 EL ontologies.

1 Introduction

There are several advantages to the use of an ontology with a data repository. On the one hand, the ontology can be used as a conceptual schema in order to provide an intuitive and unified view over one or more repositories, allowing queries to be independent of the structure and location of the data; on the other hand, data repositories typically provide persistence and efficient query answering over large volumes of (instance) data. The use of ontologies as conceptual schemas has been extensively studied in the context of applications, such as information integration [3]. The use of data repositories to store instance data is becoming increasingly important due to the widespread use of ontologies and the scalability requirements of many applications, such as the semantic Web.

In OWL 2—a new version of the OWL ontology language that recently became a W3C candidate recommendation—scalability requirements are addressed by *profiles*: subsets of the language that enjoy desirable computational properties. The OWL 2 QL profile has been specifically designed for answering queries via *query rewriting*: conjunctive queries posed against an OWL 2 QL ontology and a set of instance data stored in a data repository can be answered by first using the ontology to rewrite the query and then delegating the evaluation of the rewritten query to the data repository. We describe the whole process in Section

Proceedings of OWL: Experiences and Directions 2009 (OWLED 2009), Rinke Hoekstra, editor.
http://www.webont.org/owled/2009

2. We will 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. [2]. The same authors proposed a query rewriting algorithm, which we will refer to as CGLLR, that transforms a conjunctive query Q and a DL-Lite$_R$ ontology $\mathcal{O}$ into a *union of conjunctive queries* (UCQ) $Q_\mathcal{O}$ such that the answers to Q over $\mathcal{O}$ and any set of instance data $\mathcal{A}$ can be obtained by evaluating $Q_\mathcal{O}$ over $\mathcal{A}$ only. CGLLR is used in reasoners such as QuOnto[1] and 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 $\mathcal{O}$. This means that, on the one hand, $Q_\mathcal{O}$ may be costly to compute, and, on the other hand, its evaluation by current database systems may be costly or even unfeasible. Trying to produce smaller 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 [7] 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, called RQR (Resolution-based Query Rewriting), takes as input a conjunctive query Q and an $\mathcal{ELHIO}^\neg$ ontology $\mathcal{O}$, and uses a resolution-based calculus to produce a rewritten query $Q_\mathcal{O}$. Although $Q_\mathcal{O}$ will, in general, be a (possibly recursive) *datalog query* (DQ)—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 UCQ; the algorithm can therefore be seen as a generalization and extension of CGLLR. An advantage of using RQR as opposed to CGLLR is that, in addition to OWL 2 QL, RQR can handle most of the EL profile.

In order to test the likely practicality of query rewriting and the performance of the different rewriting techniques for DL-Lite$_R$, we implemented RQR in a query rewriting system that we call REQUIEM[3] (REsolution-based QUery rewrIting for Expressive Models). We recently conducted an empirical evaluation [6] in which we compared REQUIEM to an implementation of CGLLR. 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. REQUIEM often produced significantly smaller rewritings than its counterpart; however, our results show that, even when using REQUIEM, typical queries over realistic ontologies can lead to very large rewritings (e.g., containing many thousands of queries).

Both algorithms are clearly be amenable to optimizations aimed at reducing the size of the rewritings. One obvious optimization would be to use query subsumption checks to eliminate redundant conjunctive queries from the rewriting.

[1] `http://www.dis.uniroma1.it/~quonto/`
[2] `http://pellet.owldl.com/owlgres/`
[3] `http://www.comlab.ox.ac.uk/projects/requiem/`

Our empirical evaluation showed that the query subsumption check can significantly reduce the size of the rewritings, and that the optimized versions of RQR and CGLLR produce very similar rewritings. However, the resulting rewritings can still be very large (e.g., containing many hundreds of queries). In order to address this problem, in Section 3 we describe various optimizations that can be used in order to further reduce the size of the rewritings.

Finally, in Section 4 we go beyond OWL 2 QL and discuss the consequences of using RQR to rewrite queries w.r.t. OWL 2 EL ontologies. We present a further optimization that can be used to reduce the size of the rewritings in case they are DQs. Moreover, we present a simple procedure that can be used to transform non-recursive DQs into UCQs. We have conducted a preliminary empirical evaluation of REQUIEM in which we consider typical queries posed over realistic OWL 2 EL ontologies. Our results suggest that in many cases the rewritings can be transformed into relatively small UCQs, which means that it will often be possible to use a database system for answering conjunctive queries even over ontologies that go beyond OWL 2 QL in realistic scenarios.

2 Ontology-based Data Access via Query Rewriting

We now describe how to answer queries posed over an OWL 2 QL ontology and a database using query rewriting. We illustrate the process by means of an example. Suppose we have a relational database DB containing a table `Professor` with attributes name and department; and a table `Student` with attributes name, major, 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 OWL 2 QL ontology $\mathcal{O}$ to describe DB:[4]

$$\begin{aligned} \text{Professor} &\sqsubseteq \exists \text{teaches} \\ \exists \text{teaches} &\sqsubseteq \text{Teacher} \\ \exists \text{hasTutor}^{-} &\sqsubseteq \text{Professor} \end{aligned}$$

The ontology $\mathcal{O}$ states that professors teach at least someone, that the domain of the property teaches is Teacher, and 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. 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 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 $\mathcal{M}$ between $\mathcal{O}$ and DB is defined as follows:

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

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

Queries posed over the ontology are answered in two steps: first, the ontology is used to rewrite the query into a UCQ—so-called *rewriting*—and second, the mappings are used to transform the rewriting into an SQL query that is then evaluated using the database system where the instance data resides. Intuitively, the rewriting is an expanded query that incorporates the knowledge encoded in the ontology that is relevant for answering the original query. Consider, for example, the query $Q = Q(x) \leftarrow \text{Teacher}(x)$. A piece of relevant information encoded in $\mathcal{O}$ for answering Q is, for instance, that all professors are teachers; therefore, the rewriting $Q_\mathcal{O}$ of Q w.r.t. $\mathcal{O}$ should reflect this fact. In particular, $Q_\mathcal{O}$ should retrieve instances of Professor as well as instances of Teacher.

There are currently two main algorithms that can be used to compute the rewriting of a query w.r.t. an OWL 2 QL ontology: CGLLR and RQR. Although the algorithms compute the rewritings quite differently—CGLLR uses the axioms of the ontology as 'rewriting' rules, whereas RQR employs a resolution-based calculus—both algorithms are guaranteed to produce UCQs when rewriting queries w.r.t. OWL 2 QL ontologies. Given the inputs Q and $\mathcal{O}$ as above, either algorithm will produce the following rewriting $Q_\mathcal{O}$:

$$Q(x) \leftarrow \text{Teacher}(x) \tag{1}$$
$$Q(x) \leftarrow \text{teaches}(x, y) \tag{2}$$
$$Q(x) \leftarrow \text{Professor}(x) \tag{3}$$
$$Q(x) \leftarrow \text{hasTutor}(y, x) \tag{4}$$

Once $Q_\mathcal{O}$ has been computed, we can proceed to evaluate it over the database DB. In order to do so, we need to transform the rewriting into an SQL query. Transforming $Q_\mathcal{O}$ into an SQL query $\mathsf{sql}(Q_\mathcal{O})$ basically amounts to using the mappings $\mathcal{M}$ to replace each class or property D occurring in a query contained in $Q_\mathcal{O}$ with the corresponding SQL query Q_D, and forming the union of the resulting queries. Note that in this case, $\mathcal{M}$ does not contain a mapping for every class and property of $\mathcal{O}$. 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. In this case, queries (1) and (2) can be discarded. It is easy to see that, as a result, $\mathsf{sql}(Q_\mathcal{O})$ is

```
SELECT Name FROM Professor UNION SELECT Tutor FROM Student.
```

Finally, the evaluation of $\mathsf{sql}(Q_\mathcal{O})$ is delegated to the database system where DB resides.

3 Query Rewriting in Practice

Calvanese et al. showed that the size of the rewriting $Q_\mathcal{O}$ of a query Q w.r.t. an OWL 2 QL ontology $\mathcal{O}$ is worst-case exponential w.r.t. the size of Q and $\mathcal{O}$ [2]. Consider, for instance, the ontology $\mathcal{O} = \{R_1 \sqsubseteq R_2, R_2 \sqsubseteq R_3, ..., R_{n-1} \sqsubseteq R_n\}$ and the query $Q = Q(x_0) \leftarrow R_n(x_0, x_1) \wedge R_n(x_1, x_2) \wedge ... \wedge R_n(x_{m-1}, x_m)$.

It can be shown that the rewriting $Q_{\mathcal{O}}$ of Q w.r.t. $\mathcal{O}$ will contain n^m queries. As the example shows, queries containing classes or properties with many subsumers can lead to large rewritings. This means that, on the one hand, $Q_{\mathcal{O}}$ may be costly to compute, and, on the other hand, the evaluation of $\mathsf{sql}(Q_{\mathcal{O}})$ by existing database systems may be costly or even unfeasible.

One obvious optimization that can help to reduce the size of the rewritings is based on the notion of *query subsumption*. We say that a query Q_1 *subsumes* another query Q_2 if there is a substitution σ such that $Q_1\sigma \subseteq Q_2$, where both queries are regarded as *Horn clauses* [4]. The query subsumption optimization consists in checking subsumption between pairs of queries in $Q_{\mathcal{O}}$ and eliminating every clause that is subsumed by another.

In order to test the likely practicality of query rewriting, we recently conducted an empirical evaluation of REQUIEM—our implementation of RQR—in which we compare it to an implementation of CGLLR [6]. The results of our evaluation indicate that, while RQR produced significantly smaller rewritings than the CGLLR algorithm in most cases, with the query subsumption optimization the two techniques produced almost identical rewritings. Unfortunately, our evaluation showed that, even when using query subsumption, the rewritings can be extremely large. Therefore, is it vital to devise other optimization techniques to further reduce the size of the rewritings.

Other optimizations similar to query subsumption are the well-known *forward and backward subsumption* [1]. Both techniques compare each new clause C_1 produced in the saturation step with the set of previously generated clauses. In forward subsumption, C_1 is discarded if the set of clauses already contains a clause C_2 that subsumes C_1 under the multiset semantics; in backward subsumption, C_1 is removed from the set of clauses if it is subsumed by C_2 under the multiset semantics. Since RQR is based on a resolution calculus, these optimizations can be straightforwardly applied without affecting completeness [1]. In the case of CGLLR, however, forward subsumption renders the algorithm incomplete [6] and it is not clear whether backward subsumption can be applied to CGLLR without affecting completeness.

Once the rewriting has been computed, we may still be able to further reduce it by exploiting the information contained in the mappings. As explained in Section 2, the answer to any query in $Q_{\mathcal{O}}$ containing an atom for which there is no mapping will necessarily be empty; therefore, such queries can be safely discarded before computing $\mathsf{sql}(Q_{\mathcal{O}})$. We believe that, in practice, it is likely that the set of mappings does not contain a mapping for every class and property occurring in the ontology. In these cases, we expect that pruning $Q_{\mathcal{O}}$ w.r.t. the mappings would significantly reduce the size of $\mathsf{sql}(Q_{\mathcal{O}})$, hopefully producing an SQL query of manageable size.

4 Going Beyond OWL 2 QL

An advantage of using RQR as opposed to CGLLR is that it can handle fragments of OWL 2 that go beyond the QL profile. In fact, RQR supports ontologies

expressed in $\mathcal{ELHIO}^{\neg}$—a description logic that captures most of the EL profile of OWL 2. Unlike DL-Lite$_R$, the DL $\mathcal{ELHIO}^{\neg}$ allows for axioms containing conjunction on the l.h.s. (e.g., Student $\sqcap$ $\exists$hasSupervisor $\sqsubseteq$ GraduateStudent), qualified existential restrictions on the l.h.s. (e.g., $\exists$studies.Course $\sqsubseteq$ Student), and nominals—classes that are to be interpreted as *singletons* or sets with only one element (e.g., OxfordStudent $\sqsubseteq$ $\exists$studiesAt.{OxfordUniversity}).

When dealing with OWL 2 EL ontologies, however, RQR is no longer guaranteed to produce UCQs; instead, the rewriting $Q_{\mathcal{O}}$ might be a possibly recursive DQ. In order to understand why this is so, consider the following example. Suppose we want to rewrite the query $Q = Q(x) \leftarrow \text{Student}(x)$ w.r.t. an OWL 2 EL ontology $\mathcal{O}$ containing the axiom

$$\exists\text{hasClassmate.Student} \sqsubseteq \text{Student}, \tag{5}$$

which states that anybody who has a classmate who is a student is also a student. It can be shown that RQR will produce the following rewriting $Q_{\mathcal{O}}$:

$$Q(x) \leftarrow \text{Student}(x) \tag{6}$$

$$\text{Student}(x) \leftarrow \text{hasClassmate}(x, y) \wedge \text{Student}(y) \tag{7}$$

Intuitively, the reason RQR produced such a DQ is that clause (7) is recursive. In order to evaluate such a DQ, we require a *deductive* database system. Existing deductive database systems, such as IRIS[5] or XSB[6], allow for the specification of external data sources. Therefore, assuming the data resides in a data repository DB, the mappings between $\mathcal{O}$ and DB can be used to specify an external data source in the deductive database system in order for $Q_{\mathcal{O}}$ to be evaluated.

A straightforward optimization to reduce the size of $Q_{\mathcal{O}}$ before evaluation is based on the so-called *dependency graph*. We define the dependency graph $\mathcal{G}(Q_{\mathcal{O}})$ of $Q_{\mathcal{O}}$ as follows: every predicate P occurring in $Q_{\mathcal{O}}$ is a node in $\mathcal{G}(Q_{\mathcal{O}})$ and there is an edge from a predicate P_1 to a predicate P_2 if there is a clause $C \in Q_{\mathcal{O}}$ such that P_1 occurs in the head of C and P_2 occurs in the body of C or vice versa. We can use $\mathcal{G}(Q_{\mathcal{O}})$ to identify and discard the set of clauses in $Q_{\mathcal{O}}$ that are *unreachable* from the query predicate of $Q_{\mathcal{O}}$ as such clauses are clearly irrelevant to answer $Q_{\mathcal{O}}$. Consider, for example, an OWL 2 EL ontology $\mathcal{O}$ consisting of the axioms

$$\exists\text{studies.Course} \sqsubseteq \text{Student},$$

$$\exists\text{hasTutor}^{-} \sqsubseteq \text{Professor},$$

that state that anybody that studies a course is a student and that the range of the property hasTutor is Professor; and the query $Q = Q(x) \leftarrow \text{Professor}(x)$. On input Q and $\mathcal{O}$, RQR will produce the following rewriting $Q_{\mathcal{O}}$:

$$\text{Student}(x) \leftarrow \text{studies}(x, y) \wedge \text{Course}(y) \tag{8}$$

$$Q(x) \leftarrow \text{Professor}(x) \tag{9}$$

$$Q(x) \leftarrow \text{hasTutor}(y, x) \tag{10}$$

[5] http://www.iris-reasoner.org/

[6] http://xsb.sourceforge.net/

```
Input: DQ Q_O
Q'_O = Q_O;
foreach IDB predicate P occurring in Q_O except the query predicate do
    C_P = {C | C ∈ Q'_O with head predicate P};
    foreach clause C ∈ C_P do
        if P occurs in the body of C then
            return Q'_O;
        end
    end
    foreach clause C ∈ C_P do
        Unfold C into every clause in Q'_O;
        Q'_O = Q'_O \ {C};
    end
end
return Q'_O;
```

Algorithm 1: Greedy Unfolding

It can be readily verified that clause (8) is not reachable from the query predicate of $Q_{\mathcal{O}}$ in $\mathcal{G}(Q_{\mathcal{O}})$; therefore, we can deduce that such a clause is not relevant for answering $Q_{\mathcal{O}}$ and discard it.

Even though RQR is not guaranteed to produce UCQs for OWL 2 EL, examination of a large corpus of ontologies [5] suggests that in many realistic cases ontologies do not contain (or imply) cyclic axioms such as (5), which might allow us to transform $Q_{\mathcal{O}}$ into a UCQ even for OWL 2 EL ontologies. Suppose, for example, that RQR computed the following rewriting $Q_{\mathcal{O}}$:

$$Q(x) \leftarrow \text{Teacher}(x) \tag{11}$$

$$\text{Teacher}(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y) \tag{12}$$

It is not difficult to see that by *unfolding* (12) into (11), such a datalog query can be transformed into the following UCQ:

$$Q(x) \leftarrow \text{Teacher}(x)$$
$$Q(x) \leftarrow \text{teaches}(x, y) \wedge \text{Student}(y)$$

A simple procedure for transforming non-recursive DQs into UCQs is shown in Algorithm 1. Soundness of this procedure follows from [7]. As can be seen, the idea is to unfold and discard non-recursive non-query clauses. Clearly, if we are able to unfold and discard every non-query clause in the original DQ, in the end we will obtain a UCQ.

We have recently conducted a preliminary empirical evaluation of our system REQUIEM—enhanced with the greedy unfolding procedure and the optimizations described in this paper—in which we consider real OWL 2 EL ontologies. Our test data includes the well-known ontology NCI[7] and an $\mathcal{ELHI}$ version of the

[7] `http://www.mindswap.org/2003/CancerOntology/`

university bechmark ontology developed at Lehigh University[8]. Our first results suggest that REQUIEM will often perform well and produce small rewritings in practice, especially when dealing with relatively small ontologies. Additionally, our evaluation suggests that we can often use the greedy unfolding procedure in order to obtain relatively small UCQs. This is an encouraging result since it means that we will often be able to use an off-the-shelf database system for answering conjunctive queries even over ontologies that go beyond OWL 2 QL in realistic scenarios.

5 Future Work

We plan to implement an OWL 2 QL ontology-based data access system using REQUIEM. Based on our results, we expect such a system to be useful for answering many realistic queries even over OWL 2 EL ontologies. Moreover, we expect the system to perform well both w.r.t. the size of the rewritings and the time needed to compute them; its practicality 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 the optimization that uses the mappings to prune irrelevant queries (see Section 3) 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. Additionally, we plan to extend the system to support all of OWL 2 QL, which mainly involves adding support for datatypes.

References

1. 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.
2. 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.
3. 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.
4. M. Fitting. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
5. T. Gardiner, D. Tsarkov, and I. Horrocks. Framework for an automated comparison of description logic reasoners. volume 4273, pages 654–667, 2006.
6. H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient Query Answering for OWL 2. In *Proceedings of the 8th International Semantic Web Conference (ISWC 2009), Chantilly, Virginia, USA*, 2009.
7. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable Query Answering and Rewriting under Description Logic Constraints. *Journal of Applied Logic*, 2009. To appear.

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